

# The Principles of Diffusion Models

## From Origins to Advances

---

**Chieh-Hsin Lai**  
Sony AI

**Yang Song**  
OpenAI

**Dongjun Kim**  
Stanford University

**Yuki Mitsufuji**  
Sony Corporation, Sony AI

**Stefano Ermon**  
Stanford University

# Contents

---

<b>Acknowledgements</b>	<b>3</b>
<b>A Introduction to Deep Generative Modeling</b>	<b>14</b>
<b>1 Deep Generative Modeling</b>	<b>15</b>
1.1 What is Deep Generative Modeling? . . . . .	16
1.2 Prominent Deep Generative Models . . . . .	22
1.3 Taxonomy of Modelings . . . . .	26
1.4 Closing Remarks . . . . .	28
<b>B Origins and Foundations of Diffusion Models</b>	<b>30</b>
<b>2 Variational Perspective: From VAEs to DDPMs</b>	<b>32</b>
2.1 Variational Autoencoder . . . . .	33
2.2 Variational Perspective: DDPM . . . . .	43
2.3 Closing Remarks . . . . .	55
<b>3 Score-Based Perspective: From EBMs to NCSN</b>	<b>56</b>
3.1 Energy-Based Models . . . . .	57
3.2 From Energy-Based to Score-Based Generative Models . . . . .	64
3.3 Denoising Score Matching . . . . .	68
3.4 Multi-Noise Levels of Denoising Score Matching (NCSN) . . . . .	79
3.5 Summary: A Comparative View of NCSN and DDPM . . . . .	84

3.6	Closing Remarks . . . . .	85
<b>4</b>	<b>Diffusion Models Today: Score SDE Framework</b>	<b>86</b>
4.1	Score SDE: Its Principles . . . . .	87
4.2	Score SDE: Its Training and Sampling . . . . .	105
4.3	Instantiations of SDEs . . . . .	110
4.4	(Optional) Rethinking Forward Kernels in Score-Based and Variational Diffusion Models . . . . .	115
4.5	(Optional) Fokker–Planck Equation and Reverse-Time SDEs via Marginalization and Bayes' Rule . . . . .	121
4.6	Closing Remarks . . . . .	126
<b>5</b>	<b>Flow-Based Perspective: From NFs to Flow Matching</b>	<b>127</b>
5.1	Flow-Based Models: Normalizing Flows and Neural ODEs . . . . .	129
5.2	Flow Matching Framework . . . . .	136
5.3	Constructing Probability Paths and Velocities Between Distributions	148
5.4	(Optional) Properties of the Canonical Affine Flow . . . . .	159
5.5	Closing Remarks . . . . .	165
<b>6</b>	<b>A Unified and Systematic Lens on Diffusion Models</b>	<b>166</b>
6.1	Conditional Tricks: The Secret Sauce of Diffusion Models . . . . .	168
6.2	A Roadmap for Elucidating Training Losses in Diffusion Models	170
6.3	Equivalence in Diffusion Models . . . . .	175
6.4	Beneath It All: The Fokker–Planck Equation . . . . .	186
6.5	Closing Remarks . . . . .	190
<b>7</b>	<b>(Optional) Diffusion Models and Optimal Transport</b>	<b>191</b>
7.1	Prologue of Distribution-to-Distribution Translation . . . . .	192
7.2	Taxonomy of the Problem Setups . . . . .	194
7.3	Relationship of Variant Optimal Transport Formulations . . . . .	206
7.4	Is Diffusion Model's SDE Optimal Solution to SB Problem? . . . . .	212
7.5	Is Diffusion Model's ODE an Optimal Map to OT Problem? . . . . .	216
<b>C</b>	<b>Sampling of Diffusion Models</b>	<b>224</b>
<b>8</b>	<b>Guidance and Controllable Generation</b>	<b>226</b>
8.1	Prologue . . . . .	227
8.2	Classifier Guidance . . . . .	232
8.3	Classifier-Free Guidance . . . . .	235

8.4	(Optional) Training-Free Guidance . . . . .	238
8.5	From Reinforcement Learning to Direct Preference Optimization for Model Alignment . . . . .	243
8.6	Closing Remarks . . . . .	253
<b>9</b>	<b>Sophisticated Solvers for Fast Sampling</b>	<b>254</b>
9.1	Prologue . . . . .	255
9.2	DDIM . . . . .	263
9.3	DEIS . . . . .	275
9.4	DPM-Solver . . . . .	282
9.5	DPM-Solver++ . . . . .	295
9.6	PF-ODE Solver Families and Their Numerical Analogues . . . . .	301
9.7	(Optional) DPM-Solver-v3 . . . . .	304
9.8	(Optional) ParaDiGMs . . . . .	315
9.9	Closing Remarks . . . . .	321
<b>D</b>	<b>Toward Learning Fast Diffusion-Based Generators</b>	<b>322</b>
<b>10</b>	<b>Distillation-Based Methods for Fast Sampling</b>	<b>323</b>
10.1	Prologue . . . . .	324
10.2	Distribution-Based Distillation . . . . .	329
10.3	Progressive Distillation . . . . .	334
10.4	Closing Remarks . . . . .	340
<b>11</b>	<b>Learning Fast Generators from Scratch</b>	<b>341</b>
11.1	Prologue . . . . .	343
11.2	Special Flow Map: Consistency Model in Discrete Time . . . . .	348
11.3	Special Flow Map: Consistency Model in Continuous Time . . . . .	356
11.4	General Flow Map: Consistency Trajectory Model . . . . .	365
11.5	General Flow Map: Mean Flow . . . . .	375
11.6	Closing Remarks . . . . .	380
<b>Appendices</b>		<b>381</b>
<b>A</b>	<b>Crash Course on Differential Equations</b>	<b>382</b>
A.1	Foundation of Ordinary Differential Equations . . . . .	383
A.2	Foundation of Stochastic Differential Equations . . . . .	394

<b>B Density Evolution: From Change of Variable to Fokker–Planck</b>	<b>398</b>
B.1 Change-of-Variable Formula:	
From Deterministic Maps to Stochastic Flows . . . . .	399
B.2 Intuition of the Continuity Equation . . . . .	409
<b>C Behind the Scenes of Diffusion Models:</b>	
<b>Itô’s Calculus and Girsanov’s Theorem</b>	<b>412</b>
C.1 Itô’s Formula: The Chain Rule for Random Processes . . . . .	413
C.2 Change-of-Variable For Measures: Girsanov’s Theorem in Diffusion Models . . . . .	422
<b>D Supplementary Materials and Proofs</b>	<b>426</b>
D.1 Variational Perspective . . . . .	426
D.2 Score-Based Perspective . . . . .	430
D.3 Flow-Based Perspective . . . . .	441
D.4 Theoretical Supplement: A Unified and Systematic View on Diffusion Models . . . . .	445
D.5 Theoretical Supplement: Learning Fast Diffusion-Based Generators	446
D.6 (Optional) Elucidating Diffusion Model (EDM) . . . . .	450
<b>References</b>	<b>454</b>

# The Principles of Diffusion Models

Chieh-Hsin Lai<sup>1</sup>, Yang Song<sup>2</sup>, Dongjun Kim<sup>3</sup>, Yuki Mitsufuji<sup>4</sup> and Stefano Ermon<sup>5</sup>

<sup>1</sup>*Sony AI*; [chieh-hsin.lai@sony.com](mailto:chieh-hsin.lai@sony.com) / [chiehhsinlai@gmail.com](mailto:chiehhsinlai@gmail.com)

<sup>2</sup>*OpenAI\**; [thusongyang@gmail.com](mailto:thusongyang@gmail.com)

<sup>3</sup>*Stanford University*; [dongjun@stanford.edu](mailto:dongjun@stanford.edu)

<sup>4</sup>*Sony Corporation, Sony AI*; [yuhki.mitsufuji@sony.com](mailto:yuhki.mitsufuji@sony.com)

<sup>5</sup>*Stanford University*; [ermon@cs.stanford.edu](mailto:ermon@cs.stanford.edu)

---

## ABSTRACT

This monograph focuses on the principles that have shaped the development of diffusion models, tracing their origins and showing how different formulations arise from common mathematical ideas.

Diffusion modeling begins by specifying a *forward corruption process* that gradually turns data into noise. This forward process links the data distribution to a simple noise distribution by defining a continuous family of intermediate distributions. The core objective of a diffusion model is to construct another process that runs in the opposite direction, transforming noise into data while recovering the same intermediate distributions defined by the forward corruption process.

We describe three complementary ways to formalize this idea. The *variational view*, inspired by variational autoencoders, sees diffusion as learning to remove noise step by step, solving small denoising objectives that together teach the model to turn noise back into data. The *score-based view*, rooted in energy-based modeling, learns the gradient of the evolving data distribution, which indicates how to nudge samples toward more likely regions. The *flow-based view*, related to normalizing flows, treats generation as following a smooth path that moves samples from noise to data under a learned velocity field.

These perspectives share a common backbone: a learned time-dependent velocity field whose flow transports a simple prior to the

---

\*Affiliation reflects the institution at the time of the work.

data. With this in hand, sampling amounts to solving a differential equation that evolves noise into data along a continuous generative trajectory. On this foundation, the monograph discusses *guidance* for controllable generation, *advanced numerical solvers* for efficient sampling, and diffusion-motivated *flow-map models* that learn direct mappings between arbitrary times along this trajectory.

This monograph is written for readers with a basic deep learning background who seek a clear, conceptual, and mathematically grounded understanding of diffusion models. It clarifies the theoretical foundations, explains the reasoning behind their diverse formulations, and provides a stable footing for further study and research in this rapidly evolving field. It serves both as a principled reference for researchers and as an accessible entry point for learners.

---

## Acknowledgements

---

The authors are deeply grateful to **Professor Dohyun Kwon** from the University of Seoul and KIAS for his generous time and effort in engaging with this work. He carefully reviewed parts of Chapter 7, helping to ensure the correctness of statements and proofs, and he contributed to several valuable discussions that clarified the presentation. Beyond technical suggestions, his thoughtful feedback and willingness to share perspectives have been a source of encouragement throughout the writing of this monograph. We sincerely appreciate his support and collegial spirit, which have enriched the final version.

## Preface and Roadmap

---

Diffusion models have rapidly become a central paradigm in generative modeling, with a vast body of work spanning machine learning, computer vision, natural language processing, and beyond. This literature is dispersed across communities and highlights different dimensions of progress, including theoretical foundations that concern modeling principles, training objectives, sampler design, and the mathematical ideas behind them; implementation advances that cover engineering practices and architectural choices; practical applications that adapt the models to specific domains or tasks; and system level optimizations that improve efficiency in computation, memory, and deployment.

This monograph sets out to provide a *principled foundation* of diffusion models, focusing on the following central themes:

- We present the essential concepts and formulations that anchor diffusion model research, giving readers the core understanding needed to navigate the broader literature. We do not survey all variants or domain specific applications; instead we establish a stable conceptual foundation from which such developments can be understood.
- Unlike classical generative models that learn a direct mapping from noise to data, diffusion models view generation as a gradual transformation over time, refining coarse structures into fine details. This central idea has been developed through three main perspectives, i.e., *variational*, *score-based*, and *flow-based* methods, which offer complementary ways to understand and implement diffusion modeling. We focus on the core principles and foundations of these formulations, aiming to trace the

origins of their key ideas, clarify the relations among different formulations, and develop a coherent understanding that connects intuitive insight with rigorous mathematical formulation.

- Building on these foundations, we examine how diffusion models can be further developed to generate samples more efficiently, provide greater control over the generative process, and inspire standalone forms of generative modeling grounded in the principles of diffusion.

This monograph is intended for researchers, graduate students, and practitioners who have a basic understanding of deep learning (for example, what a neural network is and how training works), or more specifically, deep generative modeling, and who wish to deepen their grasp of diffusion models beyond surface-level familiarity. By the end, readers will have a principled understanding of the foundations of diffusion modeling, the ability to interpret different formulations within a coherent framework, and the background needed to both apply existing models with confidence and pursue new research directions.

## Roadmap of This Monograph

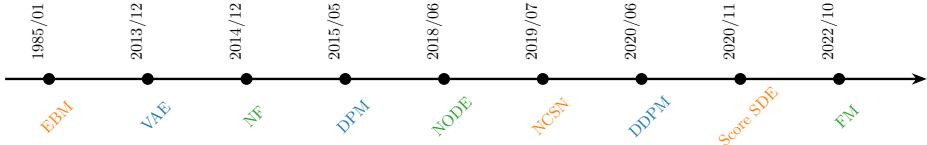
This monograph systematically introduces the foundations of diffusion models, tracing them back to their core underlying principles.

**Suggested Reading Path.** We recommend reading this monograph in the presented order to build a comprehensive understanding. Sections marked as *Optional* can be skipped by readers already familiar with the fundamentals. For instance, those comfortable with deep generative models (DGM) may bypass the overview in Chapter 1. Similarly, prior knowledge of Variational Autoencoders (Section 2.1), Energy-Based Models (Section 3.1), or Normalizing Flows (Section 5.1) allows skipping these introductory sections. Other optional parts provide deeper insights into advanced or specialized topics and can be consulted as needed.

The monograph is organized into four main parts.

**Parts A & B: Foundations of Diffusion Models.** This section traces the origins of diffusion models by reviewing three foundational perspectives that have shaped the field. Figure 2 provides an overview of this part.

**Part A: Introduction to Deep Generative Modeling (DGM).** We begin in Chapter 1 with a review of the fundamental goals of deep generative modeling. Starting from a collection of data examples, the aim is to build a model that can produce new examples that appear to come from the same underlying, and generally unknown, data distribution. Many approaches achieve this by learning how the data are distributed, either explicitly through a probability model or implicitly through a learned transformation. We then explain how such models represent the data distribution with neural networks, how they learn from examples, and how they generate new samples. The chapter concludes with a taxonomy of major generative frameworks, highlighting their central ideas and key distinctions.



**Figure 1: Timeline of diffusion model perspectives.** Each group shares the same color.

- In Chapter 2, Variational Autoencoder (VAE) (Kingma and Welling, 2013) → Diffusion Probabilistic Models (DPM) (Sohl-Dickstein *et al.*, 2015) → DDPM (Ho *et al.*, 2020).
- In Chapters 3 and 4, Energy-Based Model (EBM) (Ackley *et al.*, 1985) → Noise Conditional Score Network (NCSN) (Song and Ermon, 2019) → Score SDE (Song *et al.*, 2020c).
- In Chapter 5, Normalizing Flow (NF) (Rezende and Mohamed, 2015) → Neural ODE (NODE) (Chen *et al.*, 2018) → Flow Matching (FM) (Lipman *et al.*, 2022).

**Part B: Core Perspectives on Diffusion Models.** Having outlined the general goals and mechanisms of deep generative modeling, we now turn to diffusion models, a class of methods that realize generation as a gradual transformation from noise to data. We examine three interconnected frameworks, each characterized by a forward process that gradually adds noise and a reverse-time process approximated by a sequence of models performing gradual denoising:

- **Variational View** (Chapter 2): Originating from Variational Autoencoders (VAEs) (Kingma and Welling, 2013), it frames diffusion as learning a denoising process through a variational objective, giving rise to Denoising Diffusion Probabilistic Models (DDPMs) (Sohl-Dickstein *et al.*, 2015; Ho *et al.*, 2020).
- **Score-Based View** (Chapter 3): Rooted in Energy-Based Models (EBMs) (Ackley *et al.*, 1985) and developed into Noise Conditional Score Networks

(NCSN) (Song and Ermon, 2019). It learns the score function, the gradient of the log data density, which guides how to gradually remove noise from samples. In continuous time, Chapter 4 introduces the *Score SDE framework*, which describes this denoising process as a Stochastic Differential Equation (SDE) and its deterministic counterpart as an Ordinary Differential Equation (ODE). This view connects diffusion modeling with classical differential equation theory, providing a clear mathematical basis for analysis and algorithm design.

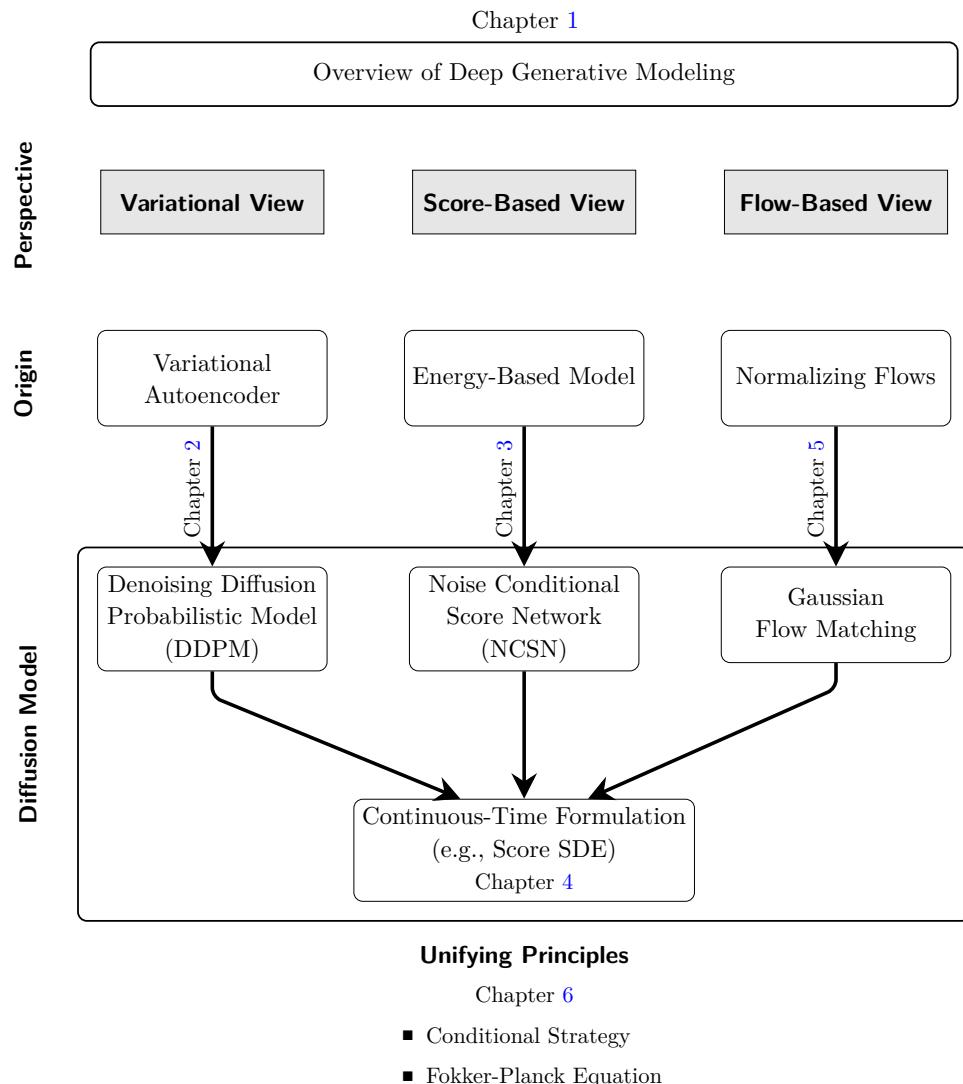
- **Flow-Based View** (Chapter 5): Building on Normalizing Flows (Rezende and Mohamed, 2015) and generalized by Flow Matching (Lipman *et al.*, 2022), this view models generation as a continuous transformation that transports samples from a simple prior toward the data distribution. The evolution is governed by a velocity field through an ODE, which explicitly defines how probability mass moves over time. This flow-based formulation naturally extends beyond prior-to-data generation to more general *distribution-to-distribution translation* problems, where one seeks to learn a flow connecting any pair of source and target distributions.

Although these perspectives may seem different at first, Chapter 6 shows that they are deeply connected. Each uses a *conditioning strategy* that turns the learning objective into a tractable regression problem. At a deeper level, they all describe the same temporal evolution of probability distributions, from the prior toward the data. This evolution is governed by the *Fokker–Planck equation*, which can be viewed as the continuous-time change of variables for densities, ensuring consistency between the stochastic and deterministic formulations.

Since diffusion models can be viewed as approaches for transporting one distribution to another, Chapter 7 develops their connections to classical optimal transport and the Schrödinger bridge, interpreted as optimal transport with entropy regularization. We review both the static and dynamic formulations and explain their relations to the continuity equation and the Fokker–Planck perspective. This chapter is optional for readers focused on practical aspects, but it provides rigorous mathematical background and pointers to the classical literature for those who wish to study these links in depth.

**Part C & D: Controlling and Accelerating the Diffusion Sampling.** With the foundational principles unified, we now turn to practical aspects of utilizing diffusion models for efficient generation. Sampling from a diffusion model corresponds to solving a differential equation. However, this procedure is

**Figure 2: Part B. Unifying and Principled Perspectives on Diffusion Models.** This diagram visually connects classical generative modeling approaches—Variational Autoencoders, Energy-Based Models, and Normalizing Flows—with their corresponding diffusion model formulations. Each vertical path illustrates a conceptual lineage, culminating in the continuous-time framework. The three views (Variational, Score-Based, and Flow-Based) offer distinct yet mathematically equivalent interpretations.



typically computationally expensive. Parts C and D focus on improving generation quality, controllability, and efficiency through enhanced sampling and learned acceleration techniques.

**Part C: Sampling from Diffusion Models.** The generation process of diffusion models exhibits a distinctive coarse-to-fine refinement: noise is removed step by step, yielding samples with increasingly coherent structure and detail. This property comes with trade-offs. On the positive side, it affords fine-grained control; by adding a guidance term to the learned, time-dependent velocity field, we can steer the ODE flow to reflect user intent and make sampling controllable. On the negative side, the required iterative integration makes sampling slow compared with single-shot generators. This part focuses on improving the generative process at inference time, without retraining.

- **Steering Generation** (Chapter 8): Techniques such as classifier guidance and classifier-free guidance make it possible to condition the generation process on user-defined objectives or attributes. Building on this, we next discuss how the use of a preference dataset can further align diffusion models with such preferences.
- **Fast Generation with Numerical Solvers** (Chapter 9): Sampling can be significantly accelerated using advanced numerical solvers that approximate the reverse process in fewer steps, reducing cost while preserving quality.

**Part D: Learning Fast Generative Models.** Beyond improving existing sampling algorithms, we investigate how to directly learn fast generators that approximate the diffusion process.

- **Distillation-Based Methods** (Chapter 10): This approach focuses on training a student model to imitate the behavior of a pre-trained, slow diffusion model (the teacher). Instead of reducing the teacher’s size, the goal is to reproduce its sampling trajectory or output distribution with far fewer integration steps, often only a few or even one.
- **Learning from Scratch** (Chapter 11): Since sampling in diffusion models can be seen as solving an ODE, this approach learns the solution map (i.e., the flow map) directly from scratch, without relying on a teacher model. The learned map can take noise directly to data, or more generally perform anytime-to-anytime jumps along the solution trajectory.

**Appendices.** To ensure our journey is accessible to all, the appendices provide background for foundational concepts. Chapter A offers a crash course on the differential equations that have become the language of diffusion models.

The core insight behind diffusion models, despite their varied perspectives and origins, lies in the *change-of-variables formula*. This foundation naturally extends to deeper concepts such as the *Fokker–Planck equation* and the *continuity equation*, which describe how probability densities transform and evolve under mappings defined by functions (discrete time) or differential equations (continuous time). Chapter B offers a gentle introduction that bridges these foundational ideas to more advanced concepts. In Chapter C, we present two powerful but often overlooked tools underlying diffusion models: *Itô’s formula* and *Girsanov’s theorem*, which provide rigorous support for the Fokker–Planck equation and the reverse-time sampling process. Finally, Chapter D gathers proofs of selected propositions and theorems discussed in the main chapters.

**What This Monograph Covers and What It Does Not.** We aim for durability. From a top-down viewpoint, this monograph begins with a single principle: construct continuous-time dynamics that transport a simple prior to the data distribution while ensuring that the marginal distribution at each time matches the marginal induced by a prescribed forward process from data to noise. From this principle, we develop the stochastic and deterministic flows that enable sampling, show how to steer the trajectory (guidance), and explain how to accelerate it (numerical solvers). We then study diffusion-motivated fast generators, including distillation methods and flow-map models. With these tools, readers can place new papers within a common template, understand why methods work, and design improved models.

We do not attempt to provide an exhaustive survey of the diffusion model literature, nor do we catalog architectures, training practices, hyperparameters, compare empirical results across methods, cover datasets and leaderboards, describe domain- or modality-specific applications, address system-level deployment, provide recipes for large-scale training, or discuss hardware engineering. These topics evolve rapidly and are better covered by focused surveys, open repositories, and implementation guides.

## Notations

---

### Numbers and Arrays

$a$	A scalar.
$\mathbf{a}$	A column vector (e.g., $\mathbf{a} \in \mathbb{R}^D$ ).
$\mathbf{A}$	A matrix (e.g., $\mathbf{A} \in \mathbb{R}^{m \times n}$ ).
$\mathbf{A}^\top$	Transpose of $\mathbf{A}$ .
$\text{Tr}(\mathbf{A})$	Trace of $\mathbf{A}$ .
$\mathbf{I}_D$	Identity matrix of size $D \times D$ .
$\mathbf{I}$	Identity matrix; dimension implied by context.
$\text{diag}(\mathbf{a})$	Diagonal matrix with diagonal entries given by $\mathbf{a}$ .
$\phi, \theta$	Learnable neural network parameters.
$\phi^\times, \theta^\times$	Parameters after training (fixed during inference).
$\phi^*, \theta^*$	Optimal parameters of an optimization problem.

### Calculus

$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	Partial derivatives of $\mathbf{y}$ w.r.t. $\mathbf{x}$ (componentwise).
$\frac{d\mathbf{y}}{d\mathbf{x}}$ or $D\mathbf{y}(\mathbf{x})$	Total (Fréchet) derivative of $\mathbf{y}$ w.r.t. $\mathbf{x}$ .
$\nabla_{\mathbf{x}} y$	Gradient of scalar $y : \mathbb{R}^D \rightarrow \mathbb{R}$ ; a column in $\mathbb{R}^D$ .
$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ or $\nabla_{\mathbf{x}} \mathbf{F}$	Jacobian of $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ ; shape $m \times n$ .
$\nabla \cdot \mathbf{y}$	Divergence of a vector field $\mathbf{y} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ ; a scalar.
$\nabla_{\mathbf{x}}^2 f(\mathbf{x})$ or $\mathbf{H}(f)(\mathbf{x})$	Hessian of $f : \mathbb{R}^D \rightarrow \mathbb{R}$ ; shape $D \times D$ .
$\int f(\mathbf{x}) d\mathbf{x}$	Integral of $f$ over the domain of $\mathbf{x}$ .

### Probability and Information Theory

$p(\mathbf{a})$	Density/distribution over a continuous vector $\mathbf{a}$ .
$p_{\text{data}}$	Data distribution.
$p_{\text{prior}}$	Prior distribution (e.g., standard normal).
$p_{\text{src}}$	Source distribution.
$p_{\text{tgt}}$	Target distribution.
$\mathbf{a} \sim p$	Random vector $\mathbf{a}$ is distributed as $p$ .
$\mathbb{E}_{\mathbf{x} \sim p}[\mathbf{f}(\mathbf{x})]$	Expectation of $\mathbf{f}(\mathbf{x})$ under $p(\mathbf{x})$ .
$\mathbb{E}[\mathbf{f}(\mathbf{x}) \mathbf{z}]$ , or $\mathbb{E}_{\mathbf{x} \sim p(\cdot \mathbf{z})}[\mathbf{f}(\mathbf{x})]$	Conditional expectation of $\mathbf{f}(\mathbf{x})$ given $\mathbf{z}$ , with $\mathbf{x}$ distributed as $p(\cdot \mathbf{z})$ .
$\text{Var}(\mathbf{f}(\mathbf{x}))$	Variance under $p(\mathbf{x})$ .
$\text{Cov}(\mathbf{f}(\mathbf{x}), \mathbf{g}(\mathbf{x}))$	Covariance under $p(\mathbf{x})$ .
$\mathcal{D}_{\text{KL}}(p\ q)$	Kullback–Leibler divergence from $q$ to $p$ .
$\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$	Standard normal sample.
$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian over $\mathbf{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ .

**Clarification.** We use the same symbol for a random vector and its realized value. This convention, common in deep learning and generative modeling, keeps notation compact and uncluttered. The intended meaning is determined by context.

For example, in expressions such as  $p(\mathbf{x})$ , the symbol  $\mathbf{x}$  serves as a dummy variable, and the expression denotes the distribution or density as a function

of its input. Thus  $p(\mathbf{x})$  refers to the functional form rather than evaluation at a particular sample. When evaluation at a given point is intended, we state it explicitly (for instance, “evaluate  $p$  at the given point  $\mathbf{x}$ ”).

Conditional expressions are read by context. For  $p(\mathbf{x}|\mathbf{y})$ , fixing  $\mathbf{y}$  makes it a density in  $\mathbf{x}$ ; fixing  $\mathbf{x}$  makes it a function of  $\mathbf{y}$ .

For conditional expectations,  $\mathbb{E}[\mathbf{f}(\mathbf{x})|\mathbf{z}]$  denotes a function of  $\mathbf{z}$ , giving the expected value of  $\mathbf{f}(\mathbf{x})$  conditional on  $\mathbf{z}$ . When conditioning on a specific realized value, we write  $\mathbb{E}[\mathbf{f}(\mathbf{x})|\mathbf{Z} = \mathbf{z}]$ . Equivalently, this can be written as an integral with respect to the conditional distribution,

$$\mathbb{E}_{\mathbf{x} \sim p(\cdot|\mathbf{z})}[\mathbf{f}(\mathbf{x})] = \int \mathbf{f}(\mathbf{x}) p(\mathbf{x}|\mathbf{z}) d\mathbf{x}.$$

This distinction clarifies whether  $\mathbf{z}$  is treated as a variable defining a function,  $\mathbf{z} \mapsto \mathbb{E}[\mathbf{f}(\mathbf{x})|\mathbf{z}]$ , or as a fixed value at which that function is evaluated.

## **Part A**

# **Introduction to Deep Generative Modeling**

# 1

---

## Deep Generative Modeling

---

*What I cannot create, I do not understand.*

---

Richard P. Feynman

Deep generative models (DGMs) are neural networks that learn a probability distribution over high-dimensional data (e.g., images, text, audio) so they can generate new examples that resemble the dataset. We denote the model distribution by  $p_\phi$  and the data distribution by  $p_{\text{data}}$ . Given a finite dataset, we fit  $\phi$  by minimizing a loss that measures how far  $p_\phi$  is from  $p_{\text{data}}$ . After training, generation amounts to running the model’s sampling procedure to draw  $\mathbf{x} \sim p_\phi$  (the density  $p_\phi(\mathbf{x})$  may or may not be directly computable, depending on the model class). Model quality is judged by how well generated samples and their summary statistics match those of  $p_{\text{data}}$ , together with task-specific or perceptual metrics.

This chapter builds the mathematical and conceptual foundations behind these ideas. We formalize the problem in Section 1.1, present representative model classes in Section 1.2, and summarize a practical taxonomy in Section 1.3.

## 1.1 What is Deep Generative Modeling?

DGMs take as input a large collection of real-world examples (e.g., images, text) drawn from an unknown and complex distribution  $p_{\text{data}}$  and output a trained neural network that parameterizes an approximate distribution  $p_{\phi}$ . Their goals are twofold:

1. **Realistic Generation:** To generate novel, realistic samples indistinguishable from real data.
2. **Controllable Generation:** To enable fine-grained and interpretable control over the generative process.

This section presents the fundamental concepts and motivations behind DGMs, preparing for a detailed exploration of their mathematical framework and practical applications.

### 1.1.1 Mathematical Setup

We assume access to a finite set of samples drawn independently and identically distributed (i.i.d.) from an underlying, complex data distribution  $p_{\text{data}}(\mathbf{x})^1$ .

**Goal of DGM.** The primary goal of DGM is to learn a tractable probability distribution from a finite dataset. These data points are observations assumed to be sampled from an unknown and complex true distribution  $p_{\text{data}}(\mathbf{x})$ . Since the form of  $p_{\text{data}}(\mathbf{x})$  is unknown, we cannot draw new samples from it directly. The core challenge is therefore to create a model that approximates this distribution well enough to enable the generation of new, realistic samples.

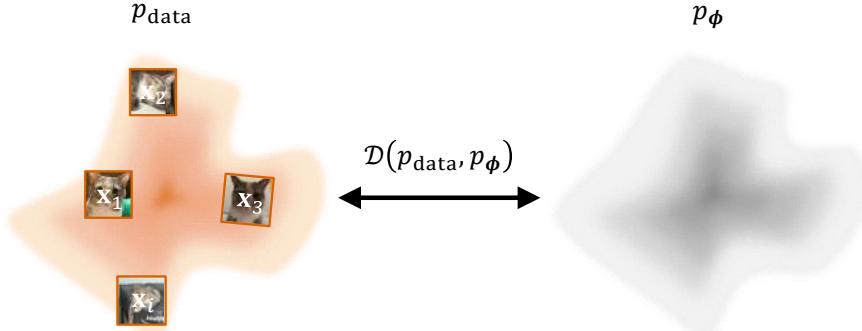
To this end, a DGM uses a deep neural network to parameterize a model distribution  $p_{\phi}(\mathbf{x})$ , where  $\phi$  represents the network's trainable parameters. The training objective is to find the optimal parameters  $\phi^*$  that minimize the divergence between the model distribution  $p_{\phi}(\mathbf{x})$  and the true data distribution  $p_{\text{data}}(\mathbf{x})$ . Conceptually,

$$p_{\phi^*}(\mathbf{x}) \approx p_{\text{data}}(\mathbf{x}).$$

When the statistical model  $p_{\phi^*}(\mathbf{x})$  closely approximates the data distribution  $p_{\text{data}}(\mathbf{x})$ , it can serve as a proxy for generating new samples and evaluating probability values. This model  $p_{\phi}(\mathbf{x})$  is commonly referred to as a *generative model*.

---

<sup>1</sup>This is a common assumption in machine learning. For simplicity, we use the symbol  $p$  to represent either a probability distribution or its probability density/mass function, depending on the context.



**Figure 1.1: Illustration of the target in DGM.** Training a DGM is essentially minimizing the discrepancy between the model distribution  $p_\phi$  and the unknown data distribution  $p_{\text{data}}$ . Since  $p_{\text{data}}$  is not directly accessible, this discrepancy must be estimated efficiently using a finite set of independent and identically distributed (i.i.d.) samples,  $\mathbf{x}_i$ , drawn from it.

**Capability of DGM.** Once a proxy of the data distribution,  $p_\phi(\mathbf{x})$ , is available, we can generate an arbitrary number of new data points using sampling methods such as Monte Carlo sampling from  $p_\phi(\mathbf{x})$ . Additionally, we can compute the probability (or likelihood) of any given data sample  $\mathbf{x}'$  by evaluating  $p_\phi(\mathbf{x}')$ .

**Training of DGM.** We learn parameters  $\phi$  of a model family  $\{p_\phi\}$  by minimizing a discrepancy  $\mathcal{D}(p_{\text{data}}, p_\phi)$ :

$$\phi^* \in \arg \min_{\phi} \mathcal{D}(p_{\text{data}}, p_\phi). \quad (1.1.1)$$

Because  $p_{\text{data}}$  is unknown, a practical choice of  $\mathcal{D}$  must admit efficient estimation from i.i.d. samples from  $p_{\text{data}}$ . With sufficient capacity,  $p_{\phi^*}$  can closely approximate  $p_{\text{data}}$ .

**Forward KL and Maximum Likelihood Estimation (MLE).** A standard choice is the (forward) Kullback–Leibler divergence<sup>2</sup>

$$\begin{aligned} \mathcal{D}_{\text{KL}}(p_{\text{data}} \| p_\phi) &:= \int p_{\text{data}}(\mathbf{x}) \log \frac{p_{\text{data}}(\mathbf{x})}{p_\phi(\mathbf{x})} d\mathbf{x} \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log p_{\text{data}}(\mathbf{x}) - \log p_\phi(\mathbf{x})]. \end{aligned}$$

which is asymmetric, i.e.,

$$\mathcal{D}_{\text{KL}}(p_{\text{data}} \| p_\phi) \neq \mathcal{D}_{\text{KL}}(p_\phi \| p_{\text{data}}).$$

---

<sup>2</sup>All integrals are in the Lebesgue sense and reduce to sums under counting measures.

Importantly, minimizing  $\mathcal{D}_{\text{KL}}(p_{\text{data}} \| p_{\phi})$  encourages *mode covering*: if there exists a set of positive measure  $A$  with  $p_{\text{data}}(A) > 0$  but  $p_{\phi}(\mathbf{x}) = 0$  for  $\mathbf{x} \in A$ , then the integrand contains  $\log(p_{\text{data}}(\mathbf{x})/0) = +\infty$  on  $A$ , so  $\mathcal{D}_{\text{KL}} = +\infty$ . Thus minimizing forward KL forces the model to assign probability wherever the data has support.

Although the data density  $p_{\text{data}}(\mathbf{x})$  cannot be evaluated explicitly, the forward KL divergence can be decomposed as

$$\begin{aligned}\mathcal{D}_{\text{KL}}(p_{\text{data}} \| p_{\phi}) &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[ \log \frac{p_{\text{data}}(\mathbf{x})}{p_{\phi}(\mathbf{x})} \right] \\ &= -\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log p_{\phi}(\mathbf{x})] + \mathcal{H}(p_{\text{data}}),\end{aligned}$$

where  $\mathcal{H}(p_{\text{data}}) := -\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log p_{\text{data}}(\mathbf{x})]$  is the entropy of the data distribution, which is constant with respect to  $\phi$ . This observation implies the following equivalence:

**Lemma 1.1.1: Minimizing KL  $\Leftrightarrow$  MLE**

$$\min_{\phi} \mathcal{D}_{\text{KL}}(p_{\text{data}} \| p_{\phi}) \iff \max_{\phi} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log p_{\phi}(\mathbf{x})]. \quad (1.1.2)$$

In other words, minimizing the forward KL divergence is equivalent to performing MLE.

In practice we replace the population expectation by its Monte Carlo estimate from i.i.d. samples  $\{\mathbf{x}^{(i)}\}_{i=1}^N \sim p_{\text{data}}$ , yielding the empirical MLE objective

$$\hat{\mathcal{L}}_{\text{MLE}}(\phi) := -\frac{1}{N} \sum_{i=1}^N \log p_{\phi}(\mathbf{x}^{(i)}),$$

optimized via stochastic gradients over minibatches; no evaluation of  $p_{\text{data}}(\mathbf{x})$  is required.

**Fisher Divergence.** The Fisher divergence is another important concept for (score-based) diffusion modeling (see Chapter 3). For two distributions  $p$  and  $q$ , it is defined as

$$\mathcal{D}_F(p \| q) := \mathbb{E}_{\mathbf{x} \sim p} \left[ \|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \nabla_{\mathbf{x}} \log q(\mathbf{x})\|_2^2 \right]. \quad (1.1.3)$$

It measures the discrepancy between the *score functions*  $\nabla_{\mathbf{x}} \log p(\mathbf{x})$  and  $\nabla_{\mathbf{x}} \log q(\mathbf{x})$ , which are vector fields pointing toward regions of higher probability. In short,  $\mathcal{D}_F(p \| q) \geq 0$  with equality if and only if  $p = q$  almost everywhere.

It is invariant to normalization constants, since scores depend only on gradients of log-densities, and it forms the basis of *score matching* (Equations (3.1.3) and (3.2.1)): a method that learns the gradient of the log-density for generation (score-based models). In this setting, the data distribution  $p = p_{\text{data}}$  serves as the target, while the model  $q = p_\phi$  is trained to align its score field with that of the data.

**Beyond KL.** Although the KL divergence is the most widely used measure of difference between probability distributions, it is not the only one. Different divergences capture different geometric or statistical notions of discrepancy, which in turn affect the optimization dynamics of learning algorithms. A broad family is the *f-divergences* (Csiszár, 1963):

$$\mathcal{D}_f(p\|q) = \int q(\mathbf{x}) f\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right) d\mathbf{x}, \quad f(1) = 0, \quad (1.1.4)$$

where  $f : \mathbb{R}_+ \rightarrow \mathbb{R}$  is a convex function. By changing  $f$ , we obtain many well-known divergences:

$$\begin{aligned} f(u) &= u \log u & \Rightarrow \mathcal{D}_f &= \mathcal{D}_{\text{KL}}(p\|q) \quad (\text{forward KL}), \\ f(u) &= \frac{1}{2} \left[ u \log u - (u + 1) \log \frac{1+u}{2} \right] & \Rightarrow \mathcal{D}_f &= \mathcal{D}_{\text{JS}}(p\|q) \quad (\text{Jensen-Shannon}), \\ f(u) &= \frac{1}{2}|u - 1| & \Rightarrow \mathcal{D}_f &= \mathcal{D}_{\text{TV}}(p, q) \quad (\text{total variation}). \end{aligned}$$

For clarity, the explicit forms are

$$\mathcal{D}_{\text{JS}}(p\|q) = \frac{1}{2}\mathcal{D}_{\text{KL}}(p\|\frac{1}{2}(p+q)) + \frac{1}{2}\mathcal{D}_{\text{KL}}(q\|\frac{1}{2}(p+q)),$$

and

$$\mathcal{D}_{\text{TV}}(p, q) = \frac{1}{2} \int_{\mathbb{R}^D} |p - q| d\mathbf{x} = \sup_{A \subset \mathbb{R}^D} |p(A) - q(A)|.$$

Intuitively, the JS divergence provides a smooth and symmetric measure that balances both distributions and avoids the unbounded penalties of KL (we will later see that it helps interpret the Generative Adversarial Network (GAN) framework), while the total variation distance captures the largest possible probability difference between the two.

A different viewpoint comes from *optimal transport* (see Chapter 7), whose representative is the Wasserstein distance (see . It measures the minimal cost of moving probability mass from one distribution to another. Unlike *f*-divergences, which compare density ratios, Wasserstein distances depend on the geometry of the sample space and remain meaningful even when the supports of  $p$  and  $q$  do not overlap.

Each divergence embodies a different notion of closeness between distributions and thus induces distinct learning behavior. We will revisit these divergences when they arise naturally in the context of generative modeling throughout this monograph.

### 1.1.2 Challenges in Modeling Distributions

To model a complex data distribution, we can parameterize the probability density function  $p_{\text{data}}$  using a neural network with parameters  $\phi$ , creating a model we denote as  $p_\phi$ . For  $p_\phi$  to be a valid probability density function, it must satisfy two fundamental properties:

- (i) **Non-Negativity:**  $p_\phi(\mathbf{x}) \geq 0$  for all  $\mathbf{x}$  in the domain.
- (ii) **Normalization:** The integral over the entire domain must equal one, i.e.,  $\int p_\phi(\mathbf{x}) d\mathbf{x} = 1$ .

A network can naturally produce a real scalar  $E_\phi(\mathbf{x}) \in \mathbb{R}$  for input  $\mathbf{x}$ . To interpret this output as a valid density, it must be transformed to satisfy conditions (i) and (ii). A practical alternative is to view  $E_\phi: \mathbb{R}^D \rightarrow \mathbb{R}$  as defining an *unnormalized* density and then enforce these properties explicitly.

**Step 1: Ensuring Non-Negativity.** We can guarantee that our model's output is always non-negative by applying a positive function to the raw output of the neural network  $E_\phi(\mathbf{x})$ , such as  $|E_\phi(\mathbf{x})|$ ,  $E_\phi^2(\mathbf{x})$ . A standard and convenient choice is the exponential function. This gives us an unnormalized density,  $\tilde{p}_\phi(\mathbf{x})$ , that is guaranteed to be positive:

$$\tilde{p}_\phi(\mathbf{x}) = \exp(E_\phi(\mathbf{x})).$$

**Step 2: Enforcing Normalization.** The function  $\tilde{p}_\phi(\mathbf{x})$  is positive but does not integrate to one. To create a valid probability density, we must divide it by its integral over the entire space. This leads to the final form of our model:

$$p_\phi(\mathbf{x}) = \frac{\tilde{p}_\phi(\mathbf{x})}{\int \tilde{p}_\phi(\mathbf{x}') d\mathbf{x}'} = \frac{\exp(E_\phi(\mathbf{x}))}{\int \exp(E_\phi(\mathbf{x}')) d\mathbf{x}'}.$$

The denominator in this expression is known as the *normalizing constant* or *partition function*, denoted by  $Z(\phi)$ :

$$Z(\phi) := \int \exp(E_\phi(\mathbf{x}')) d\mathbf{x}'.$$

While this procedure provides a valid construction for  $p_\phi(\mathbf{x})$ , it introduces a major computational challenge. For most high-dimensional problems, the integral required to compute the normalizing constant  $Z(\phi)$  is intractable. This intractability is a central problem that motivates the development of many different families of deep generative models.

In the following sections, we introduce several prominent approaches of DGM. Each is designed to circumvent or reduce the computational cost of evaluating this normalizing constant.

## 1.2 Prominent Deep Generative Models

A central challenge in generative modeling is to learn expressive probabilistic models that can capture the rich and complex structure of high-dimensional data. Over the years, various modeling strategies have been developed, each making different trade-offs between tractability, expressiveness, and training efficiency. In this section, we explore some of the most influential strategies that have shaped the field, accompanied by a comparison of their computation graphs in Figure 1.2.

**Energy-Based Models (EBMs).** EBMs (Ackley *et al.*, 1985; LeCun *et al.*, 2006) define a probability distribution through an energy function  $E_\phi(\mathbf{x})$  that assigns lower energy to more probable data points. The probability of a data point is defined as:

$$p_\phi(\mathbf{x}) := \frac{1}{Z(\phi)} \exp(-E_\phi(\mathbf{x})),$$

where

$$Z(\phi) = \int \exp(-E_\phi(\mathbf{x})) d\mathbf{x}$$

is the partition function. Training EBMs typically involves maximizing the log-likelihood of the data. However, this requires techniques to address the computational challenges arising from the intractability of the partition function. In the following chapter, we will explore how Diffusion Models offer an alternative by generating data from *the gradient of the log density*, which does not depend on the normalizing constant, thereby circumventing the need for partition function computation.

**Autoregressive Models.** Deep autoregressive (AR) models (Frey *et al.*, 1995; Larochelle and Murray, 2011; Uria *et al.*, 2016) factorize the joint data distribution  $p_{\text{data}}$  into a product of conditional probabilities using the *chain rule of probability*:

$$p_{\text{data}}(\mathbf{x}) = \prod_{i=1}^D p_\phi(x_i | \mathbf{x}_{<i}),$$

where  $\mathbf{x} = (x_1, \dots, x_D)$  and  $\mathbf{x}_{<i} = (x_1, \dots, x_{i-1})$ .

Each conditional  $p_\phi(x_i | \mathbf{x}_{<i})$  is parameterized by a neural network, such as a Transformer, allowing flexible modeling of complex dependencies. Because each term is normalized by design (e.g., via softmax for discrete or parameterized Gaussian for continuous variables), global normalization is trivial.

Training proceeds by maximizing the exact likelihood, or equivalently minimizing the negative log-likelihood,

While AR models achieve strong density estimation and exact likelihoods, their sequential nature limits sampling speed and may restrict flexibility due to fixed ordering. Nevertheless, they remain a foundational class of likelihood-based generative models and key approaches in modern research.

**Variational Autoencoders (VAEs).** VAEs (Kingma and Welling, 2013) extend classical autoencoders by introducing latent variables  $\mathbf{z}$  that capture hidden structure in the data  $\mathbf{x}$ . Instead of directly learning a mapping between  $\mathbf{x}$  and  $\mathbf{z}$ , VAEs adopt a probabilistic view: they learn both an *encoder*,  $q_{\theta}(\mathbf{z}|\mathbf{x})$ , which approximates the unknown distribution of latent variables given the data, and a *decoder*,  $p_{\phi}(\mathbf{x}|\mathbf{z})$ , which reconstructs data from these latent variables. To make training feasible, VAEs maximize a tractable surrogate to the true log-likelihood, called the Evidence Lower Bound (ELBO):

$$\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_{\theta}(\mathbf{z}|\mathbf{x})} [\log p_{\phi}(\mathbf{x}|\mathbf{z})] - \mathcal{D}_{\text{KL}}(q_{\theta}(\mathbf{z}|\mathbf{x}) \parallel p_{\text{prior}}(\mathbf{z})).$$

Here, the first term encourages accurate reconstruction of the data, while the second regularizes the latent variables by keeping them close to a simple prior distribution  $p_{\text{prior}}(\mathbf{z})$  (often Gaussian).

VAEs provide a principled way to combine neural networks with latent-variable models and remain one of the most widely used likelihood-based approaches. However, they also face practical challenges, such as limited sample sharpness and training pathologies (e.g., the tendency of the encoder to ignore latent variables). Despite these limitations, VAEs laid important foundations for later advances, including diffusion models.

**Normalizing Flows.** Classic flow-based models, such as Normalizing Flows (NFs) (Rezende and Mohamed, 2015) and Neural Ordinary Differential Equations (NODEs) (Chen *et al.*, 2018), aim to learn a bijective mapping  $\mathbf{f}_{\phi}$  between a simple latent distribution  $\mathbf{z}$  and a complex data distribution  $\mathbf{x}$  via an invertible operator. This is achieved either through a sequence of bijective transformations (in NFs) or by modeling the transformation as an Ordinary Differential Equation (in NODEs). These models leverage the “change-of-variable formula for densities”, enabling MLE training:

$$\log p_{\phi}(\mathbf{x}) = \log p(\mathbf{z}) + \log \left| \det \frac{\partial \mathbf{f}_{\phi}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right|,$$

where  $\mathbf{f}_\phi$  represents the invertible transformation mapping  $\mathbf{z}$  to  $\mathbf{x}$ . NPs explicitly model normalized densities using invertible transformations with tractable Jacobian determinants. The normalization constant is absorbed analytically via the change-of-variables formula, making likelihood computation exact and tractable.

Despite their conceptual elegance, classic flow-based models often face practical limitations. For instance, NPs typically impose restrictive architectural constraints to ensure bijectivity, while NODEs may encounter training inefficiencies due to the computational overhead of solving ODEs. Both approaches face challenges when scaling to high-dimensional data. In later chapters, we will explore how Diffusion Models relate to and build upon these classic flow-based methods.

**Generative Adversarial Networks (GANs).** GANs (Goodfellow *et al.*, 2014) consist of two neural networks, a generator  $\mathbf{G}_\phi$  and a discriminator  $D_\zeta$ , that compete against each other. The generator aims to create realistic samples  $\mathbf{G}_\phi(\mathbf{z})$  from random noise  $\mathbf{z} \sim p_{\text{prior}}$ , while the discriminator attempts to distinguish between real samples  $\mathbf{x}$  and generated samples  $\mathbf{G}_\phi(\mathbf{z})$ . The objective function for GANs can be formulated as:

$$\min_{\mathbf{G}_\phi} \max_{D_\zeta} \underbrace{\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D_\zeta(\mathbf{x})]}_{\text{real}} + \underbrace{\mathbb{E}_{\mathbf{z} \sim p_{\text{prior}}(\mathbf{z})} [\log(1 - D_\zeta(\mathbf{G}_\phi(\mathbf{z})))]}_{\text{fake}}.$$

GANs do not define an explicit density function and therefore bypass likelihood estimation entirely. Instead of computing a normalization constant, they focus on generating samples that closely mimic the data distribution.

From a divergence perspective, the discriminator implicitly measures the discrepancy between the true data distribution  $p_{\text{data}}$  and the generator distribution  $p_{\mathbf{G}_\phi}$ , where  $p_{\mathbf{G}_\phi}$  denotes the distribution of generated samples  $\mathbf{G}_\phi(\mathbf{z})$  obtained from noise  $\mathbf{z} \sim p_{\text{prior}}$ . With an optimal discriminator for a fixed generator  $\mathbf{G}_\phi$  computed as

$$\frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\mathbf{G}_\phi}(\mathbf{x})},$$

the generator's minimization reduces to

$$\min_{\mathbf{G}_\phi} 2 \mathcal{D}_{\text{JS}}(p_{\text{data}} \| p_{\mathbf{G}_\phi}) - \log 4.$$

Here,  $\mathcal{D}_{\text{JS}}$  denotes the Jensen–Shannon divergence, defined as

$$\mathcal{D}_{\text{JS}}(p \| q) := \frac{1}{2} \mathcal{D}_{\text{KL}}\left(p \middle\| \frac{p+q}{2}\right) + \frac{1}{2} \mathcal{D}_{\text{KL}}\left(q \middle\| \frac{p+q}{2}\right).$$

This shows that GANs implicitly minimize  $\mathcal{D}_{\text{JS}}(p_{\text{data}} \parallel p_{\mathbf{G}_\phi})$ . More broadly, extensions such as  $f$ -GANs (Nowozin *et al.*, 2016) generalize this view by demonstrating that adversarial training can minimize a family of  $f$ -divergences, placing GANs within the same divergence-minimization framework as other generative models.

Although GANs are capable of generating high-quality data, their min-max training process is notoriously unstable, often requiring carefully designed architectures and engineering techniques to achieve satisfactory performance. However, GANs have since been revived as an auxiliary component to enhance other generative models, particularly Diffusion Models.

### 1.3 Taxonomy of Modelings

As we have seen, DGMs span a wide spectrum of modeling strategies. A fundamental distinction lies in how these models *parameterize* the underlying data distribution, that is, whether they specify  $p_\phi(\mathbf{x})$  *explicitly* or only *implicitly*, irrespective of the training objective.

- **Explicit Models:** These models directly parameterize a probability distribution  $p_\phi(\mathbf{x})$  via a tractable or approximately tractable density or mass function. Examples include ARs, NFs, VAEs, and DMs, all of which define  $p_\phi(\mathbf{x})$  either exactly or through a tractable bound.
- **Implicit Models:** These models specify a distribution only through a sampling procedure, typically of the form  $\mathbf{x} = \mathbf{G}_\phi(\mathbf{z})$  for some noise variable  $\mathbf{z} \sim p_{\text{prior}}$ . In this case,  $p_\phi(\mathbf{x})$  is not available in closed form and may not be defined at all.

The table in Table 1.1 offers a concise summary of these contrasting approaches.

**Table 1.1:** Comparison of Explicit and Implicit Generative Models

	Explicit		Implicit
	Exact Likelihood	Approx. Likelihood	
<b>Likelihood</b>	Tractable	Bound/Approx.	Not Directly Modeled/ Intractable
<b>Objective</b>	MLE	ELBO	Adversarial
<b>Examples</b>	NFs, ARs	VAEs, DMs	GANs

**Connection to Diffusion Models.** Taken together, these classical families of DGMs illustrate complementary strategies for modeling complex distributions. Beyond their standalone importance, they also provide guiding principles for understanding diffusion models. Diffusion methods inherit ideas from several of these perspectives: they connect to VAEs through variational training objectives, to EBMs through score-matching approaches that learn gradients of the log-density (closely tied to energy functions), and to NFs through continuous-time transformations.

To lay the groundwork for the diffusion methods discussed in later chapters, we will focus on three central paradigms: VAEs (Section 2.1), EBMs

(Section 3.1), and NFs (Section 5.1). This exploration provides a foundation for the core principles that underlie modern diffusion-based generative modeling, which will be developed further in the chapters that follow.

## 1.4 Closing Remarks

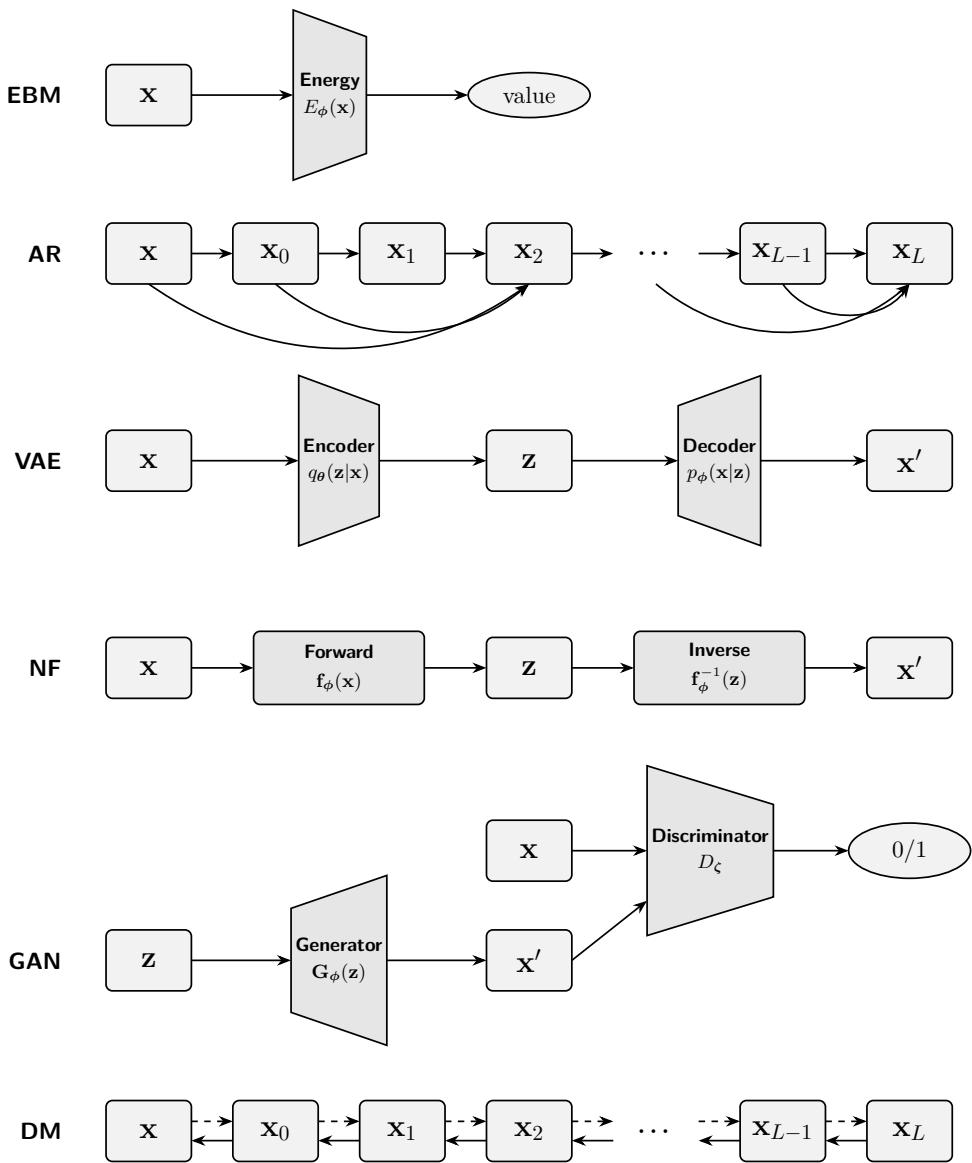
This chapter has established the foundational concepts of deep generative modeling. We begin by defining the primary objective: to learn a tractable model distribution  $p_{\text{model}}$  (parametrized by  $\phi$ ) that approximates an unknown, complex data distribution  $p_{\text{data}}$ . A central challenge is the computational intractability of the normalizing constant, or partition function  $Z(\phi)$ , which is required to define a valid probability density.

To circumvent this problem, various families of deep generative models have been developed, each employing a distinct strategy. We surveyed several prominent approaches, including Energy-Based Models (EBMs), Autoregressive Models (ARs), Variational Autoencoders (VAEs), Normalizing Flows (NFs), and Generative Adversarial Networks (GANs). These models can be broadly categorized into explicit models, which define a tractable density, and implicit models, which define a distribution only through a sampling procedure.

While each of these classical frameworks is significant, three in particular serve as the conceptual origins for the diffusion models that are the focus of this monograph: VAEs, EBMs, and NFs. In the chapters that follow, we will trace the evolution of diffusion models from these three foundational paradigms:

1. Part B will begin by exploring the variational perspective (Chapter 2), showing how (the hierarchical latent variable structure of) VAEs leads naturally to the formulation of Denoising Diffusion Probabilistic Models (DDPMs).
2. Next, we will examine the score-based perspective (Chapter 3), which originates from EBMs and score matching, and develops into Noise Conditional Score Networks (NCSN) and the more general Score SDE framework (Chapter 4).
3. Finally, we will investigate the flow-based perspective (Chapter 5), which builds upon the principles of Normalizing Flows to frame generation as a continuous transformation, generalized by the concept of Flow Matching.

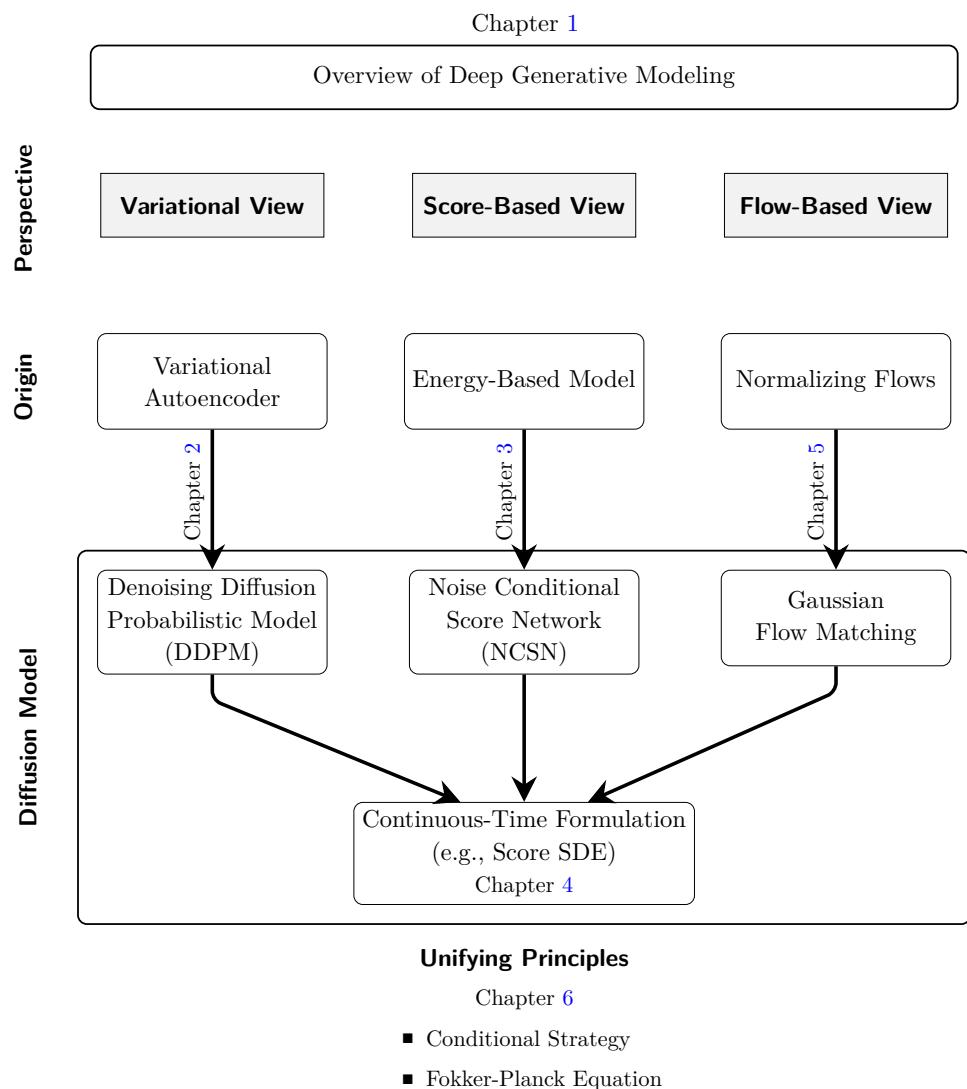
By understanding these origins, we will build a coherent framework for interpreting the diverse formulations of diffusion models and uncovering the deep principles that unify them.



**Figure 1.2: Computation graphs of prominent deep generative models.** Top to bottom: **EBM** maps an input  $\mathbf{x}$  to a scalar energy; **AR** generates a sequence  $\{\mathbf{x}_\ell\}$  left to right with causal dependencies; **VAE** encodes  $\mathbf{x}$  to a latent  $\mathbf{z}$  and decodes to a reconstruction  $\mathbf{x}'$ ; **NF** applies an invertible map  $f_\phi$  between  $\mathbf{x}$  and  $\mathbf{z}$  and uses  $f_\phi^{-1}$  to produce  $\mathbf{x}'$ ; **GAN** transforms noise  $\mathbf{z}$  to a sample  $\mathbf{x}'$  that is judged against real  $\mathbf{x}$  by a discriminator  $D_\zeta$ ; **DM** iteratively refines a noisy sample through a multi-step denoising chain  $\{\mathbf{x}_\ell\}$ . Boxes denote variables, trapezoids are learnable networks, ovals are scalars; arrows indicate computation flow.

## **Part B**

# **Origins and Foundations of Diffusion Models**



# 2

---

## Variational Perspective: From VAEs to DDPMs

---

In this chapter we view diffusion models through a variational lens. We begin with the Variational Autoencoders (VAEs), which represents data with latent variables and is trained by maximizing a tractable lower bound on the log likelihood. In this setting a learned encoder maps observations to latents, and a learned decoder maps latents back to observations, closing the modeling loop.

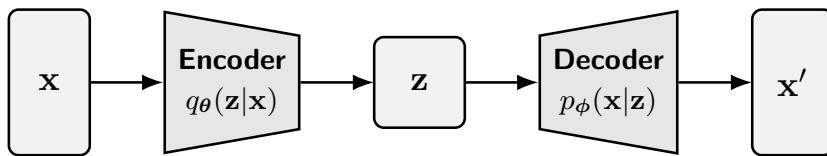
Building on this pattern, hierarchical variants (Hierarchical VAEs) stack several latent layers to capture structure at multiple scales. With this setup, Denoising Diffusion Probabilistic Models (DDPM) follow the same template: instead of jointly training both the encoder and decoder, the encoder is fixed as a forward noising process that gradually maps data to noise, and training learns a decoder that reverses this path in successive denoising steps. In this view, VAEs, hierarchical VAEs, and diffusion models all optimize a likelihood surrogate defined by a variational bound, providing a common foundation for the methods introduced here.

## 2.1 Variational Autoencoder

How can a neural network learn to generate realistic data? A natural starting point is the *autoencoder*, which consists of two networks: a deterministic *encoder* that compresses an input to a low-dimensional latent code, and a deterministic *decoder* that reconstructs the input from this code. Training minimizes the reconstruction error between the original input and its reconstruction. While this setup enables accurate reconstruction, the latent space is unstructured: randomly sampling latent codes usually produces meaningless outputs, limiting the model’s use for generation.

The *Variational Autoencoder (VAE)* (Kingma and Welling, 2013) solves this by imposing a probabilistic structure on the latent space. This transforms the model from a simple reconstruction tool into a true generative model, capable of producing novel and realistic data.

### 2.1.1 Probabilistic Encoder and Decoder



**Figure 2.1: Illustration of a VAE.** It consists of a stochastic encoder  $q_{\theta}(z|x)$  that maps data  $x$  to a latent variable  $z$ , and a decoder  $p_{\phi}(x|z)$  that reconstructs data from the latent.

**Construction of Decoder (Generator).** In VAEs, we distinguish between two types of variables: *observed variables*  $\mathbf{x}$ , which correspond to the data we see (e.g., an image), and *latent variables*  $\mathbf{z}$ , which capture the hidden factors of variation (e.g., object shape, color, or style). The model assumes that each observation  $\mathbf{x}$  is generated from a latent variable sampled from a simple *prior distribution*, typically a standard Gaussian,  $\mathbf{z} \sim p_{\text{prior}} := \mathcal{N}(\mathbf{0}, \mathbf{I})$ .

To map  $\mathbf{z}$  back to data space, we define a *decoder (generator)* distribution  $p_{\phi}(\mathbf{x}|\mathbf{z})$ . In practice, this decoder is kept simple, often a factorized Gaussian (see Section 2.1.3) or similar distribution, so that learning focuses on extracting useful latent features rather than memorizing data. Intuitively, directly generating pixels one by one is extremely hard; instead, the latent variable provides a compact representation, from which decoding the exact pixel arrangement becomes much easier. New samples are drawn by first sampling  $\mathbf{z} \sim p_{\text{prior}}$  and then decoding via  $\mathbf{x} \sim p_{\phi}(\mathbf{x}|\mathbf{z})$ .

The VAE thereby defines a latent-variable generative model through the marginal likelihood:

$$p_\phi(\mathbf{x}) = \int p_\phi(\mathbf{x}|\mathbf{z})p(\mathbf{z}) d\mathbf{z}.$$

Ideally, the decoder parameters  $\phi$  are learned by maximizing this marginal likelihood, as in maximum likelihood estimation (see Equation (1.1.2)). However, because the integral over  $\mathbf{z}$  is intractable for expressive, non-linear decoders, direct MLE is computationally infeasible, motivating the variational approach used in VAEs.

**Construction of Encoder (Inference Network).** To connect our intractable generator to real data, consider the reverse question: given an observation  $\mathbf{x}$ , what latent codes  $\mathbf{z}$  could have produced it? By Bayes’ rule, the posterior distribution is

$$p_\phi(\mathbf{z}|\mathbf{x}) = \frac{p_\phi(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p_\phi(\mathbf{x})}.$$

The difficulty is that the denominator involves the marginal likelihood  $p_\phi(\mathbf{x})$ , which requires integrating over all latent variables and is intractable for nonlinear decoders. Thus, exact inference of  $\mathbf{z}$  from  $\mathbf{x}$  is computationally prohibitive.

The “variational” step in VAEs addresses this by replacing the intractable posterior with a tractable approximation. We introduce an *encoder* (or inference network)  $q_\theta(\mathbf{z}|\mathbf{x})$ , parameterized by a neural network, whose role is to serve as a learnable proxy:

$$q_\theta(\mathbf{z}|\mathbf{x}) \approx p_\phi(\mathbf{z}|\mathbf{x}).$$

In practice, the encoder maps each observed data point  $\mathbf{x}$  to a distribution over latent codes, providing a feasible and trainable pathway from  $\mathbf{x}$  back to  $\mathbf{z}$  that enables learning.

### 2.1.2 Training via the Evidence Lower Bound (ELBO)

We now define a computable training objective. While we cannot directly optimize  $\log p_\phi(\mathbf{x})$ , we can maximize a lower bound on it—the *Evidence Lower Bound (ELBO)*:

### Theorem 2.1.1: Evidence Lower Bound (ELBO)

For any data point  $\mathbf{x}$ , the log-likelihood satisfies:

$$\log p_\phi(\mathbf{x}) \geq \mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}),$$

where the ELBO is given by:

$$\mathcal{L}_{\text{ELBO}} = \underbrace{\mathbb{E}_{\mathbf{z} \sim q_\theta(\mathbf{z}|\mathbf{x})} [\log p_\phi(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Term}} - \underbrace{\mathcal{D}_{\text{KL}}(q_\theta(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))}_{\text{Latent Regularization}}. \quad (2.1.1)$$

#### **Proof for Theorem.**

The ELBO arises from Jensen's inequality:

$$\begin{aligned} \log p_\phi(\mathbf{x}) &= \log \int p_\phi(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \log \int q_\theta(\mathbf{z}|\mathbf{x}) \frac{p_\phi(\mathbf{x}, \mathbf{z})}{q_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\ &= \log \mathbb{E}_{\mathbf{z} \sim q_\theta(\mathbf{z}|\mathbf{x})} \left[ \frac{p_\phi(\mathbf{x}, \mathbf{z})}{q_\theta(\mathbf{z}|\mathbf{x})} \right] \geq \mathbb{E}_{\mathbf{z} \sim q_\theta(\mathbf{z}|\mathbf{x})} \left[ \log \frac{p_\phi(\mathbf{x}, \mathbf{z})}{q_\theta(\mathbf{z}|\mathbf{x})} \right]. \end{aligned}$$

The ELBO objective naturally decomposes into two parts:

- **Reconstruction:** Encourages accurate recovery of  $\mathbf{x}$  from its latent code  $\mathbf{z}$ . With Gaussian encoder and decoder assumptions, this term reduces exactly to the familiar reconstruction loss of an autoencoder (cf. Section 2.1.3). However, as in autoencoders, optimizing this term alone risks memorizing the training data, motivating an additional regularization.
- **Latent KL:** Encourages the encoder distribution  $q_\theta(\mathbf{z}|\mathbf{x})$  to stay close to a simple Gaussian prior  $p_{\text{prior}}(\mathbf{z})$ . This regularization shapes the latent space into a smooth and continuous structure, enabling meaningful generation by ensuring that samples drawn from the prior can be reliably decoded.

This trade-off ensures both faithful reconstructions and coherent sampling.

**Information-Theoretic View: ELBO as a Divergence Bound.** The ELBO objective has a natural information-theoretic interpretation. Recall that maximum likelihood training amounts to minimizing the KL divergence

$$\mathcal{D}_{\text{KL}}(p_{\text{data}}(\mathbf{x}) \| p_\phi(\mathbf{x})),$$

which measures how well the model distribution approximates the data distribution. Since this term is intractable in general, the variational framework introduces a joint comparison.

Specifically, consider two joint distributions:

- The **generative joint**,  $p_\phi(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p_\phi(\mathbf{x}|\mathbf{z})$ , which describes how the model generates data;
- The **inference joint**,  $q_\theta(\mathbf{x}, \mathbf{z}) = p_{\text{data}}(\mathbf{x})q_\theta(\mathbf{z}|\mathbf{x})$ , which couples real data with its inferred latent.

Comparing these distributions yields the inequality

$$\mathcal{D}_{\text{KL}}(p_{\text{data}}(\mathbf{x}) \| p_\phi(\mathbf{x})) \leq \mathcal{D}_{\text{KL}}(q_\theta(\mathbf{x}, \mathbf{z}) \| p_\phi(\mathbf{x}, \mathbf{z})), \quad (2.1.2)$$

sometimes referred to as the chain rule for KL divergence. Intuitively, comparing only marginals ( $\mathbf{x}$ ) can hide mismatches that are revealed when the full latent–data joint is considered.

Formally, one can expand the joint KL as

$$\begin{aligned} & \underbrace{\mathcal{D}_{\text{KL}}(q_\theta(\mathbf{x}, \mathbf{z}) \| p_\phi(\mathbf{x}, \mathbf{z}))}_{\text{Total Error Bound}} \\ &= \mathbb{E}_{q_\theta(\mathbf{x}, \mathbf{z})} \left[ \log \frac{p_{\text{data}}(\mathbf{x})q_\theta(\mathbf{z}|\mathbf{x})}{p_\phi(\mathbf{x})p_\phi(\mathbf{z}|\mathbf{x})} \right] \\ &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[ \log \frac{p_{\text{data}}(\mathbf{x})}{p_\phi(\mathbf{x})} + \mathcal{D}_{\text{KL}}(q_\theta(\mathbf{z}|\mathbf{x}) \| p_\phi(\mathbf{z}|\mathbf{x})) \right] \\ &= \underbrace{\mathcal{D}_{\text{KL}}(p_{\text{data}} \| p_\phi)}_{\text{True Modeling Error}} + \underbrace{\mathbb{E}_{p_{\text{data}}(\mathbf{x})} [\mathcal{D}_{\text{KL}}(q_\theta(\mathbf{z}|\mathbf{x}) \| p_\phi(\mathbf{z}|\mathbf{x}))]}_{\text{Inference Error}}, \end{aligned}$$

where the first term is the true modeling error and the second is the inference error, i.e., the gap between the approximate and true posteriors. The latter is always non-negative, which explains Equation (2.1.2).

Finally, note that

$$\log p_\phi(\mathbf{x}) - \mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathcal{D}_{\text{KL}}(q_\theta(\mathbf{z}|\mathbf{x}) \| p_\phi(\mathbf{z}|\mathbf{x})).$$

Thus the inference error is exactly the gap between the log-likelihood and the ELBO. Maximizing the ELBO therefore corresponds to directly reducing inference error, ensuring that training minimizes a meaningful part of the overall bound.

### 2.1.3 Gaussian VAE

A standard formulation of the VAE employs Gaussian distributions for both the encoder and decoder.

**Encoder Part.** The encoder  $q_{\theta}(\mathbf{z}|\mathbf{x})$  is typically modeled as a Gaussian distribution as:

$$q_{\theta}(\mathbf{z}|\mathbf{x}) := \mathcal{N}\left(\mathbf{z}; \boldsymbol{\mu}_{\theta}(\mathbf{x}), \text{diag}(\boldsymbol{\sigma}_{\theta}^2(\mathbf{x}))\right),$$

where  $\boldsymbol{\mu}_{\theta} : \mathbb{R}^D \rightarrow \mathbb{R}^d$  and  $\boldsymbol{\sigma}_{\theta} : \mathbb{R}^D \rightarrow \mathbb{R}_+^d$  are deterministic outputs of the encoder network.

**Decoder Part.** The decoder is typically modeled as a Gaussian distribution with fixed variance:

$$p_{\phi}(\mathbf{x}|\mathbf{z}) := \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{\phi}(\mathbf{z}), \sigma^2 \mathbf{I}),$$

where  $\boldsymbol{\mu}_{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^D$  is a neural network, and  $\sigma > 0$  is a small constant controlling the variance.

Under this assumption, the reconstruction term in the ELBO simplifies as

$$\mathbb{E}_{q_{\theta}(\mathbf{z}|\mathbf{x})} [\log p_{\phi}(\mathbf{x}|\mathbf{z})] = -\frac{1}{2\sigma^2} \mathbb{E}_{q_{\theta}(\mathbf{z}|\mathbf{x})} [\|\mathbf{x} - \boldsymbol{\mu}_{\phi}(\mathbf{z})\|^2] + C,$$

where  $C$  is a constant independent of  $\theta$  and  $\phi$ . The ELBO objective thus reduces to:

$$\min_{\theta, \phi} \mathbb{E}_{q_{\theta}(\mathbf{z}|\mathbf{x})} \left[ \frac{1}{2\sigma^2} \|\mathbf{x} - \boldsymbol{\mu}_{\phi}(\mathbf{z})\|^2 \right] + \mathcal{D}_{\text{KL}}(q_{\theta}(\mathbf{z}|\mathbf{x}) \| p_{\text{prior}}(\mathbf{z})),$$

where the KL term admits a closed-form solution due to the Gaussian assumption. Training the VAE therefore reduces to minimizing a regularized reconstruction loss.

### 2.1.4 Drawbacks of Standard VAE

Despite the theoretical appeal of the VAE framework, it suffers from a critical drawback: it often produces blurry outputs.

**Blurry Generations in VAEs.** To understand this phenomenon, consider a fixed Gaussian encoder  $q_{\text{enc}}(\mathbf{z}|\mathbf{x})$ , and a decoder of the form

$$p_{\text{dec}}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}(\mathbf{z}), \sigma^2 \mathbf{I}),$$

where  $\mu(\mathbf{z})$  denotes the decoder network. With an arbitrary encoder, optimizing the ELBO reduces (up to an additive constant) to minimizing the expected reconstruction error:

$$\arg \min_{\mu} \mathbb{E}_{p_{\text{data}}(\mathbf{x})q_{\text{enc}}(\mathbf{z}|\mathbf{x})} [\|\mathbf{x} - \mu(\mathbf{z})\|^2].$$

This is a standard least squares problem in  $\mu(\mathbf{z})$ , and its solution is given in closed form by the conditional mean:

$$\mu^*(\mathbf{z}) = \mathbb{E}_{q_{\text{enc}}(\mathbf{x}|\mathbf{z})}[\mathbf{x}],$$

where  $q_{\text{enc}}(\mathbf{x}|\mathbf{z})$  is the encoder-induced posterior on inputs given latents, defined via Bayes' rule:

$$q_{\text{enc}}(\mathbf{x}|\mathbf{z}) = \frac{q_{\text{enc}}(\mathbf{z}|\mathbf{x})p_{\text{data}}(\mathbf{x})}{p_{\text{prior}}(\mathbf{z})}.$$

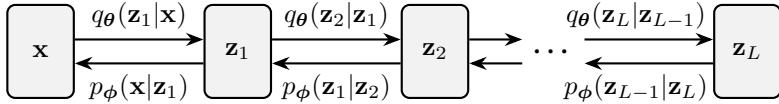
An equivalent form of the optimal generator via Bayes' rule is:

$$\mu^*(\mathbf{z}) = \frac{\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[q_{\text{enc}}(\mathbf{z}|\mathbf{x}) \cdot \mathbf{x}]}{\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[q_{\text{enc}}(\mathbf{z}|\mathbf{x})]}.$$

Now suppose that two distinct inputs  $\mathbf{x} \neq \mathbf{x}'$  are mapped to overlapping regions in latent space, i.e., the supports of  $q_{\text{enc}}(\cdot|\mathbf{x})$  and  $q_{\text{enc}}(\cdot|\mathbf{x}')$  intersect. That is,  $\mu^*(\mathbf{z})$  averages over multiple, potentially unrelated inputs, which leads to blurry, non-distinct outputs. This averaging effect over conflicting modes is a fundamental reason for the characteristic blurriness in VAE-generated samples.

### 2.1.5 (Optional) From Standard VAE to Hierarchical VAEs

To model complex data, Hierarchical Variational Autoencoders (HVAEs) (Vahdat and Kautz, 2020) enhance VAEs by introducing a hierarchy of latent variables. This deep, layered structure allows the model to capture data features at multiple levels of abstraction, significantly boosting expressive power and mirroring the compositional nature of real-world data.



**Figure 2.2: Computation graph of the HVAE.** It has a hierarchical structure with stacked, trainable encoders and decoders across multiple latent layers.

**HVAE's Modeling.** Unlike standard VAEs that use a single latent code  $\mathbf{z}$ , hierarchical VAEs (HVAEs) introduce multiple layers of latent variables arranged in a top-down hierarchy. Each latent layer conditions the one below it, forming a chain of conditional priors that captures structure at progressively finer levels of abstraction. This leads to the following top-down factorization of the joint distribution:

$$p_{\phi}(\mathbf{x}, \mathbf{z}_{1:L}) = p_{\phi}(\mathbf{x}|\mathbf{z}_1) \prod_{i=2}^L p_{\phi}(\mathbf{z}_{i-1}|\mathbf{z}_i)p(\mathbf{z}_L).$$

This structure defines the marginal data distribution,

$$p_{\text{HVAE}}(\mathbf{x}) := \int p_{\phi}(\mathbf{x}, \mathbf{z}_{1:L}) d\mathbf{z}_{1:L}.$$

Generation proceeds progressively: starting from the top latent variable  $\mathbf{z}_L$ , each latent is decoded sequentially down to  $\mathbf{z}_1$ , followed by generating the final observation  $\mathbf{x}$ .

For encoding part, HVAEs utilize a structured, learnable variational encoder  $q_{\theta}(\mathbf{z}_{1:L}|\mathbf{x})$  that mirrors the generative hierarchy. A common choice is a bottom-up Markov factorization:

$$q_{\theta}(\mathbf{z}_{1:L}|\mathbf{x}) = q_{\theta}(\mathbf{z}_1|\mathbf{x}) \prod_{i=2}^L q_{\theta}(\mathbf{z}_i|\mathbf{z}_{i-1}).$$

**HVAE's ELBO.** Similar to Equation (2.1.1), ELBO is derived via Jensen's inequality:

$$\begin{aligned}
\log p_{\text{HVAE}}(\mathbf{x}) &= \log \int p_{\phi}(\mathbf{x}, \mathbf{z}_{1:L}) d\mathbf{z}_{1:L} \\
&= \log \int \frac{p_{\phi}(\mathbf{x}, \mathbf{z}_{1:L})}{q_{\theta}(\mathbf{z}_{1:L}|\mathbf{x})} q_{\theta}(\mathbf{z}_{1:L}|\mathbf{x}) d\mathbf{z}_{1:L} \\
&= \log \mathbb{E}_{q_{\theta}(\mathbf{z}_{1:L}|\mathbf{x})} \left[ \frac{p_{\phi}(\mathbf{x}, \mathbf{z}_{1:L})}{q_{\theta}(\mathbf{z}_{1:L}|\mathbf{x})} \right] \\
&\geq \mathbb{E}_{q_{\theta}(\mathbf{z}_{1:L}|\mathbf{x})} \left[ \log \frac{p_{\phi}(\mathbf{x}, \mathbf{z}_{1:L})}{q_{\theta}(\mathbf{z}_{1:L}|\mathbf{x})} \right] \\
&=: \mathcal{L}_{\text{ELBO}}(\phi).
\end{aligned} \tag{2.1.3}$$

Substituting the factorized forms yields:

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q_{\theta}(\mathbf{z}_{1:L}|\mathbf{x})} \left[ \log \frac{p(\mathbf{z}_L) \prod_{i=2}^L p_{\phi}(\mathbf{z}_{i-1}|\mathbf{z}_i) p_{\phi}(\mathbf{x}|\mathbf{z}_1)}{q_{\theta}(\mathbf{z}_1|\mathbf{x}) \prod_{i=2}^L q_{\theta}(\mathbf{z}_i|\mathbf{z}_{i-1})} \right].$$

This hierarchical ELBO decomposes into interpretable terms, including a reconstruction term and KL divergences between each generative conditional and its corresponding variational approximation.

The leap from shallow to deep networks revolutionized machine learning, and a similar idea transformed generative models. HVAEs showed the power of using deep, stacked layers to build data. This concept of a layered hierarchy is a cornerstone of modern generative modeling, appearing again in score-based methods (Section 3.4) and normalizing flows (Section 5.1). The core insight is simple yet powerful:

### Observation 2.1.1:

Stacking layers allows the model to generate data progressively, starting with coarse details and adding finer ones at each step. This process makes it far easier to capture the complex structure of high-dimensional data.

**Why Deeper Networks in a Flat VAE are Not Enough.** There are two fundamental limitations of a standard flat VAE that are not resolved by simply making the encoder and decoder deeper.

The first limitation is the variational family. In a standard VAE,

$$q_{\theta}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_{\theta}(\mathbf{x}), \text{diag}(\sigma_{\theta}^2(\mathbf{x}))),$$

so for each fixed  $\mathbf{x}$  the encoder posterior is a single Gaussian with diagonal covariance. Greater network depth improves the accuracy of  $\mu_\theta$  and  $\sigma_\theta$  but does not expand the family; even a full covariance remains one unimodal ellipsoid. When  $p_\phi(\mathbf{z}|\mathbf{x})$  is multi-peaked, this family cannot match it, which loosens the ELBO and weakens inference. Addressing this requires a richer posterior class, not merely deeper networks.

Second, if the decoder is too expressive, the model may suffer from posterior collapse. To see why, let us recall that the objective of the VAE is

$$\begin{aligned} & \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\mathcal{L}_{\text{ELBO}}(\mathbf{x})] \\ &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})q_\theta(\mathbf{z}|\mathbf{x})}[\log p_\phi(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\mathcal{D}_{\text{KL}}(q_\theta(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))] \\ &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})q_\theta(\mathbf{z}|\mathbf{x})}[\log p_\phi(\mathbf{x}|\mathbf{z})] - \mathcal{I}_q(\mathbf{x}; \mathbf{z}) - \mathcal{D}_{\text{KL}}(q_\theta(\mathbf{z})\|p(\mathbf{z})), \end{aligned}$$

where  $\mathcal{I}_q(\mathbf{x}; \mathbf{z})$  is the mutual information defined by

$$\mathcal{I}_q(\mathbf{x}; \mathbf{z}) = \mathbb{E}_{q(\mathbf{x}, \mathbf{z})}\left[\log \frac{q_\theta(\mathbf{z}|\mathbf{x})}{q(\mathbf{z})}\right] = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\mathcal{D}_{\text{KL}}(q_\theta(\mathbf{z}|\mathbf{x})\|q(\mathbf{z}))],$$

and the aggregated posterior is  $q_\theta(\mathbf{z}) = \int p_{\text{data}}(\mathbf{x})q_\theta(\mathbf{z}|\mathbf{x}) d\mathbf{x}$ .

If the decoder class can model the data well without using  $\mathbf{z}$  (i.e., it contains some  $p_\phi(\mathbf{x}|\mathbf{z}) = r(\mathbf{x})$  close to  $p_{\text{data}}$ ), then a maximizer of the ELBO sets  $q_\theta(\mathbf{z}|\mathbf{x}) = p(\mathbf{z})$ , so  $\mathcal{I}_q(\mathbf{x}; \mathbf{z}) = 0$  and  $q_\theta(\mathbf{z}) = p(\mathbf{z})$ . This “ignore  $\mathbf{z}$ ” solution does not disappear by making the networks deeper: (1) the learned code becomes independent of  $\mathbf{x}$  (so it carries no data-dependent structure useful for downstream tasks), and (2) conditioning or moving in  $\mathbf{z}$  has no effect on generated samples, so controllable generation fails.

**What Hierarchy Changes?** An HVAE introduces multiple latent levels,

$$p_\phi(\mathbf{x}, \mathbf{z}_{1:L}) = p_\phi(\mathbf{x}|\mathbf{z}_1) \prod_{i=2}^L p_\phi(\mathbf{z}_{i-1}|\mathbf{z}_i)p(\mathbf{z}_L),$$

with ELBO

$$\begin{aligned} \mathcal{L}_{\text{ELBO}}(\mathbf{x}) &= \mathbb{E}_q[\log p_\phi(\mathbf{x}|\mathbf{z}_1)] - \mathbb{E}_q\left[\mathcal{D}_{\text{KL}}(q_\theta(\mathbf{z}_1|\mathbf{x})\|p_\phi(\mathbf{z}_1|\mathbf{z}_2))\right] \\ &\quad - \sum_{i=2}^{L-1} \mathbb{E}_q\left[\mathcal{D}_{\text{KL}}(q_\theta(\mathbf{z}_i|\mathbf{z}_{i-1})\|p_\phi(\mathbf{z}_i|\mathbf{z}_{i+1}))\right] \\ &\quad - \mathbb{E}_q\left[\mathcal{D}_{\text{KL}}(q_\theta(\mathbf{z}_L|\mathbf{z}_{L-1})\|p(\mathbf{z}_L))\right]. \end{aligned}$$

Here, we denote  $\mathbb{E}_q := \mathbb{E}_{p_{\text{data}}(\mathbf{x})q_\theta(\mathbf{z}_{1:L}|\mathbf{x})}$ . Each inference conditional is aligned with its top-down generative counterpart:  $q_\theta(\mathbf{z}_1|\mathbf{x})$  with  $p_\phi(\mathbf{z}_1|\mathbf{z}_2)$ , intermediate

layers with  $p_\phi(\mathbf{z}_i|\mathbf{z}_{i+1})$ , and the top with the prior  $p(\mathbf{z}_L)$ . This distributes the information penalty across levels and localizes learning signals through these adjacent KL terms. These properties stem from the hierarchical latent graph, not from simply deepening networks in a flat VAE.

**What Will be Ahead?** While HVAEs extend the VAE framework with multiple latent layers for expressiveness, their training poses unique challenges. Because the encoder and decoder must be optimized jointly, learning becomes unstable: lower layers and the decoder can already reconstruct  $\mathbf{x}$ , leaving higher-level latents with little effective signal. Moreover, gradient information reaching deeper variables is often indirect and weak, making it difficult for them to contribute meaningfully. An additional difficulty lies in balancing model capacity, since overly expressive conditionals can dominate the reconstruction task and suppress the utility of higher latents.

Interestingly, the core idea of a deep, layered hierarchy finds a more powerful incarnation in variational diffusion models, a topic we explore in Section 2.2. Diffusion models inherit the progressive structure of HVAEs but elegantly sidestep their central weakness. By fixing the encoding process and focusing solely on learning the generative reversal, they unlock newfound stability and modeling flexibility, leading to a significant leap in the quality of generated outputs.

For notational simplicity, we deviate from the common VAE convention that uses  $q$  for the encoder and  $p$  for the generator. To avoid ambiguity, we denote distributions as  $p$  and will always specify their roles through appropriate subscripts or superscripts, clarifying them in context.

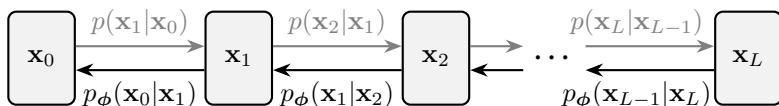
## 2.2 Variational Perspective: DDPM

Denoising Diffusion Probabilistic Models (DDPMs) (Sohl-Dickstein *et al.*, 2015; Ho *et al.*, 2020) represent a cornerstone of diffusion modeling. Conceptually, they operate within a variational framework, much like VAEs and HVAEs. However, DDPMs introduce a clever twist that tackles some of the challenges faced by their predecessors.

At their core, DDPMs involve two distinct stochastic processes:

- **The Forward Pass (Fixed Encoder):** This process gradually corrupts data by injecting Gaussian noise over multiple steps via a transition kernel  $p(\mathbf{x}_i|\mathbf{x}_{i-1})$ . The data evolves into an isotropic Gaussian distribution, effectively becoming pure noise. This means the encoder is fixed and not learned.
- **The Reverse Denoising Process (Learnable Decoder):** Here, a neural network learns to reverse the noise corruption through a parameterized distribution  $p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)$ . Starting from pure noise, this process iteratively denoises to generate realistic samples. Crucially, each individual denoising step is a more manageable task than generating a complete sample from scratch, as VAEs often attempt to do.

By fixing the encoder and concentrating learning on the gradual generative trajectory, DDPMs achieve remarkable stability and expressive power.



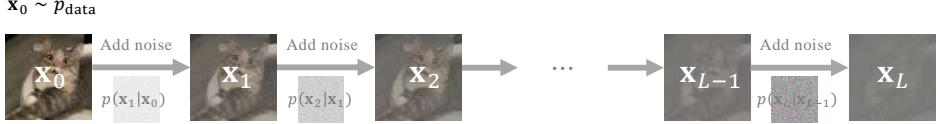
**Figure 2.3: Illustration of DDPM.** It consists of a fixed forward process (in gray) that gradually adds Gaussian noise to the data, and a learned reverse process that denoises step-by-step to generate new samples.

In this section, we focus on DDPMs, postponing the broader discussion to Section 4.4, where we present a more general and flexible framework.

### 2.2.1 Forward Process (Fixed Encoder)

In DDPMs, the forward process is a fixed, non-trainable operation that serves as an encoder. It progressively corrupts the original data by adding noise over multiple steps, eventually transforming it into a simple prior distribution

$p_{\text{prior}} := \mathcal{N}(\mathbf{0}, \mathbf{I})$ . This transformation is depicted as the forward chain in Figure 2.3 or illustration in Figure 2.4.



**Figure 2.4:** Illustration of the DDPM forward process, wherein Gaussian noise is incrementally added to corrupt a data sample into pure noise.

Let us formalize this step-by-step degradation:

**Fixed Gaussian Transitions.** Each step in the forward process is governed by a fixed Gaussian transition kernel<sup>1</sup>:

$$p(\mathbf{x}_i | \mathbf{x}_{i-1}) := \mathcal{N}(\mathbf{x}_i; \sqrt{1 - \beta_i^2} \mathbf{x}_{i-1}, \beta_i^2 \mathbf{I}).$$

Here, the process begins with  $\mathbf{x}_0$ , representing a sample drawn from the real data distribution  $p_{\text{data}}$ . The sequence  $\{\beta_i\}_{i=1}^L$  denotes a pre-determined, monotonically increasing noise schedule, where each  $\beta_i \in (0, 1)$  controls the variance of the Gaussian noise injected at step  $i$ . For convenience, we define  $\alpha_i := \sqrt{1 - \beta_i^2}$ . This mathematical definition is precisely equivalent to the following intuitive iterative update:

$$\mathbf{x}_i = \alpha_i \mathbf{x}_{i-1} + \beta_i \boldsymbol{\epsilon}_i,$$

where  $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  are independently and identically distributed. This means at each step  $i$ , we scale down the previous state  $\mathbf{x}_{i-1}$  by  $\alpha_i$  and add a controlled amount of Gaussian noise scaled by  $\beta_i$ .

**Perturbation Kernel and Prior Distribution.** By recursively applying the transition kernels, we obtain a closed-form expression for the distribution of noisy samples at step  $i$  given the original data  $\mathbf{x}_0$ :

$$p_i(\mathbf{x}_i | \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_i; \bar{\alpha}_i \mathbf{x}_0, (1 - \bar{\alpha}_i^2) \mathbf{I}\right),$$

where

$$\bar{\alpha}_i := \prod_{k=1}^i \sqrt{1 - \beta_k^2} = \prod_{k=1}^i \alpha_k.$$

---

<sup>1</sup>This formulation, while potentially appearing different, is mathematically equivalent to the original DDPM transition kernel.

This means we can sample  $\mathbf{x}_i$  directly from  $\mathbf{x}$  as<sup>2</sup>

$$\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_i^2} \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (2.2.1)$$

Let the noise schedule  $\{\beta_i\}_{i=1}^L$  be an increasing sequence, then the marginal distribution of the forward process converges as

$$p_L(\mathbf{x}_L | \mathbf{x}_0) \rightarrow \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad \text{as } L \rightarrow \infty,$$

which motivates the choice of the prior distribution as

$$p_{\text{prior}} := \mathcal{N}(\mathbf{0}, \mathbf{I})$$

with no reliance on data  $\mathbf{x}_0$ .

## 2.2.2 Reverse Denoising Process (Learnable Decoder)

At its core, the essence of DDPMs lies in their ability to *reverse* the controlled degradation imposed by the forward diffusion process. Starting from pure, unstructured noise,  $\mathbf{x}_L \sim p_{\text{prior}}$ , the objective is to progressively *denoise* this randomness, step by step, until a coherent and meaningful data sample emerges. This reverse generation proceeds through a Markov chain, illustrated by Figure 2.5.

The fundamental challenge, and the central question guiding DDPM development, then becomes:

### Question 2.2.1

*Can we precisely compute, or at least effectively approximate, these reverse transition kernels  $p(\mathbf{x}_{i-1} | \mathbf{x}_i)$ , especially when considering the complex distribution of  $\mathbf{x}_i \sim p_i(\mathbf{x}_i)$ ?*

---

<sup>2</sup>For a fixed index  $t$ , we will often use, both here and later on, the Gaussian perturbation form

$$p_t(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}),$$

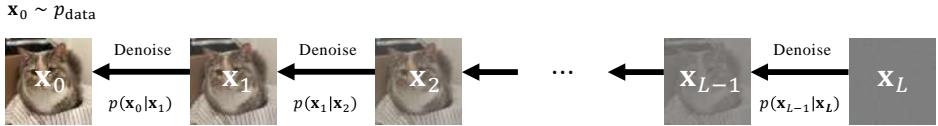
which we equivalently write as the identity *in distribution*

$$\mathbf{x}_t \stackrel{d}{=} \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}, \quad \text{that is} \quad \text{Law}(\mathbf{x}_t) = \text{Law}(\alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}),$$

where  $\mathbf{x}_0 \sim p_{\text{data}}$ ,  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  is independent of  $\mathbf{x}_0$ , and  $\alpha_t, \sigma_t$  are deterministic scalars. Equality “ $\stackrel{d}{=}$ ” means the two random variables have the same probability density (i.e., *law*), hence the same expectations for any test function  $\phi$ :

$$\mathbb{E}[\phi(\mathbf{x}_t)] = \mathbb{E}[\phi(\alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon})].$$

For brevity, we will write  $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$ , understood either as an equality in distribution or, by context, as a sample realization; this shorthand will be used throughout.



**Figure 2.5: Illustration of DDPM reverse (denoising) process.** Starting from noise  $\mathbf{x}_L \sim p_{\text{prior}}$ , the model sequentially samples  $\mathbf{x}_{i-1} \sim p(\mathbf{x}_{i-1}|\mathbf{x}_i)$  for  $i = L, \dots, 1$  to obtain a newly generated data  $\mathbf{x}$ . The oracle transition  $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$  is unknown; thus, we aim to approximate it.

Rather than diving immediately into the mathematically intricate derivation of the Evidence Lower Bound (ELBO), as the original DDPM paper does (for which a detailed discussion awaits in Section 2.2.5), we will instead approach the training objective from a more intuitive perspective: by leveraging conditional probabilities to achieve a tractable formulation.

**Overview: Modeling and Training Objective.** To enable the generative process, our goal is to approximate the unknown true reverse transition kernel,  $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$ . We achieve this by introducing a learnable parametric model,  $p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)$ , and training it to minimize the expected KL divergence:

$$\mathbb{E}_{p_i(\mathbf{x}_i)} [\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i) \| p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))] . \quad (2.2.2)$$

However, a direct computation of the target distribution  $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$  is challenging. By Bayes' theorem, we would need to evaluate:

$$p(\mathbf{x}_{i-1}|\mathbf{x}_i) = p(\mathbf{x}_i|\mathbf{x}_{i-1}) \underbrace{\frac{p_{i-1}(\mathbf{x}_{i-1})}{p_i(\mathbf{x}_i)}}_{\text{intractable}} .$$

The marginals  $p_i(\mathbf{x}_i)$  and  $p_{i-1}(\mathbf{x}_{i-1})$  are expectations over the unknown data distribution  $p_{\text{data}}$ , given by:

$$p_i(\mathbf{x}_i) = \int p_i(\mathbf{x}_i|\mathbf{x}_0)p_{\text{data}}(\mathbf{x}_0)d\mathbf{x}_0,$$

and analogously for  $p_{i-1}(\mathbf{x}_{i-1})$ . Since  $p_{\text{data}}$  is unknown, these integrals have no closed-form evaluation; at best they can be approximated from samples, so the exact densities are not available in practice.

**Overcoming Intractability with Conditioning.** A central insight in DDPMs resolves this intractability: we condition the reverse transition on a clean data

sample  $\mathbf{x}$ . This subtle yet powerful step transforms the intractable kernel into one that is mathematically tractable:

$$p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}) = p(\mathbf{x}_i|\mathbf{x}_{i-1}) \frac{p(\mathbf{x}_{i-1}|\mathbf{x})}{p(\mathbf{x}_i|\mathbf{x})}.$$

This tractability arises from two key properties of the forward process: its Markov property, meaning  $p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{x}) = p(\mathbf{x}_i|\mathbf{x}_{i-1})$ , and the Gaussian nature of all involved distributions. As a result,  $p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})$  itself is Gaussian and admits a closed-form expression (which we saw in Equation (2.2.4)). Crucially, this elegant conditioning strategy allows us to derive a tractable objective that is functionally equivalent to the seemingly intractable marginal KL divergence in Equation (2.2.2).

### Theorem 2.2.1: Equivalence Between Marginal and Conditional KL Minimization

The following equality holds:

$$\begin{aligned} & \mathbb{E}_{p_i(\mathbf{x}_i)} [\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i) \| p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))] \\ &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{p(\mathbf{x}_i|\mathbf{x})} [\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}) \| p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))] + C, \end{aligned} \quad (2.2.3)$$

where  $C$  is a constant independent of  $\phi$ . Moreover, the minimizer of Equation (2.2.3) satisfies

$$p^*(\mathbf{x}_{i-1}|\mathbf{x}_i) = \mathbb{E}_{p(\mathbf{x}|\mathbf{x}_i)} [p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})] = p(\mathbf{x}_{i-1}|\mathbf{x}_i), \quad \mathbf{x}_i \sim p_i.$$

#### Proof for Theorem.

The proof rewrites a KL-divergence expectation by expanding definitions, applying the chain rule of probability, and using a logarithmic identity to decompose it into the sum of an expected conditional KL divergence and a marginal KL divergence. A complete derivation is in Section D.1.1. ■

This alternative viewpoint: conditioning to obtain a tractable objective, forms the foundation of DDPMs and reveals a profound commonality with other influential diffusion models, as we will explore in Chapter 3 and Chapter 5.

It reveals a powerful equivalence: minimizing the KL divergence between marginal distributions is mathematically identical to minimizing the KL divergence between specific conditional distributions. This latter formulation is exceptionally useful because the crucial conditional distribution,  $p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})$ , possesses a convenient closed-form expression:

**Lemma 2.2.2: Reverse Conditional Transition Kernel**

$p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})$  is Gaussian with the closed-form expression:

$$p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}) = \mathcal{N}\left(\mathbf{x}_{i-1}; \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}, i), \sigma^2(i)\mathbf{I}\right),$$

where

$$\boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}, i) := \frac{\bar{\alpha}_{i-1}\beta_i^2}{1-\bar{\alpha}_i^2}\mathbf{x} + \frac{(1-\bar{\alpha}_{i-1}^2)\alpha_i}{1-\bar{\alpha}_i^2}\mathbf{x}_i, \quad \sigma^2(i) := \frac{1-\bar{\alpha}_{i-1}^2}{1-\bar{\alpha}_i^2}\beta_i^2. \quad (2.2.4)$$

Later in Lemma 4.4.2, we present a more general formula that extends beyond the DDPM noising process described in Equation (2.2.1).

**2.2.3 Modeling of Reverse Transition Kernel  $p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)$** 

Leveraging the gradient-level equivalence as in Theorem 2.2.1 and the Gaussian form of the reverse conditional  $p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})$  as in Lemma 2.2.2, DDPM assumes that each reverse transition  $p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)$  is Gaussian, parameterized as

$$p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i) := \mathcal{N}(\mathbf{x}_{i-1}; \boldsymbol{\mu}_\phi(\mathbf{x}_i, i), \sigma^2(i)\mathbf{I}), \quad (2.2.5)$$

where  $\boldsymbol{\mu}_\phi(\cdot, i) : \mathbb{R}^D \rightarrow \mathbb{R}^D$  is a learnable mean function, and  $\sigma^2(i) > 0$  is fixed as defined in Equation (2.2.4).

We denote the KL divergence, averaged over time steps  $i$  and conditioned on data  $\mathbf{x}_0 \sim p_{\text{data}}$ , to match all layers of distributions as:

$$\mathcal{L}_{\text{diffusion}}(\mathbf{x}_0; \phi) := \sum_{i=1}^L \mathbb{E}_{p(\mathbf{x}_i|\mathbf{x}_0)} [\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \| p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))]. \quad (2.2.6)$$

Thanks to the Gaussian forms of both distributions and the parameterization defined in Equation (2.2.5), the objective admits a closed-form expression and can be simplified as:

$$\mathcal{L}_{\text{diffusion}}(\mathbf{x}_0; \phi) = \sum_{i=1}^L \frac{1}{2\sigma^2(i)} \|\boldsymbol{\mu}_\phi(\mathbf{x}_i, i) - \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i)\|_2^2 + C, \quad (2.2.7)$$

where  $C$  is a constant independent of  $\phi$ . Averaging over the data distribution and omitting the constant  $C$  (which does not affect the optimization), the final DDPM training objective is

$$\mathcal{L}_{\text{DDPM}}(\phi) := \sum_{i=1}^L \frac{1}{2\sigma^2(i)} \mathbb{E}_{\mathbf{x}_0} \mathbb{E}_{p(\mathbf{x}_i|\mathbf{x}_0)} [\|\boldsymbol{\mu}_\phi(\mathbf{x}_i, i) - \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i)\|_2^2], \quad (2.2.8)$$

where  $\mathbf{x}_0 \sim p_{\text{data}}$ .

### 2.2.4 Practical Choices of Predictions and Loss

**$\epsilon$ -Prediction.** In typical DDPM implementations, training is not conducted directly using the original loss based on the *mean prediction* parameterization from Equation (2.2.8). Instead, an equivalent reparameterization, known as the  $\epsilon$ -*prediction* (noise prediction) formulation, is commonly adopted.

Recall that in the DDPM forward process, a noisy sample  $\mathbf{x}_i \sim p(\mathbf{x}_i | \mathbf{x})$  at noise level  $i$  is generated by

$$\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_i^2} \boldsymbol{\epsilon}, \quad \mathbf{x}_0 \sim p_{\text{data}}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (2.2.9)$$

Using this expression, the reverse mean  $\mu(\mathbf{x}_i, \mathbf{x}_0, i)$  from Equation (2.2.4) can be rewritten as:

$$\mu(\mathbf{x}_i, \mathbf{x}_0, i) = \frac{1}{\alpha_i} \left( \mathbf{x}_i - \frac{1 - \alpha_i^2}{\sqrt{1 - \bar{\alpha}_i^2}} \boldsymbol{\epsilon} \right).$$

This motivates a parameterization of the model mean  $\mu_\phi$  using a neural network  $\epsilon_\phi(\mathbf{x}_i, i)$  that directly predicts the noise:

$$\mu_\phi(\mathbf{x}_i, i) = \frac{1}{\alpha_i} \left( \mathbf{x}_i - \frac{1 - \alpha_i^2}{\sqrt{1 - \bar{\alpha}_i^2}} \underbrace{\epsilon_\phi(\mathbf{x}_i, i)}_{\epsilon\text{-prediction}} \right).$$

Substituting this into the original loss leads to a squared  $\ell_2$  error between predicted and true noise:

$$\|\mu_\phi(\mathbf{x}_i, i) - \mu(\mathbf{x}_i, \mathbf{x}_0, i)\|_2^2 \propto \|\epsilon_\phi(\mathbf{x}_i, i) - \boldsymbol{\epsilon}\|_2^2,$$

up to a weighting factor depending on  $i$ . Intuitively, the model acts as a “noise detective”, estimating the random noise added at each step of the forward process. Subtracting this estimate from the corrupted sample moves it closer to the clean original, and repeating this step-by-step reconstructs the data from pure noise.

**Simplified Loss with  $\epsilon$ -Prediction.** In practice, this expression is further simplified by omitting the weighting term, yielding the widely used DDPM training loss:

$$\mathcal{L}_{\text{simple}}(\phi) := \mathbb{E}_i \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[ \|\epsilon_\phi(\mathbf{x}_i, i) - \boldsymbol{\epsilon}\|_2^2 \right], \quad (2.2.10)$$

where  $\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_i^2} \boldsymbol{\epsilon}$  with  $\mathbf{x}_0 \sim p_{\text{data}}$ . Since the target noise has unit variance at every timestep  $t$ , the  $\ell_2$  loss in Equation (2.2.10) maintains a

consistent scale across all  $t$ . This prevents vanishing or exploding targets and eliminates the need for explicit loss weighting.

Importantly, both  $\mathcal{L}_{\text{DDPM}}$  and  $\mathcal{L}_{\text{simple}}$  share the same optimal solution  $\epsilon^*$ , this is because Equation (2.2.10) essentially reduces to a least-squares problem (as shown similarly in Proposition 4.2.1 or Proposition 6.3.1):

$$\epsilon^*(\mathbf{x}_i, i) = \mathbb{E}[\epsilon|\mathbf{x}_i], \quad \mathbf{x}_i \sim p_i.$$

Intuitively, the  $\epsilon$ -prediction network  $\epsilon_\phi(\mathbf{x}_i, i)$  estimates the noise added by the forward process to produce  $\mathbf{x}_i$ . At optimality, this estimate coincides with the conditional expectation of the true noise, even though  $\mathbf{x}_i$  does not uniquely determine the original clean sample.

**Another Equivalent Parametrization:  $\mathbf{x}$ -Prediction.** Equation (2.2.4) motivates an alternative yet equivalent parameterization, known as  $\mathbf{x}$ -*prediction* (clean prediction), in which a neural network  $\mathbf{x}_\phi(\mathbf{x}_i, i)$  is trained to predict a clean (denoised) sample from a given noisy input  $\mathbf{x}_i \sim p_i(\mathbf{x}_i)$  at noise level  $i$ . Replacing the ground-truth clean sample  $\mathbf{x}$  in the reverse mean expression with  $\mathbf{x}_\phi(\mathbf{x}_i, i)$  leads to the following model:

$$\boldsymbol{\mu}_\phi(\mathbf{x}_i, i) = \frac{\bar{\alpha}_{i-1}\beta_i^2}{1 - \bar{\alpha}_i^2} \mathbf{x}_\phi(\mathbf{x}_i, i) + \frac{(1 - \bar{\alpha}_{i-1}^2)\alpha_i}{1 - \bar{\alpha}_i^2} \mathbf{x}_i.$$

Analogous to the  $\epsilon$ -prediction formulation, the training objective can be expressed as

$$\|\boldsymbol{\mu}_\phi(\mathbf{x}_i, i) - \boldsymbol{\mu}(\mathbf{x}_i, \mathbf{x}_0, i)\|_2^2 \propto \|\mathbf{x}_\phi(\mathbf{x}_i, i) - \mathbf{x}_0\|_2^2, \quad \mathbf{x}_0 \sim p_{\text{data}},$$

where the model is trained to predict the original data sample  $\mathbf{x}$  from its noisy version  $\mathbf{x}_i$ . This equivalence reduces the mean-matching loss in Equation (2.2.8) to

$$\mathbb{E}_i \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}} \mathbb{E}_{\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[ \omega_i \|\mathbf{x}_\phi(\mathbf{x}_i, i) - \mathbf{x}_0\|_2^2 \right],$$

for some weighting function  $\omega_i$ . Since this is a least-squares problem, the optimal solution is given by (see Proposition 4.2.1 or Proposition 6.3.1)

$$\mathbf{x}^*(\mathbf{x}_i, i) = \mathbb{E}[\mathbf{x}_0|\mathbf{x}_i], \quad \mathbf{x}_i \sim p_i, \tag{2.2.11}$$

that is, the model should predict the expected clean data given a noisy observation  $\mathbf{x}_i$  at timestep  $i$ .

The  $\mathbf{x}$ -prediction and  $\epsilon$ -prediction parameterizations are mathematically equivalent and connected via the forward process:

$$\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_\phi(\mathbf{x}_i, i) + \sqrt{1 - \bar{\alpha}_i^2} \epsilon_\phi(\mathbf{x}_i, i). \tag{2.2.12}$$

That is, one may either predict the clean sample  $\mathbf{x}_\phi(\mathbf{x}_i, i)$  or the noise  $\epsilon_\phi(\mathbf{x}_i, i)$ , such that their combination reproduces  $\mathbf{x}_i$  under the forward noising process.

### 2.2.5 DDPM's ELBO

With the reverse transitions defined as in Equation (2.2.5), this leads to the definition of the joint generative distribution in DDPM as:

$$p_\phi(\mathbf{x}_0, \mathbf{x}_{1:L}) := p_\phi(\mathbf{x}_0|\mathbf{x}_1)p_\phi(\mathbf{x}_1|\mathbf{x}_2) \cdots p_\phi(\mathbf{x}_{L-1}|\mathbf{x}_L)p_{\text{prior}}(\mathbf{x}_L),$$

and the marginal generative model for data is given by:

$$p_\phi(\mathbf{x}_0) := \int p_\phi(\mathbf{x}_0, \mathbf{x}_{1:L}) d\mathbf{x}_{1:L}.$$

Indeed, DDPM training via Equation (2.2.6) can be rigorously grounded in maximum likelihood estimation (Equation (1.1.2)). Specifically, its objective forms an ELBO, similar to those in VAEs and HVAEs from Section 2.1, which serves as a lower bound on the log-density:

#### Theorem 2.2.3: DDPM's ELBO

$$\begin{aligned} -\log p_\phi(\mathbf{x}_0) &\leq -\mathcal{L}_{\text{ELBO}}(\mathbf{x}_0; \phi) \\ &:= \mathcal{L}_{\text{prior}}(\mathbf{x}_0) + \mathcal{L}_{\text{recon.}}(\mathbf{x}_0; \phi) + \mathcal{L}_{\text{diffusion}}(\mathbf{x}_0; \phi) \end{aligned} \quad (2.2.13)$$

Here, each component of losses are defined as:

$$\begin{aligned} \mathcal{L}_{\text{prior}}(\mathbf{x}_0) &:= \mathcal{D}_{\text{KL}}(p(\mathbf{x}_L|\mathbf{x}_0) \| p_{\text{prior}}(\mathbf{x}_L)) \\ \mathcal{L}_{\text{recon.}}(\mathbf{x}_0; \phi) &:= \mathbb{E}_{p(\mathbf{x}_1|\mathbf{x}_0)} [-\log p_\phi(\mathbf{x}_0|\mathbf{x}_1)] \\ \mathcal{L}_{\text{diffusion}}(\mathbf{x}_0; \phi) &= \sum_{i=1}^L \mathbb{E}_{p(\mathbf{x}_i|\mathbf{x}_0)} \left[ \mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \| p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)) \right]. \end{aligned}$$

#### Proof for Theorem.

The derivation applies Jensen's inequality, as in the HVAE/VAE ELBO (Equation (2.1.3)), with further simplifications. The detailed proof is deferred to Section D.1.2.

The ELBO  $\mathcal{L}_{\text{ELBO}}$  consists of three terms:

- $\mathcal{L}_{\text{prior}}$  can be made negligible by choosing the noise schedule  $\{\beta_i\}$  such that  $p(\cdot|\mathbf{x}_0) \approx p_{\text{prior}}(\cdot)$ .
- For  $\mathcal{L}_{\text{recon.}}$ , this can be approximated and optimized using a Monte Carlo estimate; see (Ho *et al.*, 2020; Kingma *et al.*, 2021) for practical implementations.

- $\mathcal{L}_{\text{diffusion}}$  (cf. Equation (2.2.6)) matches the reverse conditionals  $p_{\phi}(\mathbf{x}_{i-1}|\mathbf{x}_i)$  to  $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$  at all steps  $i$ .

The ELBO objective  $\mathcal{L}_{\text{ELBO}}$  can also be interpreted through the lens of the Data Processing Inequality with latents  $\mathbf{z} = \mathbf{x}_{1:L}$ , as illustrated in Equation (2.1.2):

$$\mathcal{D}_{\text{KL}}(p_{\text{data}}(\mathbf{x}_0)\|p_{\phi}(\mathbf{x}_0)) \leq \mathcal{D}_{\text{KL}}(p(\mathbf{x}_0, \mathbf{x}_{1:L})\|p_{\phi}(\mathbf{x}_0, \mathbf{x}_{1:L})) ,$$

where  $p(\mathbf{x}_0, \mathbf{x}_{1:L}) := p_{\text{data}}(\mathbf{x}_0)p(\mathbf{x}_1|\mathbf{x}_0)p(\mathbf{x}_2|\mathbf{x}_1) \cdots p(\mathbf{x}_L|\mathbf{x}_{L-1})$  denotes the joint distribution along the forward process.

### Remark.

Diffusion’s variational view fits the HVAE template: the “encoder” is the fixed forward noising chain, and the latents  $\mathbf{x}_{1:T}$  share the data dimensionality. Training maximizes the same ELBO. There is no learned encoder and no per-level KL terms; instead, the objective decomposes into well-conditioned denoising subproblems from large to small noise (coarse to fine), yielding stable optimization, and high sample quality while preserving a coarse-to-fine hierarchy over time/noise.

## 2.2.6 Sampling

After training the  $\epsilon$ -prediction model,  $\epsilon_{\phi^{\times}}(\mathbf{x}_i, i)$ <sup>3</sup>, sampling is performed sequentially as illustrated in Figure 2.5, using the parametrized transition  $p_{\phi^{\times}}(\mathbf{x}_{i-1}|\mathbf{x}_i)$  instead.

More specifically, starting from a random seed  $\mathbf{x}_L \sim p_{\text{prior}} = \mathcal{N}(\mathbf{0}, \mathbf{I})$ , we recursively sample from  $p_{\phi^{\times}}(\mathbf{x}_{i-1}|\mathbf{x}_i)$  following the update rule below for  $i = L, L-1, \dots, 1$ :

$$\mathbf{x}_{i-1} \leftarrow \underbrace{\frac{1}{\alpha_i} \left( \mathbf{x}_i - \frac{1 - \alpha_i^2}{\sqrt{1 - \bar{\alpha}_i^2}} \epsilon_{\phi^{\times}}(\mathbf{x}_i, i) \right)}_{\mu_{\phi^{\times}}(\mathbf{x}_i, i)} + \sigma(i) \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (2.2.14)$$

This “denoising” process continues until  $\mathbf{x}_0$  is obtained as the final clean generated sample.

**Another Interpretation of DDPM’s Sampling.** From Equation (2.2.12), the clean sample prediction corresponding to a noise estimate  $\epsilon_{\phi^{\times}}(\mathbf{x}_i, i)$  can be

---

<sup>3</sup>We use the symbol “ $\times$ ” to indicate that the model has been trained and is now frozen.

expressed as

$$\mathbf{x}_{\phi^\times}(\mathbf{x}_i, i) = \frac{\mathbf{x}_i - \sqrt{1 - \bar{\alpha}_i^2} \boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}_i, i)}{\bar{\alpha}_i}.$$

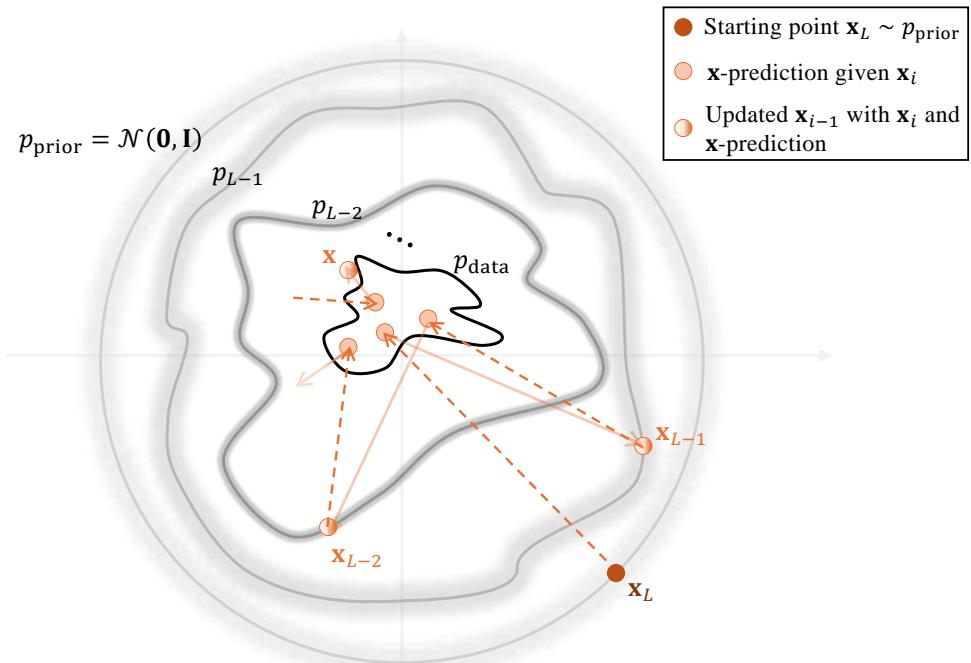
Plugging this into the DDPM sampling rule in Equation (2.2.14) yields the equivalent update:

$$\mathbf{x}_{i-1} \leftarrow (\text{interpolation between } \mathbf{x}_i \text{ and clean prediction } \mathbf{x}_{\phi^\times}) + \sigma(i) \boldsymbol{\epsilon}_i$$

indicating that each step is centered around the predicted clean sample, with added Gaussian noise scaled by  $\sigma(i)$ .

This reveals that DDPM sampling can be viewed as an iterative denoising process that alternates between:

1. Estimating the clean data  $\mathbf{x}_{\phi^\times}(\mathbf{x}_i, i)$  from the current noisy input  $\mathbf{x}_i$ ,
2. Sampling a less noisy latent  $\mathbf{x}_{i-1}$  via the update rule using this clean estimate.



**Figure 2.6:** Illustration of DDPM sampling with clean prediction: estimate  $\mathbf{x}_{\phi^\times}(\mathbf{x}_i, i)$  from  $\mathbf{x}_i$ , then update to  $\mathbf{x}_{i-1}$ .

However, even if  $\mathbf{x}_{\phi^\times}$  is trained as the optimal denoiser (i.e., the conditional expectation minimizer; see Equation (2.2.11)), it can only predict the *average* clean sample given  $\mathbf{x}_i$ . This limitation leads to blurry predictions, particularly at high noise levels, where recovering detailed structure from severely corrupted inputs becomes difficult.

From this viewpoint, diffusion sampling typically moves from high to low noise and progressively refines an estimate of the clean signal. Early steps set the global structure, later steps add fine detail, and the sample becomes more realistic as the noise is removed.

**Slow Sampling Speed of DDPM.** DDPM (a.k.a., diffusion model) sampling is inherently slow<sup>4</sup> due to the sequential nature of its reverse process, constrained by the following factors.

Theorem 2.2.1 shows that an expressive  $p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)$  can theoretically match the true reverse distribution  $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$ . However, in practice,  $p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)$  is typically modeled as a Gaussian to approximate  $p(\mathbf{x}_{i-1}|\mathbf{x}_i)$ , limiting its expressiveness.

For small forward noise scales  $\beta_i$ , the true reverse distribution is approximately Gaussian, enabling accurate approximation. Conversely, large  $\beta_i$  induce multimodality or strong non-Gaussianity that a single Gaussian cannot capture. To maintain accuracy, DDPM employs many small  $\beta_i$  steps, forming a sequential chain where each step depends on the previous and requires a neural network evaluation  $\epsilon_{\phi^\times}(\mathbf{x}_i, i)$ . This results in  $\mathcal{O}(L)$  sequential passes, preventing parallelization and slowing generation.

Later in Chapter 4 we show a more principled interpretation of this inherent sampling bottleneck as a differential-equation problem, which motivates continuous-time numerical strategies for accelerating generation.

---

<sup>4</sup>DDPM typically needs 1,000 denoising steps.

## 2.3 Closing Remarks

In this chapter, we have traced the origins of diffusion models through the variational lens. We began with the Variational Autoencoder (VAE), a foundational generative model that learns a probabilistic mapping between data and a structured latent space via the Evidence Lower Bound (ELBO). We saw how Hierarchical VAEs (HVAEs) extended this idea by stacking latent layers, introducing the powerful concept of progressive, coarse-to-fine generation. However, these models face challenges with training stability and sample quality.

We then framed Denoising Diffusion Probabilistic Models (DDPMs) as a pivotal evolution within this variational framework. By fixing the encoder to a gradual noising process and learning only the reverse denoising steps, DDPMs elegantly sidestep the training instabilities of HVAEs. Crucially, we demonstrated that DDPMs are also trained by maximizing a variational bound on the log-likelihood, with a training objective that decomposes into a series of simple denoising tasks. This tractability is enabled by a powerful conditioning strategy that transforms an intractable marginal objective into a tractable conditional one, a recurring theme in diffusion models.

While this variational framework provides a complete and powerful foundation for DDPMs, it is not the only way to understand them. An alternative and equally fundamental perspective emerges from the principles of energy-based modeling. In the next chapter, we will explore this score-based perspective:

1. We will shift our focus from learning the denoising transition probabilities  $p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)$  to directly learning the gradient of the data's log-density, i.e., the score function.
2. We will see how this approach, originating from EBMs, gives rise to Noise Conditional Score Networks (NCSN) and reveals a deep, mathematical equivalence between the noise prediction ( $\epsilon$ -prediction) learned in DDPMs and the score function itself.

This alternative viewpoint will not only offer new insights but also serve as another cornerstone for the unified, continuous-time framework of diffusion models to be developed later.

# 3

---

## Score-Based Perspective: From EBMs to NCSN

---

In the previous chapters we traced diffusion models to their variational roots and showed how they arise within the framework of VAEs. We now turn to a second, equally fundamental viewpoint: *Energy Based Models* (EBMs) (Ackley *et al.*, 1985; LeCun *et al.*, 2006). An EBM represents a distribution by an energy landscape that is low on data and high elsewhere. Sampling typically relies on Langevin dynamics, which moves samples toward high density regions by following the gradient of this landscape. This gradient field, known as the *score*, points toward directions of higher probability.

The central observation is that knowing the score is enough for generation: it moves samples toward likely regions without computing the intractable normalization constant. *Score-based* diffusion models build directly on this idea. Instead of focusing only on the clean data distribution, they consider a sequence of Gaussian noise-perturbed distributions whose scores are easier to approximate. Learning these scores yields a family of vector fields that guide noisy samples step by step back to data, turning generation into progressive denoising.

### 3.1 Energy-Based Models

For readers already familiar with EBMs, this section is meant as a concise refresher and a bridge to the score-based view of diffusion.

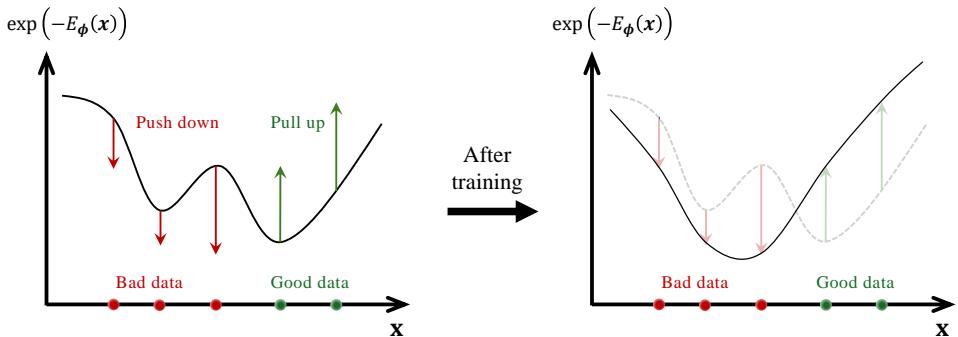
#### 3.1.1 Modeling Probability Distributions Using Energy Functions

Let  $\mathbf{x} \in \mathbb{R}^D$  denote a data point. EBMs define a probability density via an energy function  $E_\phi(\mathbf{x})$ , parameterized by  $\phi$ , which assigns lower energy to more likely configurations. The resulting distribution is given by

$$p_\phi(\mathbf{x}) := \frac{\exp(-E_\phi(\mathbf{x}))}{Z_\phi}, \quad Z_\phi := \int_{\mathbb{R}^D} \exp(-E_\phi(\mathbf{x})) d\mathbf{x},$$

where  $Z_\phi$  is called the *partition function* ensuring normalization:

$$\int_{\mathbb{R}^D} p_\phi(\mathbf{x}) d\mathbf{x} = 1.$$



**Figure 3.1: Illustration of EBM training.** The model lowers density (raises energy) at “bad” data points (red arrows), and raises density (lowers energy) at “good” data points (green arrows).

In this view, points with lower energy correspond to higher probability, much like a ball rolling down into a valley. The partition function  $Z_\phi$  ensures that all probabilities add up to one, and as a result only the *relative* values of energy matter. For instance, adding a constant to all energies multiplies both numerator and denominator by the same factor, leaving the distribution unchanged.

Moreover, because the partition function  $Z_\phi$  enforces that probabilities sum to one, it follows mathematically that decreasing the energy within a region increases its probability, while the probability of its complement

decreases accordingly. Thus, EBMs obey a strict global trade-off: making one valley deeper inevitably makes others shallower, and probability mass is redistributed across the entire space rather than assigned independently to each region.

**Challenges of Maximum Likelihood Training in EBMs.** In principle, EBMs can be trained by maximum likelihood, which naturally balances fitting the data with global regularization (see Equation (1.1.2)):

$$\begin{aligned}\mathcal{L}_{\text{MLE}}(\phi) &= \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[ \log \frac{\exp(-E_\phi(\mathbf{x}))}{Z_\phi} \right] \\ &= - \underbrace{\mathbb{E}_{p_{\text{data}}} [E_\phi(\mathbf{x})]}_{\text{lowers energy of data}} - \underbrace{\log \int \exp(-E_\phi(\mathbf{x})) d\mathbf{x}}_{\text{global regularization}},\end{aligned}\tag{3.1.1}$$

with  $Z_\phi = \int \exp(-E_\phi(\mathbf{x})) d\mathbf{x}$ . The first term lowers the energy of real data, while the second enforces normalization via the partition function.

However, in high dimensions computing  $\log Z_\phi$  and its gradient is intractable, as it requires expectations under the model distribution. This motivates alternative objectives that either approximate the term, such as contrastive divergence (Hinton, 2002), or avoid it altogether through *score matching*.

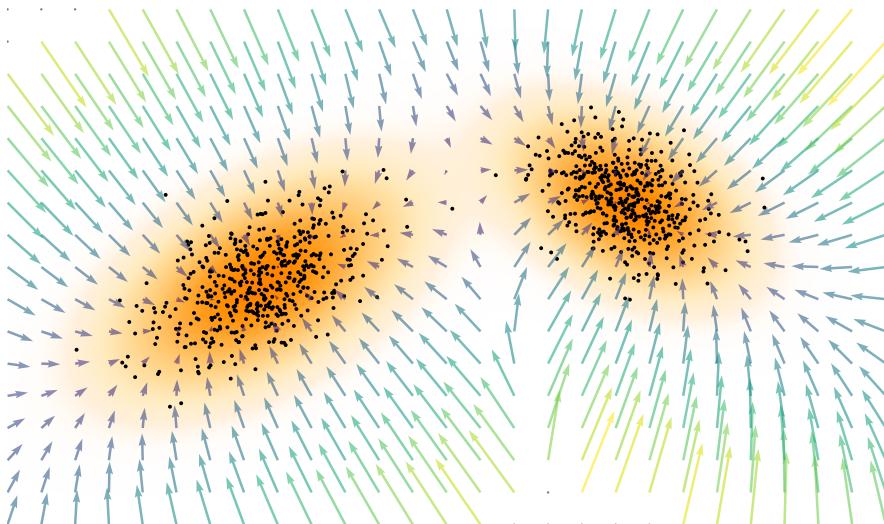
In what follows, we first introduce the notion of the score function in Section 3.1.2 and present score matching as a tractable training objective that bypasses the partition function in Section 3.1.3, and then discuss Langevin dynamics as a practical sampling method with score functions in Section 3.1.4.

### 3.1.2 Motivation: What Is the Score?

For a density  $p(\mathbf{x})$  on  $\mathbb{R}^D$ , the *score function* is the gradient of the log-density:

$$\mathbf{s}(\mathbf{x}) := \nabla_{\mathbf{x}} \log p(\mathbf{x}), \quad \mathbf{s}: \mathbb{R}^D \rightarrow \mathbb{R}^D.$$

Intuitively, the score forms a vector field that points toward regions of higher probability, providing a local guide to where the data is most likely to occur (see Figure 3.2).



**Figure 3.2: Illustration of score vector fields.** Score vector fields  $\nabla_{\mathbf{x}} \log p(\mathbf{x})$  indicate directions of increasing density.

**Why Model Scores Instead of Densities?** Modeling the score offers both theoretical and practical benefits:

**1. Freedom from Normalization Constants.** Many distributions are defined only up to an unnormalized density  $\tilde{p}(\mathbf{x})$ , e.g.,  $\exp(-E_{\phi}(\mathbf{x}))$  in EBMs:

$$p(\mathbf{x}) = \frac{\tilde{p}(\mathbf{x})}{Z}, \quad Z = \int \tilde{p}(\mathbf{x}) d\mathbf{x}.$$

While computing  $Z$  is intractable, the score depends only on  $\tilde{p}$ :

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) = \nabla_{\mathbf{x}} \log \tilde{p}(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z}_{=0} = \nabla_{\mathbf{x}} \log \tilde{p}(\mathbf{x}), \quad (3.1.2)$$

since  $Z$  is constant in  $\mathbf{x}$ . This bypasses the partition function entirely.

**2. A Complete Representation.** The score function fully characterizes the underlying distribution. Since it is the gradient of the log-density, the density can be recovered (up to a constant) via

$$\log p(\mathbf{x}) = \log p(\mathbf{x}_0) + \int_0^1 \mathbf{s}(\mathbf{x}_0 + t(\mathbf{x} - \mathbf{x}_0))^\top (\mathbf{x} - \mathbf{x}_0) dt,$$

where  $\mathbf{x}_0$  is a reference point and  $\log p(\mathbf{x}_0)$  is fixed by normalization. Thus, modeling the score is as expressive as modeling  $p(\mathbf{x})$  itself, while often more tractable for generative modeling.

### 3.1.3 Training EBMs via Score Matching

In EBMs, the density is defined as  $p_\phi(\mathbf{x}) = \frac{\exp(-E_\phi(\mathbf{x}))}{Z_\phi}$ . Maximum likelihood training requires computing  $Z_\phi$ , which is generally intractable. A key observation is that the model score  $p_\phi$  simplifies to:  $-\nabla_{\mathbf{x}} E_\phi(\mathbf{x})$ , independent of  $Z_\phi$  (see Equation (3.1.2)).

*Score matching* (Hyvärinen and Dayan, 2005) leverages the fact that scores depend only on the energy function. Instead of fitting normalized probabilities, it trains EBMs by aligning the model score with the (unknown) data score:

$$\mathcal{L}_{\text{SM}}(\phi) = \frac{1}{2} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \|\nabla_{\mathbf{x}} \log p_\phi(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})\|_2^2. \quad (3.1.3)$$

Although the data score is inaccessible, integration by parts yields an equivalent expression involving only the energy and its derivatives (see Proposition 3.2.1 for more details):

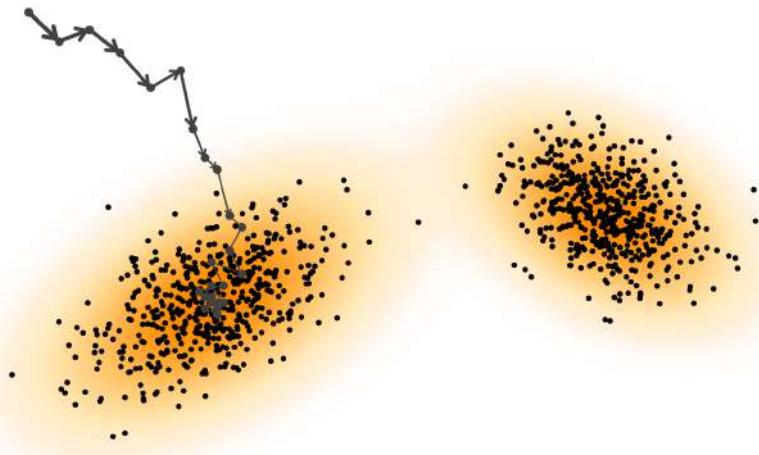
$$\mathcal{L}_{\text{SM}}(\phi) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[ \text{Tr}(\nabla_{\mathbf{x}}^2 E_\phi(\mathbf{x})) + \frac{1}{2} \|\nabla_{\mathbf{x}} E_\phi(\mathbf{x})\|_2^2 \right] + C,$$

where  $\nabla_{\mathbf{x}}^2 E_\phi(\mathbf{x})$  is the Hessian of  $E_\phi$  and  $C$  is a constant independent of  $\phi$ .

This formulation is attractive because it eliminates the partition function and avoids sampling from the model during training. Its main drawback is the need for second-order derivatives, which can be computationally prohibitive in high dimensions. We will revisit approaches to addressing this limitation later in the chapter.

### 3.1.4 Langevin Sampling with Score Functions

Sampling from EBMs, defined by the energy function  $E_\phi(\mathbf{x})$ , can be performed using *Langevin dynamics*. We first present the discrete-time Langevin update and then its continuous-time limit as a stochastic differential equation (SDE). Finally, we discuss the physical intuition behind how Langevin dynamics enables efficient exploration of complex energy landscapes.



**Figure 3.3: Illustration of Langevin sampling.** Langevin sampling using the score function  $\nabla_{\mathbf{x}} \log p_{\phi}(\mathbf{x})$  to guide trajectories toward high-density regions via the update in Equation (3.1.5) (indicating by arrows).

**Discrete-Time Langevin Dynamics.** The discrete-time Langevin update is

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta \nabla_{\mathbf{x}} E_{\phi}(\mathbf{x}_n) + \sqrt{2\eta} \epsilon_n, \quad n = 0, 1, 2, \dots, \quad (3.1.4)$$

where  $\mathbf{x}_0$  is initialized from some distribution (often Gaussian),  $\eta > 0$  is the step size, and  $\epsilon_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  is Gaussian noise. The noise enables exploration beyond local minima by adding stochasticity.

Since the score function can be computed as

$$\nabla_{\mathbf{x}} \log p_{\phi}(\mathbf{x}) = -\nabla_{\mathbf{x}} E_{\phi}(\mathbf{x}).$$

the update can equivalently be written as

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \eta \nabla_{\mathbf{x}} \log p_{\phi}(\mathbf{x}_n) + \sqrt{2\eta} \epsilon_n, \quad (3.1.5)$$

where the score function guides the samples toward high-density regions. This formulation is central to diffusion models, as will be detailed later.

**Continuous-Time Langevin Dynamics.** As the step size  $\eta$  approaches zero, the discrete Langevin updates naturally converge to a continuous-time process described by the *Langevin Stochastic Differential Equation (SDE)*<sup>1</sup> :

$$d\mathbf{x}(t) = \nabla_{\mathbf{x}} \log p_{\phi}(\mathbf{x}(t)) dt + \sqrt{2} d\mathbf{w}(t), \quad (3.1.6)$$

---

<sup>1</sup>With the factor  $\sqrt{2}$ , the Langevin dynamics leave  $p_{\phi}$  unchanged in time. Namely,  $p_{\phi}$  is stationary: if  $\mathbf{x}(0) \sim p_{\phi}$  then  $\mathbf{x}(t) \sim p_{\phi}$  for all  $t \geq 0$ . Equivalently,  $p_{\phi}$  is the stationary

where  $\mathbf{w}(t)$  denotes a standard Brownian motion (also known as a Wiener process<sup>2</sup>). It is important to understand that the discrete update rule in Equation (3.1.4) serves as the Euler–Maruyama discretization of this continuous SDE.

Under standard regularity assumptions (e.g.,  $p_\phi \propto e^{-E_\phi}$  with a confining, sufficiently smooth  $E_\phi$ ), the distribution of  $\mathbf{x}(t)$  converges (exponentially fast) to  $p_\phi$  as  $t \rightarrow \infty$ ; thus we can sample by simulating (solving) the SDE Equation (3.1.6).

**Why Langevin Sampling?** A natural way to understand Langevin sampling is through the lens of physics, where the energy function  $E_\phi(\mathbf{x})$  defines a potential landscape that shapes the behavior of particles. According to Newtonian dynamics, the motion of a particle under the force field derived from this energy is described by the ordinary differential equation (ODE)

$$d\mathbf{x}(t) = -\nabla_{\mathbf{x}} E_\phi(\mathbf{x}(t)) dt,$$

which deterministically drives the particle downhill toward a local minimum of the energy function. However, such deterministic dynamics can become trapped in local minima, preventing exploration of the full data distribution.

To overcome this limitation, Langevin dynamics introduces stochastic perturbations, resulting in the SDE

$$d\mathbf{x}(t) = -\nabla_{\mathbf{x}} E_\phi(\mathbf{x}(t)) dt + \underbrace{\sqrt{2} d\mathbf{w}(t)}_{\text{injected noise}},$$

where  $\mathbf{w}(t)$  is a standard Brownian motion. The noise term allows the particle to escape local minima by crossing energy barriers, making the trajectory a stochastic process whose stationary distribution converges to the Boltzmann distribution

$$p_\phi(\mathbf{x}) \propto e^{-E_\phi(\mathbf{x})}.$$

From this perspective, EBMs can be viewed as learning a force field that pushes samples toward regions of high probability. Langevin sampling is

---

solution of the Fokker–Planck equation (see Chapter B):

$$\partial_t \rho = -\nabla \cdot (\rho \nabla \log p_\phi) + \frac{\sigma^2}{2} \Delta \rho.$$

Setting  $\rho = p_\phi$  gives  $(\frac{\sigma^2}{2} - 1)\Delta p_\phi = 0$ , which holds only if  $\sigma = \sqrt{2}$ .

<sup>2</sup>Brownian increments satisfy  $\mathbf{w}(t + \eta) - \mathbf{w}(t) \sim \mathcal{N}(\mathbf{0}, \eta \mathbf{I})$ . Euler–Maruyama therefore uses a step noise  $\sqrt{2}[\mathbf{w}(t + \eta) - \mathbf{w}(t)] = \sqrt{2\eta}\epsilon_n$  with  $\epsilon_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , which explains the  $\sqrt{\eta}$  factor.; this is the source of the square-root scaling. For a detailed introduction to Brownian motion and SDEs, please refer to Chapter A.

particularly useful for EBMs because it provides a practical method to generate samples from the model distribution  $p_\phi(\mathbf{x})$  without explicitly computing the partition function. By iteratively applying the Langevin update, one obtains samples that approximate the target distribution.

**Inherent Challenges of Langevin Sampling.** Langevin dynamics, a widely used MCMC-based sampler, faces serious limitations in high-dimensional spaces. Its efficiency is highly sensitive to the choice of step size  $\eta$ , noise scale, and the number of iterations required to approximate the target distribution accurately.

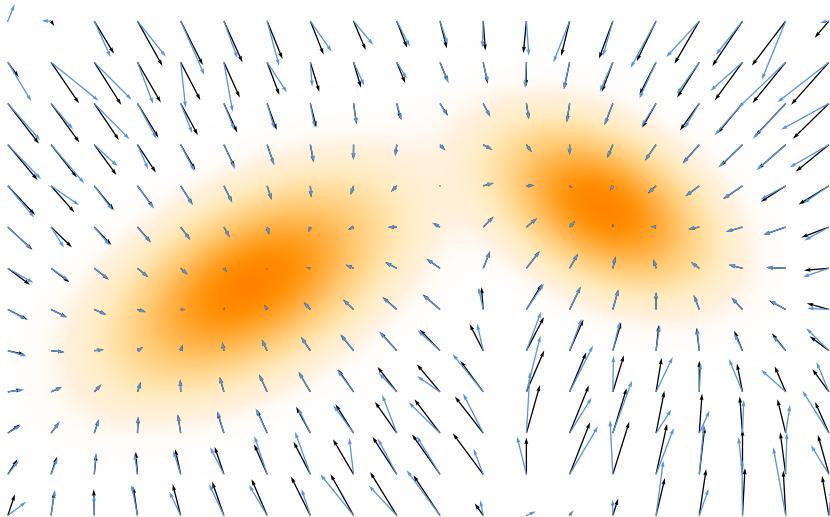
At the heart of this inefficiency lies the issue of poor “mixing time”: In complex data distributions with many isolated modes, Langevin sampling often requires an extremely long time to transition between regions of high probability. This problem becomes significantly worse as dimensionality increases, leading to prohibitively slow convergence.

One can think of sampling as exploring a vast and rugged landscape with many distant valleys, each corresponding to a different data mode. Langevin dynamics, relying on local stochastic updates, struggles to traverse between these valleys efficiently. As a result, it often fails to capture the full diversity of the distribution.

This inefficiency hints the need for more structured and guided sampling methods that can navigate complex data manifolds more effectively than purely random exploration.

### 3.2 From Energy-Based to Score-Based Generative Models

EBMs show that generation depends only on the score, which points toward regions of higher probability, rather than on the full normalized density. While score matching avoids the partition function, training through the energy still requires expensive second derivatives. The key idea is that since sampling with Langevin dynamics needs only the score, we can learn it directly with a neural network. This shift, from modeling energies to modeling scores, forms the foundation of score-based generative models.



**Figure 3.4: Illustration of Score Matching.** The neural network score  $s_\phi(\mathbf{x})$  is trained to match the ground truth score  $s(\mathbf{x})$  using a MSE loss. Both are represented as vector fields.

#### 3.2.1 Training with Score Matching

**Score Matching.** To approximate the score function  $\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$  from samples of  $p_{\text{data}}$ , we approximate it directly as a vector field parameterized by a neural network  $\mathbf{s}_\phi(\mathbf{x})$  (see Figure 3.4):

$$\mathbf{s}_\phi(\mathbf{x}) \approx \mathbf{s}(\mathbf{x}).$$

*Score matching* fits this vector field by minimizing the mean squared error (MSE) between the true and estimated scores:

$$\mathcal{L}_{\text{SM}}(\phi) := \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[ \|\mathbf{s}_\phi(\mathbf{x}) - \mathbf{s}(\mathbf{x})\|_2^2 \right]. \quad (3.2.1)$$

**Tractable Score Matching.** At first glance, this objective seems infeasible because the true score  $\mathbf{s}(\mathbf{x})$ , which serves as the regression target, is unknown. Fortunately, Hyvärinen and Dayan (2005) showed that integration by parts yields an equivalent objective that depends only on the model  $\mathbf{s}_\phi$  and the data samples, without requiring access to the true score. We state this key result in the following proposition:

### Proposition 3.2.1: Hyvärinen's Tractable Form of SM

We can express the following equation as:

$$\mathcal{L}_{\text{SM}}(\phi) = \tilde{\mathcal{L}}_{\text{SM}}(\phi) + C.$$

where

$$\tilde{\mathcal{L}}_{\text{SM}}(\phi) := \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \left[ \text{Tr}(\nabla_{\mathbf{x}} \mathbf{s}_\phi(\mathbf{x})) + \frac{1}{2} \|\mathbf{s}_\phi(\mathbf{x})\|_2^2 \right], \quad (3.2.2)$$

and  $C$  is a constant that does not depend on  $\phi$ . The minimizer  $\mathbf{s}^*$  is obtained as:  $\mathbf{s}^*(\cdot) = \nabla_{\mathbf{x}} \log p(\cdot)$ .

### Proof for Proposition.

The result follows by expanding the MSE in  $\mathcal{L}_{\text{SM}}$  and applying integration by parts. The proof is given in Section D.2.1. ■

Using the equivalent objective in Equation (3.2.2), we train the score model  $\mathbf{s}_\phi(\mathbf{x})$  solely from observed samples of  $p_{\text{data}}$ , eliminating the need for the true score function.

**Intuition of Equation (3.2.2).** The alternative score matching objective  $\tilde{\mathcal{L}}_{\text{SM}}(\phi)$  can be understood directly from its two terms. The norm term  $\frac{1}{2} \|\mathbf{s}_\phi(\mathbf{x})\|^2$  suppresses the score in regions where  $p_{\text{data}}$  is large, making them stationary. The divergence term  $\text{Tr}(\nabla_{\mathbf{x}} \mathbf{s}_\phi(\mathbf{x}))$  favors negative values, so these stationary points act as attractive sinks. Together, the loss shapes high-density regions into stable and contracting points of the score field. We explain this in detail below.

**Stationarity from the Magnitude Term.** Since the expectation in  $\tilde{\mathcal{L}}_{\text{SM}}(\phi)$  is taken under  $p_{\text{data}}$ , so regions where  $p_{\text{data}}(\mathbf{x})$  is large (high data density) contribute most to the loss. The magnitude term  $\frac{1}{2}\|\mathbf{s}_\phi(\mathbf{x})\|^2$  therefore drives  $\mathbf{s}_\phi(\mathbf{x}) \rightarrow 0$  precisely in those high-probability areas, i.e., those locations become *stationary*.

**Concavity When the Field is (Approximately) a Gradient.** The divergence term  $\text{Tr}(\nabla_{\mathbf{x}}\mathbf{s}_\phi(\mathbf{x}))$  encourages the vector field to have negative divergence in regions of high data density. Negative divergence means that nearby vectors converge rather than spread out, so a stationary point in such a region acts as a *sink*: nearby trajectories are pulled inward. To make this precise, assume  $\mathbf{s}_\phi = \nabla_{\mathbf{x}}u$  for a scalar function  $u : \mathbb{R}^D \rightarrow \mathbb{R}$ , as is natural when matching a log density. Then  $\nabla_{\mathbf{x}}\mathbf{s}_\phi = \nabla_{\mathbf{x}}^2u$  (the Hessian) and  $\nabla \cdot \mathbf{s}_\phi(\mathbf{x}) = \text{Tr}(\nabla_{\mathbf{x}}^2u(\mathbf{x}))$  (the divergence).

At a stationary point  $\mathbf{x}_*$ , where  $\mathbf{s}_\phi(\mathbf{x}_*) = \nabla_{\mathbf{x}}u(\mathbf{x}_*) = \mathbf{0}$ , a second order Taylor expansion gives

$$u(\mathbf{x}) = u(\mathbf{x}_*) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_*)^\top \nabla_{\mathbf{x}}^2u(\mathbf{x}_*)(\mathbf{x} - \mathbf{x}_*) + o(\|\mathbf{x} - \mathbf{x}_*\|^2).$$

If the Hessian  $\nabla_{\mathbf{x}}^2u(\mathbf{x}_*)$  is negative definite, then  $u$  is locally concave at  $\mathbf{x}_*$  and the log density attains a strict local maximum<sup>3</sup> there. Because all eigenvalues of the Hessian are negative, the trace is also negative:  $\text{Tr}(\nabla_{\mathbf{x}}^2u(\mathbf{x}_*)) < 0$ . Thus the learned vector field has negative divergence and the stationary point is a *sink*: small perturbations are contracted back toward  $\mathbf{x}_*$ .

### 3.2.2 Sampling with Langevin Dynamics

Once trained by minimizing Equation (3.2.2), the score model  $\mathbf{s}_{\phi^\times}(\mathbf{x})$  can replace the oracle score in Langevin dynamics for sampling:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \eta \mathbf{s}_{\phi^\times}(\mathbf{x}_n) + \sqrt{2\eta}\boldsymbol{\epsilon}_n, \quad \boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (3.2.3)$$

for  $n = 0, 1, 2, \dots$ , initialized at  $\mathbf{x}_0$ . As in the EBM case Equation (3.1.6), this recursion is precisely the Euler–Maruyama discretization of the continuous-time Langevin SDE:

$$d\mathbf{x}(t) = \mathbf{s}_{\phi^\times}(\mathbf{x}(t)) dt + \sqrt{2} d\mathbf{w}(t),$$

---

<sup>3</sup>We remark that strict concavity (and thus a strict local maximum of the log density) requires the entire Hessian  $\nabla_{\mathbf{x}}^2u$  to be negative definite, not merely to have negative trace. A negative trace guarantees that the sum of eigenvalues is negative, but some eigenvalues could still be positive, leading to a saddle point rather than a maximum.

with initialization  $\mathbf{x}(0)$ . Hence, in the limit of small step size, the discrete and continuous formulations coincide. In practice, one can either run the discrete sampler or directly simulate the SDE.

### 3.2.3 Prologue: Score-Based Generative Models

In the remainder of this chapter, we examine the foundational role of the score function in modern diffusion models. Initially introduced to enable efficient training of EBMs, the score function has evolved into a central component of a new generation of generative models. Building on this foundation, we explore how the score function informs the theoretical formulation and practical implementation of *score-based diffusion models*, offering a principled framework for data generation via stochastic processes.

### 3.3 Denoising Score Matching

#### 3.3.1 Motivation

Although the alternative objective in Equation (3.2.2)

$$\tilde{\mathcal{L}}_{\text{SM}}(\phi) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[ \text{Tr}(\nabla_{\mathbf{x}} \mathbf{s}_\phi(\mathbf{x})) + \frac{1}{2} \|\mathbf{s}_\phi(\mathbf{x})\|_2^2 \right]$$

is more tractable, it still requires computing the trace of the Jacobian,  $\text{Tr}(\nabla_{\mathbf{x}} \mathbf{s}_\phi(\mathbf{x}))$ , which has worst-case complexity  $\mathcal{O}(D^2)$ . Such complexity limits scalability to high-dimensional data.

To address this, sliced score matching (Song *et al.*, 2020b) replaces the trace term with a stochastic estimate based on random projections. We briefly outline the idea below.

**Sliced Score Matching and Hutchinson’s Estimator.** Sliced score matching replaces the trace in score matching by averaging directional derivatives along random “slices”. Let  $\mathbf{u} \in \mathbb{R}^D$  be an *isotropic* random vector (e.g., Rademacher or standard Gaussian) with  $\mathbb{E}[\mathbf{u}] = 0$  and  $\mathbb{E}[\mathbf{u}\mathbf{u}^\top] = \mathbf{I}$ . By Hutchinson’s identity

$$\text{Tr}(\mathbf{A}) = \mathbb{E}_{\mathbf{u}}[\mathbf{u}^\top \mathbf{A} \mathbf{u}], \quad \text{and} \quad \mathbb{E}_{\mathbf{u}}[(\mathbf{u}^\top \mathbf{s}_\phi(\mathbf{x}))^2] = \|\mathbf{s}_\phi(\mathbf{x})\|_2^2,$$

we obtain the exact form

$$\tilde{\mathcal{L}}_{\text{SM}}(\phi) = \mathbb{E}_{\mathbf{x}, \mathbf{u}} \left[ \mathbf{u}^\top (\nabla_{\mathbf{x}} \mathbf{s}_\phi(\mathbf{x})) \mathbf{u} + \frac{1}{2} (\mathbf{u}^\top \mathbf{s}_\phi(\mathbf{x}))^2 \right].$$

This objective can be evaluated efficiently with automatic differentiation, using Jacobian- and vector-Jacobian-product operations (JVP/VJP) instead of explicitly computing large Jacobian or Hessian matrices. Averaging over  $K$  random probes yields an unbiased estimator with variance  $\mathcal{O}(1/K)$ , and the directional term  $\mathbf{u}^\top (\nabla_{\mathbf{x}} \mathbf{s}_\phi) \mathbf{u}$  can be computed efficiently using JVP/VJP routines without explicit Jacobians. Intuitively, this means we only check the model’s behavior along random directions: the projected score is nudged to align with regions of higher data density, so data points become stationary in expectation.

**From Sliced to Denoising Score Matching.** Sliced score matching sidesteps Jacobians but still relies on the raw data distribution. This makes it fragile: for image data lying on low-dimensional manifolds, the score  $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$  may be undefined or unstable, and the method only constrains the vector field

at observed points, providing weak control in their neighborhoods. It further suffers from probe-induced variance and repeated JVP/VJP costs.

A more robust alternative, which we focus on here, is *Denoising Score Matching* (DSM) (Vincent, 2011), which offers a principled and scalable solution.

### 3.3.2 Training

Let us revisit the SM loss in Equation (3.2.1):

$$\mathcal{L}_{\text{SM}}(\phi) = \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \left[ \|\mathbf{s}_\phi(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})\|_2^2 \right],$$

where the issue arises from the intractable term  $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$ .

**Vincent (2011)'s Solution by Conditioning.** To overcome the intractability of  $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$ , Vincent (2011) proposed injecting noise into the data  $\mathbf{x} \sim p_{\text{data}}$  via a known conditional distribution  $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$  with scale  $\sigma$ . The neural network  $\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)$  is trained to approximate the score of the marginal perturbed distribution

$$p_\sigma(\tilde{\mathbf{x}}) = \int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) p_{\text{data}}(\mathbf{x}) d\mathbf{x}$$

by minimizing the loss

$$\mathcal{L}_{\text{SM}}(\phi; \sigma) := \frac{1}{2} \mathbb{E}_{\tilde{\mathbf{x}} \sim p_\sigma} \left[ \|\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma) - \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})\|_2^2 \right]. \quad (3.3.1)$$

Even though  $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$  is generally intractable, Vincent (2011) showed that *conditioning* on  $\mathbf{x} \sim p_{\text{data}}$  yields an equivalent, tractable objective—the *Denoising Score Matching* (DSM) loss:

$$\mathcal{L}_{\text{DSM}}(\phi; \sigma) := \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}, \tilde{\mathbf{x}} \sim p_\sigma(\cdot|\mathbf{x})} \left[ \|\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma) - \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\|_2^2 \right]. \quad (3.3.2)$$

The optimal minimizer  $\mathbf{s}^*$  of Equation (3.3.2) satisfies

$$\mathbf{s}^*(\tilde{\mathbf{x}}; \sigma) = \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}),$$

which is also optimal for Equation (3.3.1).

For example, when  $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$  is Gaussian noise with variance  $\sigma^2$ ,

$$p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \mathbf{I}),$$

the gradient  $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$  has a closed form (see Equation (3.3.4)), making the regression target explicit and computationally tractable.

Moreover, as  $\sigma \approx 0$ ,  $p_\sigma(\tilde{\mathbf{x}}) \approx p_{\text{data}}(\mathbf{x})$  and

$$\mathbf{s}^*(\tilde{\mathbf{x}}; \sigma) = \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}) \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}),$$

indicating the learned score approximates the original data score, enabling its use in generation.

We formalize this discussion on the gradient equivalence between  $\mathcal{L}_{\text{SM}}$  and  $\mathcal{L}_{\text{DSM}}$  in the following theorem:

### Theorem 3.3.1: Equivalence of $\mathcal{L}_{\text{SM}}$ and $\mathcal{L}_{\text{DSM}}$

For any fixed noise scale  $\sigma > 0$ , the following holds:

$$\mathcal{L}_{\text{SM}}(\phi; \sigma) = \mathcal{L}_{\text{DSM}}(\phi; \sigma) + C, \quad (3.3.3)$$

where  $C$  is a constant independent of the parameter  $\phi$ . Furthermore, the minimizer  $\mathbf{s}^*(\cdot; \sigma)$  of both losses satisfies

$$\mathbf{s}^*(\tilde{\mathbf{x}}; \sigma) = \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}), \quad \text{for almost every } \tilde{\mathbf{x}}.$$

#### **Proof for Theorem.**

The equivalence follows from a direct computation: by expanding the MSE in  $\mathcal{L}_{\text{SM}}$  and  $\mathcal{L}_{\text{DSM}}$ , all  $\phi$ -dependent terms cancel, leaving only a constant difference independent of  $\phi$ .

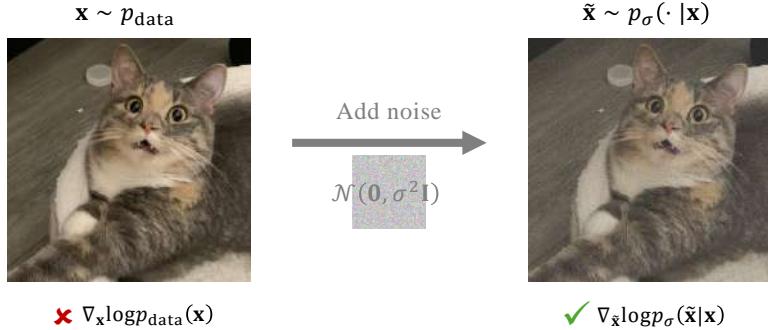
The derivation of the minimizer follows the same argument as in Proposition 4.2.1. ■

This theorem, like Theorem 2.2.1 in DDPM, illustrates a key shared principle:

### Insight 3.3.1: Conditioning Technique

The conditioning technique also appears in the variational view of diffusion models in DDPM (see Theorem 2.2.1), where conditioning on a data point  $\mathbf{x}$  turns an intractable loss into a tractable one for Monte Carlo estimation. A similar idea arises in the flow-based perspective (e.g., Flow Matching (Lipman *et al.*, 2022)), as we will see in Section 5.2.

**Special Case: Additive Gaussian Noise.** We now consider the common case where Gaussian noise  $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$  with variance  $\sigma^2$  is added to each data point



**Figure 3.5: Illustration of DSM via the conditioning technique.** By perturbing the data distribution  $p_{\text{data}}$  with small additive Gaussian noise  $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ , the resulting conditional distribution  $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \mathbf{I})$  admits a closed-form score function.

$\mathbf{x} \sim p_{\text{data}}$ :

$$\tilde{\mathbf{x}} = \mathbf{x} + \sigma \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

so that the corrupted data  $\tilde{\mathbf{x}}$  follows

$$p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \mathbf{I}).$$

In this setting, the conditional score is analytically given by

$$\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \frac{\mathbf{x} - \tilde{\mathbf{x}}}{\sigma^2}.$$

Hence, the DSM loss simplifies to:

$$\begin{aligned} \mathcal{L}_{\text{DSM}}(\phi; \sigma) &= \frac{1}{2} \mathbb{E}_{\mathbf{x}, \tilde{\mathbf{x}}|\mathbf{x}} \left[ \left\| \mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma) - \frac{\mathbf{x} - \tilde{\mathbf{x}}}{\sigma^2} \right\|_2^2 \right] \\ &= \frac{1}{2} \mathbb{E}_{\mathbf{x}, \boldsymbol{\epsilon}} \left[ \left\| \mathbf{s}_\phi(\mathbf{x} + \sigma \boldsymbol{\epsilon}; \sigma) + \frac{\boldsymbol{\epsilon}}{\sigma} \right\|_2^2 \right], \end{aligned} \quad (3.3.4)$$

where  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . This objective forms the core of the (score-based) Diffusion Model.

When the noise level  $\sigma$  is small, the Gaussian smoothed marginal  $p_\sigma = p_{\text{data}} * \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ , so their high density regions and scores nearly coincide:  $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}) \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$ . Consequently, taking a small step along the noisy score direction  $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma$  moves a noisy sample toward essentially the same high likelihood regions of the clean distribution, which is similar to the intuition behind score matching summarized in Section 3.2.1. By contrast,

when  $\sigma$  is large, the smoothing “over simplifies” the landscape:  $p_\sigma$  washes out local modes and its score mostly pulls toward global mass (think shrinkage toward the mean), yielding coarse denoising that can over smooth. In practice, however, DSM typically assumes that the injected noise is small and mild.

To better see why the objective naturally corresponds to a “denoising” process, we expand on the discussion in Sections 3.3.4 and 3.3.5.

### 3.3.3 Sampling

Once we have a trained score model  $s_{\phi^\times}(\tilde{\mathbf{x}}; \sigma)$  at noise level  $\sigma$ , we generate samples using Langevin dynamics by replacing the true score with the learned model. The update rule is:

$$\tilde{\mathbf{x}}_{n+1} = \tilde{\mathbf{x}}_n + \eta \underbrace{s_{\phi^\times}(\tilde{\mathbf{x}}_n; \sigma)}_{\approx \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}_n)} + \sqrt{2\eta} \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (3.3.5)$$

for  $n = 0, 1, 2, \dots$ , starting from an initial value  $\tilde{\mathbf{x}}_0$ . If  $\sigma$  is sufficiently small, then after enough iterations,  $\tilde{\mathbf{x}}_n$  approximates samples from  $p_{\text{data}}$ .

**Advantages of Noise Injection.** We additionally remark that, compared to vanilla score matching in Equation (3.2.1), injecting Gaussian noise to form  $p_\sigma$  (e.g., Equation (3.3.4)) provides two key advantages (Song and Ermon, 2019):

- **Well-Defined Gradients.** The noise perturbs data away from its lower-dimensional manifold, resulting in a distribution  $p_\sigma$  with full support in  $\mathbb{R}^D$ . Consequently, the score function  $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$  is well-defined everywhere.
- **Improved Coverage.** The noise smooths out sparse regions between modes, enhancing training signal quality and facilitating Langevin dynamics to traverse low-density regions more effectively.

### 3.3.4 Why DSM is Denoising: Tweedie’s Formula

We begin with *Tweedie’s formula* (Efron, 2011), which provides a principled basis for denoising from noisy observations alone. Concretely, it states that: given a single Gaussian-corrupted observation  $\tilde{\mathbf{x}} \sim \mathcal{N}(\cdot; \alpha \mathbf{x}, \sigma^2 \mathbf{I})$  from an unknown  $\mathbf{x} \sim p_{\text{data}}$ , a denoised estimate (the average over all plausible clean signals given  $\tilde{\mathbf{x}}$ ) is obtained by nudging  $\tilde{\mathbf{x}}$  a step of size  $\sigma^2$  in the direction of the score  $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$  of its noisy marginal defined as:

$$p_\sigma(\tilde{\mathbf{x}}) := \int \mathcal{N}(\tilde{\mathbf{x}}; \alpha \mathbf{x}_0, \sigma^2 \mathbf{I}) p_{\text{data}}(\mathbf{x}) d\mathbf{x}.$$

We present the proposition formally below.

### Lemma 3.3.2: Tweedie's Formula

Assume  $\mathbf{x} \sim p_{\text{data}}$  and, conditionally on  $\mathbf{x}$ ,  $\tilde{\mathbf{x}} \sim \mathcal{N}(\cdot; \alpha\mathbf{x}, \sigma^2\mathbf{I})$  with  $\alpha \neq 0$ . Then Tweedie's formula states

$$\alpha \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}|\tilde{\mathbf{x}})} [\mathbf{x}|\tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}), \quad (3.3.6)$$

where the expectation is taken over the posterior distribution  $p(\mathbf{x}|\tilde{\mathbf{x}})$  of  $\mathbf{x}$  given  $\tilde{\mathbf{x}}$ .

#### Proof for Lemma.

The proof proceeds by computing the score of the marginal  $p(\tilde{\mathbf{x}}) = \int p(\tilde{\mathbf{x}}|\mathbf{x})p_{\text{data}}(\mathbf{x}) d\mathbf{x}$ . Differentiating under the integral and using the Gaussian form of the conditional density leads directly to an expression that rearranges into the desired identity linking the score with the posterior mean. See Section D.2.3 for details.

Tweedie's formula plays a central role in diffusion models, where multiple layers of noise are introduced as in DDPM. It enables the estimation of clean samples from noisy observations via the score function, thereby establishing a fundamental link between score prediction and denoiser:

$$\underbrace{\mathbb{E}[\mathbf{x}|\tilde{\mathbf{x}}]}_{\substack{\text{denoiser} \\ \text{estimated from } \tilde{\mathbf{x}}}} = \frac{1}{\alpha} \left( \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}) \right).$$

Especially, a single gradient-ascent step on the noisy log-likelihood with the particular step size  $\sigma^2$  is the denoised estimate (the conditional average clean signal). This makes DSM training and denoising tightly related: if  $\mathbf{s}_\phi(\tilde{\mathbf{x}}) \approx \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$  trained from DSM, then

$$\frac{1}{\alpha} \left( \tilde{\mathbf{x}} + \sigma^2 \mathbf{s}_\phi(\tilde{\mathbf{x}}) \right)$$

is the denoiser.

**(Optional) Higher Order Tweedie's Formula.** The classical Tweedie's formula expresses the posterior mean  $\mathbb{E}[\mathbf{x}_0|\tilde{\mathbf{x}}]$  through the gradient  $\nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}})$ . Higher order extensions (Meng *et al.*, 2021a) express the posterior covariance and higher cumulants through higher derivatives of  $\log p(\tilde{\mathbf{x}})$ .

**Exponential Family Setup with the Log-Normalizer  $\lambda(\tilde{\mathbf{x}})$ .** Assume the conditional law of  $\tilde{\mathbf{x}}$  given a latent natural parameter  $\boldsymbol{\eta} \in \mathbb{R}^D$  belongs to a natural exponential family written as

$$q_\sigma(\tilde{\mathbf{x}}|\boldsymbol{\eta}) = \exp(\boldsymbol{\eta}^\top \tilde{\mathbf{x}} - \psi(\boldsymbol{\eta})) q_0(\tilde{\mathbf{x}}).$$

Here  $q_0(\tilde{\mathbf{x}})$  is the *base measure*, namely the part that does not depend on  $\boldsymbol{\eta}$ ; for additive Gaussian noise with variance  $\sigma^2 \mathbf{I}$  it equals  $(2\pi\sigma^2)^{-D/2} \exp(-\|\tilde{\mathbf{x}}\|^2/2\sigma^2)$ . Let  $p(\boldsymbol{\eta})$  be the pre-defined distribution of the latent natural parameter, which can be viewed as the reparameterized clean-data distribution (for Gaussian location,  $\boldsymbol{\eta} = \mathbf{x}/\sigma^2$ ). The observed noisy marginal is

$$p_\sigma(\tilde{\mathbf{x}}) = \int q_\sigma(\tilde{\mathbf{x}}|\boldsymbol{\eta}) p(\boldsymbol{\eta}) d\boldsymbol{\eta}.$$

Define the *log-partition* (log-normalizer) in  $\tilde{\mathbf{x}}$  by

$$\lambda(\tilde{\mathbf{x}}) := \log p_\sigma(\tilde{\mathbf{x}}) - \log q_0(\tilde{\mathbf{x}}).$$

Then the posterior of  $\boldsymbol{\eta}$  given  $\tilde{\mathbf{x}}$  is

$$p(\boldsymbol{\eta}|\tilde{\mathbf{x}}) \propto \exp(\boldsymbol{\eta}^\top \tilde{\mathbf{x}} - \psi(\boldsymbol{\eta}) - \lambda(\tilde{\mathbf{x}})) p(\boldsymbol{\eta}),$$

which shows that, as a function of  $\tilde{\mathbf{x}}$ , the posterior has exponential-family form with natural parameter  $\tilde{\mathbf{x}}$ , sufficient statistic  $\boldsymbol{\eta}$ , and log-partition  $\lambda(\tilde{\mathbf{x}})$ .

**Derivatives of  $\lambda$  Produce Posterior Cumulants.** Two simple rules are at play. First, normalization: for every  $\tilde{\mathbf{x}}$ ,

$$\int \exp(\boldsymbol{\eta}^\top \tilde{\mathbf{x}} - \psi(\boldsymbol{\eta}) - \lambda(\tilde{\mathbf{x}})) p(\boldsymbol{\eta}) d\boldsymbol{\eta} = 1.$$

Differentiating this identity with respect to  $\tilde{\mathbf{x}}$  brings down powers of  $\boldsymbol{\eta}$  from the exponential and derivatives of  $\lambda(\tilde{\mathbf{x}})$ ; setting the result to zero yields equalities between derivatives of  $\lambda$  and posterior moments of  $\boldsymbol{\eta}$ . Second, a standard property of exponential families: the log-partition is the cumulant generating function of the sufficient statistic. Therefore

$$\nabla_{\tilde{\mathbf{x}}} \lambda(\tilde{\mathbf{x}}) = \mathbb{E}[\boldsymbol{\eta}|\tilde{\mathbf{x}}], \quad \nabla_{\tilde{\mathbf{x}}}^2 \lambda(\tilde{\mathbf{x}}) = \text{Cov}[\boldsymbol{\eta}|\tilde{\mathbf{x}}], \quad \nabla_{\tilde{\mathbf{x}}}^{(k)} \lambda(\tilde{\mathbf{x}}) = \kappa_k(\boldsymbol{\eta}|\tilde{\mathbf{x}}) \quad (k \geq 3),$$

where  $\kappa_k$  are the *conditional cumulants* of order  $k$  of the random vector  $\boldsymbol{\eta}$  given  $\tilde{\mathbf{x}}$ , obtained via the standard moment–cumulant relations.

These are the higher order Tweedie's formulas. Specializing to the Gaussian location model with  $\boldsymbol{\eta} = \mathbf{x}/\sigma^2$  yields the familiar forms in terms of derivatives of  $\log p_\sigma(\tilde{\mathbf{x}})$ :

$$\mathbb{E}[\mathbf{x}|\tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}), \quad \text{Cov}[\mathbf{x}|\tilde{\mathbf{x}}] = \sigma^2 \mathbf{I} + \sigma^4 \nabla_{\tilde{\mathbf{x}}}^2 \log p_\sigma(\tilde{\mathbf{x}}),$$

and higher cumulants scale with higher derivatives of  $\log p_\sigma(\tilde{\mathbf{x}})$ .

Several studies have explored training neural networks to estimate higher order scores (Meng *et al.*, 2021a; Lu *et al.*, 2022a; Lai *et al.*, 2023a). In contrast, our aim is to clarify their relationship with statistical quantities, and we refer the reader to these works for methodological details.

### 3.3.5 (Optional) Why DSM is Denoising: SURE

**SURE (Stein's Unbiased Risk Estimator).** At a high level, Stein's Unbiased Risk Estimator (SURE) is a technique that allows one to estimate the mean squared error (MSE) of a denoiser  $\mathbf{D}$  *without knowing the clean signal*. In other words, SURE provides a way to select or train denoisers when only noisy data are available.

For clarity, consider the additive Gaussian noise setting:

$$\tilde{\mathbf{x}} = \mathbf{x} + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

where  $\mathbf{x} \in \mathbb{R}^D$  is the unknown clean signal and  $\tilde{\mathbf{x}}$  is the observed noisy version. A denoiser is any (weakly differentiable) mapping  $\mathbf{D} : \mathbb{R}^D \rightarrow \mathbb{R}^D$  that produces an estimate  $\mathbf{D}(\tilde{\mathbf{x}})$  of  $\mathbf{x}$ .

The natural quality measure is the conditional MSE

$$R(\mathbf{D}; \mathbf{x}) := \mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{x}} [\|\mathbf{D}(\tilde{\mathbf{x}}) - \mathbf{x}\|_2^2 | \mathbf{x}] .$$

This quantity depends on the unknown ground truth  $\mathbf{x}$ , and therefore cannot be computed directly. Stein's identity (see Section D.2.4), however, yields the following *observable* surrogate:

$$\text{SURE}(\mathbf{D}; \tilde{\mathbf{x}}) = \|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|_2^2 + 2\sigma^2 \nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{D}(\tilde{\mathbf{x}}) - D\sigma^2, \quad (3.3.7)$$

where  $\nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{D}(\tilde{\mathbf{x}})$  denotes the divergence of  $\mathbf{D}$ . We emphasize that  $\text{SURE}(\mathbf{D}; \tilde{\mathbf{x}})$  requires only the noisy observation  $\tilde{\mathbf{x}}$ , not the clean  $\mathbf{x}$ .

Intuitively, SURE consists of two parts that complement each other. The term  $\|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|_2^2$  measures how far the denoiser's output is from the noisy input; by itself this underestimates the true error since  $\tilde{\mathbf{x}}$  is already corrupted. The divergence term acts as a correction: it captures how sensitive the denoiser is to small perturbations in its input, effectively accounting for the variance introduced by the noise.

Importantly, for any fixed but unknown  $\mathbf{x}$ ,

$$\mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{x}} [\text{SURE}(\mathbf{D}; \mathbf{x} + \sigma\epsilon) | \mathbf{x}] = R(\mathbf{D}; \mathbf{x}),$$

where the expectation is over the Gaussian noise  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . Thus, minimizing SURE (in expectation or empirically) is equivalent to minimizing the true

MSE, while relying only on noisy data. In practice, averaging SURE over both  $\mathbf{x} \sim p_{\text{data}}$  and the corruption noise  $\epsilon$  yields an unbiased estimate of the global MSE risk.

**Link to Tweedie's Formula and Bayes Optimality.** Let  $p_\sigma(\tilde{\mathbf{x}}) = (p_{\text{data}} * \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}))(\tilde{\mathbf{x}})$  denote the noisy marginal considered in this section.

SURE is an unbiased estimator of the mean squared error with respect to the noise, conditional on  $\mathbf{x}$ :

$$\mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{x}}[\text{SURE}(\mathbf{D}; \tilde{\mathbf{x}})] = \mathbb{E}_{\tilde{\mathbf{x}}|\mathbf{x}}[\|\mathbf{D}(\tilde{\mathbf{x}}) - \mathbf{x}\|^2].$$

Hence minimizing the expected SURE equals minimizing the Bayes risk  $\mathbb{E}_{(\mathbf{x}, \tilde{\mathbf{x}})}[\|\mathbf{D}(\tilde{\mathbf{x}}) - \mathbf{x}\|^2] = \mathbb{E}_{\tilde{\mathbf{x}}}[\mathbb{E}_{\mathbf{x}|\tilde{\mathbf{x}}}[\|\mathbf{D}(\tilde{\mathbf{x}}) - \mathbf{x}\|^2]]$  by the law of total expectation (tower property). This decomposition yields a pointwise optimization: for almost every  $\tilde{\mathbf{x}}$ ,

$$\mathbf{D}^*(\tilde{\mathbf{x}}) = \arg \min_{\mathbf{z}} \mathbb{E}_{\mathbf{x}|\tilde{\mathbf{x}}}[\|\mathbf{z} - \mathbf{x}\|^2] = \mathbb{E}[\mathbf{x}|\tilde{\mathbf{x}}].$$

Therefore the SURE-optimal denoiser coincides with the Bayes estimator in Section 3.3.4, and by Tweedie's identity:

$$\mathbf{D}^*(\tilde{\mathbf{x}}) = \mathbb{E}[\mathbf{x}|\tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}). \quad (3.3.8)$$

**Relationship of SURE and Score Matching.** The identity in Equation (3.3.8) motivates parameterizing the denoiser  $\mathbf{D}$  via a score field:

$$\mathbf{D}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}} + \sigma^2 \mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma),$$

with  $\mathbf{s}_\phi(\cdot; \sigma)$  meant to approximate the noisy score  $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\cdot)$ . Plugging  $\mathbf{D}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}} + \sigma^2 \mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)$  in Equation (3.3.7) gives

$$\frac{1}{2\sigma^4} \text{SURE}(\mathbf{D}; \tilde{\mathbf{x}}) = \text{Tr}(\nabla_{\tilde{\mathbf{x}}} \mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)) + \frac{1}{2} \|\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)\|_2^2 + \text{const}(\sigma).$$

Therefore, taking expectation with respect to  $\tilde{\mathbf{x}} \sim p_\sigma$ , minimizing SURE is equivalent (up to an additive constant) to minimizing Hyvärinen's alternative score matching objective at noise level  $\sigma$ , with the expectation taken under  $p_\sigma$  (see Equation (3.2.2)). Consequently, both objectives share the same minimizer, namely the denoiser in Equation (3.3.8).

### 3.3.6 (Optional) Generalized Score Matching

**Motivation.** Classical score matching, denoising score matching, and higher order variants all target

$$\frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})}, \quad \text{for some density } p$$

with a linear operator  $\mathcal{L}$  acting on the density. In the classical case  $\mathcal{L} = \nabla_{\mathbf{x}}$ , this gives

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) = \frac{\nabla_{\mathbf{x}} p(\mathbf{x})}{p(\mathbf{x})}$$

The  $\frac{\mathcal{L}p}{p}$  structure allows integration by parts to remove normalizing constants, yielding a tractable objective that depends only on samples from  $p$  and the learned field  $\mathbf{s}_{\phi}$ . This viewpoint motivates the generalized score matching framework.

**Generalized Fisher Divergence.** Let  $p$  be the data distribution and  $q$  any model distribution. For a linear operator  $\mathcal{L}$  on scalar functions of  $\mathbf{x}$ , define the generalized Fisher divergence

$$\mathcal{D}_{\mathcal{L}}(p \| q) := \int p(\mathbf{x}) \left\| \frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})} - \frac{\mathcal{L}q(\mathbf{x})}{q(\mathbf{x})} \right\|_2^2 d\tilde{\mathbf{x}}.$$

If  $\mathcal{L}$  is *complete*, i.e.,

$$\frac{\mathcal{L}p_1}{p_1} = \frac{\mathcal{L}p_2}{p_2} \text{ a.e. implies } p_1 = p_2 \text{ a.e.},$$

then  $\mathcal{D}_{\mathcal{L}}(p \| q) = 0$  identifies  $q = p$ . For  $\mathcal{L} = \nabla_{\tilde{\mathbf{x}}}$  this recovers the classical Fisher divergence (see Equation (1.1.3)).

**Score Parameterization.** In practice we do not model a normalized density  $q$ . Instead, we directly parameterize a vector field  $\mathbf{s}_{\phi}(\mathbf{x})$  to approximate the generalized score  $\frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})}$ . Consider

$$\mathcal{D}_{\mathcal{L}}(p \| \mathbf{s}_{\phi}) := \mathbb{E}_{\mathbf{x} \sim p} \left[ \left\| \mathbf{s}_{\phi}(\mathbf{x}) - \frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})} \right\|_2^2 \right].$$

Although  $\frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})}$  is unknown, “integration by parts” makes the loss depend only on  $\mathbf{s}_{\phi}$ . Let  $\mathcal{L}^{\dagger}$  be the adjoint of  $\mathcal{L}$ , defined by

$$\int (\mathcal{L}f)^{\top} g = \int f (\mathcal{L}^{\dagger}g) \quad \text{for all test functions } f, g,$$

which formally “moves”  $\mathcal{L}$  across the integral when boundary terms vanish. Expanding the square and applying this identity yields the tractable objective

$$\mathcal{L}_{\text{GSM}}(\phi) = \mathbb{E}_{\mathbf{x} \sim p} \left[ \frac{1}{2} \|\mathbf{s}_{\phi}(\mathbf{x})\|_2^2 - (\mathcal{L}^{\dagger} \mathbf{s}_{\phi})(\mathbf{x}) \right] + \text{const},$$

where the constant does not depend on  $\phi$ . We use  $p$  only through expectations, so the generalized score matching loss admits an empirical estimator from training data, exactly as in classical score matching.

For  $\mathcal{L} = \nabla$  we have  $\mathcal{L}^\dagger = -\nabla \cdot$ , which recovers Hyvärinen's score matching objective  $\mathbb{E}_p[\frac{1}{2}\|\mathbf{s}_\phi\|_2^2 + \nabla \cdot \mathbf{s}_\phi]$  in Equation (3.2.2).

### Examples of Operators.

- **Classical Score Matching.** Consider  $\mathcal{L} = \nabla_{\mathbf{x}}$ . Then the generalized score reduces to the classical score function

$$\frac{\mathcal{L}p(\mathbf{x})}{p(\mathbf{x})} = \nabla_{\mathbf{x}} \log p(\mathbf{x}).$$

- **Denoising Score Matching.** For additive Gaussian noise, define the operator

$$(\mathcal{L}f)(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}} f'(\tilde{\mathbf{x}}) + \sigma^2 \nabla_{\tilde{\mathbf{x}}} f(\tilde{\mathbf{x}}).$$

Then

$$\frac{\mathcal{L}p_\sigma(\tilde{\mathbf{x}})}{p_\sigma(\tilde{\mathbf{x}})} = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}) = \mathbb{E}[\mathbf{x}_0 | \tilde{\mathbf{x}}],$$

with  $p_\sigma(\tilde{\mathbf{x}}) := \int \mathcal{N}(\tilde{\mathbf{x}}; \alpha \mathbf{x}_0, \sigma^2 \mathbf{I}) p_{\text{data}}(\mathbf{x}) d\mathbf{x}$  and  $\tilde{\mathbf{x}} = \mathbf{x} + \sigma \epsilon$ . This is exactly the Tweedie's identity. Minimizing  $\mathcal{L}_{\text{GSM}}$  with this operator trains  $\mathbf{s}_\phi$  to approximate the denoiser, recovering the denoising score matching objective.

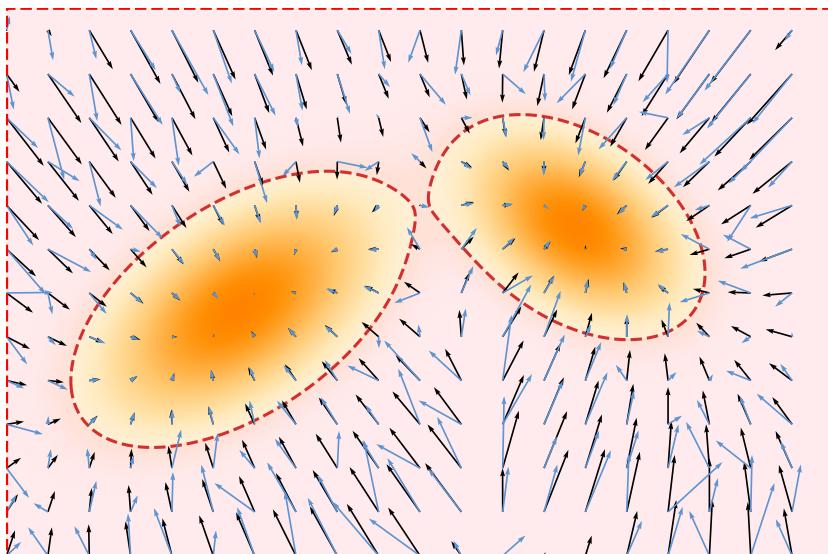
- **Higher Order Targets.** Stacking derivatives inside  $\mathcal{L}$  exposes  $\nabla^2 \log p$  and higher derivatives, which align with posterior covariance and higher order cumulants.

**Extensions and Use Cases.** Generalized score matching extends beyond continuous variables to discrete settings, including language modeling (Meng *et al.*, 2022; Lou *et al.*, 2024). It also motivates score inspired training that yields denoising style objectives. This operator view unifies a range of objectives, admits empirical estimation from data, and offers a general principle for designing loss functions through suitable choices of  $\mathcal{L}$ .

## 3.4 Multi-Noise Levels of Denoising Score Matching (NCSN)

### 3.4.1 Motivation

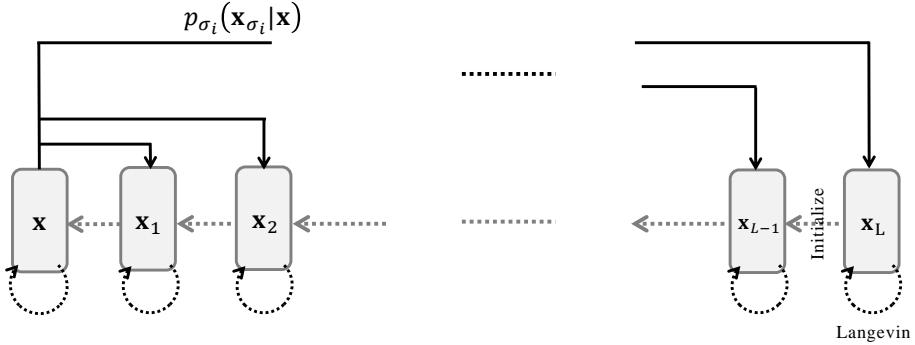
Adding Gaussian noise with a single fixed variance to the data distribution smooths it to a certain extent, but training a score-based model at only one noise level introduces key limitations. At low levels of injected noise, Langevin dynamics struggles to traverse modes in multi-modal distributions due to vanishing gradients in low-density regions. In contrast, at high noise levels, sampling becomes easier, but the model captures only coarse structures, resulting in blurry samples that lack fine detail. Furthermore, Langevin dynamics can be slow to converge or even fail in high-dimensional spaces. Since it depends on the gradient of the log-density for guidance, poor initialization, particularly in plateau regions or near saddle points, can impede exploration or cause the sampler to get trapped in a single mode.



**Figure 3.6: Illustration of SM inaccuracy (revisiting Figure 3.4).** the red region indicates low-density areas with potentially inaccurate score estimates due to limited sample coverage, while high-density regions tend to yield more accurate estimates.

To address these challenges, Song and Ermon (2019) propose injecting Gaussian noise at multiple levels into the data distribution and jointly training a noise-conditional score network (NCSN) to estimate score functions across a range of noise scales. During generation, Langevin dynamics is applied in a noise-annealed fashion: beginning with high-noise levels to enable coarse exploration, and gradually refining toward low-noise levels to recover fine

details.



**Figure 3.7: Illustration of NCSN.** The forward process perturbs the data with multiple levels of additive Gaussian noise  $p_\sigma(\mathbf{x}_\sigma | \mathbf{x})$ . Generation proceeds via Langevin sampling at each noise level, using the result from the current level to initialize sampling at the next lower variance.

### 3.4.2 Training

To overcome the limitations of score-based models trained at a single noise level, Song and Ermon (2019) propose adding Gaussian noise at multiple levels to the data distribution. Specifically, a sequence of  $L$  noise levels  $\{\sigma_i\}_{i=1}^L$  is chosen such that

$$0 < \sigma_1 < \sigma_2 < \dots < \sigma_L,$$

where  $\sigma_1$  is small enough to preserve most of the data's fine details, and  $\sigma_L$  is large enough to sufficiently smooth the distribution, facilitating easier training.

Each noisy sample is constructed by perturbing a clean data point  $\mathbf{x} \sim p_{\text{data}}$  as  $\mathbf{x}_\sigma = \mathbf{x} + \sigma \boldsymbol{\epsilon}$  with  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . This defines the

#### Perturbation Kernel:

$$p_\sigma(\mathbf{x}_\sigma | \mathbf{x}) := \mathcal{N}(\mathbf{x}_\sigma; \mathbf{x}, \sigma^2 \mathbf{I}),$$

which induces the

#### Marginal Distribution:

$$p_\sigma(\mathbf{x}_\sigma) = \int p_\sigma(\mathbf{x}_\sigma | \mathbf{x}) p_{\text{data}}(\mathbf{x}) d\mathbf{x},$$

at each noise level  $\sigma$ . It presents the Gaussian smoothed data distribution.

**Training Objective of NCSN.** The goal is to train a noise-conditional score network  $s_\phi(\mathbf{x}, \sigma)$  to estimate the score function  $\nabla_{\mathbf{x}} \log p_\sigma(\mathbf{x})$  for all  $\sigma \in \{\sigma_i\}_{i=1}^L$ . This is achieved by minimizing the DSM objective across all noise levels:

$$\mathcal{L}_{\text{NCSN}}(\phi) := \sum_{i=1}^L \lambda(\sigma_i) \mathcal{L}_{\text{DSM}}(\phi; \sigma_i), \quad (3.4.1)$$

where

$$\mathcal{L}_{\text{DSM}}(\phi; \sigma) = \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x}), \tilde{\mathbf{x}} \sim p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})} \left[ \left\| s_\phi(\tilde{\mathbf{x}}, \sigma) - \left( \frac{\mathbf{x} - \tilde{\mathbf{x}}}{\sigma^2} \right) \right\|_2^2 \right],$$

and  $\lambda(\sigma_i) > 0$  is a weighting function for each scale.

Minimizing this objective yields the score model  $\mathbf{s}^*(\mathbf{x}, \sigma)$  that recovers the true score at each noise level:

$$\mathbf{s}^*(\cdot, \sigma) = \nabla_{\mathbf{x}} \log p_\sigma(\cdot), \quad \text{for all } \sigma \in \{\sigma_i\}_{i=1}^L,$$

as it is essentially DSM minimization (see Theorem 3.3.1).

**Relationship with DDPM Loss.** Let  $\mathbf{x}_\sigma = \mathbf{x} + \sigma \epsilon$  with  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  and let  $p_\sigma$  denote the marginal distribution. By Tweedie's formula,

$$\nabla_{\mathbf{x}_\sigma} \log p_\sigma(\mathbf{x}_\sigma) = -\frac{1}{\sigma} \mathbb{E}[\epsilon | \mathbf{x}_\sigma].$$

Thus the NCSN optimum is the true score  $\mathbf{s}^*(\mathbf{x}_\sigma, \sigma) = \nabla_{\mathbf{x}_\sigma} \log p_\sigma(\mathbf{x}_\sigma)$ , while the Bayes optimal noise predictor under the DDPM loss Equation (2.2.10) is  $\epsilon^*(\mathbf{x}_\sigma, \sigma) = \mathbb{E}[\epsilon | \mathbf{x}_\sigma]$ . They are exactly equivalent via

$$\mathbf{s}^*(\mathbf{x}_\sigma, \sigma) = -\frac{1}{\sigma} \epsilon^*(\mathbf{x}_\sigma, \sigma), \quad \epsilon^*(\mathbf{x}_\sigma, \sigma) = -\sigma \mathbf{s}^*(\mathbf{x}_\sigma, \sigma).$$

In the DDPM's perturbation Equation (2.2.9) with discrete index  $i$ ,

$$\mathbf{x}_i = \bar{\alpha}_i \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_i^2}$$

the same relation gives

$$\mathbf{s}^*(\mathbf{x}_i, i) = -\frac{1}{\sigma_i} \mathbb{E}[\epsilon | \mathbf{x}_i],$$

so minimizing Equation (2.2.10) learns the conditional denoiser for  $\epsilon$ , which is a scaled reparameterization of the true score at noise level  $i$ .

We will systematically compare and summarize this equivalence of parameterizations in Chapter 6.

---

**Algorithm 1** Annealed Langevin Dynamics

---

**Input:** Trained score  $\mathbf{s}_{\phi^\times}(\cdot, \sigma_\ell)$ , step sizes  $\eta_\ell$ , and Langevin iteration budgets  $N_\ell$  for each noise level  $\ell = L, \dots, 2$

- 1:  $\mathbf{x}^{\sigma_L} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- 2: **for**  $\ell = L, \dots, 2$  **do**
- 3:    $\tilde{\mathbf{x}}_0 \leftarrow \mathbf{x}^{\sigma_\ell}$   
       ▷ Initialize Langevin from previous noise level's output
- 4:   **for**  $n = 0$  **to**  $N_\ell - 1$  **do**
- 5:      $\epsilon_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- 6:      $\tilde{\mathbf{x}}_{n+1} \leftarrow \tilde{\mathbf{x}}_n + \eta_\ell \mathbf{s}_{\phi^\times}(\tilde{\mathbf{x}}_n, \sigma_\ell) + \sqrt{2\eta_\ell} \epsilon_n$
- 7:   **end for**
- 8:    $\mathbf{x}^{\sigma_{\ell-1}} \leftarrow \tilde{\mathbf{x}}_{N_\ell}$   
       ▷ Output used as initialization for next noise level
- 9: **end for**

**Output:**  $\mathbf{x}^{\sigma_1}$

---

### 3.4.3 Sampling

With trained score networks available at multiple noise levels

$$\mathbf{s}_{\phi^\times}(\cdot, \sigma_1), \quad \mathbf{s}_{\phi^\times}(\cdot, \sigma_2), \quad \dots, \quad \mathbf{s}_{\phi^\times}(\cdot, \sigma_{L-1}), \quad \mathbf{s}_{\phi^\times}(\cdot, \sigma_L),$$

the sampling procedure known as *annealed Langevin dynamics* (Song and Ermon, 2019) generates data by progressively denoising from a high noise level  $\sigma_L$  down to a low noise level  $\sigma_1 \approx 0$ .

Starting from a Gaussian noise  $\mathbf{x}^{\sigma_L} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , the algorithm applies Langevin dynamics at each noise level  $\sigma_\ell$  to approximately sample from the perturbed distribution  $p_{\sigma_\ell}(\mathbf{x})$ . The output at level  $\sigma_\ell$  is used to provide a *better initialization* at the next lower noise level  $\sigma_{\ell-1}$ .

At each level, Langevin dynamics iteratively updates:

$$\tilde{\mathbf{x}}_{n+1} = \tilde{\mathbf{x}}_n + \eta_\ell \mathbf{s}_{\phi^\times}(\tilde{\mathbf{x}}_n, \sigma_\ell) + \sqrt{2\eta_\ell} \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

starting from  $\tilde{\mathbf{x}}_0 := \mathbf{x}^{\sigma_\ell}$ . The step size is typically scaled by the noise level:

$$\eta_\ell = \delta \cdot \frac{\sigma_\ell^2}{\sigma_1^2}, \quad \text{for some fixed } \delta > 0.$$

This noise-annealed refinement proceeds down to the lowest noise level  $\sigma_1$ , where the final sample  $\mathbf{x}^{\sigma_1}$  is obtained. By progressively using the output of the previous level as better initialization for the next, this strategy enables more effective exploration and improved coverage of complex data distributions. Algorithm 1 summarizes the procedure.

**Slow Sampling Speed of NCSN.** NCSN generates samples using annealed MCMC (commonly Langevin dynamics) across noise scales  $\{\sigma_i\}_{i=1}^L$ . For each scale  $\sigma_i$ , it performs  $K$  iterative updates of the form “update  $\tilde{\mathbf{x}}_n$  using the score  $\mathbf{s}_{\phi^\times}(\tilde{\mathbf{x}}_n, \sigma_i)$  plus a small random perturbation”, each requiring a forward pass through the score network. Two factors necessitate large  $L \times K$ :

- (i) **Local Accuracy and Stability:** the learned score is reliable only for small perturbations, requiring small step sizes and many iterations per noise level to avoid bias or instability;
- (ii) **Slow Mixing in High Dimensions:** local MCMC moves explore multi-modal, high-dimensional targets inefficiently, demanding many iterations to reach typical data regions.

Because updates are strictly sequential (each iteration depends on the previous one) and each requires an expensive network evaluation, the overall cost is  $\mathcal{O}(LK)$  sequential network passes, making sampling computationally slow.

### 3.5 Summary: A Comparative View of NCSN and DDPM

**Comparison.** We begin by comparing the graphical models of NCSN and DDPM in Figure 3.7, with key differences and similarities summarized in Table 3.1.

**Table 3.1:** Comparisons of NCSN and DDPM

	NCSN	DDPM
$\mathbf{x}_{i+1}   \mathbf{x}_i$	Derive as $\mathbf{x}_{i+1} = \mathbf{x}_i + \sqrt{\sigma_{i+1}^2 - \sigma_i^2} \epsilon$	Define as $\mathbf{x}_{i+1} = \sqrt{1 - \beta_i} \mathbf{x}_i + \sqrt{\beta_i} \epsilon$
$\mathbf{x}_i   \mathbf{x}$	Define as $\mathbf{x}_i = \mathbf{x} + \sigma_i^2 \epsilon$	Derive as $\mathbf{x}_i = \bar{\alpha}_i \mathbf{x} + \sqrt{1 - \bar{\alpha}_i^2} \epsilon$
$p_{\text{prior}}$	$\mathcal{N}(\mathbf{0}, \sigma_L^2 \mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$
Loss	$\mathbb{E}_i \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[ \left\  \mathbf{s}_{\phi}(\mathbf{x}_i, \sigma_i) + \frac{\epsilon}{\sigma_i} \right\ _2^2 \right]$	$\mathbb{E}_i \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \mathbb{E}_{\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[ \left\  \mathbf{\epsilon}_{\phi}(\mathbf{x}_i, i) - \epsilon \right\ _2^2 \right]$
Sampling	Apply Langevin per layer; use output to initialize the next	Traversing the Markovian chain with $p_{\phi^x}(\mathbf{x}_{i-1}   \mathbf{x}_i)$

**A Shared Bottleneck.** Despite their different formulations, both NCSN and DDPM rely on dense time discretization. This leads to a critical limitation: sampling often requires hundreds or even thousands of iterations, making generation slow and computationally intensive.

#### Question 3.5.1

*How can we accelerate sampling in diffusion models?*

We will revisit this challenge in Chapter 9 and Chapter 10.

### 3.6 Closing Remarks

This chapter has charted a second major path to diffusion models, beginning from the score-based perspective rooted in Energy-Based Models (EBMs). We started by identifying the core challenge of EBMs—the intractable partition function—and introduced the score function,  $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ , as a powerful tool that circumvents this issue entirely.

Our journey led us from classic score matching to its more scalable and robust variant, Denoising Score Matching (DSM). Through DSM, we saw how perturbing data with noise enables a tractable training objective, once again leveraging a conditioning strategy to create a simple regression target. Furthermore, we established a profound connection between score estimation and the act of denoising via Tweedie’s formula, which showed that the score provides the precise direction needed to estimate a clean signal from its noisy observation.

This principle was then extended from a single noise level to a continuum with Noise Conditional Score Networks (NCSN), which learn a single score model conditioned on multiple noise scales and generate samples via annealed Langevin dynamics. By the end of our exploration, we found that NCSN and the DDPM from the variational view, despite their different origins, share a strikingly similar structure and a common bottleneck: slow, sequential sampling.

This convergence is no coincidence; it hints at a deeper, unified mathematical structure. The limitations of these discrete-time models motivate the need for a more general framework. In the next chapter, we will take this crucial step:

1. We will move into a continuous-time perspective, showing that both DDPMs and NCSNs can be elegantly unified as different discretizations of a single, powerful process described by a Stochastic Differential Equation (SDE).
2. This Score SDE framework will formally connect the variational and score-based views, recasting the problem of generation as one of solving a differential equation.

This unifying lens will not only provide profound theoretical clarity but also unlock a new class of advanced numerical methods designed to tackle the fundamental challenge of slow sampling.

# 4

---

## Diffusion Models Today: Score SDE Framework

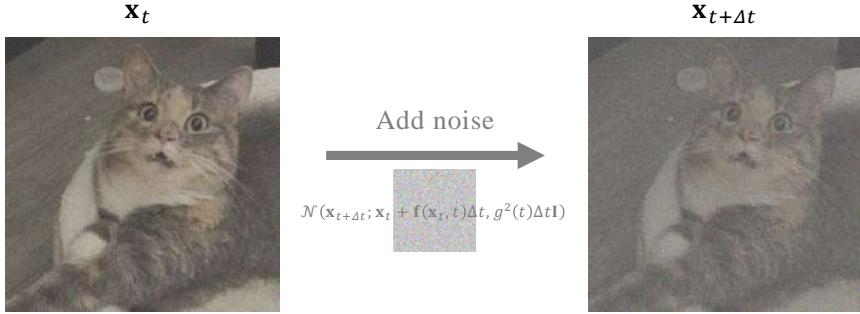
---

*There is only one precise way of presenting the laws, and that is by means of differential equations. They have the advantage of being fundamental and, so far as we know, precise.*

---

Richard P. Feynman

So far, we have studied diffusion models from two perspectives: the variational view and the score-based view, the latter naturally emerging from the EBM formulation. We now take the next step and move to the *continuous-time framework*. At its core lies the *Score SDE*, the continuous limit that unifies DDPM and NCSN into a single formulation. This perspective is powerful because it extends discrete updates with a clean, principled description grounded in differential equations (DE). In this view, generation reduces to solving a DE over time. This lets us directly apply tools from numerical analysis: for example, the basic Euler method can simulate the dynamics, while more advanced solvers improve stability and efficiency. By working in continuous time, we also gain a richer mathematical structure and a unified foundation for understanding, analyzing, and improving diffusion models. This perspective will be developed further in this monograph.



**Figure 4.1: Illustration of the discrete-time noise-adding step.** It adds noise from  $t$  to  $t + \Delta t$  with mean drift  $\mathbf{f}(\mathbf{x}_t, t)$  and diffusion coefficient  $g(t)$ .

## 4.1 Score SDE: Its Principles

The use of multiple noise scales has been a crucial ingredient in the success of NCSN and DDPM frameworks. In this section, we introduce the foundation of the *Score SDE* (Song *et al.*, 2020c), which elevates this idea by considering a continuum of noise levels. A continuous-time limit of forward and reverse diffusion processes had already been noted by Sohl-Dickstein *et al.* (2015), but Song *et al.* (2020c) make this perspective central by formulating the data evolution as a stochastic/ordinary differential equation, where the noise level increases smoothly over time. This continuous-time formulation not only unifies prior discrete-time models but also provides a principled and flexible foundation for generative modeling by casting it as the problem of solving differential equations.

### 4.1.1 Motivation: From Discrete to Continuous-Time Processes

We revisit the forward noise injection schemes of NCSN and DDPM. NCSN uses a sequence of increasing noise levels  $\{\sigma_i\}_{i=1}^L$ . Each clean sample  $\mathbf{x} \sim p_{\text{data}}$  is perturbed as

$$\mathbf{x}_{\sigma_i} = \mathbf{x} + \sigma_i \boldsymbol{\epsilon}_i, \quad \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

DDPM instead injects noise incrementally with a variance schedule  $\{\beta_i\}_{i=1}^L$ :

$$\mathbf{x}_i = \sqrt{1 - \beta_i^2} \mathbf{x}_{i-1} + \beta_i \boldsymbol{\epsilon}_i, \quad \boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

We view them together on a discrete time grid, where the sequential update

from  $\mathbf{x}_t$  to  $\mathbf{x}_{t+\Delta t}$  takes the form<sup>1</sup>:

$$\begin{aligned}\text{NCSN: } \mathbf{x}_{t+\Delta t} &= \mathbf{x}_t + \sqrt{\sigma_{t+\Delta t}^2 - \sigma_t^2} \boldsymbol{\epsilon}_t &\approx \mathbf{x}_t + \sqrt{\frac{d\sigma_t^2}{dt} \Delta t} \boldsymbol{\epsilon}_t \\ \text{DDPM: } \mathbf{x}_{t+\Delta t} &= \sqrt{1 - \beta_t} \mathbf{x}_t + \sqrt{\beta_t} \boldsymbol{\epsilon}_t &\approx \mathbf{x}_t - \frac{1}{2} \beta_t \mathbf{x}_t \Delta t + \sqrt{\beta_t \Delta t} \boldsymbol{\epsilon}_t,\end{aligned}$$

where  $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . Interestingly, both noise injection processes follow a common structural pattern:

$$\mathbf{x}_{t+\Delta t} \approx \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t) \Delta t + g(t) \sqrt{\Delta t} \boldsymbol{\epsilon}_t, \quad (4.1.1)$$

with  $\mathbf{f} : \mathbb{R}^D \times \mathbb{R} \rightarrow \mathbb{R}^D$  and  $g : \mathbb{R} \rightarrow \mathbb{R}$  given by:

$$\begin{aligned}\text{NCSN: } \mathbf{f}(\mathbf{x}, t) &= 0, & g(t) &= \sqrt{\frac{d\sigma^2(t)}{dt}} \\ \text{DDPM: } \mathbf{f}(\mathbf{x}, t) &= -\frac{1}{2} \beta(t) \mathbf{x}, & g(t) &= \sqrt{\beta(t)}.\end{aligned}$$

This formulation corresponds to the following Gaussian transition:

$$p(\mathbf{x}_{t+\Delta t} | \mathbf{x}_t) := \mathcal{N} \left( \mathbf{x}_{t+\Delta t}; \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t) \Delta t, g^2(t) \Delta t \mathbf{I} \right), \quad (4.1.2)$$

where, by a slight abuse of notation, we treat  $\mathbf{x}_t$  as a fixed sample and  $\mathbf{x}_{t+\Delta t}$  as a random variable.

As  $\Delta t \rightarrow 0$  (which can be conceptually understood as preparing *infinitely many* layers of noises), the discrete time process converges to a continuous time SDE evolving forward in time<sup>2</sup>:

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t) dt + g(t) d\mathbf{w}(t),$$

where  $\mathbf{w}(t)$  is a standard Wiener process (or Brownian motion).

### Remark.

While a full formal definition is not necessary here, a *Wiener process* is a continuous-time stochastic process  $\mathbf{w}(t)$  that starts at zero, has independent increments, and satisfies that for any  $s < t$ , the increment  $\mathbf{w}(t) - \mathbf{w}(s)$  is normally distributed with mean zero and variance  $t - s$ . It represents the

<sup>1</sup>For convenience, we use  $\mathbf{x}(t)$  and  $\mathbf{x}_t$  interchangeably (and similarly for other time-dependent variables) to denote samples at time  $t$ .

<sup>2</sup>The forward kernel in Equation (4.1.2) converges, as  $\Delta t \rightarrow 0$ , to the solution of the corresponding Itô SDE. A fully rigorous proof relies on advanced results which we defer to the literature.

accumulation of independent Gaussian fluctuations over time, and although it is almost surely continuous, it is nowhere differentiable.

Over an infinitesimal time interval  $[t, t + dt]$ , the increment of a Wiener process is defined as

$$d\mathbf{w}(t) := \mathbf{w}(t + dt) - \mathbf{w}(t),$$

which is modeled as a Gaussian random variable with zero mean and variance  $dt$ :

$$d\mathbf{w}(t) \sim \mathcal{N}(\mathbf{0}, dt\mathbf{I}).$$

A brief introduction to the foundations of SDEs is provided in Section A.2, with a more advanced discussion in Chapter C. However, we can conceptually understand the connection between the discrete and continuous formulations as follows:

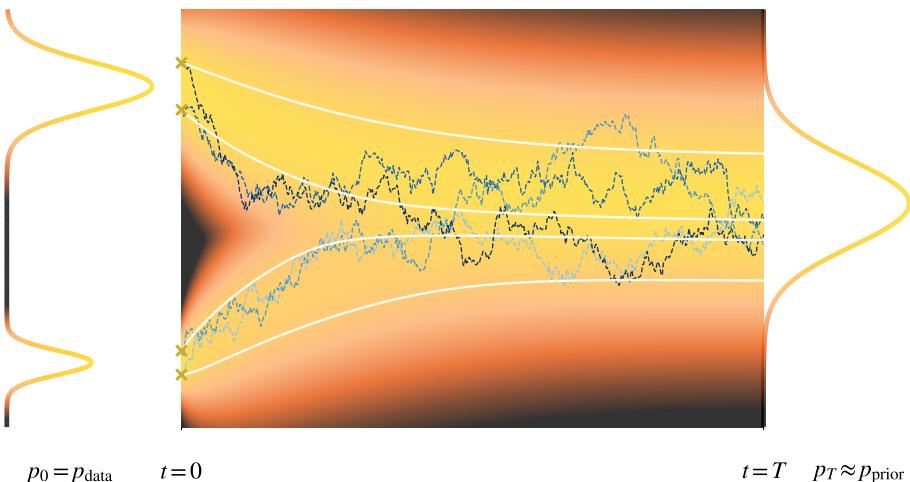
- $\mathbf{x}(t + \Delta t) - \mathbf{x}(t) \approx d\mathbf{x}(t),$
- $\Delta t \approx dt,$
- $\sqrt{\Delta t}\epsilon_t \sim \mathcal{N}(\mathbf{0}, \Delta t\mathbf{I}) \approx d\mathbf{w}(t).$

Once the drift  $\mathbf{f}(\mathbf{x}, t)$  and diffusion  $g(t)$  are specified, the forward time SDE automatically induces a reverse time SDE that transports the terminal noise distribution back to the data distribution. The reverse dynamics involve only a single unknown term, surprisingly the *score function at each continuous-time level*. This identifies score matching as the training objective; once the score is learned, sampling amounts to numerically integrating the reverse time SDE with the learned score.

While Section 4.2 presents practical implementations, we first examine the theoretical foundations of the forward and reverse processes in Section 4.1.2 and Section 4.1.3.

### 4.1.2 Forward-Time SDEs: From Data to Noise

With this formulation, earlier methods based on discrete time, such as NCSN (Song and Ermon, 2019) and DDPM (Sohl-Dickstein *et al.*, 2015; Ho *et al.*, 2020), can be unified under the *continuous-time* framework through a stochastic process  $\mathbf{x}(t)$  governed by a forward SDE defined on the interval  $[0, T]$ :



**Figure 4.2: (1D) Visualization of the forward process in a diffusion model.** The process starts from initial points sampled (denoted as “ $\times$ ”) from a complex bimodal data distribution ( $p_0 = p_{\text{data}}$ ) and evolves toward a simple, unimodal Gaussian prior ( $p_T \approx p_{\text{prior}}$ ). The background heatmap illustrates the evolving marginal probability density,  $p_t$ , which smooths over time. Sample trajectories are shown evolving from  $t = 0$  to  $t = T$ , comparing the stochastic forward SDE process (blue paths) with its deterministic counterpart, the PF-ODE (white paths). Note that the PF-ODE is a deterministic transport map for densities, not generally the mean of sample paths started from a single point.

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t) dt + g(t) d\mathbf{w}(t), \quad \mathbf{x}(0) \sim p_{\text{data}}. \quad (4.1.3)$$

Here,  $\mathbf{f}(\cdot, t): \mathbb{R}^D \rightarrow \mathbb{R}^D$  is the drift,  $g(t) \in \mathbb{R}$  is the scalar diffusion coefficient, and  $\mathbf{w}(t)$  denotes a standard Wiener process. We refer to this as the *forward SDE*, which describes how clean data is gradually perturbed into noise over time.

Once the drift  $\mathbf{f}$  and diffusion coefficient  $g$  are specified, the forward process is fully determined, describing how the data variable is progressively corrupted through the injection of Gaussian noise. In particular, two families of time-dependent densities are induced:

**Perturbation Kernels.** The conditional law

$$p_t(\mathbf{x}_t | \mathbf{x}_0)$$

describes how a clean data sample  $\mathbf{x}_0 \sim p_{\text{data}}$  evolves into its noisy counterpart  $\mathbf{x}_t$  at time  $t$ . In general, the drift term  $\mathbf{f}(\mathbf{x}, t)$  in Equation (4.1.3) can be an

arbitrary function of  $\mathbf{x}$ , but a common and analytically convenient choice is to assume it is affine:

$$\mathbf{f}(\mathbf{x}, t) = f(t)\mathbf{x}, \quad (4.1.4)$$

where  $f(t)$  is a scalar function of  $t$ , typically taken to be non-positive. Under this structure, the process remains Gaussian at every time, and the conditional distribution admits a closed-form solution obtained by solving the associated mean-variance ODEs (Särkkä and Solin, 2019) (see also Section 4.3.3). In particular,

$$p_t(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \mathbf{m}(t), P(t)\mathbf{I}_D),$$

with

$$\mathbf{m}(t) = \exp\left(\int_0^t f(u) du\right)\mathbf{x}_0, \quad P(t) = \int_0^t \exp\left(2 \int_s^t f(u) du\right) g^2(s) ds,$$

and initial conditions  $\mathbf{m}(0) = \mathbf{x}_0$ ,  $P(0) = 0$ .

This explicit form allows one to sample  $\mathbf{x}_t$  given  $\mathbf{x}_0$  directly, without numerically integrating the SDE, hence the term *simulation-free*. Both NCSN and DDPM fall into this affine-drift setting.

In the remainder, we develop the general theory for arbitrary drifts  $\mathbf{f}(\mathbf{x}, t)$ , but will return to the affine drift when closed-form analysis is useful.

**Marginal Densities.** The time-marginal density  $p_t(\mathbf{x}_t)$  is obtained by integrating over the perturbation kernel:

$$p_t(\mathbf{x}_t) := \int p_t(\mathbf{x}_t | \mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0, \quad \text{with } p_0 = p_{\text{data}}. \quad (4.1.5)$$

By choosing the coefficients  $f(t)$  and  $g(t)$  appropriately, the forward process gradually adds noise until the influence of the initial state is effectively forgotten. As  $T$  becomes large, the conditional distribution  $p_T(\mathbf{x}_T | \mathbf{x}_0)$  no longer depends on  $\mathbf{x}_0$ , because its mean evolves as

$$\mathbf{m}(T) = \exp\left(\int_0^T f(u) du\right)\mathbf{x}_0 \longrightarrow \mathbf{0}, \quad \text{as } T \rightarrow \infty,$$

provided  $f(u)$  is non-positive so that the exponential factor decays. At the same time, the variance grows and stabilizes to match a chosen prior distribution. Consequently, the marginal

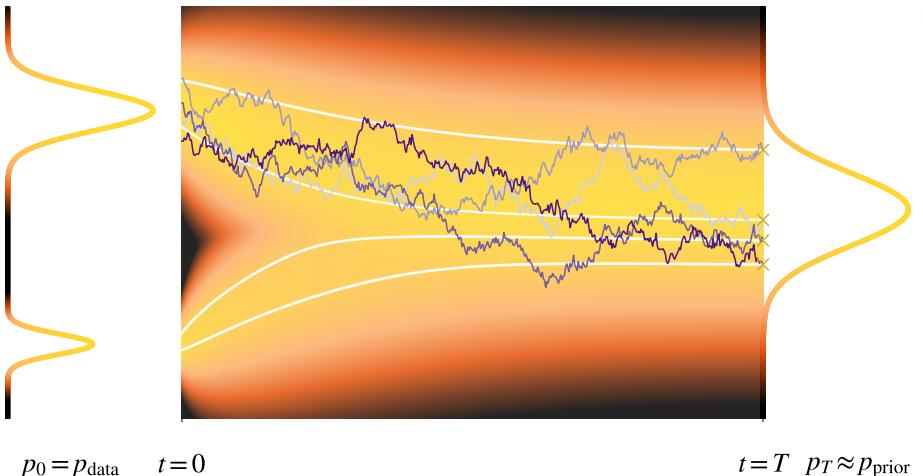
$$p_T(\mathbf{x}_T) = \int p_T(\mathbf{x}_T | \mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0,$$

which initially represents a complicated mixture over data samples, converges to a simple prior  $p_{\text{prior}}$ , typically Gaussian. In this limit,

$$p_T(\mathbf{x}_T) \approx p_{\text{prior}}(\mathbf{x}_T) \quad \text{and} \quad p_T(\mathbf{x}_T | \mathbf{x}_0) \approx p_{\text{prior}}(\mathbf{x}_T),$$

so the forward process maps any data distribution into a tractable prior, providing a convenient starting point for reversal and generation.

#### 4.1.3 Reverse-Time Stochastic Process for Generation



**Figure 4.3: Visualization of the reverse-time stochastic process for data generation.** It begins from samples drawn from a simple prior distribution ( $p_{\text{prior}}$ ) at  $t = T$  (denoted as “ $\times$ ”), which are evolved backward in time using a reverse-SDE. The resulting trajectories terminate at  $t = 0$  and collectively form the target bimodal data distribution ( $p_0 = p_{\text{data}}$ ). The background heatmap illustrates how the probability density is gradually transformed from a simple Gaussian into the complex target distribution.

Intuitively, data generation from noise can be achieved by “reversing” the forward process: starting from a random point sampled from the prior distribution and evolving it backward in time to obtain a generated sample. For deterministic systems (that is, ODEs), this idea works naturally. Since no randomness is involved, reversing time simply means tracing the trajectory of a point in the opposite direction along the same path as in the forward process<sup>3</sup>. In contrast, SDEs incorporate stochasticity at every time step, meaning that a single point can evolve along many plausible random trajectories. As a result, reversing such processes is more subtle<sup>4</sup>.

While individual stochastic trajectories are not reversible, the remarkable insight is that the distribution over these trajectories can be reversed. This is formalized by a foundational result from Anderson (1982), which shows that the

<sup>3</sup>Technically, this corresponds to solving the ODE with a time-flipping substitution  $t \leftrightarrow T - t$ .

<sup>4</sup>Naively flipping time does not yield the correct reverse process.

time-reversed process  $\{\bar{\mathbf{x}}(t)\}_{t \in [0,T]}$ <sup>5</sup> of the forward process in Equation (4.1.3) is itself governed by a well-defined SDE. This reverse-time process evolves from  $T$  to 0, and its dynamics are given by:

$$\begin{aligned} d\bar{\mathbf{x}}(t) &= \left[ \mathbf{f}(\bar{\mathbf{x}}(t), t) - g^2(t) \nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t)) \right] dt + g(t) d\bar{\mathbf{w}}(t), \\ \bar{\mathbf{x}}(T) &\sim p_{\text{prior}} \approx p_T. \end{aligned} \quad (4.1.6)$$

Here,  $\bar{\mathbf{w}}(t)$  denotes a standard Wiener process in reverse time, defined as  $\bar{\mathbf{w}}(t) := \mathbf{w}(T-t) - \mathbf{w}(T)$ .

To build intuition for Equation (4.1.6), we present a concrete example in Section 4.1.6 with a Gaussian data distribution and linear–Gaussian dynamics. This setting is analytically tractable: one can derive the time-reversal formula directly using basic calculus and linear algebra, without invoking the full general theory of Anderson (1982).

Note that the presence of stochasticity ( $g \neq 0$ ) introduces an additional correction term,  $-g^2(t) \nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t))$ , which accounts for the effect of diffusion and ensures that the reversed dynamics correctly reproduce the evolution of marginal distributions induced by the forward SDE (see Section 4.1.5).

**Conceptually, Why Does the Reverse Process Work?** Section 4.5.2 presents an intuitive derivation of the reverse-time SDE by connecting it to the DDPM variational framework (optional but insightful). Here, we provide complementary intuition for how the reverse-time dynamics recover structured data from noise.

At first glance, the presence of Brownian noise in the reverse time process may seem paradoxical. If the forward diffusion spreads data into increasingly noisy configurations, it is unclear how reversing this process, particularly one that introduces additional randomness through  $\bar{\mathbf{w}}(t)$ , can produce clean, structured samples concentrated near the data manifold. The key point is that the reverse time SDE does not inject arbitrary randomness. The diffusion term  $g(t) d\bar{\mathbf{w}}(t)$  is always coupled with the score–driven drift  $-g^2(t) \nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t))$ . Together, these terms balance one another: the score guides trajectories toward regions of higher density, while the noise introduces controlled stochasticity that allows exploration without overwhelming the dynamics.

---

<sup>5</sup>We use the “bar” notation to distinguish the reverse process  $\{\bar{\mathbf{x}}(t)\}_{t \in [0,T]}$  from the forward process  $\{\mathbf{x}(t)\}_{t \in [0,T]}$ , defined by the forward-time SDE.

To see this more clearly, return to the Langevin intuition in Equation (3.1.6). When  $f(t) \equiv 0$ , Equation (4.1.6) reads

$$d\bar{\mathbf{x}}(t) = -g^2(t)\nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t)) dt + g(t) d\bar{\mathbf{w}}(t).$$

Reparameterize time forward via  $s := T - t$  (so  $dt = -ds$ ), and rename the Brownian motion in law so that  $d\bar{\mathbf{w}}(t) = -d\mathbf{w}_s$ . Writing  $\bar{\mathbf{x}}_s := \bar{\mathbf{x}}(T - s)$  and  $\pi_s := p_{T-s}$  then gives

$$\begin{aligned} d\bar{\mathbf{x}}_s &= g^2(T-s)\nabla \log \pi_s(\bar{\mathbf{x}}_s) ds + g(T-s) d\mathbf{w}_s \\ &= 2\tau(s)\nabla \log \pi_s(\bar{\mathbf{x}}_s) ds + \sqrt{2\tau(s)} d\mathbf{w}_s, \quad \tau(s) := \frac{1}{2}g^2(T-s). \end{aligned}$$

This has the Langevin form with a time-varying temperature  $\tau(s)$ , targeting the evolving density  $\pi_s$ . By Tweedie's formula (Equation (3.3.6)), the score direction  $\nabla \log \pi_s$  points toward the conditional clean signal at each time slice, so the drift continually “pulls back” denoised structure.

Crucially,  $g(t)$  is *annealing* along the reverse trajectory. Early on ( $s \approx 0$ , i.e.,  $t \approx T$ ),  $g(T-s)$  is typically larger, so the injected noise is stronger and the process explores broadly. As  $s$  increases,  $g(T-s)$  decreases, the stochastic term weakens, and the score term dominates, pulling samples into high-density regions of  $\pi_s$ ; by  $s = T$  (i.e.,  $t = 0$ ), trajectories concentrate near the data manifold.

**Overview of Reverse-Time SDE Capabilities.** It is fascinating how the time-dependent score function

$$\mathbf{s}(\mathbf{x}, t) := \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$$

naturally appears in Equation (4.1.6). Once the forward coefficients  $f(t)$  and  $g(t)$  are specified, the score is the only remaining unknown in the reverse dynamics. This highlights its central role: with the score in hand, the reverse process is determined, and sampling amounts to numerically integrating Equation (4.1.6) with the learned score.

Since the oracle score generally lacks a closed-form expression, we adopt the approach of Chapter 3 and train a neural network  $\mathbf{s}_\phi(\mathbf{x}, t)$  to approximate it via score matching; see Section 4.2.1 for details. Substituting  $\mathbf{s}(\mathbf{x}, t)$  with  $\mathbf{s}_\phi(\mathbf{x}, t)$  in Equation (4.1.6) then specifies the reverse dynamics completely.

Generation corresponds to solving the reverse-time SDE reversely from  $t = T$ , starting with  $\mathbf{x}_T \sim p_{\text{prior}}$ , to  $t = 0$ . Importantly, Anderson (1982) proves that the marginal densities of the forward and reverse processes coincide, ensuring that samples at  $t = 0$  approximately follow  $p_{\text{data}}$  when  $p_{\text{prior}} \approx p_T$ . We will explore this further in Section 4.2.2.

#### 4.1.4 Deterministic Process (Probability Flow ODE) for Generation

Although the SDE in Equation (4.1.6) introduces stochasticity and potentially increases the diversity of generated samples, a question arises:

##### Question 4.1.1

*Is it necessary to sample using the SDE in Equation (4.1.6)?*

Inspired by Maoutsa *et al.* (2020), Song *et al.* (2020c) also introduced a deterministic process, an ODE, that evolves samples with the same marginal distributions as the forward SDE. This process  $\{\tilde{\mathbf{x}}(t)\}_{t \in [0, T]}$ <sup>6</sup>, called the *Probability Flow ODE (PF-ODE)*, is given by:

$$\frac{d}{dt} \tilde{\mathbf{x}}(t) = \mathbf{f}(\tilde{\mathbf{x}}(t), t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\tilde{\mathbf{x}}(t)). \quad (4.1.7)$$

Analogous to the SDE case, one can replace the true score with a learned approximation and integrate the reverse-time ODE from  $t = T$  to  $t = 0$  to generate samples. Concretely, the generated sample (solution of PF-ODE at time  $t = 0$ ) takes the form

$$\tilde{\mathbf{x}}(T) + \int_T^0 \left[ \mathbf{f}(\tilde{\mathbf{x}}(\tau), \tau) - \frac{1}{2} g^2(\tau) \nabla_{\mathbf{x}} \log p_{\tau}(\tilde{\mathbf{x}}(\tau)) \right] d\tau,$$

where the initial condition  $\tilde{\mathbf{x}}(T) \sim p_{\text{prior}}$ . Since this integral is intractable in closed form, practical generation relies on numerical solvers (e.g., Euler method, see Equation (4.2.4)).

Compared to the reverse-time SDE, the PF-ODE offers two key advantages:

- The ODE can be integrated in either direction, from  $t = 0$  to  $t = T$  or from  $t = T$  to  $t = 0$ , using the same formulation of equation, provided the corresponding initial condition is specified at the chosen endpoint. This bidirectionality contrasts with SDEs, which generally admit only forward time integration.
- It benefits from a wide range of well-established, off-the-shelf numerical solvers developed for ODEs.

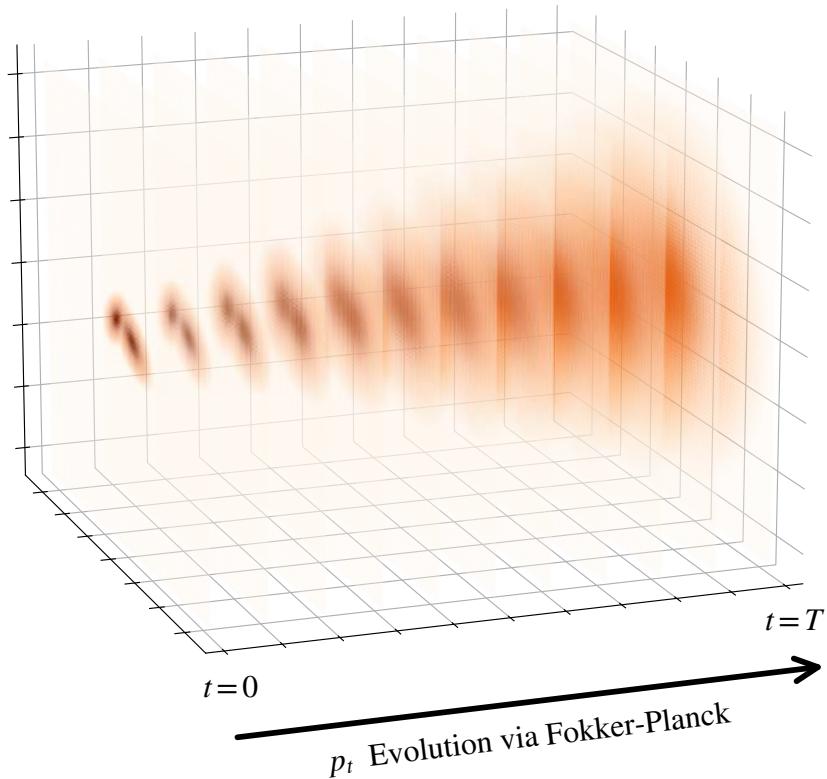
We emphasize that the PF-ODE is not obtained by simply removing the diffusion term in Equation (4.1.6); notably, the factor of  $\frac{1}{2}$  in its drift term

---

<sup>6</sup>We use a tilde to distinguish processes associated with the forward and reverse-time SDEs. Going forward, we omit this notational distinction for simplicity.

has a principled origin. At a high level, Equation (4.1.7) arises by choosing the drift of an ODE such that its evolution preserves the same marginal densities as the forward SDE in Equation (4.1.3). The underlying principle (i.e., Fokker-Planck Equation (Øksendal, 2003)) ensuring this alignment of marginals will be detailed in the next section.

#### 4.1.5 Matching Marginal Distributions in Forward/Reverse-Time SDEs and PF-ODE



**Figure 4.4: (2D) Temporal evolution of the marginal density  $p_t$ .** The forward SDE has  $\mathbf{f} \equiv \mathbf{0}$  and  $g(t) = \sqrt{2t}$  on  $[0, T]$ . It starts with  $p_0 = p_{\text{data}}$  a two-mode Gaussian mixture and ends at  $p_T \approx p_{\text{prior}} := \mathcal{N}(\mathbf{0}, T^2 \mathbf{I})$ . The temporal-spatial evolution of  $p_t$  follows the Fokker–Planck equation.

**Fokker-Planck Equation to Ensure Alignment of Marginal Densities.** A central concept in diffusion models is that *different processes can lead to the same sequence of marginal distributions* (as we will illustrate later in this subsection). The objective is to construct a process that transforms  $p_{\text{prior}}$  into  $p_{\text{data}}$  by aligning the marginals across time, and in particular at  $t = 0$ . The exact form of the process is secondary, provided it is tractable and admits efficient sampling. This naturally leads to a fundamental question:

### Question 4.1.2

*How can we ensure that different processes yield identical marginal distributions?*

Returning to our setup, once the forward SDE is specified, it defines the evolution of marginal densities from  $p_{\text{data}}$  to  $p_{\text{prior}}$ . The reverse-time SDE and PF-ODE are then constructed so that their trajectories yield marginal distributions that exactly match those of the forward process. The key to this correspondence lies in the Fokker–Planck equation, which governs how marginal densities evolve under diffusion processes. The following theorem (Anderson, 1982; Song *et al.*, 2020c) establishes the foundation for this connection:

### Theorem 4.1.1: Fokker–Planck Equation Ensures Marginals Alignment

Let  $\{\mathbf{x}(t)\}_{t \in [0, T]}$  evolves with the forward SDE

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t) dt + g(t) d\mathbf{w}(t),$$

with initial condition  $\mathbf{x}(0) \sim p_0 = p_{\text{data}}$ . Then its marginal densities  $p_t$  satisfy the Fokker–Planck equation

$$\begin{aligned}\partial_t p_t(\mathbf{x}) &= -\nabla_{\mathbf{x}} \cdot [\mathbf{f}(\mathbf{x}, t)p_t(\mathbf{x})] + \frac{1}{2}g^2(t)\Delta_{\mathbf{x}} p_t(\mathbf{x}) \\ &= -\nabla_{\mathbf{x}} \cdot [\tilde{\mathbf{f}}(\mathbf{x}, t)p_t(\mathbf{x})],\end{aligned}\tag{4.1.8}$$

where  $\Delta_{\mathbf{x}}$  denotes the Laplacian operator, and

$$\mathbf{v}(\mathbf{x}, t) = \mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}).$$

Then, both the PF-ODE and the reverse-time SDE yield the same family  $\{p_t\}_{t \in [0, T]}$ , with the latter evolving in reverse time:

- (i) The *PF-ODE*  $\{\tilde{\mathbf{x}}(t)\}_{t \in [0, T]}$

$$\frac{d\tilde{\mathbf{x}}(t)}{dt} = \mathbf{v}(\tilde{\mathbf{x}}(t), t),$$

if started with  $\tilde{\mathbf{x}}(0) \sim p_0$  and run forward in  $t$ , or equivalently started with  $\tilde{\mathbf{x}}(T) \sim p_T$  and run backward in  $t$ , has marginals  $\tilde{\mathbf{x}}(t) \sim p_t$  for all  $t \in [0, T]$ .

- (ii) The *reverse-time SDE*  $\{\bar{\mathbf{x}}(t)\}_{t \in [0, T]}$

$$d\bar{\mathbf{x}}(t) = [\mathbf{f}(\bar{\mathbf{x}}(t), t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t))] dt + g(t) d\bar{\mathbf{w}}(t),$$

with  $\bar{\mathbf{x}}(0) \sim p_T$  and  $\bar{\mathbf{w}}(t)$  a standard Wiener process in reverse time, has marginals  $\bar{\mathbf{x}}(t) \sim p_{T-t}$ .

#### **Proof for Theorem.**

The proof is provided in Section D.2.5, while Section 4.5.1 offers further intuition behind the Fokker–Planck equation using the marginalization technique of probability. ■

**Multiple Conditional Distributions for a Fixed Marginal.** To understand how the PF-ODE transports  $p_{\text{data}}$  forward in time (or equivalently  $p_{\text{prior}}$  in reverse), consider the *flow map*  $\Psi_{s \rightarrow t} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ , where  $\Psi_{s \rightarrow t}(\mathbf{x}_s)$  denotes

the PF-ODE solution at time  $t$  initialized from  $\mathbf{x}_s$  at time  $s$ , for any time  $s, t \in [0, T]$ . In other words, this map takes an initial state  $\mathbf{x}_s$  and directly jumps to its state at  $t$ :

$$\Psi_{s \rightarrow t}(\mathbf{x}_s) := \mathbf{x}_s + \int_s^t \mathbf{v}(\mathbf{x}_\tau, \tau), d\tau, \quad (4.1.9)$$

with velocity field

$$\mathbf{v}(\mathbf{x}, \tau) := \mathbf{f}(\mathbf{x}, \tau) - \frac{1}{2} g^2(\tau) \nabla_{\mathbf{x}} \log p_\tau(\mathbf{x}).$$

Here, the integral captures the net displacement accumulated along the PF-ODE trajectory  $\mathbf{x}_\tau$ . Under mild smoothness assumptions on  $\mathbf{v}$ , the flow map  $\Psi_{s \rightarrow t} : \mathbb{R}^D \rightarrow \mathbb{R}^D$  is a smooth bijection<sup>7</sup>.

For any  $t \in [0, T]$ , the *pushforward density* is defined as

$$p_t^{\text{fwd}}(\mathbf{x}_t) := \int \delta(\mathbf{x}_t - \Psi_{t \rightarrow 0}(\mathbf{x}_0)) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0,$$

denoted  $\Psi_{t \rightarrow 0} \# p_{\text{data}}$ , representing the distribution at time  $t$  under  $\Psi_{t \rightarrow 0}$ . Theorem 4.1.1 ensures  $p_t^{\text{fwd}} = p_t$ , where  $p_t$  is the marginal density of the forward SDE, equating the deterministic PF-ODE and stochastic kernel:

$$p_t(\mathbf{x}_t) = \int p_t(\mathbf{x}_t | \mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0 = \int \delta(\mathbf{x}_t - \Psi_{t \rightarrow 0}(\mathbf{x}_0)) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0.$$

This implies infinitely many conditionals  $Q_t(\mathbf{x}_t | \mathbf{x}_0)$  yield the same  $p_t(\mathbf{x}_t)$ , for instance:

- **Stochastic (Simulation-Free):**  $Q_t(\mathbf{x}_t | \mathbf{x}_0) = p_t(\mathbf{x}_t | \mathbf{x}_0)$ ,
- **Deterministic (Requires ODE Solving):**  $Q_t(\mathbf{x}_t | \mathbf{x}_0) = \delta(\mathbf{x}_t - \Psi_{t \rightarrow 0}(\mathbf{x}_0))$ ,
- **Mixture:**  $Q_t(\mathbf{x}_t | \mathbf{x}_0) = \lambda p_t(\mathbf{x}_t | \mathbf{x}_0) + (1 - \lambda) \delta(\mathbf{x}_t - \Psi_{t \rightarrow 0}(\mathbf{x}_0))$ ,  $\lambda \in [0, 1]$ .

This nonuniqueness of  $Q_t(\mathbf{x}_t | \mathbf{x}_0)$  arises from the fact that the marginal constraint does not uniquely determine the conditional distribution. This concept reappears in Section 5.2.2 and Section 9.2.3. In particular, there exists an entire family of reverse-time SDEs that are consistent with the same marginal  $p_t$ .

---

<sup>7</sup>Spoiler: the PF-ODE flow map  $\Psi_{s \rightarrow t}$  is exactly the Normalizing Flow (NF) bijection carrying  $p_s$  to  $p_t$  (to be detailed in Section 5.1). The difference is that PF-ODE fixes the unique vector field dictated by the SDE's Fokker–Planck dynamics, whereas NF (or continuous-time NF) parameterizes this field but relies on the same change-of-variables principle.

### Observation 4.1.1: Matching Prescribed Marginal Densities

Multiple processes can give rise to the same sequence of marginal densities; what truly matters is satisfying the Fokker–Planck equation. This fundamental insight affords us remarkable flexibility in designing generative processes that transition from  $p_{\text{prior}}$  to  $p_{\text{data}}$ , or vice versa.

The Fokker–Planck equation lies at the heart of diffusion models and is rooted in the fundamental *change-of-variable formula* for probability densities (see Chapter B for a systematic treatment). Far from being a minor technical detail, this principle recurs throughout our development, most notably in Section 5.2.

#### 4.1.6 A Computable Example: Evolutions of Gaussian Dynamics

When  $p_{\text{data}}$  is a normal distribution (or a mixture of Gaussians), the score function admits a closed-form expression. This makes it an ideal setting for building intuition about diffusion processes: we can explicitly derive the reverse-time SDE and the PF-ODE using only basic calculus, without resorting to advanced mathematical tools. In this subsection, we illustrate how these equations behave in such a tractable case.

**Exact Computation of the Reverse-Time SDE with a Gaussian.** When  $p_{\text{data}}$  is Gaussian, the formula in Equation (4.1.6) can be derived directly, without relying on the general theory and proofs of Anderson (1982). To illustrate the core idea, we consider the one-dimensional case; the extension to higher dimensions follows in the same way.

Start from the forward SDE

$$dx(t) = f(t)x(t) dt + g(t) dw_t,$$

and take one small Euler step of size  $\Delta t > 0$ :

$$x_{t+\Delta t} = ax_t + r\epsilon,$$

where  $a := 1 + f(t)\Delta t$ ,  $r := g(t)\sqrt{\Delta t}$ , and  $\epsilon \sim \mathcal{N}(0, 1)$ . Equivalently, the forward one-step transition kernel is Gaussian:

$$x_{t+\Delta t}|x_t \sim \mathcal{N}(ax_t, r^2).$$

Since  $p_{\text{data}}$  is assumed to be Gaussian, the current marginal at time  $t$  is also Gaussian, which takes the following form:

$$x_t \sim \mathcal{N}(m_t, s_t^2),$$

for some scalar  $m_t$  and  $s_t$ . So conditioning will amount to multiplying two Gaussians and renormalizing. This keeps the algebra elementary.

By Bayes' rule the conditional density is, up to a constant, the product of the prior and the transition kernel:

$$\begin{aligned} p(x_t|x_{t+\Delta t}) &\propto p(x_{t+\Delta t}|x_t)p_t(\mathbf{x}_t) \\ &\propto \exp\left(-\frac{(x_t - m_t)^2}{2s_t^2}\right) \exp\left(-\frac{(x_{t+\Delta t} - ax_t)^2}{2r^2}\right). \end{aligned}$$

The exponent is a quadratic in  $x_t$ . Expanding both squares and grouping terms shows exactly which coefficients matter:

$$-2 \log p(x_t|x_{t+\Delta t}) = Ax_t^2 - 2Bx_t + \text{const},$$

with

$$A := \frac{1}{s_t^2} + \frac{a^2}{r^2}, \quad B := \frac{m_t}{s_t^2} + \frac{ax_{t+\Delta t}}{r^2}.$$

Here  $A$  is the sum of precisions (prior precision plus the transition-kernel precision transported through  $a$ ), while  $B$  is the corresponding precision-weighted sum of targets. With these in hand, completing the square gives the posterior in one line:

$$Ax_t^2 - 2Bx_t = A\left(x_t - \frac{B}{A}\right)^2 - \frac{B^2}{A},$$

so the conditional distribution is Gaussian with variance  $1/A$  and mean  $B/A$ :

$$\text{Var}(x_t|x_{t+\Delta t}) = \frac{1}{\frac{1}{s_t^2} + \frac{a^2}{r^2}}, \quad \mathbb{E}[x_t|x_{t+\Delta t}] = \frac{\frac{m_t}{s_t^2} + \frac{ax_{t+\Delta t}}{r^2}}{\frac{1}{s_t^2} + \frac{a^2}{r^2}}.$$

These closed forms already describe the reverse transition for any small  $\Delta t$ . To read off a reverse-time SDE, we now expand them for small  $\Delta t$ .

Use  $a = 1 + f(t)\Delta t$  and  $r^2 = g^2(t)\Delta t$ . As  $\Delta t \rightarrow 0$ , the contribution  $\frac{a^2}{r^2} \sim \frac{1}{g^2(t)\Delta t}$  dominates the precision, so the variance becomes

$$\text{Var}(x_t|x_{t+\Delta t}) = \left(\frac{1}{s_t^2} + \frac{a^2}{r^2}\right)^{-1} = g^2(t)\Delta t + \mathcal{O}(\Delta t^2),$$

which tells us the reverse step has the same diffusion scale  $g(t)$  as the forward step. For the mean, expand the ratio  $B/A$  to first order:

$$\mathbb{E}[x_t|x_{t+\Delta t}] = x_{t+\Delta t} + \Delta t \left[ -\left(f(t) + \frac{g^2(t)}{s_t^2}\right)x_{t+\Delta t} + \frac{g^2(t)}{s_t^2}m_t \right] + \mathcal{O}(\Delta t^2).$$

Putting the mean and variance together yields the one-step reverse transition kernel

$$x_t | x_{t+\Delta t} \sim \mathcal{N}\left(x_{t+\Delta t} + \Delta t \left[-\left(f + \frac{g^2}{s_t^2}\right)x_{t+\Delta t} + \frac{g^2}{s_t^2}m_t\right], g^2\Delta t\right) + \mathcal{O}(\Delta t^2).$$

This is recognized as the Euler–Maruyama update, run backward from  $t + \Delta t$  to  $t$ :

$$x_t - x_{t+\Delta t} = \Delta t \left[-\left(f + \frac{g^2}{s_t^2}\right)x_{t+\Delta t} + \frac{g^2}{s_t^2}m_t\right] + g\sqrt{\Delta t}\epsilon + O(\Delta t^2).$$

Letting  $\Delta t \rightarrow 0$  gives the SDE on the original clock (time decreasing along the path)

$$dx(t) = \left[-\left(f(t) + \frac{g^2(t)}{s_t^2}\right)x(t) + \frac{g^2(t)}{s_t^2}m_t\right] dt + g(t) d\bar{w}_t.$$

This drift can be written with the score because for a Gaussian marginal  $p_t = \mathcal{N}(m_t, s_t^2)$ ,

$$\partial_x \log p_t(x) = -\frac{x - m_t}{s_t^2} \implies -\left(f + \frac{g^2}{s_t^2}\right)x + \frac{g^2}{s_t^2}m_t = -fx + g^2\partial_x \log p_t(x).$$

To express the conventional *forward-in-t* reverse-time parametrization, define the reversed process  $\bar{x}(t) := x(T - t)$  (so that we now evolve forward in  $t$ ). The time flip turns the drift into

$$d\bar{x}(t) = \left[f(t)\bar{x}(t) - g^2(t)\partial_x \log p_t(\bar{x}(t))\right] dt + g(t) d\bar{w}_t,$$

where  $\bar{x}(T) \sim p_{\text{prior}} \approx p_T$ . This is exactly the conventional reverse-time SDE. In vector form this matches the general Equation (4.1.6) with  $\nabla_{\mathbf{x}} \log p_t$  in place of the 1D derivative.

**Exact Computation of PF–ODE with a Gaussian.** When the data distribution is assumed to be Gaussian, we can also directly derive the PF-ODE formula, avoiding heavy machinery such as the Fokker–Planck equation. In the end, we will see that the marginal densities of the PF-ODE coincide with those of both the forward SDE and the reverse-time SDE, providing a constructive verification of the Fokker–Planck theory to be discussed in Section 4.1.5.

Assume  $x_t \sim \mathcal{N}(m_t, s_t^2)$  at time  $t$ . A small deterministic step of size  $\Delta t$  can be written as a smooth map

$$x_{t+\Delta t} = \Phi_{t,\Delta t}(x_t) = x_t + \Delta t v_t(x_t) + \mathcal{O}(\Delta t^2),$$

which is simply the first-order Taylor expansion in  $\Delta t$ . Our goal is to see what form  $v_t$  must take so that, whenever the input is Gaussian, the output remains Gaussian.

To this end, expand  $v_t$  around the current mean  $m_t$ :

$$v_t(x) = v_t(m_t) + v'_t(m_t)(x - m_t) + \frac{1}{2}v''_t(m_t)(x - m_t)^2 + \dots$$

Now set  $y := x_t - m_t$ , so that  $y \sim \mathcal{N}(0, s_t^2)$ . Next, center the output by subtracting its mean (to first order in  $\Delta t$ ):

$$z := x_{t+\Delta t} - \mathbb{E}[x_{t+\Delta t}] = y + \Delta t \left( v'_t(m_t)y + \frac{1}{2}v''_t(m_t)(y^2 - s_t^2) \right) + \mathcal{O}(\Delta t^2).$$

At this point, recall that a Gaussian has zero skewness; in other words, its third centered moment is zero. Therefore, computing  $\mathbb{E}[z^3]$  to first order and using  $\mathbb{E}[y] = 0$ ,  $\mathbb{E}[y^2] = s_t^2$ ,  $\mathbb{E}[y^3] = 0$ ,  $\mathbb{E}[y^4] = 3s_t^4$ , we obtain

$$\mathbb{E}[z^3] = 3\Delta t \cdot \frac{1}{2}v''_t(m_t)(\mathbb{E}[y^4] - s_t^2\mathbb{E}[y^2]) + \mathcal{O}(\Delta t^2) = 3\Delta t v''_t(m_t)s_t^4 + \mathcal{O}(\Delta t^2).$$

For the output to stay Gaussian for all small  $\Delta t$ , this quantity must vanish at order  $\Delta t$ , which forces  $v''_t(m_t) = 0$ . Repeating the same argument for higher derivatives rules out higher powers as well. Consequently,  $v_t$  must be linear plus a shift:

$$v_t(x) = a_t x + b_t.$$

Plugging this back into the step gives

$$x_{t+\Delta t} = (1 + \alpha_t \Delta t)x_t + \beta_t \Delta t + \mathcal{O}(\Delta t^2), \quad \alpha_t := a_t, \quad \beta_t := b_t.$$

We now push  $x_t \sim \mathcal{N}(m_t, s_t^2)$  through this map and track mean and variance to first order:

$$\begin{aligned} \mathbb{E}[x_{t+\Delta t}] &= m_t + \Delta t(\alpha_t m_t + \beta_t) + \mathcal{O}(\Delta t^2), \\ \text{Var}(x_{t+\Delta t}) &= s_t^2 + \Delta t(2\alpha_t s_t^2) + \mathcal{O}(\Delta t^2). \end{aligned}$$

On the other hand, the forward SDE  $dx = f(t)x dt + g(t) dw_t$  has the elementary moment formulas (see Equation (4.3.3)):

$$m'_t = f(t)m_t, \quad (s_t^2)' = 2f(t)s_t^2 + g^2(t).$$

Matching the coefficients of  $\Delta t$  gives

$$\alpha_t = f(t) + \frac{g^2(t)}{2s_t^2}, \quad \beta_t = -\frac{g^2(t)}{2s_t^2}m_t.$$

With these choices, the step becomes

$$x_{t+\Delta t} = x_t + \Delta t \left[ \left( f(t) + \frac{g^2(t)}{2s_t^2} \right) x_t - \frac{g^2(t)}{2s_t^2} m_t \right] + \mathcal{O}(\Delta t^2).$$

Since for a Gaussian  $p_t = \mathcal{N}(m_t, s_t^2)$  we have  $\partial_x \log p_t(x) = -(x - m_t)/s_t^2$ , we can rewrite the bracket as  $f(t)x_t - \frac{1}{2}g^2(t)\partial_x \log p_t(x_t)$ . Therefore,

$$x_{t+\Delta t} = x_t + \Delta t \left[ f(t)x_t - \frac{1}{2}g^2(t)\partial_x \log p_t(x_t) \right] + \mathcal{O}(\Delta t^2).$$

Finally, dividing by  $\Delta t$  and letting  $\Delta t \rightarrow 0$  yields the PF-ODE

$$x'(t) = f(t)x(t) - \frac{1}{2}g^2(t)\partial_x \log p_t(x(t)).$$

To see why this ODE has the same marginals as the forward SDE (and the reverse-time SDE), observe that the drift above is linear plus a shift. Thus  $x(t)$  depends affinely on  $x(0)$ , and affine maps send Gaussians to Gaussians. Moreover, the mean  $m_t$  and variance  $s_t^2$  along this ODE satisfy exactly the same two scalar ODEs as the forward SDE (by our matching), with the same initial values. Hence  $p_t = \mathcal{N}(m_t, s_t^2)$  is identical for both evolutions at every time  $t$ .

## 4.2 Score SDE: Its Training and Sampling

### 4.2.1 Training

Building on the philosophy as in Chapter 3, we approximate the oracle score  $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$  using a time-conditioned neural network

$$\mathbf{s}_\phi = \mathbf{s}_\phi(\mathbf{x}, t)$$

across all  $t \in [0, T]$ , by minimizing a score-matching objective as in Equation (3.2.1):

$$\mathcal{L}_{\text{SM}}(\phi; \omega(\cdot)) := \frac{1}{2} \mathbb{E}_{t \sim p_{\text{time}}} \mathbb{E}_{\mathbf{x}_t \sim p_t} [\omega(t) \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\|_2^2],$$

where  $p_{\text{time}}$  is some time distribution (e.g., uniform on  $[0, T]$ ),  $\omega(\cdot)$  is a time-weighting function.

To avoid relying on the intractable oracle score  $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ , the DSM loss in Equation (3.3.2) is employed. Conditioned on a data point  $\mathbf{x}_0$ , this approach allows the use of the analytically tractable score  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)$  via Equation (D.2.4), with concrete examples given in Section 4.3. Specifically, we exploit the following loss function:

$$\begin{aligned} \mathcal{L}_{\text{DSM}}(\phi; \omega(\cdot)) &:= \\ \frac{1}{2} \mathbb{E}_t \mathbb{E}_{\mathbf{x}_0} \mathbb{E}_{p_t(\mathbf{x}_t | \mathbf{x}_0)} &[\omega(t) \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)\|_2^2], \end{aligned} \tag{4.2.1}$$

where  $\mathbf{x}_0 \sim p_{\text{data}}$ . Equation (4.2.1) can be interpreted as the continuous-time counterpart of Equation (3.4.1), with the summation in the discrete case replaced by integration.

Similar to the result in Theorem 3.3.1, the minimizer of Equation (4.2.1) is uniquely determined as follows:

#### Proposition 4.2.1: Minimizer of DSM

The minimizer  $\mathbf{s}^*$  satisfies

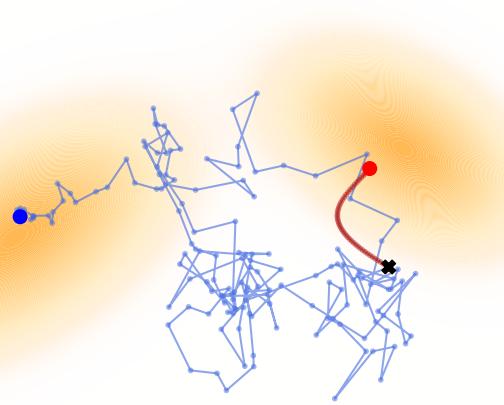
$$\mathbf{s}^*(\mathbf{x}_t, t) = \mathbb{E}_{\mathbf{x}_0 \sim p(\mathbf{x}_0 | \mathbf{x}_t)} [\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)] = \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t), \tag{4.2.2}$$

for almost every  $\mathbf{x}_t \sim p_t$  and  $t \in [0, T]$ .

#### Proof for Proposition.

DSM objective can be understood as a least-squares error problem. Specifically, at each time  $t$ , the optimal score function is given by the conditional expectation of the gradient of the log conditional density, which, under Bayes' rule, is equivalent to the gradient of the log marginal density. For a detailed proof, see Appendix D.2.6. ■

#### 4.2.2 Sampling and Inference



**Figure 4.5: (2D) Illustration of sampling from the Score SDE.** Sampling is by solving the reverse-time SDE (blue; via Equation (4.2.4)) and the PF-ODE (red; via Equation (4.2.6)) for the same forward SDE setup as in Figure 4.4. Starting from a random point  $\mathbf{x}_T \sim p_{\text{prior}}$  (dark “ $\times$ ”), both trajectories terminate near the support of  $p_{\text{data}}$  at  $t = 0$ .

After learning

$$\mathbf{s}_{\phi^\times} := \mathbf{s}_{\phi^\times}(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x}),$$

we replace the intractable oracle score  $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$  in the reverse-time SDE (Equation (4.1.6)) and PF-ODE (Equation (4.1.7)) with the learned proxy  $\mathbf{s}_{\phi^\times}(\mathbf{x}, t)$ . This substitution enables tractable inference via either the SDE or the ODE. For clarity, we distinguish the resulting processes as  $\mathbf{x}_{\phi^\times}^{\text{SDE}}(t)$  and  $\mathbf{x}_{\phi^\times}^{\text{ODE}}(t)$ , respectively, but will omit this distinction in later sections.<sup>8</sup>

**Empirical Reverse-Time SDE.** By substituting the trained score model  $\mathbf{s}_{\phi^\times}$  for the true score in Equation (4.1.6), we obtain the parameterized reverse-time

---

<sup>8</sup>This is to simplify notation after this subsection.

SDE used for generation:

$$dx_{\phi^\times}^{\text{SDE}}(t) = \left[ \mathbf{f}(\mathbf{x}_{\phi^\times}^{\text{SDE}}(t), t) - g^2(t) \mathbf{s}_{\phi^\times}(\mathbf{x}_{\phi^\times}^{\text{SDE}}(t), t) \right] dt + g(t) d\bar{\mathbf{w}}(t). \quad (4.2.3)$$

To generate a sample, we first draw an initial value  $\mathbf{x}_T$  from the prior distribution  $p_{\text{prior}}$  and then numerically solve Equation (4.2.3) backward in time from  $t = T$  to  $t = 0$ . A standard numerical solver for this is the Euler–Maruyama method, which provides the discrete update rule:

$$\mathbf{x}_{t-\Delta t} \leftarrow \mathbf{x}_t - \left[ \mathbf{f}(\mathbf{x}_t, t) - g^2(t) \mathbf{s}_{\phi^\times}(\mathbf{x}_t, t) \right] \Delta t + g(t) \sqrt{\Delta t} \cdot \boldsymbol{\epsilon}, \quad (4.2.4)$$

where  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  and  $\Delta t > 0$  is the step size.

Iterating this update rule yields a final sample  $\mathbf{x}_{\phi^\times}^{\text{SDE}}(0)$ . If the score model is accurate, the distribution of these generated samples, denoted  $p_{\phi^\times}^{\text{SDE}}(\cdot; 0)$ , provides a close approximation to the true data distribution<sup>9</sup>:

$$p_{\phi^\times}^{\text{SDE}}(\cdot; 0) \approx p_{\text{data}}(\cdot).$$

Indeed, the DDPM sampling scheme presented in Equation (2.2.14) is a special case of this Euler–Maruyama discretization applied to specific choices of  $\mathbf{f}$  and  $g$  (see Section 4.3).

**Empirical PF-ODE.** The PF-ODE defines a continuous flow connecting  $p_{\text{prior}}$  and  $p_{\text{data}}$ , enabling sampling, encoding, and exact likelihood evaluation. The following section provides further details on each of these operations.

**I. Sampling with PF-ODE.** Replacing the oracle score in Equation (4.1.7) with  $\mathbf{s}_{\phi^\times}$  yields the empirical PF-ODE:

$$\frac{d}{dt} \mathbf{x}_{\phi^\times}^{\text{ODE}}(t) = \mathbf{f}(\mathbf{x}_{\phi^\times}^{\text{ODE}}(t), t) - \frac{1}{2} g^2(t) \mathbf{s}_{\phi^\times}(\mathbf{x}_{\phi^\times}^{\text{ODE}}(t), t). \quad (4.2.5)$$

To generate samples, we begin by drawing an initial sample  $\mathbf{x}_T$  from the prior distribution,  $p_{\text{prior}}$ . We then numerically solve the PF-ODE from Equation (4.2.5) backward in time from  $t = T$  to  $t = 0$ . This process is equivalent to approximating the integral:

$$\mathbf{x}_{\phi^\times}^{\text{ODE}}(0) = \mathbf{x}_T + \int_T^0 \left[ \mathbf{f}(\mathbf{x}_{\phi^\times}^{\text{ODE}}(\tau), \tau) - \frac{1}{2} g^2(\tau) \mathbf{s}_{\phi^\times}(\mathbf{x}_{\phi^\times}^{\text{ODE}}(\tau), \tau) \right] d\tau.$$

---

<sup>9</sup>Theoretically, estimation accuracy depends on the discrepancy between  $p_T$  and  $p_{\text{prior}}$  (typically negligible), model training error, and numerical discretization error (De Bortoli, 2022). We do not pursue formal bounds here.

Solving this integral yields a final sample,  $\mathbf{x}_{\phi^\times}^{\text{ODE}}(0)$ . The distribution of samples generated via this deterministic process, denoted  $p_{\phi^\times}^{\text{ODE}}(\cdot; 0)$ , provides an approximation to the data distribution, such that  $p_{\phi^\times}^{\text{ODE}}(\cdot; 0) \approx p_{\text{data}}$ .

Let  $\Delta t > 0$  denote a discretization step size. A standard numerical integration approach is the Euler method, which estimates

$$\mathbf{f}(\mathbf{x}_\tau, \tau) - \frac{1}{2}g^2(\tau)\mathbf{s}_{\phi^\times}(\mathbf{x}_\tau, \tau) \approx \mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2}g^2(t)\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t), \quad \tau \in [t - \Delta t, t],$$

leading to the following update rule:

$$\mathbf{x}_{t-\Delta t} \leftarrow \mathbf{x}_t - \left[ \mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2}g^2(t)\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t) \right] \Delta t. \quad (4.2.6)$$

This connection allows us to reframe the process of generation with the following core insight:

#### Insight 4.2.1: Generation $\Leftrightarrow$ ODE/SDE Solving

Sampling from diffusion models is fundamentally equivalent to solving a corresponding probability flow ODE or a reverse-time SDE.

This equivalence provides a clear explanation for the slow sampling speeds of diffusion models, as raised in Question 3.5.1. Generation is computationally intensive because numerical solvers for these differential equations are inherently iterative, often requiring many steps to accurately approximate a solution trajectory<sup>10</sup>. However, the PF-ODE formulation is also advantageous, as it allows us to leverage the extensive literature on accelerated numerical solvers. Exploring these techniques to speed up diffusion model sampling is the primary focus of Chapter 9.

**II. Inversion with PF-ODE.** As discussed, unlike in the case of SDEs, we can solve the same Equation (4.2.5) both forward (from 0 to  $T$ ) and reverse (from  $T$  to 0) in time. When solving it forward, the ODE flow maps data to its (noisy) latent representations across all  $t \in [0, T]$ , which plays a role of an encoder. This concept enables powerful applications for controllable generation, such as image translation and editing and beyond (Mokady *et al.*, 2023; Su *et al.*, 2022).

---

<sup>10</sup>For example, DDPM and Score SDE typically use 1,000 function evaluations for generation.

**III. Exact Log-Likelihood Computation via PF-ODE.** We reinterpret the dynamics in Equation (4.2.5) as a Neural ODE (Chen *et al.*, 2018) variant (introduced in Section 5.1.2) that parameterizes only the score function, rather than the full velocity field. This PF-ODE formulation enables exact log-likelihood computation via the change-of-variables formula.

Applying the identity from Equation (5.1.9) to the PF-ODE in Equation (4.2.5), we define the velocity field as

$$\mathbf{v}_{\phi^\times}(\mathbf{x}, t) := \mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g^2(t)\mathbf{s}_{\phi^\times}(\mathbf{x}, t),$$

with the learned score  $\mathbf{s}_{\phi^\times}$ . The time evolution of the log-density  $p_{\phi^\times}^{\text{ODE}}(\cdot; t)$  along the PF-ODE trajectory  $\{\mathbf{x}_{\phi^\times}^{\text{ODE}}(t)\}_{t \in [0, T]}$  satisfies

$$\frac{d}{dt} \log p_{\phi^\times}^{\text{ODE}}(\mathbf{x}_{\phi^\times}^{\text{ODE}}(t), t) = -\nabla \cdot \mathbf{v}_{\phi^\times}(\mathbf{x}_{\phi^\times}^{\text{ODE}}(t), t),$$

where  $\nabla \cdot \mathbf{v}$  denotes the divergence in  $\mathbf{x}$ .

To evaluate the likelihood of a data point  $\mathbf{x}_0 \sim p_{\text{data}}$ , we integrate the following augmented ODE system from  $t = 0$  to  $t = T$ :

$$\frac{d}{dt} \begin{bmatrix} \mathbf{x}(t) \\ \delta(t) \end{bmatrix} = \begin{bmatrix} \mathbf{v}_{\phi^\times}(\mathbf{x}(t), t) \\ \nabla \cdot \mathbf{v}_{\phi^\times}(\mathbf{x}(t), t) \end{bmatrix}, \quad \begin{bmatrix} \mathbf{x}(0) \\ \delta(0) \end{bmatrix} = \begin{bmatrix} \mathbf{x}_0 \\ 0 \end{bmatrix}, \quad (4.2.7)$$

where  $\delta(t)$  accumulates the log-density change over time. Upon solving the system up to  $t = T$ , we obtain the terminal state:

$$\begin{bmatrix} \mathbf{x}(T) \\ \delta(T) \end{bmatrix}.$$

The log-likelihood of the original sample  $\mathbf{x}_0$  under the model can then be evaluated as

$$\log p_{\phi^\times}^{\text{ODE}}(\mathbf{x}_0; 0) = \log p_{\text{prior}}(\mathbf{x}(T)) + \delta(T),$$

where  $p_{\text{prior}}(\mathbf{x}(T))$  denotes the closed-form prior density evaluated at  $\mathbf{x}(T)$ .

### 4.3 Instantiations of SDEs

Song *et al.* (2020c) categorize the drift term  $\mathbf{f}(\mathbf{x}, t)$  and the diffusion term  $g(t)$  in the forward SDE into three types based on the behavior of the variance during evolution. Here, we focus on two commonly used types: the *Variance Explosion* (VE) SDE and *Variance Preserving* (VP) SDE. While it is possible to design custom noise schedulers, their design can substantially influence empirical performance. Table 4.1 summarizes these two SDE instantiations.

**Table 4.1:** Summary of the forward SDEs

	<b>VE SDE</b>	<b>VP SDE</b>
$\mathbf{f}(\mathbf{x}, t)$	<b>0</b>	$-\frac{1}{2}\beta(t)\mathbf{x}$
$g(t)$	$\sqrt{\frac{d\sigma^2(t)}{dt}}$	$\sqrt{\beta(t)}$
SDE	$d\mathbf{x}(t) = g(t) d\mathbf{w}(t)$	$d\mathbf{x}(t) = -\frac{1}{2}\beta(t)\mathbf{x}(t) dt + \sqrt{\beta(t)} d\mathbf{w}(t)$
$p_t(\mathbf{x}_t   \mathbf{x}_0)$	$\mathcal{N}(\mathbf{x}_t; \mathbf{x}_0, (\sigma^2(t) - \sigma^2(0)) \mathbf{I})$	$\mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_0 e^{-\frac{1}{2} \int_0^t \beta(\tau) d\tau}, \mathbf{I} - \mathbf{I} e^{-\int_0^t \beta(\tau) d\tau}\right)$
$p_{\text{prior}}$	$\mathcal{N}(\mathbf{0}, \sigma^2(T) \mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$

#### 4.3.1 VE SDE

VE SDE has the following components:

- **Drift Term:** A zero drift term  $\mathbf{f} = \mathbf{0}$ .
- **Diffusion Term:**  $g(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}$  for some function  $\sigma(t)$ .

The forward SDE then takes the following form:

$$d\mathbf{x}(t) = \sqrt{\frac{d\sigma^2(t)}{dt}} d\mathbf{w}(t). \quad (4.3.1)$$

Similarly, the results from Section 4.3.3 imply the perturbation kernel for the VE SDE and suggest selecting an appropriate prior distribution:

- **Perturbation Kernel:**

$$p_t(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_0, \left(\sigma^2(t) - \sigma^2(0)\right) \mathbf{I}\right)$$

- **Prior Distribution:** Assume that  $\sigma(t)$  is an increasing function for  $t \in [0, T]$  and that  $\sigma^2(T) \gg \sigma^2(0)$ . The prior distribution is given by:

$$p_{\text{prior}} := \mathcal{N}(\mathbf{0}, \sigma^2(T) \mathbf{I}).$$

A typical instance of a VE SDE is NCSN with the following design:

$$\sigma(t) := \sigma_{\min} \left( \frac{\sigma_{\max}}{\sigma_{\min}} \right)^t, \quad \text{for } t \in (0, 1],$$

where  $\sigma_{\min}$  and  $\sigma_{\max}$  are pre-specified constants. Namely, the sequence of variances is designed as a geometric sequence. With this, NCSN is viewed as a discretized version of VE SDE, as discussed in Section 4.1.1.

### 4.3.2 VP SDE

Let  $\beta: [0, T] \rightarrow \mathbb{R}_{\geq 0}$  be a non-negative function of  $t$ . A VP SDE is defined with the following components:

- **Drift Term:** A linear drift given by  $\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}$ .
- **Diffusion Term:**  $g(t) = \sqrt{\beta(t)}$ .

Thus, the forward SDE is expressed as:

$$d\mathbf{x}(t) = -\frac{1}{2}\beta(t)\mathbf{x}(t) dt + \sqrt{\beta(t)} d\mathbf{w}(t). \quad (4.3.2)$$

Using the results from Section 4.3.3, we can derive the perturbation kernel for the VP SDE and select an appropriate prior distribution:

- **Perturbation Kernel:**

$$p_t(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N} \left( \mathbf{x}_t; \mathbf{x}_0 e^{-\frac{1}{2} \int_0^t \beta(\tau) d\tau}, \mathbf{I} - \mathbf{I} e^{-\int_0^t \beta(\tau) d\tau} \right).$$

- **Prior Distribution:**  $p_{\text{prior}} := \mathcal{N}(\mathbf{0}, \mathbf{I})$ .

We remark that since the perturbation kernel is a Gaussian with a known mean and covariance, we can apply Equation (D.2.5) to compute its score function.

A classic example of a VP SDE is the DDPM, where the noise schedule  $\beta(t)$  is defined as:

$$\beta(t) := \beta_{\min} + t(\beta_{\max} - \beta_{\min}), \quad \text{for all } t \in [0, 1].$$

Here,  $\beta_{\min}$  and  $\beta_{\max}$  are pre-defined constants. With this, DDPM can be interpreted as a discretization of the VP SDE, as discussed in Section 4.1.1.

### 4.3.3 (Optional) How Is the Perturbation Kernel $p_t(\mathbf{x}_t|\mathbf{x}_0)$ Derived?

If the drift term in the forward SDE Equation (4.1.3) is linear in  $\mathbf{x}$ , taking the form

$$\mathbf{f}(\mathbf{x}, t) = f(t)\mathbf{x},$$

for some scalar-valued, time-dependent function  $f(t) \in \mathbb{R}$ , then Equation (4.1.3) becomes a linear SDE:

$$d\mathbf{x}(t) = f(t)\mathbf{x}(t) dt + g(t) dw(t).$$

Even if the initial distribution  $p_{\text{data}}$  is non-Gaussian, the linearity of the drift ensures that the conditional process remains Gaussian. In particular, for  $t > 0$ , the transition kernel admits the form:

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \mathbf{m}(t), P(t)\mathbf{I}_D),$$

where  $\mathbf{x}_0 \sim p_{\text{data}}$ , and  $\mathbf{m}(t) \in \mathbb{R}^D$ ,  $P(t) \in \mathbb{R}_{\geq 0}$  denote the conditional mean and (scalar) variance given  $\mathbf{x}_0$ , defined as:

$$\mathbf{m}(t) = \mathbb{E}[\mathbf{x}_t | \mathbf{x}(0) = \mathbf{x}_0], \quad P(t)\mathbf{I}_D = \text{Cov}[\mathbf{x}_t | \mathbf{x}(0) = \mathbf{x}_0].$$

These first and second moments evolve according to the following ODEs (Särkkä and Solin, 2019):

$$\begin{aligned} \frac{d\mathbf{m}(t)}{dt} &= f(t)\mathbf{m}(t), \\ \frac{dP(t)}{dt} &= 2f(t)P(t) + g^2(t), \end{aligned} \tag{4.3.3}$$

provided that the initial mean  $\mathbf{m}(0)$  and variance  $P(0)$  are finite.

Since both ODEs are linear, they admit closed-form solutions via the integrating factor method. Given the initial condition  $\mathbf{x}_0$ , the mean and variance evolve as

$$\mathbf{m}(t) = \mathcal{E}(0 \rightarrow t)\mathbf{x}_0, \quad P(t) = \int_0^t \mathcal{E}^2(s \rightarrow t)g(s)^2 ds, \tag{4.3.4}$$

with  $\mathbf{m}(0) = \mathbf{x}_0$  and  $P(0) = 0$ . Here  $\mathcal{E}(s \rightarrow t)$  denotes the exponential integrating factor

$$\mathcal{E}(s \rightarrow t) := \exp\left(\int_s^t f(u) du\right),$$

which captures the accumulated effect of the drift from time  $s$  to  $t$ . Consequently, the transition kernel  $p_t(\mathbf{x}_t|\mathbf{x}_0)$  also admits a closed-form expression.

We defer the justification that the conditional covariance of  $p_t(\mathbf{x}_t | \mathbf{x}_0)$  is isotropic, that is  $\text{Cov}[\mathbf{x}_t | \mathbf{x}_0] = P(t)\mathbf{I}_D$  under a  $D$ -dimensional Wiener process with independent coordinates and diffusion  $g(t)\mathbf{I}_D$ , as well as the derivation of Equation (4.3.3), to Section C.1.5, which relies on Itô calculus.

### Example: VE SDE's Transition Kernel

In the special case of VE SDE:  $\mathbf{f} \equiv 0$  and  $g(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}$ , the mean and covariance of the solution to the SDE evolve as follows.

#### Mean.

$$\frac{d\mathbf{m}(t)}{dt} = 0, \quad \text{with } \mathbf{m}(0) = \mathbf{x}_0 \implies \mathbf{m}(t) = \mathbf{x}_0.$$

#### Variance.

$$\frac{dP(t)}{dt} = \frac{d\sigma^2(t)}{dt}, \quad \text{with } P(0) = 0 \implies P(t) = \sigma^2(t) - \sigma^2(0).$$

Therefore

$$p_t(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_0, (\sigma^2(t) - \sigma^2(0))\mathbf{I}_D\right).$$



### Example: VP SDE's Transition Kernel

In the VP SDE case with drift  $\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}$  and diffusion  $g(t) = \sqrt{\beta(t)}$ :

#### Mean $\mathbf{m}(t)$ .

$$\frac{d\mathbf{m}}{dt} = -\frac{1}{2}\beta(t)\mathbf{m}(t), \quad B(t) := \int_0^t \beta(s) ds, \quad \mathbf{m}(t) = e^{-\frac{1}{2}B(t)}\mathbf{x}_0.$$

**Variance  $P(t)$ .** The variance satisfies

$$\frac{dP}{dt} = -\beta(t)P(t) + \beta(t).$$

Applying the integrating factor  $e^{B(t)}$  with  $B(t) = \int_0^t \beta(s) ds$ , we obtain

$$\frac{d}{dt} \left[ P(t)e^{B(t)} \right] = \beta(t)e^{B(t)}.$$

Integrating both sides gives

$$P(t) = 1 - e^{-B(t)}.$$

Hence the covariance is isotropic with

$$\mathbf{P}(t) = P(t)\mathbf{I}_D = (1 - e^{-B(t)})\mathbf{I}_D.$$

**Final Closed-Form Transition Kernel.**

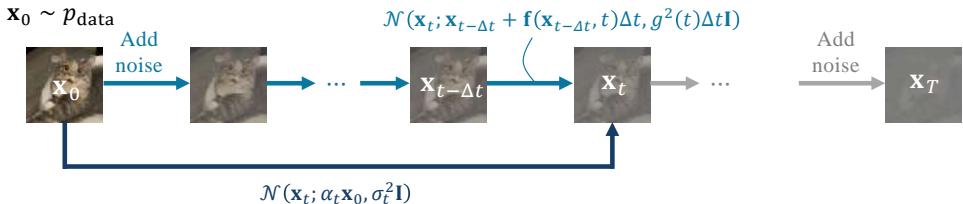
$$p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N} \left( \mathbf{x}_t; \underbrace{e^{-\frac{1}{2}B(t)}\mathbf{x}_0}_{\mathbf{m}(t)}, \underbrace{(1 - e^{-B(t)})\mathbf{I}_D}_{P(t)\mathbf{I}_D} \right), \quad B(t) = \int_0^t \beta(s) \, ds.$$



## 4.4 (Optional) Rethinking Forward Kernels in Score-Based and Variational Diffusion Models

DDPM and Score SDE are typically introduced via the forward transition kernel  $p(\mathbf{x}_t|\mathbf{x}_{t-\Delta t})$ , discretely defined in DDPM and as a continuous-time SDE in Score SDE. However, what is most relevant in practice, especially in their loss functions (Equations (2.2.8) and (4.2.1)), is the accumulated transition kernel from the data,  $p_t(\mathbf{x}_t|\mathbf{x}_0)$ . Both frameworks ultimately rely on this kernel, either through recursive computation (DDPM) or by solving an ODE, as detailed in Section 4.3.3 (Score SDE).

In this section, we start by defining  $p_t(\mathbf{x}_t|\mathbf{x}_0)$  (in continuous time), which provides a neater and more direct perspective. Overall, while  $p(\mathbf{x}_t|\mathbf{x}_{t-\Delta t})$  and  $p_t(\mathbf{x}_t|\mathbf{x}_0)$  are theoretically equivalent, defining the latter often results in a cleaner and more interpretable formulation. In particular,  $p_t(\mathbf{x}_t|\mathbf{x}_0)$  offers direct insight into the prior as  $t \rightarrow T$ , and aligns naturally with the practical loss design.



**Figure 4.6: Illustration of Lemma 4.4.1.** Incremental noise injection via continuous-time SDE ( $\Delta t \rightarrow 0$ ) and direct perturbation of Equation (4.4.1) are mathematically equivalent.

### 4.4.1 A General Affine Forward Process $p_t(\mathbf{x}_t|\mathbf{x}_0)$

We begin with defining a general forward perturbation kernel:

$$p_t(\mathbf{x}_t|\mathbf{x}_0) := \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}), \quad (4.4.1)$$

where  $\mathbf{x}_0 \sim p_{\text{data}}$ , and  $\alpha_t$ ,  $\sigma_t$  are nonnegative scalar functions of  $t \in [0, T]$  satisfying:

- (i)  $\alpha_t > 0$  and  $\sigma_t > 0$  for all  $t \in (0, 1]$  (allowing  $\sigma_0 = 0$ ), and
- (ii) Typically,  $\alpha_0 = 1$  and  $\sigma_0 = 0$ .

That is,  $\mathbf{x}_t \sim p_t(\mathbf{x}_t|\mathbf{x}_0)$  can be sampled as

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

This framework subsumes several well-known instances, including the VE (e.g., NCSN), the VP (e.g., DDPM), and the Flow Matching (FM) forward kernel (Lipman *et al.*, 2022; Liu, 2022), which linearly interpolates between  $\mathbf{x}_0$  and  $\epsilon$  (see later in Section 5.2).

- **VE (NCSN) Kernel:**  $\alpha_t \equiv 1$ ,  $\sigma_T \gg 1$ ;
- **VP (DDPM) Kernel:**  $\alpha_t := \sqrt{1 - \sigma_t^2}$ , so that  $\alpha_t^2 + \sigma_t^2 = 1$ ;
- **FM Kernel:**  $\alpha_t = 1 - t$ ,  $\sigma_t = t$ .

#### 4.4.2 Connection to Score SDE

For Score SDE, specifying  $p_t(\mathbf{x}_t | \mathbf{x}_0)$  in a linear form naturally induces an SDE with affine coefficients, providing a more intuitive alternative to starting from drift and diffusion terms and solving ODEs for the moments (see Section 4.3.3).

Given the forward perturbation kernel in Equation (4.4.1), the corresponding forward SDE takes the linear-in- $\mathbf{x}$  form as in Equation (4.3.2):

$$d\mathbf{x}(t) = \underbrace{f(t)\mathbf{x}(t)}_{\mathbf{f}(\mathbf{x}(t), t)} dt + g(t) d\mathbf{w}(t),$$

where  $f, g: [0, T] \rightarrow \mathbb{R}$  are real-valued functions of time. The coefficients  $f(t)$  and  $g(t)$  can be expressed analytically in terms of  $\alpha_t$  and  $\sigma_t$ , as summarized in the following lemma.

**Lemma 4.4.1: Forward Perturbation Kernel  $\Leftrightarrow$  Linear SDE**

Define  $\lambda_t := \log \frac{\alpha_t}{\sigma_t}$  for  $t \in (0, T]$ . Given the forward perturbation kernel

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

the linear SDE

$$d\mathbf{x}(t) = f(t)\mathbf{x}(t) dt + g(t) d\mathbf{w}(t),$$

with coefficients

$$\begin{aligned} f(t) &= \frac{d}{dt} \log \alpha_t, \\ g^2(t) &= \frac{d\sigma_t^2}{dt} - 2 \frac{d}{dt} \log \alpha_t \sigma_t^2 = -2\sigma_t^2 \frac{d}{dt} \lambda_t, \end{aligned} \tag{4.4.2}$$

has the conditional transition  $p_t(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$  for all  $t \in (0, T]$ . Conversely, if a linear SDE has conditional transitions  $\mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$  so that  $\alpha_t > 0$  and  $\sigma_t > 0$  for all  $t \in (0, T]$ , then its coefficients satisfy Equation (4.4.2) for  $t \in (0, T]$ .

**Proof for Lemma.**

From Section 4.3.3, the proof matches the mean and covariance ODEs  $\mathbf{m}'(t) = f(t)\mathbf{m}(t)$ ,  $\mathbf{P}'(t) = 2f(t)\mathbf{P}(t) + g^2(t)\mathbf{I}$  with  $\mathbf{m}(t) = \alpha_t \mathbf{x}_0$  and  $\mathbf{P}(t) = \sigma_t^2 \mathbf{I}$  on  $(0, T]$ .

**Remark.**

To exactly match a Gaussian prior at the terminal time, the process must completely forget  $\mathbf{x}_0$  and attain the target variance; this requires  $\alpha_T = 0$  and  $\sigma_T^2$  equal to the prior variance.

In the SDE formulation, one has

$$\alpha_t = \exp \left( \int_0^t f(u) du \right).$$

Thus, enforcing  $\alpha_T = 0$  at a finite  $T$  forces

$$\int_0^T f(u) du = -\infty,$$

meaning the drift must contract infinitely fast as  $t \rightarrow T$ . At the same time, the diffusion must diverge in order to maintain the prescribed variance,

which is reflected by

$$g^2(t) = \sigma_t^{2t} - 2\frac{\alpha'_t}{\alpha_t}\sigma_t^2 \rightarrow \infty \quad \text{as } t \rightarrow T.$$

If  $f$  and  $g$  remain bounded on  $[0, T]$ , then necessarily  $\alpha_T > 0$  and a residual dependence on  $\mathbf{x}_0$  remains. In that case, the Gaussian prior is attained only asymptotically: either in the limit  $t \rightarrow T$  (without exact attainment) or exactly on an infinite horizon after an appropriate time reparameterization as  $T \rightarrow \infty$ .

From the above lemma, specifying the incremental noise injection via a linear SDE with coefficients  $f(t)$  and  $g(t)$  is mathematically equivalent to defining the perturbation kernel with parameters  $\alpha_t$  and  $\sigma_t$ . In the diffusion model literature, these two viewpoints are used interchangeably. Therefore, we conclude:

#### Observation 4.4.1:

Defining  $p_t(\mathbf{x}_t | \mathbf{x}_0)$  is equivalent to specifying the linear SDE coefficients  $f(t)$  and  $g(t)$ .

### 4.4.3 Connection to Variational-Based Diffusion Model

We revisit a core identity from DDPM, derived via Bayes' rule:

$$p(\mathbf{x}_{t-\Delta t} | \mathbf{x}_t, \mathbf{x}) = p(\mathbf{x}_t | \mathbf{x}_{t-\Delta t}) \cdot \frac{p_{t-\Delta t}(\mathbf{x}_{t-\Delta t} | \mathbf{x})}{p_t(\mathbf{x}_t | \mathbf{x})}, \quad (4.4.3)$$

for any  $\mathbf{x}$  (usually  $\mathbf{x} \sim p_{\text{data}}$ ). This reverse conditional  $p(\mathbf{x}_{t-\Delta t} | \mathbf{x}_t, \mathbf{x})$  is central to modeling, enabling both tractable training targets and efficient sampling.

Although DDPM typically defines the incremental kernel  $p(\mathbf{x}_t | \mathbf{x}_{t-\Delta t})$  first, the accumulated transition  $p_t(\mathbf{x}_t | \mathbf{x}_0)$  often provides a more interpretable and practical formulation, especially for the prior and loss design.

**Deriving Transition Kernels.** We now extend this to the continuous-time setting. Let  $0 \leq t < s \leq T$  be two (continuous) time points. Given the perturbation kernel  $p_t(\mathbf{x}_t | \mathbf{x}_0)$ , we can compute the reverse conditional  $p(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x})$  for any  $\mathbf{x}$  by applying Equation (4.4.3)<sup>11</sup>, using the forward kernel  $p(\mathbf{x}_s | \mathbf{x}_t)$  as an intermediate. The following lemma summarizes this derivation, extending Lemma 2.2.2 without assuming  $\alpha_t^2 + \sigma_t^2 = 1$ .

<sup>11</sup>This identity extends naturally to continuous time by treating  $s$  as a general earlier time.

**Lemma 4.4.2: Reverse Conditional Transition Kernels**

Let  $0 \leq t < s \leq T$ . The reverse *conditional* transition kernel is:

$$p(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}) = \mathcal{N} \left( \mathbf{x}_t; \boldsymbol{\mu}(\mathbf{x}_s, \mathbf{x}; s, t), \sigma^2(s, t) \mathbf{I} \right),$$

where

$$\boldsymbol{\mu}(\mathbf{x}_s, \mathbf{x}; s, t) := \frac{\alpha_{s|t} \sigma_t^2}{\sigma_s^2} \mathbf{x}_s + \frac{\alpha_t \sigma_{s|t}^2}{\sigma_s^2} \mathbf{x}, \quad \sigma^2(s, t) := \sigma_{s|t}^2 \frac{\sigma_t^2}{\sigma_s^2}. \quad (4.4.4)$$

Here,  $\alpha_{s|t}$  and  $\sigma_{s|t}$  are defined as:

$$\alpha_{s|t} := \frac{\alpha_s}{\alpha_t}, \quad \sigma_{s|t}^2 := \sigma_s^2 - \alpha_{s|t}^2 \sigma_t^2.$$

**Proof for Lemma.**

We first compute the forward transition kernel:

$$p(\mathbf{x}_s | \mathbf{x}_t) = \mathcal{N} \left( \mathbf{x}_s; \alpha_{s|t} \mathbf{x}_t, \sigma_{s|t}^2 \mathbf{I} \right). \quad (4.4.5)$$

The reverse kernel then follows from Bayes' rule, and since all involved distributions are Gaussian, the result can be derived by direct computation. For further details, see Appendix A of (Kingma *et al.*, 2021).

Although  $p(\mathbf{x}_{t+\Delta t} | \mathbf{x}_t)$  and  $p_t(\mathbf{x}_t | \mathbf{x}_0)$  are theoretically equivalent,  $p_t(\mathbf{x}_t | \mathbf{x}_0)$  often takes a more central role. The step-wise transition in Equation (4.4.5) mainly serves to obtain a closed-form reverse kernel. Recent works (Kingma *et al.*, 2021) thus favor directly specifying  $p_t(\mathbf{x}_t | \mathbf{x}_0)$  for its clarity and interpretability.

**Reverse Process Modeling, Training, and Sampling.** The training objective (ELBO in Equation (2.2.13)) and the modeling framework introduced in Section 2.2 remain applicable under our generalized setting. For clarity, we adopt the  $\mathbf{x}$ -prediction formulation, denoted by  $\mathbf{x}_\phi(\mathbf{x}_s, s)$ , following Kingma *et al.* (2021). However, the equivalent  $\epsilon$ -prediction perspective, represented by  $\epsilon_\phi(\mathbf{x}_s, s)$ , is also valid due to the relationship (as in Equation (2.2.12))

$$\mathbf{x}_s = \alpha_s \mathbf{x}_\phi(\mathbf{x}_s, s) + \sigma_s \epsilon_\phi(\mathbf{x}_s, s), \quad \text{for any given } \mathbf{x}_s \sim q_s.$$

**Modeling and Diffusion Loss  $\mathcal{L}_{\text{diffusion}}$ .** Similar to DDPM, the conditional distribution  $p(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x})$  in Equation (4.4.4) motivates replacing the clean signal

$\mathbf{x}$  with a learnable predictor  $\mathbf{x}_\phi(\mathbf{x}_s, s)$ , yielding a parameterized reverse model of the form:

$$p_\phi(\mathbf{x}_t | \mathbf{x}_s) := \mathcal{N} \left( \mathbf{x}_t; \boldsymbol{\mu}_\phi(\mathbf{x}_s, s, t), \sigma^2(s, t) \mathbf{I} \right), \quad (4.4.6)$$

with the mean parametrized as:

$$\boldsymbol{\mu}_\phi(\mathbf{x}_s, s, t) = \frac{\alpha_{s|t}\sigma_t^2}{\sigma_s^2} \mathbf{x}_s + \frac{\alpha_t\sigma_{s|t}^2}{\sigma_s^2} \mathbf{x}_\phi(\mathbf{x}_s, s).$$

Given the forward kernel in Equation (4.4.1), the KL divergence in  $\mathcal{L}_{\text{diffusion}}(\mathbf{x}; \phi)$  reduces to a weighted regression loss:

$$\begin{aligned} \mathcal{D}_{\text{KL}}(p(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0) \| p_\phi(\mathbf{x}_t | \mathbf{x}_s)) &= \frac{1}{2\sigma^2(s, t)} \|\boldsymbol{\mu}(\mathbf{x}_s, \mathbf{x}_0; s, t) - \boldsymbol{\mu}_\phi(\mathbf{x}_s, s, t)\|_2^2 \\ &= \frac{1}{2} (\text{SNR}(t) - \text{SNR}(s)) \|\mathbf{x}_0 - \mathbf{x}_\phi(\mathbf{x}_s, s)\|_2^2, \end{aligned} \quad (4.4.7)$$

where  $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$ , with  $\mathbf{x}_0 \sim p_{\text{data}}$ ,  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , and  $\text{SNR}(s) := \alpha_s^2 / \sigma_s^2$  denotes the signal-to-noise ratio at time  $s$ .

### Remark.

In (Kingma *et al.*, 2021), the authors study the continuous-time limit of Equation (4.4.7) as  $t \rightarrow s$ , yielding:

$$\mathcal{L}_{\text{VDM}}^\infty(\mathbf{x}_0) = -\frac{1}{2} \mathbb{E}_{s, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \text{SNR}'(s) \|\mathbf{x}_0 - \mathbf{x}_\phi(\mathbf{x}_s, s)\|_2^2.$$

This setup also introduces a learnable noise schedule, and while it generalizes beyond continuous data, such extensions fall outside the scope of our current discussion.

**Sampling.** Sampling proceeds similarly to DDPM using the parameterized kernel from Equation (4.4.6):

$$\mathbf{x}_t = \underbrace{\frac{\alpha_{s|t}\sigma_t^2}{\sigma_s^2} \mathbf{x}_s + \frac{\alpha_t\sigma_{s|t}^2}{\sigma_s^2} \mathbf{x}_\phi(\mathbf{x}_s, s) + \sigma_{s|t} \frac{\sigma_t}{\sigma_s} \boldsymbol{\epsilon}_s}_{\boldsymbol{\mu}_\phi(\mathbf{x}_s, t, s)}, \quad \boldsymbol{\epsilon}_s \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (4.4.8)$$

## 4.5 (Optional) Fokker–Planck Equation and Reverse-Time SDEs via Marginalization and Bayes’ Rule

In this section, we offer a probabilistic perspective on the structure of the Fokker–Planck equation and the reverse-time SDE. By leveraging fundamental tools such as the marginalization trick and Bayes’ rule, we illuminate the connection between the statistical formulation of stochastic processes and their corresponding differential equations.

We emphasize that the “derivation” presented here is not mathematically rigorous proofs, but rather heuristic arguments intended to convey the underlying connections.

### 4.5.1 Fokker–Planck Equation from the Marginalization of Transition Kernels

Given the forward transition probability as in Equation (4.1.2)

$$p(\mathbf{x}_{t+\Delta t} | \mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_{t+\Delta t}; \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\Delta t, g^2(t)\Delta t \mathbf{I}\right),$$

and the marginal distributions

$$p_t(\mathbf{x}_t), \quad p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}),$$

we aim to derive the Fokker–Planck equation that governs the time evolution of the marginal distribution  $p_t$ .

**Change of Variables.** By the Markov property, the marginal distribution at time  $t + \Delta t$  can be expressed as an integral over the previous state  $\mathbf{x}_t$  (i.e., Chapman–Kolmogorov equation):

$$p_{t+\Delta t}(\mathbf{x}) = \int \mathcal{N}(\mathbf{x}; \mathbf{y} + \mathbf{f}(\mathbf{y}, t)\Delta t, g^2(t)\Delta t \mathbf{I}) p_t(\mathbf{y}) d\mathbf{y}.$$

We introduce a new variable

$$\mathbf{u} := \mathbf{y} + \mathbf{f}(\mathbf{y}, t)\Delta t,$$

so the Gaussian is centered at  $\mathbf{u}$ . For small  $\Delta t$ , this map is invertible with

$$\mathbf{y} = \mathbf{u} - \mathbf{f}(\mathbf{u}, t)\Delta t + \mathcal{O}(\Delta t^2), \quad \left| \det \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \right| = 1 - (\nabla_{\mathbf{u}} \cdot \mathbf{f})(\mathbf{u}, t)\Delta t + \mathcal{O}(\Delta t^2).$$

Hence, change-of-variable formula leads us to:

$$p_{t+\Delta t}(\mathbf{x}) = \int \mathcal{N}\left(\mathbf{x}; \mathbf{u}, g^2(t)\Delta t \mathbf{I}\right) \cdot \\ \left[ p_t(\mathbf{u}) - \Delta t \mathbf{f}(\mathbf{u}, t) \cdot \nabla_{\mathbf{u}} p_t(\mathbf{u}) - \Delta t (\nabla_{\mathbf{u}} \cdot \mathbf{f})(\mathbf{u}, t) p_t(\mathbf{u}) \right] d\mathbf{u} + \mathcal{O}(\Delta t^2),$$

**Taylor Expansion.** For any smooth function  $\phi : \mathbb{R}^D \rightarrow \mathbb{R}$  and scale  $\sigma > 0$ , if  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , the following approximation holds (known as the *Taylor–Gaussian smoothing formula*):

$$\int \mathcal{N}(\mathbf{x}; \mathbf{u}, \sigma^2 \mathbf{I}) \phi(\mathbf{u}) d\mathbf{u} = \mathbb{E}[\phi(\mathbf{x} + \sigma \mathbf{z})] = \phi(\mathbf{x}) + \frac{\sigma^2}{2} \Delta_{\mathbf{x}} \phi(\mathbf{x}) + \mathcal{O}(\sigma^4).$$

This is because Taylor expansion for:

$$\phi(\mathbf{x} + \sigma \mathbf{z}) = \phi(\mathbf{x}) + \sigma \nabla_{\mathbf{x}} \phi(\mathbf{x}) \cdot \mathbf{z} + \frac{\sigma^2}{2} \mathbf{z}^\top \nabla_{\mathbf{x}}^2 \phi(\mathbf{x}) \mathbf{z} + \mathcal{O}(\sigma^3)$$

and  $\mathbb{E}[\mathbf{z}] = \mathbf{0}$ ,  $\mathbb{E}[\mathbf{z} \mathbf{z}^\top] = \mathbf{I}$ .

Apply this with  $\phi = p_t$ ,  $\phi = \mathbf{f} \cdot \nabla_{\mathbf{u}} p_t$ , and  $\phi = (\nabla_{\mathbf{u}} \cdot \mathbf{f}) p_t$ , and use  $\sigma^2 = g^2(t) \Delta t$ , we can obtain

$$\begin{aligned} & p_{t+\Delta t}(\mathbf{x}) - p_t(\mathbf{x}) \\ &= -\Delta t \mathbf{f}(\mathbf{x}, t) \cdot \nabla_{\mathbf{x}} p_t(\mathbf{x}) - \Delta t (\nabla_{\mathbf{x}} \cdot \mathbf{f})(\mathbf{x}, t) p_t(\mathbf{x}) + \frac{g^2(t)}{2} \Delta t \Delta_{\mathbf{x}} p_t(\mathbf{x}) + \mathcal{O}(\Delta t^2) \\ &= -\Delta t \nabla_{\mathbf{x}} \cdot (\mathbf{f}(\mathbf{x}, t) p_t(\mathbf{x})) + \frac{g^2(t)}{2} \Delta t \Delta_{\mathbf{x}} p_t(\mathbf{x}) + \mathcal{O}(\Delta t^2). \end{aligned}$$

Divide by  $\Delta t$  and let  $\Delta t \rightarrow 0$  to obtain the Fokker–Planck equation.

In Section C.1.4, we present the Itô–based derivation to complement the discrete–time view above.

### 4.5.2 Why Does Reverse-Time SDE Take The Form?

The rigorous derivation of the reverse-time SDE is technical and requires delving into the properties of the Fokker-Planck equation. However, the form of the reverse-time SDE can be understood intuitively through Bayes' theorem. Here, we present a heuristic derivation to provide insight into why Equation (4.1.6) takes its form, with the appearance of score functions<sup>12</sup>.

**Using Bayes' Rule for Inversion.** Our goal is to determine the reverse-time transition kernel by first considering the discrete-time case:

$$p(\mathbf{x}_t | \mathbf{x}_{t+\Delta t}),$$

---

<sup>12</sup>This derivation is inspired by the approach in [this post](#).

and then taking  $\Delta t \rightarrow 0$  to obtain the continuous-time formulation. Using Bayes' theorem, we express:

$$\begin{aligned} p(\mathbf{x}_t | \mathbf{x}_{t+\Delta t}) &= p(\mathbf{x}_{t+\Delta t} | \mathbf{x}_t) \frac{p_t(\mathbf{x}_t)}{p_{t+\Delta t}(\mathbf{x}_{t+\Delta t})} \\ &= p(\mathbf{x}_{t+\Delta t} | \mathbf{x}_t) \exp (\log p_t(\mathbf{x}_t) - \log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t})) . \end{aligned} \quad (4.5.1)$$

The forward transition kernel is assumed to be as in Equation (4.1.2):

$$p(\mathbf{x}_{t+\Delta t} | \mathbf{x}_t) = \mathcal{N} \left( \mathbf{x}_{t+\Delta t}; \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t) \Delta t, g^2(t) \Delta t \mathbf{I} \right)$$

**Taylor Expansion.** To handle the exponential term, we apply a first-order Taylor expansion. The key insight is to expand around the point  $(\mathbf{x}_t, t)$  in both space and time:

$$\begin{aligned} \log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) &= \log p_t(\mathbf{x}_t) + \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t) \\ &\quad + \frac{\partial \log p_t(\mathbf{x}_t)}{\partial t} \Delta t + \mathcal{O}(\|\mathbf{h}\|_2^2) \end{aligned}$$

where  $\mathbf{h} := (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t, \Delta t)$ . Therefore:

$$\begin{aligned} \log p_t(\mathbf{x}_t) - \log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) &= -\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t) \\ &\quad - \frac{\partial \log p_t(\mathbf{x}_t)}{\partial t} \Delta t + \mathcal{O}(\|\mathbf{h}\|_2^2) \end{aligned} \quad (4.5.2)$$

For the forward process with finite drift and diffusion, we have  $\mathbb{E}[\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t\|_2^2] = \mathcal{O}(\Delta t)$ , which ensures that the remainder term is  $\mathcal{O}((\Delta t)^2)$  in expectation.

**Substituting into the Reverse Transition.** Substituting equations Equation (4.1.2) and Equation (4.5.2) into Equation (4.5.1):

$$\begin{aligned} p(\mathbf{x}_t | \mathbf{x}_{t+\Delta t}) &= \frac{1}{(2\pi g^2(t) \Delta t)^{D/2}} \exp \left( -\frac{\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - \mathbf{f}(\mathbf{x}_t, t) \Delta t\|_2^2}{2g^2(t) \Delta t} \right) \\ &\quad \cdot \exp \left( -\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t) - \frac{\partial \log p_t(\mathbf{x}_t)}{\partial t} \Delta t + \mathcal{O}((\Delta t)^2) \right) . \end{aligned}$$

**Algebraic Manipulation.** The key step is to complete the square in the exponent. We have:

$$\begin{aligned} &- \frac{\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - \mathbf{f}(\mathbf{x}_t, t) \Delta t\|_2^2}{2g^2(t) \Delta t} - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t) \\ &= - \frac{[\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - \mathbf{f}(\mathbf{x}_t, t) \Delta t\|_2^2 + 2g^2(t) \Delta t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t)]}{2g^2(t) \Delta t} \end{aligned}$$

Let  $\boldsymbol{\delta} := \mathbf{x}_{t+\Delta t} - \mathbf{x}_t$  and  $\boldsymbol{\mu} := \mathbf{f}(\mathbf{x}_t, t)\Delta t$ . Then:

$$\begin{aligned} & \|\boldsymbol{\delta} - \boldsymbol{\mu}\|_2^2 + 2g^2(t)\Delta t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot \boldsymbol{\delta} \\ &= \|\boldsymbol{\delta}\|_2^2 - 2\boldsymbol{\delta} \cdot \boldsymbol{\mu} + \|\boldsymbol{\mu}\|_2^2 + 2g^2(t)\Delta t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) \cdot \boldsymbol{\delta} \\ &= \|\boldsymbol{\delta}\|_2^2 - 2\boldsymbol{\delta} \cdot [\boldsymbol{\mu} - g^2(t)\Delta t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)] + \|\boldsymbol{\mu}\|_2^2 \\ &= \|\boldsymbol{\delta} - [\boldsymbol{\mu} - g^2(t)\Delta t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)]\|_2^2 - \|g^2(t)\Delta t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\|_2^2 \end{aligned}$$

Substituting back:

$$\begin{aligned} & \|\boldsymbol{\delta} - [\mathbf{f}(\mathbf{x}_t, t)\Delta t - g^2(t)\Delta t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)]\|_2^2 \\ &= \|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - [\mathbf{f}(\mathbf{x}_t, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)]\Delta t\|_2^2. \end{aligned}$$

Therefore,

$$\begin{aligned} & p(\mathbf{x}_t | \mathbf{x}_{t+\Delta t}) \\ &= \frac{1}{(2\pi g^2(t)\Delta t)^{D/2}} \\ &\quad \cdot \exp\left(-\frac{\|\mathbf{x}_{t+\Delta t} - \mathbf{x}_t - [\mathbf{f}(\mathbf{x}_t, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)]\Delta t\|_2^2}{2g^2(t)\Delta t}\right) \\ &\quad \cdot \exp(\mathcal{O}(\Delta t)) \\ &= \mathcal{N}\left(\mathbf{x}_t; \mathbf{x}_{t+\Delta t} - [\mathbf{f}(\mathbf{x}_t, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)]\Delta t, g^2(t)\Delta t \mathbf{I}\right) \\ &\quad \cdot (1 + \mathcal{O}(\Delta t)). \end{aligned}$$

The additional term  $\|g^2(t)\Delta t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\|_2^2$  from completing the square is  $\mathcal{O}((\Delta t)^2)$  and can be absorbed into the error term. Similarly, the time derivative term  $\frac{\partial \log p_t(\mathbf{x}_t)}{\partial t} \Delta t$  is  $\mathcal{O}(\Delta t)$  and will vanish in the continuous limit.

**Taking  $\Delta t \rightarrow 0$  Limit.** As  $\Delta t \approx 0$ , under smoothness assumptions, the following approximations hold:

$$\begin{aligned} \mathbf{f}(\mathbf{x}_t, t) &\approx \mathbf{f}(\mathbf{x}_{t+\Delta t}, t + \Delta t), \\ g(t) &\approx g(t + \Delta t), \\ \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t) &\approx \nabla_{\mathbf{x}} \log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) \\ &= \mathbf{s}(\mathbf{x}_{t+\Delta t}, t + \Delta t). \end{aligned}$$

Using these approximations and some rearrangements, we obtain:

$$\begin{aligned} & p(\mathbf{x}_t | \mathbf{x}_{t+\Delta t}) \\ & \approx \frac{1}{(2\pi g^2(t)\Delta t)^{D/2}} \exp \left( -\frac{\|\mathbf{x}_t - (\mathbf{x}_{t+\Delta t} - [\mathbf{f}(\mathbf{x}_{t+\Delta t}, t + \Delta t) - g^2(t + \Delta t)\mathbf{s}(\mathbf{x}_{t+\Delta t}, t + \Delta t)]\Delta t)\|_2^2}{2g^2(t + \Delta t)\Delta t} \right). \end{aligned}$$

This implies that  $p(\mathbf{x}_t | \mathbf{x}_{t+\Delta t})$  is roughly a normal distribution with:

**Mean:**  $\mathbf{x}_{t+\Delta t} - [\mathbf{f}(\mathbf{x}_{t+\Delta t}, t + \Delta t) - g^2(t + \Delta t)\mathbf{s}(\mathbf{x}_{t+\Delta t}, t + \Delta t)]\Delta t$ ,  
**Covariance:**  $g^2(t + \Delta t)\Delta t \mathbf{I}$ .

Taking the limit as  $\Delta t \rightarrow 0$ , we “derive” the reverse-time continuous SDE given in Equation (4.1.6).

## 4.6 Closing Remarks

This chapter marked a pivotal moment in our journey, unifying the discrete-time diffusion processes from the variational and score-based perspectives into a single, elegant continuous-time framework. We demonstrated that both DDPM and NCSN can be understood as discretizations of Stochastic Differential Equations (SDEs) with different drift/volatility coefficients.

The cornerstone of this framework is the existence of a corresponding reverse-time SDE, which formally defines a generative process that reverses the noise corruption. Crucially, the drift of this reverse process depends on a single unknown quantity: the score function,  $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ , of the marginal data distributions at every point in time. This insight solidifies the score function's central role in generative modeling.

Furthermore, we introduced a purely deterministic counterpart, the Probability Flow Ordinary Differential Equation (PF-ODE), whose solution trajectories evolve along the same marginal densities  $\{p_t\}$  as the SDEs. This remarkable consistency is guaranteed by the underlying Fokker-Planck equation. The profound implication is that the complex task of generation is fundamentally equivalent to solving a differential equation. Training reduces to learning the score function that defines the equation's vector field, while sampling becomes a problem of numerical integration.

The introduction of the PF-ODE, a purely deterministic flow, provides a powerful bridge to the third and final perspective on diffusion models. This concept of learning a deterministic transformation governed by a velocity field is the central principle of recent major family of generative models. In the next chapter, we will:

1. Explore this flow-based perspective, starting from its origins in Normalizing Flows and Neural ODEs.
2. Show how this viewpoint leads to the modern framework of Flow Matching, which directly learns a velocity field to transport samples between distributions.

Ultimately, we will see how the deterministic PF-ODE, which we derived from stochastic principles, can be constructed and generalized from this entirely different, flow-based origin, completing our unified picture of diffusion modeling.

# 5

---

## Flow-Based Perspective: From NFs to Flow Matching

---

*Everything flows.*

---

Heraclitus

The *change-of-variables formula*, a cornerstone of probability theory (Tabak and Vanden-Eijnden, 2010; Turner, 2013), takes on new life in modern generative modeling. While Score SDEs offer a differential equation framework to bridge data and prior distributions via the Fokker–Planck equation (Section 4.1.5), this continuous evolution is, at its core, a dynamic form of the same fundamental principle.

**Change-of-Variables Formula of Densities.** Given an invertible transformation  $\mathbf{f}$ , the density of  $\mathbf{x} = \mathbf{f}(\mathbf{z})$  where  $\mathbf{z} \sim p_{\text{prior}}$  is:

$$p(\mathbf{x}) = p_{\text{prior}}(\mathbf{z}) \left| \det \frac{\partial \mathbf{f}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right|, \quad \text{where } \mathbf{z} = \mathbf{f}^{-1}(\mathbf{x}). \quad (5.0.1)$$

This deceptively simple formula unlocks exact, bidirectional transport of densities and samples when  $\mathbf{f}$  is tractable, forming the very foundation of Normalizing Flows that we will introduce in Section 5.1. But what if we rethink this idea through the lens of continuous-time transformations?

In this chapter, we build on this core principle to explore a fresh view on diffusion models: Flow Matching (in Section 5.2). Emerging naturally from

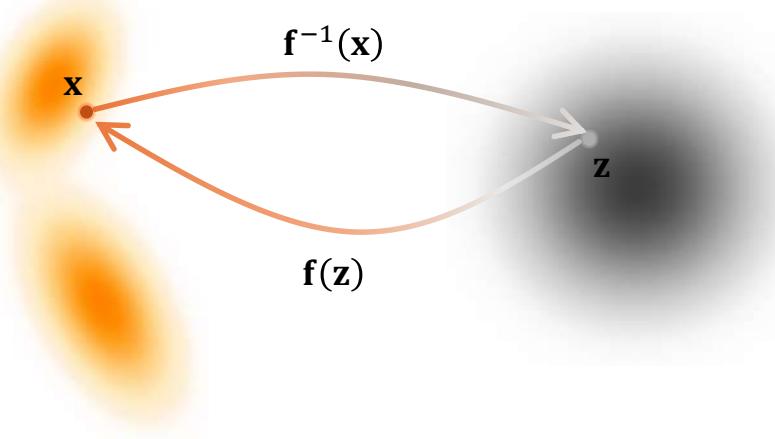
(Continuous) Normalizing Flows, Flow Matching deepens our understanding of diffusion as a powerful density transport process.

To support a solid understanding of this chapter, we provide in Chapter B an intuitive, self-contained overview of the different variants of the change-of-variables formula, progressing step by step from the basic case to the continuity equation and finally to the Fokker–Planck equation.

## 5.1 Flow-Based Models: Normalizing Flows and Neural ODEs

In this section, we will introduce Flow-Based Models, including Normalizing Flows (NFs) (Rezende and Mohamed, 2015) and Neural Ordinary Differential Equations (NODEs) (Chen *et al.*, 2018).

NFs enable flexible and tractable probability density estimation by applying a series of invertible transformations to a simple base distribution. NODEs extend this framework to continuous time, where the transformation is governed by an ODE. By treating the transformations as continuous-time dynamics, NODEs provide a smooth, scalable extension to the NF paradigm.



**Figure 5.1: Illustration of sample movement of NF under an invertible map.** It consists of a sequence of invertible functions  $\mathbf{f} : \mathbf{z} \mapsto \mathbf{x}$  that transform latent variable  $\mathbf{z}$  into a data  $\mathbf{x}$ , together with the inverse mapping  $\mathbf{f}^{-1} : \mathbf{x} \mapsto \mathbf{z}$  that reconstructs the data. An NF resembles an encoder–decoder structure, but with the encoder realized as a smooth invertible map and the decoder given exactly by its inverse. The corresponding change in density can be computed via the change-of-variables formula, as given in Equation (5.0.1).

### 5.1.1 Normalizing Flows

NFs (Rezende and Mohamed, 2015) model a complex data distribution  $p_{\text{data}}(\mathbf{x})$  by transforming a simple prior  $p_{\text{prior}}(\mathbf{z})$  (e.g., standard Gaussian  $\mathcal{N}(\mathbf{0}, \mathbf{I})$ ) via an invertible mapping

$$\mathbf{f}_\phi : \mathbb{R}^D \rightarrow \mathbb{R}^D,$$

with  $\mathbf{x} = \mathbf{f}_\phi(\mathbf{z})$  and  $\mathbf{z} \sim p_{\text{prior}}$ . Here,  $\mathbf{x}$  and  $\mathbf{z}$  share the same dimension. Using the change-of-variables formula in Equation (5.0.1), the model likelihood is<sup>1</sup>

$$\log p_\phi(\mathbf{x}) = \log p_{\text{prior}}(\mathbf{z}) + \log \left| \det \frac{\partial \mathbf{f}_\phi^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right|. \quad (5.1.1)$$

**Training Objective.** Parameters  $\phi$  are learned by maximizing the likelihood over data:

$$\mathcal{L}_{\text{NF}}(\phi) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log p_\phi(\mathbf{x})]. \quad (5.1.2)$$

Computing the Jacobian determinant in Equation (5.1.1) can be costly, scaling as  $\mathcal{O}(D^3)$  in general.

**Constructing Invertible Transformations.** A single complex invertible network can be expensive due to its Jacobian determinant. Conversely, simple transforms (e.g., linear) are efficient but lack expressivity.

To balance this, NFs employ a sequence of  $K$  trainable invertible mappings  $\{\mathbf{f}_k\}_{k=0}^{L-1}$ , each with efficiently computable Jacobians:

$$\mathbf{f}_\phi = \mathbf{f}_{L-1} \circ \mathbf{f}_{L-2} \circ \cdots \circ \mathbf{f}_0.$$

Each  $\mathbf{f}_k$  is parameterized by a neural network, though we omit the explicit dependence on  $\phi$  for notational simplicity.

Samples transform via

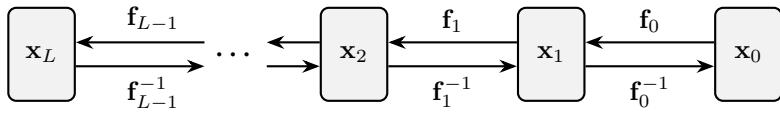
$$\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k), \quad k = 0, \dots, L-1, \quad (5.1.3)$$

with  $\mathbf{z} = \mathbf{x}_0 \sim p_{\text{prior}}$  and  $\mathbf{x} = \mathbf{x}_L$ , corresponding to data. The resulting (log-)density is derived as

$$\begin{aligned} p_\phi(\mathbf{x}) &= p_{\text{prior}}(\mathbf{x}_0) \prod_{k=0}^{L-1} \left| \det \frac{\partial \mathbf{f}_k}{\partial \mathbf{x}_k} \right|^{-1}, \text{ or equivalently,} \\ \log p_\phi(\mathbf{x}) &= \log p_{\text{prior}}(\mathbf{x}_0) + \sum_{k=0}^{L-1} \log \left| \det \frac{\partial \mathbf{f}_k}{\partial \mathbf{x}_k} \right|^{-1}. \end{aligned} \quad (5.1.4)$$

---

<sup>1</sup>If the map is further constrained to be the gradient of a convex potential,  $\mathbf{f}_\phi = \nabla \psi_\phi$  with  $\psi_\phi$  convex, then Equation (5.1.1) reduces to the Monge–Ampère relation in Equation (7.2.4). This PDE characterizes the optimal transformation of one distribution into another under the quadratic cost. See Chapter 7 and (Huang *et al.*, 2021) for further details.



**Figure 5.2: Illustration of a NF.** NF is consisting of a stack of invertible maps  $\mathbf{f}_\phi = \mathbf{f}_{L-1} \circ \mathbf{f}_{L-2} \circ \dots \circ \mathbf{f}_0$ . The transformation maps latent samples  $\mathbf{x}_0 \sim p_{\text{prior}}$  to data samples  $\mathbf{x}_L \sim p_{\text{data}}$ .

**Examples of Invertible Flows.** Extensive literature had focused on designing single-layer flow constructions that enable efficient computation of the Jacobian. Below, we introduce two representative types: Planar Flows (Rezende and Mohamed, 2015) and Residual Flows (Chen *et al.*, 2019; Behrmann *et al.*, 2019), with the latter motivating the developments in Section 5.1.2.

**Planar Flows:** It applies a simple transformation

$$\mathbf{f}(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b),$$

where  $\mathbf{u}, \mathbf{w} \in \mathbb{R}^D$ ,  $b \in \mathbb{R}$ , and  $h(\cdot)$  is an activation. The Jacobian determinant is

$$\left| 1 + \mathbf{u}^\top h'(\mathbf{w}^\top \mathbf{z} + b)\mathbf{w} \right|.$$

**Residual Flows:** Define the transform  $\mathbf{f}$  as

$$\mathbf{f}(\mathbf{z}) = \mathbf{z} + \mathbf{v}(\mathbf{z}), \quad (5.1.5)$$

with  $\mathbf{v}$  contractive (Lipschitz constant  $< 1$ ). This ensures invertibility via the Banach fixed-point theorem.

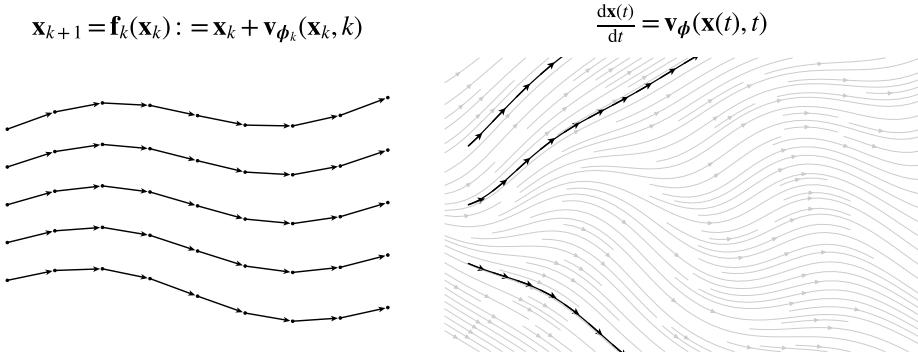
The log-determinant of the Jacobian reduces to a trace expansion:

$$\begin{aligned} \log \left| \det \frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right| &= \log \left( \det \frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right) \\ &= \text{Tr} \left( \log \left( \frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right) \right) \\ &= \text{Tr} \left( \log \left( \mathbf{I} + \frac{\partial \mathbf{v}(\mathbf{z})}{\partial \mathbf{z}} \right) \right) \\ &= \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \text{Tr} \left( \left( \frac{\partial \mathbf{v}(\mathbf{z})}{\partial \mathbf{z}} \right)^k \right), \end{aligned} \quad (5.1.6)$$

making evaluation efficient via trace estimators (Hutchinson, 1989).

**Sampling and Inference.** Sampling from NFs is straightforward: draw  $\mathbf{x}_0 \sim p_{\text{prior}}$  and compute  $\mathbf{x} = \mathbf{f}_\phi(\mathbf{x}_0)$ . Exact likelihoods are obtained from Equation (5.1.4).

### 5.1.2 Neural ODEs



**Figure 5.3: Discrete- vs. continuous-time normalizing flows.** (Left) A discrete NF transports samples by a finite sequence of invertible maps  $\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k)$ , yielding stepwise, non-crossing trajectories (dots with arrows). (Right) A continuous NF (Neural ODE) evolves states along integral curves of  $\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}_\phi(\mathbf{x}(t), t)$ , where black paths with tangent arrows are shown over the gray vector field.

**From Discrete-Time NFs to Continuous-Time NFs (Neural ODEs).** NFs are typically formulated as a sequence of  $L$  discrete, invertible transformations. Viewed through the lens of Equation (5.1.3) and the “Residual Flow” formulation in Equation (5.1.5), each layer can be written as the following:

$$\mathbf{x}_{k+1} = \mathbf{f}_k(\mathbf{x}_k) := \mathbf{x}_k + \mathbf{v}_{\phi_k}(\mathbf{x}_k, k),$$

where  $\mathbf{v}_{\phi_k}(\cdot, k)$  is a layer-dependent velocity field parameterized by neural networks. Intuitively, this velocity field is a learned vector-valued function that “pushes” the data points in the input space in small, smooth steps. Each transformation moves points along the directions suggested by this velocity, gradually morphing the simple prior distribution into the complex target distribution.

This formulation, indeed, corresponds to the Euler discretization of the continuous-time ODE with learnable parameter  $\phi$ :

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}_\phi(\mathbf{x}(t), t).$$

In the limit of infinite layers and vanishing step size ( $\Delta t \rightarrow 0$ ), the discrete NFs converges to a continuous model, yielding the framework of *Neural ODEs* (NODEs) (Chen *et al.*, 2018), also known as *Continuous Normalizing Flows* (CNFs).

**Formal Setup of Neural ODEs.** A Neural ODE defines a continuous transformation through:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}_\phi(\mathbf{x}(t), t), \quad t \in [0, T] \quad (5.1.7)$$

where:

- $\mathbf{x}(t) \in \mathbb{R}^D$  is the state at time  $t$ ; we sometimes write  $\mathbf{x}_t$  for brevity;
- $\mathbf{v}_\phi(\mathbf{x}(t), t)$  is a neural network parameterized by  $\phi$ .

**Goal of NODE.** Starting from the initial condition  $\mathbf{x}(0) \sim p_{\text{prior}}$ , the ODE evolves the state continuously over time, inducing a family of marginal distributions  $p_\phi(\mathbf{x}_t, t)$  (similar to PF-ODEs!)<sup>2</sup>.

The goal is to learn the neural vector field  $\mathbf{v}_\phi$ , which intuitively represents a velocity that transports points along continuous trajectories in data space. By learning this velocity, the terminal distribution at  $t = 0$  matches the target distribution  $p_{\text{data}}(\cdot)$ . This continuous transformation unifies discrete normalizing flows and neural ODEs within a single framework.

**Continuous-Time Change-of-Variables Formula.** Analogous to Equation (5.0.1) or Equation (5.1.4), Chen *et al.* (2018) derived a continuous-time analog of the change-of-variables formula. For the time-dependent density  $p_\phi(\mathbf{x}(t), t)$  of a process  $\mathbf{x}(t)$  evolving under Equation (5.1.7), the so-called *Instantaneous Change-of-Variables Formula* is:

$$\frac{d}{dt} \log p_\phi(\mathbf{x}(t), t) = -\nabla_{\mathbf{x}} \cdot \mathbf{v}_\phi(\mathbf{x}(t), t).$$

Thus, with the given prior  $p_{\text{prior}}(\mathbf{x}(T), T)$ , the log-density of the terminal state  $\mathbf{x}(T)$  induced by the neural ODE is given by

$$\log p_\phi(\mathbf{x}(T), T) = \log p_{\text{prior}}(\mathbf{x}(0), 0) - \int_0^T \nabla_{\mathbf{x}} \cdot \mathbf{v}_\phi(\mathbf{x}(t), t) dt. \quad (5.1.8)$$

---

<sup>2</sup>We adopt a flipped time convention, with  $t = 0$  denoting the prior (source) and  $t = 1$  the data (target) distribution. The prior is interchangeably written as  $p_\phi(\mathbf{x}(0), 0)$ ,  $p_{\text{prior}}(\mathbf{x}(0), 0)$ , or simply  $p_{\text{prior}}(\mathbf{z})$ .

This expression enables exact likelihood evaluation by numerically solving the ODE, which in turn allows for maximum likelihood training of the model. We will return to this in detail later.

Although it may appear unfamiliar at first, this instantaneous change of variable formula is a special case of the Fokker–Planck equation, specifically its deterministic form known as the *Continuity Equation* (see Chapter B). It can also be interpreted as the continuous time limit of Equation (5.1.4). We summarize this result and its derivation in the following lemma:

### Lemma 5.1.1: Instantaneous Change of Variables

Let  $\mathbf{z}(t)$  be a continuous random process with time-dependent density  $p(\mathbf{z}(t), t)$ , and suppose it evolves according to the ODE

$$\frac{d\mathbf{z}(t)}{dt} = \mathbf{F}(\mathbf{z}(t), t).$$

Assuming  $\mathbf{F}$  is uniformly Lipschitz in  $\mathbf{z}$  and continuous in  $t$ , the time derivative of the log-density satisfies:

$$\frac{\partial \log p(\mathbf{z}(t), t)}{\partial t} = -\nabla_{\mathbf{z}} \cdot \mathbf{F}(\mathbf{z}(t), t). \quad (5.1.9)$$

#### Proof for Lemma.

We present two alternative derivations in Section D.3. □

**Connection to Discrete-Time Formula.** The NODE likelihood in Equation (5.1.8),

$$\log p_{\phi}(\mathbf{x}(T), T) = \log p_{\text{prior}}(\mathbf{x}(0), 0) - \int_0^T \nabla_{\mathbf{x}} \cdot \mathbf{v}_{\phi}(\mathbf{x}(t), t) dt,$$

can be seen as the continuous-time analogue of the discrete normalizing flow formulation in Equation (5.1.4):

$$\log p_{\phi}(\mathbf{x}_L) = \log p_{\text{prior}}(\mathbf{x}_0) - \sum_{k=0}^{L-1} \log \left| \det \frac{\partial \mathbf{f}_k}{\partial \mathbf{x}_k} \right|.$$

The integral mirrors the summation, and the trace operator replaces the log-determinant, as discussed in Equation (5.1.6). These parallels are further explored in the proof of the lemma.

**Training NODEs.** Based on Equation (5.1.8), NODEs learn a parameterized velocity field  $\mathbf{v}_\phi$  such that the terminal distribution

$$p_\phi(\cdot, T) \approx p_{\text{data}},$$

where trajectories evolve from latent variables  $\mathbf{x}(0) \sim p_{\text{prior}}$  via the ODE flow. Training follows the MLE framework from Equation (1.1.2):

$$\mathcal{L}_{\text{NODE}}(\phi) := \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log p_\phi(\mathbf{x}, T)].$$

**Exact Log-Likelihood Computation.** To compute  $\log p_\phi(\mathbf{x}, T)$  for data point  $\mathbf{x}$ , we integrate the change-of-variables formula equation 5.1.8:

$$\log p_\phi(\mathbf{x}, T) = \log p_{\text{prior}}(\mathbf{z}(0)) - \int_0^T \nabla_{\mathbf{z}} \cdot \mathbf{v}_\phi(\mathbf{z}(t), t) dt. \quad (5.1.10)$$

Here,  $\mathbf{z}(t)$  solves the ODE reversely from  $t = T$  to  $t = 0$ :

$$\frac{d\mathbf{z}}{dt} = \mathbf{v}_\phi(\mathbf{z}(t), t)$$

with  $\mathbf{z}(T) = \mathbf{x}$ . The prior term  $\log p_{\text{prior}}(\mathbf{z}(0))$  is tractable for standard distributions. This enables exact likelihood-based training and evaluation in neural ODEs.

**Gradient-Based Optimization.** Maximizing  $\mathcal{L}_{\text{NODE}}$  requires backpropagation through the ODE solver. The adjoint sensitivity method (Pontryagin, 2018; Chen *et al.*, 2018) computes gradients via an auxiliary ODE with  $\mathcal{O}(1)$  memory complexity, but NODE training remains expensive due to numerical integration at each step.

**Inference with NODEs.** Sampling with a trained model  $\mathbf{v}_{\phi^\times}$  proceeds by drawing  $\mathbf{x}(0) \sim p_{\text{prior}}$  and integrating forward (by numerical solvers):

$$\mathbf{x}(T) = \mathbf{x}(0) + \int_0^T \mathbf{v}_{\phi^\times}(\mathbf{x}(t), t) dt.$$

The terminal state  $\mathbf{x}(T)$  approximates a sample from  $p_{\text{data}}$ .

Moreover, we note that for any vector field  $\mathbf{F}$ , the following identity holds:

$$\text{Tr} \left( \frac{\partial \mathbf{F}}{\partial \mathbf{z}(t)} \right) = \nabla_{\mathbf{z}} \cdot \mathbf{F}.$$

Hence, the divergence can be efficiently estimated using stochastic trace estimators, such as Hutchinson's estimator (Hutchinson, 1989), which makes exact likelihood computation more tractable in high-dimensional settings.

## 5.2 Flow Matching Framework

Score SDEs (Chapter 4) and NODEs (Section 5.1) offer an alternative perspective on generative modeling: learning a continuous-time flow, either stochastic or deterministic, that transports a simple Gaussian prior sample  $\epsilon \sim p_{\text{prior}}$  to a data-like sample from  $p_{\text{data}}$ .

The *Flow Matching* (FM) framework (Lipman *et al.*, 2022; Lipman *et al.*, 2024; Tong *et al.*, 2024) builds on this idea, but generalizes it to learn a flow between two *arbitrary* fixed endpoint distributions: a source distribution  $p_{\text{src}}$  and a target distribution  $p_{\text{tgt}}$ , both assumed to be easy to sample from. In this broader setup, the generation task becomes a special case where  $p_{\text{src}}$  is a Gaussian prior and  $p_{\text{tgt}}$  is the data distribution.

In this section, we adopt the FM viewpoint<sup>3</sup>, emphasizing its core principle: learning a time-dependent vector field  $\mathbf{v}_t(\mathbf{x}_t)$  whose associated ODE flow matches a predefined probability path  $\{p_t\}_{t \in [0,1]}$  subject to the boundary conditions

$$p_0 = p_{\text{src}}, \quad p_1 = p_{\text{tgt}}.$$

When  $p_{\text{src}}$  is Gaussian, we refer to this setting as *Gaussian Flow Matching*. Compared to classical diffusion models, FM enables efficient, simulation-free training for a broad class of transport problems using only samples from the endpoints.

### 5.2.1 Lesson from Score-Based Methods

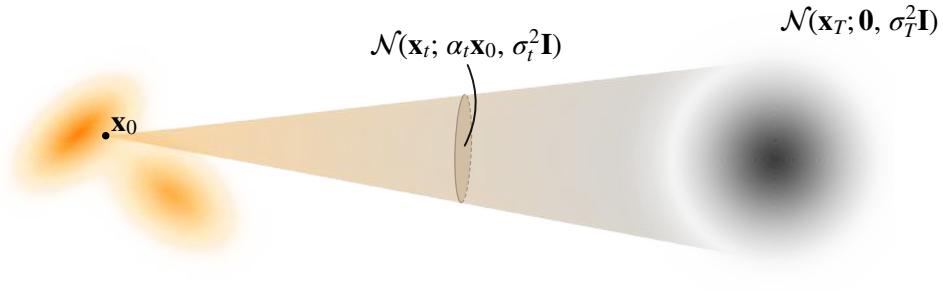
We revisit the Score SDE framework (Chapter 4) using a slightly different but equivalent formulation to extract key insights that motivate the FM approach. This analysis reveals how diffusion models implicitly learn probability flows and motivates a more direct formulation.

**Step 1: Defining a Conditional Path and Its Marginal Densities.** A diffusion model specifies a continuous-time family of densities  $\{p_t\}_{t \in [0,1]}$  that transports a simple prior  $p_{\text{prior}}$  (e.g., Gaussian) at  $t = 1$ , used as the source, to a target data distribution  $p_{\text{data}}$  at  $t = 0$ :

$$p_1(\mathbf{x}_1) = p_{\text{prior}}(\mathbf{x}_1), \quad p_0(\mathbf{x}_0) = p_{\text{data}}(\mathbf{x}_0).$$

---

<sup>3</sup>Several related approaches share the core idea of transporting between endpoint distributions using a continuous-time flow, though with slightly different formulations. These include Flow Matching (FM) (Lipman *et al.*, 2022; Neklyudov *et al.*, 2023), Rectified Flow (RF) (Liu, 2022; Heitz *et al.*, 2023), and Stochastic Interpolants (Albergo *et al.*, 2023; Albergo and Vanden-Eijnden, 2023; Ma *et al.*, 2024). Here, we use the FM terminology as a unifying representation.



**Figure 5.4: Illustration of the conditional transition distribution.**  $p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$ , defines a (Gaussian) conditional probability path from a data sample  $\mathbf{x}_0 \sim p_{\text{data}}$  (left) towards the Gaussian prior  $p_{\text{prior}}$  (right).

This path is implicitly defined via the forward conditional distribution

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}), \quad \mathbf{x}_0 \sim p_{\text{data}} \quad (5.2.1)$$

which induces the marginal density

$$p_t(\mathbf{x}_t) := \int p_t(\mathbf{x}_t|\mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0.$$

The increasing variance  $\sigma_t^2$  of the conditional Gaussian drives the evolution of  $p_t$  toward the Gaussian prior.

**Step 2: Velocity Field.** The time evolution of the marginal density  $p_t$  is governed by a velocity field  $\mathbf{v}_t : \mathbb{R}^D \rightarrow \mathbb{R}^D$ , derived from the Fokker–Planck equation:

$$\mathbf{v}_t(\mathbf{x}) := f(t)\mathbf{x} - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}), \quad (5.2.2)$$

which defines a deterministic particle flow through the PF-ODE:

$$\frac{d\mathbf{x}(t)}{dt} = \underbrace{f(t)\mathbf{x}(t) - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}(t))}_{\mathbf{v}_t(\mathbf{x}(t))}.$$

This ODE transports an initial random variable  $\mathbf{x}(0) \sim p_{\text{data}}$  forward in time or  $\mathbf{x}(1) \sim p_{\text{prior}}$  backward in time, such that the evolving marginal density of  $\mathbf{x}(t)$  matches  $p_t$  at every  $t \in [0, 1]$  (see “Underlying Rule” below).

The scalar functions  $f(t)$  and  $g(t)$  are determined by the coefficients of the associated forward SDE, or equivalently the Gaussian kernel parameters  $\alpha_t$  and  $\sigma_t$  defined in the conditional path (see Lemma 4.4.1).

**Step 3: Learning via the Conditional Strategy.** The goal is to approximate the oracle velocity field  $\mathbf{v}_t(\mathbf{x}_t)$  using a neural network  $\mathbf{s}_\phi(\mathbf{x}_t, t)$  trained via the expected squared error:

$$\mathcal{L}_{\text{SM}}(\phi) = \mathbb{E}_{t \sim \mathcal{U}[0,1], \mathbf{x}_t \sim p_t} \left[ \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\|^2 \right].$$

Since the marginal score  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$  is inaccessible, we exploit the tractable conditional distribution to define the conditional velocity:

$$\mathbf{v}_t(\mathbf{x}_t | \mathbf{x}_0) := f(t)\mathbf{x}_t - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0).$$

By the law of total expectation, the marginal score is recovered as

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \mathbb{E}_{\mathbf{x}_0 \sim p(\cdot | \mathbf{x}_t)} [\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)]. \quad (5.2.3)$$

This justifies the surrogate training objective:

$$\mathcal{L}_{\text{SM}}(\phi) = \underbrace{\mathbb{E}_{t, \mathbf{x}_0 \sim p_{\text{data}}, \mathbf{x}_t \sim p_t(\cdot | \mathbf{x}_0)} \left[ \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)\|^2 \right]}_{\mathcal{L}_{\text{DSM}}(\phi)} + C,$$

where  $C$  is a constant independent of  $\phi$ . The minimizer  $\mathbf{s}^*(\mathbf{x}_t, t)$  satisfies

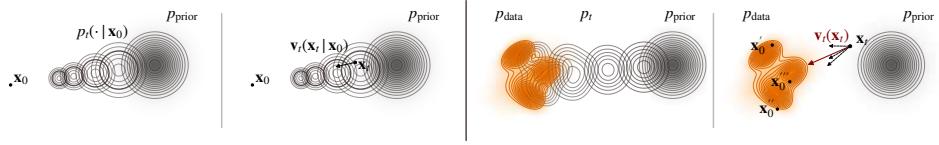
$$\mathbf{s}^*(\mathbf{x}_t, t) = \mathbb{E}_{\mathbf{x}_0 \sim p(\cdot | \mathbf{x}_t)} [\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)] = \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t),$$

where the second equality follows from Equation (5.2.3), thereby validating the conditional training objective.

**Underlying Rule: The Fokker–Planck Equation.** The marginal density  $p_t$  evolves according to the Fokker–Planck equation:

$$\frac{\partial p_t(\mathbf{x})}{\partial t} + \nabla \cdot \left( \underbrace{\left( f(t)\mathbf{x} - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \right)}_{\mathbf{v}_t(\mathbf{x})} p_t(\mathbf{x}) \right) = 0.$$

This PDE ensures that the density given by the PF-ODE matches the marginal distribution of the forward SDE. To see this, recall the *flow map*  $\Psi_{s \rightarrow t}(\mathbf{x}_s)$  of the PF-ODE as defined in Equation (4.1.9), which carries an initial state



**Figure 5.5: Illustration of conditional versus marginal perspectives in diffusion.** (This figure is motivated by Lipman *et al.* (2024).) (1) Conditional Gaussian path  $p_t(\cdot | \mathbf{x}_0)$ , showing expanding densities from a fixed  $\mathbf{x}_0$  toward the prior. (2) Conditional velocity  $\mathbf{v}_t(\mathbf{x}_t | \mathbf{x}_0) = f(t)\mathbf{x}_t - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)$ . (3) Marginal density  $p_t$ , transporting the data distribution  $p_{\text{data}}$  (orange) into the prior  $p_{\text{prior}}$  (gray). (4) Marginal velocity  $\mathbf{v}_t(\mathbf{x}_t) = f(t)\mathbf{x}_t - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ , obtained by averaging conditional directions from  $\mathbf{x}_t$  to multiple plausible origins (dashed), yielding the red arrow. In the FM framework with one-sided conditioning  $\mathbf{z} = \mathbf{x}_0$ , the same illustration applies to  $\mathbf{v}_t(\mathbf{x}_t | \mathbf{x}_0)$  and  $\mathbf{v}_t(\mathbf{x}_t)$ , without requiring them to be written explicitly in terms of the scores  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)$  or  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ .

$\mathbf{x}_s$  at time  $s$  directly to its state at  $t$ . Running the PF-ODE backward from  $t = 1$  to  $t = 0$ , starting with  $\mathbf{x}_1 \sim p_{\text{prior}}$ , we obtain time-dependent densities through the pushforward formula:

$$p_t^{\text{rev}}(\mathbf{x}) = \int \delta(\mathbf{x} - \Psi_{1 \rightarrow t}(\mathbf{x}_1)) p_{\text{prior}}(\mathbf{x}_1) d\mathbf{x}_1. \quad (5.2.4)$$

The Fokker–Planck equation ensures that the induced density path coincides with the same evolving density:

$$p_t^{\text{rev}} = p_t. \quad (5.2.5)$$

In particular, this implies  $p_0^{\text{rev}} = p_0 = p_{\text{data}}$ , thereby recovering the data distribution at time  $t = 0$ . Since the ODE solution map is bidirectional, we can similarly consider initializing at  $\mathbf{x}_0 \sim p_{\text{data}}$  and solving the ODE forward in time, enabling a parallel analysis.

### 5.2.2 Flow Matching Framework

The analysis in Section 5.2.1 reveals that diffusion models succeed by learning a velocity field, specifically, the score, that transports between distributions while satisfying boundary conditions. The design of the Gaussian conditional path in Equation (5.2.1), with increasing variance  $\sigma_t^2$ , implicitly anchors one endpoint to a Gaussian prior while allowing the conditional density to be defined over the entire space, enabling score-based gradient computation.

In this subsection, we introduce the FM framework, which builds on this insight (the same illustration in Figure 5.5 also applies to the FM framework) and extends it to learning continuous flows that transport samples between two arbitrary distributions,  $p_{\text{src}}$  and  $p_{\text{tgt}}$ .

**Step 1: Defining a Conditional Path and Its Marginal Densities.** Consider arbitrary source and target probability distributions  $p_{\text{src}}$  and  $p_{\text{tgt}}$  on  $\mathbb{R}^D$ . We set<sup>4</sup>

$$p_0(\mathbf{x}) = p_{\text{src}}(\mathbf{x}), \quad p_1(\mathbf{x}) = p_{\text{tgt}}(\mathbf{x}). \quad (5.2.6)$$

FM implicitly defines a continuous family of intermediate densities  $\{p_t\}_{t \in [0,1]}$  interpolating between these endpoints. Each marginal  $p_t$  is expressed via a latent variable  $\mathbf{z}$  drawn from a known distribution  $\pi(\mathbf{z})$  and a conditional distribution  $p_t(\mathbf{x}_t | \mathbf{z})$ :

$$p_t(\mathbf{x}_t) = \int p_t(\mathbf{x}_t | \mathbf{z}) \pi(\mathbf{z}) d\mathbf{z}, \quad (5.2.7)$$

with  $(\pi(\mathbf{z}), \{p_t(\cdot | \mathbf{z})\})$  chosen to satisfy the boundary conditions in Equation (5.2.6).

We remark that, in general, the marginal densities  $p_t$  are not tractable, since they require integrating over  $\pi(\mathbf{z})$ , and both  $\pi(\mathbf{z})$  and the conditional distributions  $p_t(\mathbf{x}_t | \mathbf{z})$  can be complex. Nonetheless, conditioning on the latent  $\mathbf{z}$  grants FM the flexibility to model a broad class of interpolation paths beyond those discussed in Section 5.2.1. Common choices for  $\mathbf{z}$  include:

- **Two-sided conditioning:**  $\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1) \sim p_{\text{src}}(\mathbf{x}_0)p_{\text{tgt}}(\mathbf{x}_1)$ , where  $\pi$  couples source and target distributions. This allows FM to define transport between arbitrary distributions.
- **One-sided conditioning:**  $\mathbf{z} = \mathbf{x}_0$  or  $\mathbf{z} = \mathbf{x}_1$ . It especially recovers diffusion-like setups when the source distribution is chosen to be Gaussian.

In all cases, the conditional distributions  $p_t(\mathbf{x}_t | \mathbf{z})$  should admit tractable closed-form expressions. We make this assumption throughout and present specific constructions in Section 5.3.2 with illustrations in Figure 5.6.

**Step 2: Velocity Field.** In standard diffusion models or Gaussian FM, the intermediate densities  $\{p_t\}_{t \in [0,1]}$  are constructed with one endpoint set to a standard Gaussian. In this setting, the velocity field  $\mathbf{v}_t$  is uniquely defined and admits a closed-form expression related to scores (see Equation (5.2.2)).

In contrast, general FM interpolates between general source and target distributions  $p_{\text{src}}$  and  $p_{\text{tgt}}$ , where the velocity field is no longer uniquely determined (as explained later).

---

<sup>4</sup>To align with the standard notation in FM literature, we reverse the time axis compared to earlier sections:  $t = 0$  corresponds to the source distribution and  $t = 1$  to the target.

The goal is to find a velocity field  $\mathbf{v}_t(\mathbf{x})$  such that the induced ODE, which enables a sample-wise transformation,

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}_t(\mathbf{x}(t)), \quad t \in [0, 1],$$

produces marginal distributions of  $\mathbf{x}(t)$  that match with  $p_t$  at each time  $t$ , whether integrating forward from  $\mathbf{x}(0) \sim p_{\text{src}}$  or backward from  $\mathbf{x}(1) \sim p_{\text{tgt}}$  (see Section 5.2.4 for a more formal discussion).

This requirement is captured by the *continuity equation*<sup>5</sup>:

$$\frac{\partial p_t(\mathbf{x})}{\partial t} + \nabla \cdot (\mathbf{v}_t(\mathbf{x}) p_t(\mathbf{x})) = 0. \quad (5.2.8)$$

Any velocity field  $\mathbf{v}_t$  that satisfies Equation (5.2.8) ensures that the ODE flow transports samples in a way that exactly follows the prescribed  $p_t$  (see Section 5.2.4 for details). Thus, solving the ODE enables transport from  $p_{\text{src}}$  to  $p_{\text{tgt}}$  while matching all intermediate distributions.

Intuitively, many different flows can induce the same marginal evolution. This is because Equation (5.2.8) is a scalar equation, while  $\mathbf{v}_t$  is a vector field in  $\mathbb{R}^D$ , so the equation admits infinitely many solutions. For example, if  $\mathbf{v}_t$  solves the equation, then so does

$$\mathbf{v}_t + \frac{1}{p_t} \tilde{\mathbf{v}}_t,$$

for any divergence-free vector field  $\tilde{\mathbf{v}}_t$  (i.e.,  $\nabla \cdot \tilde{\mathbf{v}}_t = 0$ ). FM therefore seeks a particular velocity field  $\mathbf{v}_t$  that satisfies Equation (5.2.8), enabling continuous transport of samples along the path  $\{p_t\}$ . For arbitrary distributions, however,  $p_t$  and  $\mathbf{v}_t$  are generally not available in closed form. As a concrete illustration, in Section 5.3.1 we consider the Gaussian-to-Gaussian bridge, where both quantities can be computed explicitly.

**Step 3: Learning via the Conditional Strategy.** The goal of FM training is to approximate the oracle velocity field  $\mathbf{v}_t$  using a neural network  $\mathbf{v}_\phi$ , by minimizing the expected squared error:

$$\mathcal{L}_{\text{FM}}(\phi) = \mathbb{E}_{t, \mathbf{x}_t \sim p_t} \left[ \|\mathbf{v}_\phi(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t)\|^2 \right].$$

We refer to this neural network parameterization as **v-prediction** (velocity prediction), which aims to learn the ODE drift term directly.

---

<sup>5</sup>The deterministic analogue of the Fokker–Planck equation, without the diffusion term.

As in Section 5.2.1, the oracle velocity  $\mathbf{v}_t(\mathbf{x})$  is generally intractable. To address this, we introduce a latent variable  $\mathbf{z} \sim \pi(\mathbf{z})$  and define a conditional velocity field  $\mathbf{v}_t(\mathbf{x}|\mathbf{z})$  by construction. This allows us to rewrite the loss via the law of total expectation<sup>6</sup>:

$$\mathcal{L}_{\text{FM}}(\phi) = \underbrace{\mathbb{E}_{t, \mathbf{z} \sim \pi(\mathbf{z}), \mathbf{x}_t \sim p_t(\cdot|\mathbf{z})} [\|\mathbf{v}_\phi(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t|\mathbf{z})\|^2]}_{\mathcal{L}_{\text{CFM}}(\phi)} + C, \quad (5.2.9)$$

where  $C$  is a constant independent of  $\phi$ . The main term  $\mathcal{L}_{\text{CFM}}$  is referred to as *conditional flow matching*.

That is, minimizing  $\mathcal{L}_{\text{FM}}(\phi)$  is equivalent to minimizing  $\mathcal{L}_{\text{CFM}}(\phi)$ , with the latter offering a more tractable formulation. For  $\mathcal{L}_{\text{CFM}}(\phi)$  to enable tractable, simulation-free training, two requirements must be met:

- (i) Sampling from the conditional probability path  $p_t(\mathbf{x}_t|\mathbf{z})$  should be straightforward (simulation-free).
- (ii) The conditional velocity  $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$ , used as the regression target, must admit a simple closed-form expression.

We will provide explicit constructions that satisfy these conditions in Section 5.3.2. This conditional view makes training feasible: instead of learning the intractable unconditional velocity field  $\mathbf{v}_t(\cdot)$ , the model learns the tractable conditional field  $\mathbf{v}_t(\cdot|\mathbf{z})$ : in direct analogy to denoising score matching.

Even though there are infinitely many possible unconditional velocity fields consistent with a given  $p_t$ , one such field can be recovered by marginalizing the conditional velocity fields:

$$\mathbf{v}_t(\mathbf{x}_t) := \mathbb{E}_{\mathbf{z} \sim p(\cdot|\mathbf{x}_t)} [\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})], \quad (5.2.10)$$

where the expectation is taken over  $p(\mathbf{z}|\mathbf{x}_t)$ . We can show that the minimizer  $\mathbf{v}^*$  of the conditional flow matching objective in Equation (5.2.9) recovers this marginal velocity:

$$\mathbf{v}^*(\mathbf{x}_t, t) = \mathbf{v}_t(\mathbf{x}_t). \quad (5.2.11)$$

Thus, learning to match the conditional velocity field  $\mathbf{v}_t(\cdot|\mathbf{z})$  suffices to recover a valid unconditional velocity field.

---

<sup>6</sup>This follows a standard integration-by-parts argument, as in the derivation of Equation (3.3.3). Likewise, Equation (5.2.11) is derived using a similar approach within the score matching framework.

We summarize the above discussion as follows:

**Theorem 5.2.1: Equivalence of  $\mathcal{L}_{\text{FM}}$  and  $\mathcal{L}_{\text{CFM}}$**

The following holds:

$$\mathcal{L}_{\text{FM}}(\phi) = \mathcal{L}_{\text{CFM}}(\phi) + C,$$

where  $C$  is a constant independent of the parameter  $\phi$ . Furthermore, the minimizer  $\mathbf{v}^*$  of both losses satisfies

$$\mathbf{v}^*(\mathbf{x}_t, t) = \mathbf{v}_t(\mathbf{x}_t), \quad \text{for almost every } \mathbf{x}_t \sim p_t,$$

where  $\mathbf{v}_t(\mathbf{x}_t)$  is defined in Equation (5.2.10).

**Proof for Theorem.**

The argument and derivation of the minimizer follows exactly the same reasoning as in the score matching case of Proposition 4.2.1. ■

This marks the third instance where the conditioning trick yields a tractable training objective. Notably, the variational, score based, and flow based approaches all reflect the same underlying principle.

**Remark.**

Taking  $\pi = p_{\text{data}}$ , we can apply Bayes' rule:

$$p(\mathbf{x}_0 | \mathbf{x}_t) = \frac{p_t(\mathbf{x}_t | \mathbf{x}_0)p_{\text{data}}(\mathbf{x}_0)}{p_t(\mathbf{x}_t)},$$

a similar decomposition of Equation (5.2.10) appears in score-based models:

$$\begin{aligned} \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) &= \mathbb{E}_{\mathbf{x}_0 \sim p(\cdot | \mathbf{x}_t)} [\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)] \\ &= \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}} \left[ \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0) \cdot \frac{p_t(\mathbf{x}_t | \mathbf{x}_0)}{p_t(\mathbf{x}_t)} \right], \end{aligned}$$

which mirrors the marginalization strategy in Equation (5.2.10).

As in Section 5.2.1, where the conditional density  $p_t(\mathbf{x}_t | \mathbf{z})$  and conditional velocity field  $\mathbf{v}_t(\mathbf{x}_t | \mathbf{z})$  must be explicitly specified, with  $p_t(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$  and  $\mathbf{v}_t(\mathbf{x}_t | \mathbf{x}_0) = f(t)\mathbf{x}_t - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)$ , the general conditional flow matching framework also requires these two components. However, we have not yet construct the conditional density  $p_t(\mathbf{x}_t | \mathbf{z})$  or the conditional velocity field  $\mathbf{v}_t(\mathbf{x}_t | \mathbf{z})$  in this general case. In the next section, we

introduce several common instantiations of these components.

### 5.2.3 Comparison of Diffusion Models, General Flow Matching, and NODEs

**Comparison of Diffusion Models and General Flow Matching.** The insight from Section 5.2.1 leads to an extended FM framework that retains the same underlying principles. To highlight their similarities, we summarize them in Table 5.1.

**Table 5.1:** Comparison between diffusion models (or Gaussian FM) and the general FM framework. Here, the general FM framework refers to the setting with two-sided conditioning, where  $\mathbf{x}_0 \sim p_{\text{src}}$  and  $\mathbf{x}_1 \sim p_{\text{tgt}}$  are sampled independently.

Aspect	Diffusion Model	General FM
Source dist. $p_{\text{src}}$	Gaussian prior	Any
Target dist. $p_{\text{tgt}}$	Data distribution	Any
Latent dist. $\pi(\mathbf{z})$	$p_{\text{data}}$	See Section 5.3.2
Cond. dist. $p_t(\mathbf{x}_t   \mathbf{z})$	$\mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$	See Section 5.3.2
Marginal dist. $p_t(\mathbf{x}_t)$	$\int p_t(\mathbf{x}_t   \mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0$	$\int p_t(\mathbf{x}_t   \mathbf{z}) \pi(\mathbf{z}) d\mathbf{z}$
Cond. velocity $\mathbf{v}_t(\mathbf{x}   \mathbf{z})$	$f(t)\mathbf{x} - \frac{1}{2}g^2(t)\nabla \log p_t(\mathbf{x}   \mathbf{x}_0)$	See Section 5.3.2
Marginal velocity $\mathbf{v}_t(\mathbf{x})$	$f(t)\mathbf{x} - \frac{1}{2}g^2(t)\nabla \log p_t(\mathbf{x})$	See Equation (5.2.10)
Learning objective	$\mathcal{L}_{\text{SM}} = \mathcal{L}_{\text{DSM}} + C$	$\mathcal{L}_{\text{FM}} = \mathcal{L}_{\text{CFM}} + C$
Underlying Rule	Fokker-Planck / Continuity Equation	

We remark that since Gaussian FM is essentially equivalent to the standard diffusion model (see more in Chapter 6), we will not differentiate between them unless explicitly stated.

**Connection to NODEs.** FM can be viewed as a simulation free alternative to NODEs, introduced in Section 5.1.2. While CNFs require solving ODEs during maximum likelihood training, which is computationally intensive, FM bypasses this by directly regressing a prescribed velocity field through a simple regression loss. The key insight is that when the marginal density path connecting the source and target distributions is fixed, exact simulation during training becomes unnecessary.

### 5.2.4 (Optional) Underlying Rules

**Continuity Equation: Mass Conservation Criterion.** Similar to the PF-ODE and Fokker–Planck analysis in Section 5.2.1, we now present a criterion for

verifying whether the density path induced by an ODE flow aligns with a prescribed path  $\{p_t\}_{t \in [0,1]}$ .

Consider the ODE describing the flow of particles under a time-dependent velocity field  $\mathbf{v}_t$ :

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}_t(\mathbf{x}(t)).$$

As in Equation (5.2.4), this ODE defines a flow map  $\Psi_{s \rightarrow t}(\mathbf{x}_0)$  for any  $s, t \in [0, 1]$ , which in particular transports an initial point  $\mathbf{x}_0 \sim p_{\text{src}}$  at time 0 to its state at time  $t$ . The induced distribution at time  $t$  is given by the pushforward

$$p_t^{\text{fwd}}(\mathbf{x}) = \int \delta(\mathbf{x} - \Psi_{0 \rightarrow t}(\mathbf{x}_0)) p_{\text{src}}(\mathbf{x}_0) d\mathbf{x}_0 =: \Psi_{0 \rightarrow t} \# p_{\text{src}}, \quad (5.2.12)$$

so that  $\Psi_{0 \rightarrow t}(\mathbf{x}_0) \sim p_t^{\text{fwd}}$  whenever  $\mathbf{x}_0 \sim p_{\text{src}}$ . Similarly, one can transport backward from  $\mathbf{x}_1 \sim p_{\text{tgt}}$  to  $p_{\text{src}}$  via  $\Psi_{1 \rightarrow 0}(\mathbf{x}_1)$ .

Suppose we are given a prescribed density path  $\{p_t\}_{t \in [0,1]}$ , and we construct a velocity field  $\{\mathbf{v}_t\}_{t \in [0,1]}$  to define a particle flow. This naturally raises the question:

### Question 5.2.1

Under what conditions does the flow-induced density  $p_t^{\text{fwd}}$  exactly match the target density  $p_t$  for all  $t \in [0, 1]$ ?

Once the two density evolutions align, we can leverage the ODE flow to flexibly transport samples between  $p_{\text{src}}$  and  $p_{\text{tgt}}$  by solving the ODE.

As in Equation (5.2.5), a principled way to verify this alignment is via the *continuity equation*, which captures the conservation of mass in time-evolving densities:

### Theorem 5.2.2: Mass Conservation Criterion

The flow-induced density  $p_t^{\text{fwd}}$  equals the prescribed path  $p_t$  for all  $t \in [0, 1]$ ; i.e.,

$$p_t^{\text{fwd}} = p_t, \quad \text{for all } t \in [0, 1],$$

if and only if the pair  $(p_t, \mathbf{v}_t)$  satisfies the continuity equation:

$$\partial_t p_t(\mathbf{x}) + \nabla_{\mathbf{x}} \cdot (p_t(\mathbf{x}) \mathbf{v}_t(\mathbf{x})) = 0,$$

for all  $t \in [0, 1]$  and  $\mathbf{x}$ .

### Proof for Theorem.

A conceptual derivation is provided in Section D.3.2, while a more rigorous treatment can be found in (Villani *et al.*, 2008) (see “Mass Conservation Formula”).

**From Conditional to Marginal Paths.** As seen in Section 5.2.2, we begin by defining a conditional probability path  $p_t(\cdot|\mathbf{z})$  and a corresponding conditional velocity field  $\mathbf{v}_t(\cdot|\mathbf{z})$ . We then construct the marginal velocity field via:

$$\mathbf{v}_t(\mathbf{x}) = \int \mathbf{v}_t(\mathbf{x}|\mathbf{z}) \frac{p_t(\mathbf{x}|\mathbf{z})\pi(\mathbf{z})}{p_t(\mathbf{x})} d\mathbf{z},$$

as in Equation (5.2.10). However, we still need to ensure that the resulting marginal velocity  $\mathbf{v}_t$  induces an ODE flow whose density path aligns with the prescribed  $p_t$ . Fortunately, this verification can be done entirely at the conditional level: if each conditional velocity field  $\mathbf{v}_t(\cdot|\mathbf{z})$  induces the conditional density path  $p_t(\cdot|\mathbf{z})$ , then the resulting marginal velocity  $\mathbf{v}_t$  also induces the correct marginal path. Formally, this is stated as follows:

### Proposition 5.2.3: Marginal VF Generates Given Marginal Density

If the conditional velocity fields  $\mathbf{v}_t(\cdot|\mathbf{z})$  induce conditional density paths that match  $p_t(\cdot|\mathbf{z})$  (starting from  $p_0(\cdot|\mathbf{z})$ ), then the marginal velocity field  $\mathbf{v}_t(\cdot)$  defined in Equation (5.2.10) induces a marginal density path that aligns with  $p_t(\cdot)$ , starting from  $p_0(\cdot)$ .

#### Proof for Proposition.

This result follows by verifying that the pair  $(p_t, \mathbf{v}_t)$  satisfies the Continuity Equation. We present the argument in a converse manner to provide intuition for why the marginalized velocity field takes the form in Equation (5.2.10). Since the conditional velocity fields  $\mathbf{v}_t(\cdot|\mathbf{z})$  induce density paths matching the conditional densities  $p_t(\cdot|\mathbf{z})$  for  $\mathbf{z} \sim \pi$ , the continuity equation holds for each conditional pair:

$$\frac{d}{dt} p_t(\mathbf{x}|\mathbf{z}) = -\nabla_{\mathbf{x}} \cdot (\mathbf{v}_t(\mathbf{x}|\mathbf{z}) p_t(\mathbf{x}|\mathbf{z})). \quad (5.2.13)$$

We aim to find a velocity field  $\mathbf{v}_t(\cdot)$  whose induced densities align with the marginal density  $p_t$ , i.e., satisfy

$$\frac{d}{dt} p_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot (\mathbf{v}_t(\mathbf{x}) p_t(\mathbf{x})). \quad (5.2.14)$$

Starting from the definition of  $p_t$  in Equation (5.2.7),

$$\begin{aligned}\frac{d}{dt}p_t(\mathbf{x}) &= \int \frac{d}{dt}p_t(\mathbf{x}_t|\mathbf{z})\pi(\mathbf{z}) d\mathbf{z} \\ &= - \int \nabla_{\mathbf{x}} \cdot (\mathbf{v}_t(\mathbf{x}|\mathbf{z})p_t(\mathbf{x}|\mathbf{z}))\pi(\mathbf{z}) d\mathbf{z} \\ &= -\nabla_{\mathbf{x}} \cdot \left( \int \mathbf{v}_t(\mathbf{x}|\mathbf{z})p_t(\mathbf{x}|\mathbf{z})\pi(\mathbf{z}) d\mathbf{z} \right),\end{aligned}$$

where the second equality follows by applying Equation (5.2.13). Comparing this with the right-hand side of Equation (5.2.14) shows that, up to a divergence-free term,

$$\mathbf{v}_t(\mathbf{x})p_t(\mathbf{x}) = \int \mathbf{v}_t(\mathbf{x}|\mathbf{z})p_t(\mathbf{x}|\mathbf{z})\pi(\mathbf{z}) d\mathbf{z}.$$

Therefore, we can define

$$\mathbf{v}_t(\mathbf{x}) := \int \mathbf{v}_t(\mathbf{x}|\mathbf{z}) \frac{p_t(\mathbf{x}|\mathbf{z})}{p_t(\mathbf{x})} \pi(\mathbf{z}) d\mathbf{z},$$

which is precisely the form in Equation (5.2.10). The proof of this theorem essentially follows the reverse of this argument. ■

This connection allows us to reduce the construction of the potentially intractable marginal velocity field to defining simpler conditional fields  $\mathbf{v}_t(\cdot|\mathbf{z})$ , which are easier to work with by construction.

### 5.3 Constructing Probability Paths and Velocities Between Distributions

The essence of flow matching lies in the gradual transformation of a source distribution into a target. To direct this transformation, two key elements are needed: the *probability path*  $p_t$ , which provides a snapshot of the evolving distribution at each time  $t$ , and the *velocity field*  $\mathbf{v}_t$ , which describes how individual particles move along the path. These two objects are not independent; they are linked through the continuity equation, which ensures that particle dynamics are consistent with the evolution of the distribution. Thus, the learning task reduces to finding a velocity field  $\mathbf{v}_t$  that faithfully drives the process. The difficulty, however, is that for general and complex distributions, the true marginal velocity  $\mathbf{v}_t$  is unknown, leaving us with an intractable target that cannot be accessed directly.

The core idea of Conditional Flow Matching is to address the intractability of the true marginal velocity by constructing an artificial but tractable process. To do this, we introduce a conditioning variable  $\mathbf{z}$  and design either a conditional velocity  $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$  and/or a conditional path  $p_t(\mathbf{x}_t|\mathbf{z})$ , which are deliberately chosen to be simple.

Because these conditional objects are known in closed form, they serve as surrogate targets that the model can regress against. This leads to a valid training loss  $\mathcal{L}_{\text{CFM}}$ , provided two practical requirements are met: (i) we can sample efficiently from  $p_t(\cdot|\mathbf{z})$ , and (ii) the corresponding velocity  $\mathbf{v}_t(\cdot|\mathbf{z})$  admits a closed-form expression.

How should we design a well behaved conditional process? For inspiration, we turn to the one case that is fully understood: the Gaussian to Gaussian bridge (Section 5.3.1). This example highlights two natural design strategies: adopt a Gaussian probability path at each time  $t$ , or prescribe an affine velocity field, both of which are analytically tractable.

Guided by this insight, we extend to general endpoint distributions with two complementary views (see also Section B.1.2) for constructing conditional paths and velocities:

- **Conditional Probability Path First (Eulerian View).** It begins with a conditional probability path  $p_t(\cdot|\mathbf{z})$  and derives the corresponding conditional velocity field.
- **Conditional Flow First (Lagrangian View).** It starts from a conditional flow  $\Psi_{0 \rightarrow t}(\cdot|\mathbf{z})$ , typically affine, and derives the conditional velocity field by differentiating with respect to time along trajectories.

In Section 5.3.2, we detail the first approach, which shows its close analogy to diffusion model construction discussed in Section 5.2.1, while in Section 5.3.3 we present the second. Together, these perspectives provide a practical framework for defining  $p_t(\mathbf{x}_t|\mathbf{z})$  and  $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$ , enabling simulation-free training and the construction of flows between arbitrary source and target distributions.

### 5.3.1 A Key Special Case: Marginal $p_t(\mathbf{x}_t)$ and Velocity $\mathbf{v}_t(\mathbf{x}_t)$ in the Gaussian-to-Gaussian Bridge

We begin with the Gaussian–endpoint case, where we can compute the marginal density  $p_t(\mathbf{x}_t)$  and velocity field  $\mathbf{v}_t(\mathbf{x}_t)$  analytically. This serves as a template for the general construction of the conditional density  $p_t(\mathbf{x}_t|\mathbf{z}_t)$  and velocity field  $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z}_t)$ .

When the source and target distributions,  $p_{\text{src}}$  and  $p_{\text{tgt}}$ , are both Gaussian, the velocity field  $\mathbf{v}_t(\cdot)$  admits a closed-form expression. We consider the interpolated marginal density path:

$$p_t(\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_t; \boldsymbol{\mu}(t), \sigma^2(t)\mathbf{I}), \quad (5.3.1)$$

with time-varying mean  $\boldsymbol{\mu}(t)$  and variance  $\sigma^2(t) > 0$ . The two endpoints are given by

$$p_{\text{src}} = p_0 = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}(0), \sigma^2(0)\mathbf{I}), \quad p_{\text{tgt}} = p_1 = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}(1), \sigma^2(1)\mathbf{I}),$$

so that the path  $\{p_t\}_{t \in [0,1]}$  connects these distributions.

With the given path  $\{p_t\}_{t \in [0,1]}$ , there are indeed many velocity fields that induce an ODE flow  $\Psi_{0 \rightarrow t}(\mathbf{x})$  such that  $\mathbf{x} \sim p_0$  implies  $\Psi_{0 \rightarrow t}(\mathbf{x}) \sim p_t$ . For this Gaussian path, a particularly simple realization is given by<sup>7</sup>:

$$\Psi_{0 \rightarrow t}(\mathbf{x}) := \boldsymbol{\mu}(t) + \sigma(t) \left( \frac{\mathbf{x} - \boldsymbol{\mu}(0)}{\sigma(0)} \right). \quad (5.3.2)$$

For the defined Gaussian path  $p_t$  (Gaussian for all  $t$ ), the velocity field  $\mathbf{v}_t(\cdot)$  inducing the ODE flow in Equation (5.3.2) is uniquely and analytically characterized as follows (Lipman *et al.*, 2022):

---

<sup>7</sup>In (Lipman *et al.*, 2022), the authors consider  $\Psi_{0 \rightarrow t}(\mathbf{x}) = \boldsymbol{\mu}(t) + \sigma(t)\mathbf{x}$ , which requires constraints on  $\boldsymbol{\mu}(t)$  and  $\sigma(t)$  to ensure boundary conditions. We adopt an equivalent normalized formulation that avoids such constraints.

**Proposition 5.3.1: Closed-Form Velocity Field for Gaussian Density Path**

Let  $p_t$  be the Gaussian path in Equation (5.3.1). Then the velocity field  $\mathbf{v}_t(\cdot)$  that generates the ODE flow Equation (5.3.2) is unique for the defined  $\Psi_{0 \rightarrow t}$  and has the closed-form expression:

$$\mathbf{v}_t(\mathbf{x}) = \frac{\sigma'(t)}{\sigma(t)} (\mathbf{x} - \boldsymbol{\mu}(t)) + \boldsymbol{\mu}'(t).$$

**Proof for Proposition.**

Consider the ODE with initial condition  $\mathbf{y}$ :

$$\frac{d}{dt} \Psi_{0 \rightarrow t}(\mathbf{y}) = \mathbf{v}_t(\Psi_{0 \rightarrow t}(\mathbf{y})).$$

Since  $\Psi_{0 \rightarrow t}$  is invertible (as  $\sigma(t) > 0$ ), we may set  $\mathbf{x} = \Psi_{0 \rightarrow t}(\mathbf{y})$  and  $\mathbf{y} = \Psi_{0 \rightarrow t}^{-1}(\mathbf{x}) = \Psi_{t \rightarrow 0}(\mathbf{x})$  to obtain

$$\Psi'_{0 \rightarrow t}(\Psi_{0 \rightarrow t}^{-1}(\mathbf{x})) = \mathbf{v}_t(\mathbf{x}).$$

Differentiating Equation (5.3.2) with respect to  $t$  gives

$$\Psi'_{0 \rightarrow t}(\mathbf{x}) = \boldsymbol{\mu}'(t) + \sigma'(t) \left( \frac{\mathbf{x} - \boldsymbol{\mu}(0)}{\sigma(0)} \right).$$

Solving for  $\mathbf{y} = \Psi_{0 \rightarrow t}^{-1}(\mathbf{x})$  yields

$$\mathbf{y} = \boldsymbol{\mu}(0) + \sigma(0) \left( \frac{\mathbf{x} - \boldsymbol{\mu}(t)}{\sigma(t)} \right).$$

Substituting this into  $\Psi'_{0 \rightarrow t}(\mathbf{x})$  gives

$$\mathbf{v}_t(\mathbf{x}) = \frac{\sigma'(t)}{\sigma(t)} (\mathbf{x} - \boldsymbol{\mu}(t)) + \boldsymbol{\mu}'(t),$$

as claimed. ■

We note that for a fixed flow map  $\Psi_{0 \rightarrow t}$  (flow-first view), the velocity is uniquely determined by

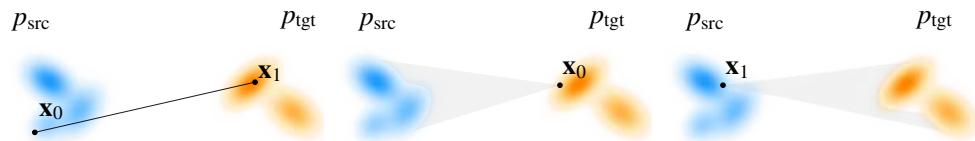
$$\mathbf{v}_t = \partial_t \Psi_{0 \rightarrow t} \circ \Psi_{0 \rightarrow t}^{-1}.$$

Under this construction, the pair  $(p_t, \mathbf{v}_t)$  automatically satisfies the continuity equation. By contrast, for a given density path  $t \mapsto p_t$  without fixing  $\Psi_{0 \rightarrow t}$  (probability-path-first view), the velocity field is not unique.

This distinction precisely characterizes the difference between the flow-first and probability-path-first perspectives.

This closed-form characterization remains valid when conditioning on a latent variable  $\mathbf{z}$ . In the following, we extend this insight to construct a conditional Gaussian path  $p_t(\cdot|\mathbf{z})$  and derive the corresponding conditional velocity field  $\mathbf{v}_t(\cdot|\mathbf{z})$  for the general marginal setting.

### 5.3.2 Conditional Probability-Path-First Construction of $\mathbf{v}_t(\cdot|\mathbf{z})$ and $p_t(\cdot|\mathbf{z})$



**Figure 5.6: Illustrations of two common types of conditioning probability paths.** It includes: (1) *two-sided*, conditioned on  $\mathbf{x}_0 \sim p_{\text{tgt}}$  and  $\mathbf{x}_1 \sim p_{\text{src}}$  with general endpoint distributions; (2) *one-sided*, conditioned at either  $\mathbf{x}_0 \sim p_{\text{tgt}}$  or  $\mathbf{x}_1 \sim p_{\text{src}}$ .

We aim to construct a conditional density path  $p_t(\cdot|\mathbf{z})$  first and then derive its corresponding conditional velocity field  $\mathbf{v}_t(\cdot|\mathbf{z})$  (via Proposition 5.3.1), under conditioning with respect to  $\pi(\mathbf{z})$ . Depending on how  $\mathbf{z}$  is chosen, there are two natural scenarios: (i) two-sided conditioning with  $\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1)$ , or (ii) one-sided conditioning with  $\mathbf{z} = \mathbf{x}_0$  or  $\mathbf{x}_1$ . In either case, the construction must match the boundary distributions:

$$p_{\text{src}}(\mathbf{x}_0) = \int p_0(\mathbf{x}_0|\mathbf{z})\pi(\mathbf{z}) d\mathbf{z}, \quad p_{\text{tgt}}(\mathbf{x}_1) = \int p_1(\mathbf{x}_1|\mathbf{z})\pi(\mathbf{z}) d\mathbf{z}.$$

Since verifying these constraints is straightforward once a concrete construction is specified, we do not emphasize the verification step here.

#### I. Two-Sided $\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1)$ — “Beam-Like” Path.

**Choice of  $\pi(\mathbf{z})$ .** Consider general distributions  $p_{\text{src}}$  and  $p_{\text{tgt}}$  over  $\mathbb{R}^D$ . Let  $\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1)$  with  $\mathbf{x}_0 \sim p_{\text{src}}$  and  $\mathbf{x}_1 \sim p_{\text{tgt}}$  independently, i.e.,

$$\pi(\mathbf{z}) = p_{\text{src}}(\mathbf{x}_0)p_{\text{tgt}}(\mathbf{x}_1).$$

**Choice of Conditional Path  $p_t(\cdot|\mathbf{z})$ .** Define the conditional path by linear interpolation with fixed variance  $\sigma > 0$ :

$$p_t(\mathbf{x}_t|\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1)) = \mathcal{N}(\mathbf{x}_t; a_t \mathbf{x}_0 + b_t \mathbf{x}_1, \sigma^2 \mathbf{I}),$$

where  $a_t$  and  $b_t$  are time-dependent functions satisfying  $a_0 = 1, b_0 = 0$  and  $a_1 = 0, b_1 = 1$ . A choice suggested by (Lipman *et al.*, 2022; Liu, 2022) is  $a_t = 1 - t, b_t = t$ . In the deterministic case  $\sigma = 0$ , we obtain

$$p_t(\mathbf{x}_t | \mathbf{z}) = \delta(\mathbf{x}_t - [a_t \mathbf{x}_0 + b_t \mathbf{x}_1]),$$

which describes a deterministic interpolating path from  $\mathbf{x}_0$  to  $\mathbf{x}_1$ .

**Derived Conditional Velocity  $\mathbf{v}_t(\cdot | \mathbf{z})$ .** By Proposition 5.3.1, the conditional velocity is

$$\mathbf{v}_t(\mathbf{x} | \mathbf{z}) = a'_t \mathbf{x}_0 + b'_t \mathbf{x}_1.$$

**CFM Loss.** When  $\sigma = 0$  so that  $\mathbf{x}_t = a_t \mathbf{x}_0 + b_t \mathbf{x}_1$ , the CFM loss reduces to

$$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t, \mathbf{x}_0 \sim p_{\text{src}}, \mathbf{x}_1 \sim p_{\text{tgt}}} \|\mathbf{v}_\phi(\mathbf{x}_t, t) - (a'_t \mathbf{x}_0 + b'_t \mathbf{x}_1)\|^2.$$

From Equations (5.2.10) and (5.2.11), the optimal velocity field is

$$\mathbf{v}^*(\mathbf{x}_t, t) = \mathbb{E} [\mathbf{x}'_t | \mathbf{x}_t] = \mathbb{E} [a'_t \mathbf{x}_0 + b'_t \mathbf{x}_1 | \mathbf{x}_t].$$

Here, the expectation is taken over  $p(\mathbf{x}_0, \mathbf{x}_1 | \mathbf{x}_t)$ , the conditional distribution over source-target pairs  $(\mathbf{x}_0, \mathbf{x}_1)$  that could have produced the observed interpolation  $\mathbf{x}_t = a_t \mathbf{x}_0 + b_t \mathbf{x}_1$  at time  $t$ .

**II. One-Sided  $\mathbf{z} = \mathbf{x}_0$  or  $\mathbf{x}_1$  — “Spotlight-Like” Path.** We illustrate the conditional probability-path-first construction in the one-sided setting, considering the standard generative setup with  $p_{\text{src}} = \mathcal{N}(\mathbf{0}, \mathbf{I})$  and  $p_{\text{tgt}} = p_{\text{data}}$ . Crucially, this Gaussian source is not an additional assumption but a direct consequence of the conditional path defined below. A more general treatment of arbitrary endpoints will be given in Section 5.3.3.

**Choice of  $\pi(\mathbf{z})$ .** We take  $\mathbf{z} = \mathbf{x}_1$  with  $\pi(\mathbf{z}) = p_{\text{data}}(\mathbf{x}_1)$  (the case  $\mathbf{z} = \mathbf{x}_0 \sim p_{\text{prior}}$  follows analogously).

**Choice of Conditional Path  $p_t(\cdot | \mathbf{z})$ .** For fixed  $\mathbf{x}_1 \sim p_{\text{data}}$ , define

$$p_t(\mathbf{x}_t | \mathbf{z} = \mathbf{x}_1) = \mathcal{N}(\mathbf{x}_t; b_t \mathbf{x}_1, a_t^2 \mathbf{I}),$$

with  $a_0 = 1, b_0 = 0, a_1 = 0, b_1 = 1$  (usually interpreted as the limit). At the boundaries,

$$p_0(\cdot | \mathbf{z} = \mathbf{x}_1) = \mathcal{N}(\cdot; \mathbf{0}, \mathbf{I}), \quad p_1(\cdot | \mathbf{z} = \mathbf{x}_1) = \delta(\cdot - \mathbf{x}_1).$$

Marginalizing over  $\mathbf{x}_1$  yields  $\{p_t\}_{t \in [0,1]}$  with  $p_0 = \mathcal{N}(\mathbf{0}, \mathbf{I})$  (independent of  $p_{\text{data}}$ ) and  $p_1 = p_{\text{data}}$ .

**Derived Conditional Velocity  $\mathbf{v}_t(\cdot|\mathbf{z})$ .** For  $t \in (0, 1)$  with  $b_t > 0$ , applying Proposition 5.3.1 to the conditional Gaussian path gives

$$\mathbf{v}_t(\mathbf{x}|\mathbf{x}_1) = b'_t \mathbf{x}_1 + \frac{a'_t}{a_t} (\mathbf{x} - b_t \mathbf{x}_1).$$

**One-Sided CFM Objective.** With  $t \sim \mathcal{U}(0, 1)$  (or any fixed sampling distribution) and  $\mathbf{x}_1 \sim p_{\text{data}}$ , the CFM loss becomes

$$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t, \mathbf{x}_1} \mathbb{E}_{\mathbf{x}_t \sim p_t(\cdot|\mathbf{x}_1)} \left\| \mathbf{v}_\phi(\mathbf{x}_t, t) - \left[ b'_t \mathbf{x}_1 + \frac{a'_t}{a_t} (\mathbf{x}_t - b_t \mathbf{x}_1) \right] \right\|_2^2. \quad (5.3.3)$$

By MSE optimality, the unique minimizer is the marginal velocity field

$$\mathbf{v}^*(\mathbf{x}, t) = \mathbb{E} [\mathbf{v}_t(\mathbf{x}|\mathbf{x}_1)|\mathbf{x}_t = \mathbf{x}] = \mathbb{E} [a'_t \mathbf{x}_0 + b'_t \mathbf{x}_1 | \mathbf{x}_t = \mathbf{x}].$$

**Equivalence to Two-Sided Target.** For paired samples  $(\mathbf{x}_0, \mathbf{x}_1)$  with  $\mathbf{x}_t = a_t \mathbf{x}_0 + b_t \mathbf{x}_1$ ,

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_1) = b'_t \mathbf{x}_1 + \frac{a'_t}{a_t} (\mathbf{x}_t - b_t \mathbf{x}_1) = a'_t \mathbf{x}_0 + b'_t \mathbf{x}_1.$$

Thus the one-sided loss regresses to the *conditional expectation* of the two-sided target given  $\mathbf{x}_t$ :

$$\mathbf{v}^*(\mathbf{x}, t) = \mathbb{E}[a'_t \mathbf{x}_0 + b'_t \mathbf{x}_1 | \mathbf{x}_t = \mathbf{x}],$$

so the one-sided and two-sided CFM objectives share the same minimizer.

**Gaussian FM = Diffusion Model.** We use the FM convention where  $t = 0$  denotes the source/prior and  $t = 1$  denotes the target/data:

$$p_{\text{src}} = p_{\text{prior}}, \quad p_{\text{tgt}} = p_{\text{data}}.$$

By contrast, diffusion models typically index time from data to noise (i.e.,  $t = 0$  is data and  $t = 1$  is prior). Here, we consistently adopt the FM convention indexing to avoid confusion. If further  $p_{\text{src}} = \mathcal{N}(\mathbf{0}, \mathbf{I})$ , then for fixed condition  $\mathbf{x}_1 \sim p_{\text{data}}$ , the conditional path  $p_t(\cdot|\mathbf{x}_1)$  is naturally chosen to be Gaussian, while the target distribution  $p_{\text{tgt}}$  itself need not be Gaussian. Some literature usually refer to this setting as *Gaussian FM*.

Choosing  $a_t = 1 - t$  and  $b_t = t$  (equivalently,  $\alpha_t = t$  and  $\sigma_t = 1 - t$  under the relabeling  $a_t := \sigma_t$ ,  $b_t := \alpha_t$  in diffusion model) recovers the familiar FM/RF schedule (Lipman *et al.*, 2022; Liu, 2022).

In the Gaussian FM setting, both the beam-like and spotlight-like conditional paths lead to training objectives that are similar to the standard diffusion losses. As we will elaborate in Chapter 6, Gaussian FM can in fact

**Table 5.2:** Summary of Different Interpolants written in FM convention  $\mathbf{x}_t = a_t \mathbf{x}_0 + b_t \mathbf{x}_1$ , where  $\mathbf{x}_0 \sim p_{\text{src}} = p_{\text{prior}}$ ,  $\mathbf{x}_1 \sim p_{\text{tgt}} = p_{\text{data}}$ . VE/VP are converted from their diffusion convention (data  $\rightarrow$  noise) via  $a_t := \sigma_t$ ,  $b_t := \alpha_t$ .

	<b>VE</b>	<b>VP</b>	<b>FM/RF</b>	<b>Trig.</b> (Albergo <i>et al.</i> , 2023)
$a_t$ (prior coeff.)	$a_t$	$\sqrt{1 - b_t^2}$	$1 - t$	$\cos\left(\frac{\pi}{2}t\right)$
$b_t$ (data coeff.)	1	$b_t$	$t$	$\sin\left(\frac{\pi}{2}t\right)$
$a_0$	0	0	1	1
$b_0$	1	1	0	0
$a_1$	$a_1$	1	0	0
$b_1$	1	0	1	1
$p_{\text{prior}}$	$\mathcal{N}(\mathbf{0}, a_1^2 \mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$	$\mathcal{N}(\mathbf{0}, \mathbf{I})$

be equivalently interpreted as a diffusion model trained to predict the *velocity*, under the linear schedule  $a_t = 1 - t$  and  $b_t = t$ . This perspective highlights that flow matching and diffusion are not fundamentally different, but rather two equivalent formulations that can be transformed into one another. The Gaussian FM objective is particularly appealing in practice: its loss function ( $\mathbb{E}_{t, \mathbf{x}_t} [\|\mathbf{v}_\phi(\mathbf{x}_t, t) - (\mathbf{x}_1 - \mathbf{x}_0)\|_2^2]$ ) is simple, and it has been shown to achieve competitive performance at scale (Esser *et al.*, 2024).

### Remark.

It is worth emphasizing that some prior works (Liu, 2022; Lipman *et al.*, 2022) suggest that adopting the canonical affine flow,  $a_t = 1 - t$  and  $b_t = t$ , yields a “straight-line” ODE trajectory enabling faster sampling. However, this claim does not hold in general. The velocity field in this formulation is given by the conditional expectation

$$\mathbf{v}(\mathbf{x}, t) = \mathbb{E}[\mathbf{x}_1 - \mathbf{x}_0 | \mathbf{X}_t = \mathbf{x}],$$

which depends on  $t$  and thus does not always align with the naive direction  $\mathbf{x}_1 - \mathbf{x}_0$ . In practice, the choice of time-weighting functions and parameterizations strongly influences training dynamics and can improve empirical performance, but such improvements cannot be attributed to the claimed “straightness” of the scheduler ( $a_t = 1 - t$ ,  $b_t = t$ ).

### 5.3.3 Conditional Flow-First Construction of $\mathbf{v}_t(\cdot|\mathbf{z})$ and $p_t(\cdot|\mathbf{z})$

We treat the general case where the endpoints  $p_{\text{src}}$  (at  $t = 0$ ) and  $p_{\text{tgt}}$  (at  $t = 1$ ) are arbitrary. Our goal is to design, directly in trajectory space, a conditional flow that transports samples from  $p_{\text{src}}$  to  $p_{\text{tgt}}$  and yields a closed-form  $\mathbf{v}_t(\mathbf{x}_t|\mathbf{z})$  usable as a regression target.

**Motivation.** Instead of first designing conditional density path, we may directly specify a conditional flow map  $\Psi_{0 \rightarrow t}(\cdot; \mathbf{z})$  that moves samples along trajectories. This has two practical advantages: (i) it immediately yields a regression target for training via a time derivative along trajectories; (ii) on geometry-structured spaces (Riemannian manifolds, Lie groups, or constrained submanifolds), it is often natural to construct the conditional flow map  $\Psi_{0 \rightarrow t}$  directly from the geometry (e.g., geodesics, exponential maps, or premetrics) (Lipman *et al.*, 2024) which yields analytic, simulation-free target velocities for training.

**Conditional Affine Flow (Link to Proposition 5.3.1).** We fix a conditioning variable  $\mathbf{z} \sim \pi$  (e.g.,  $\mathbf{z} = \mathbf{x}_1 \sim p_{\text{tgt}}$  in one-sided “spotlight” training) and push forward  $\mathbf{x}_0 \sim p_{\text{src}}$  through the time-varying *conditional affine flow*

$$\Psi_{0 \rightarrow t}(\mathbf{x}_0; \mathbf{z}) := \boldsymbol{\mu}_t(\mathbf{z}) + \mathbf{A}_t(\mathbf{z})\mathbf{x}_0, \quad t \in [0, 1],$$

where  $\boldsymbol{\mu}_t(\mathbf{z}) \in \mathbb{R}^D$  and  $\mathbf{A}_t(\mathbf{z}) \in \mathbb{R}^{D \times D}$  is invertible for  $t \in (0, 1)$ . The boundary  $\mathbf{A}_0(\mathbf{z}) = \mathbf{I}$ ,  $\boldsymbol{\mu}_0(\mathbf{z}) = \mathbf{0}$  recovers  $p_{\text{src}}$  at  $t = 0$ . It is standard to interpret boundary when  $t \rightarrow 1$  as a limit (the terminal map may concentrate mass on a lower-dimensional set or a point)<sup>8</sup>.

**Induced Conditional Path  $p_t(\cdot|\mathbf{z})$ .** The construction defines

$$p_t(\cdot|\mathbf{z}) = (\Psi_{0 \rightarrow t}(\cdot; \mathbf{z}))_{\#} p_{\text{src}}, \quad p_t(\cdot) = \int p_t(\cdot|\mathbf{z})\pi(\mathbf{z}) d\mathbf{z}.$$

What ultimately matters in  $\mathcal{L}_{\text{CFM}}$  is how to sample from it: first draw  $\mathbf{z} \sim \pi$ , then draw  $\mathbf{x}_0 \sim p_{\text{src}}$ , and finally set

$$\mathbf{x}_t = \boldsymbol{\mu}_t(\mathbf{z}) + \mathbf{A}_t(\mathbf{z})\mathbf{x}_0.$$

We remark that when  $\Psi_{0 \rightarrow t}$  is affine in  $\mathbf{x}_0$ , then  $p_t(\cdot|\mathbf{z})$  is Gaussian if and only if  $p_{\text{src}}$  is Gaussian. In particular, for arbitrary (non-Gaussian)  $p_{\text{src}}$ , an affine flow yields a generally non-Gaussian  $p_t(\cdot|\mathbf{z})$ .

---

<sup>8</sup>Allowing  $\mathbf{A}_1(\mathbf{z})$  to be singular (e.g.,  $\mathbf{0}$ ) is compatible with invertibility on  $(0, 1)$  and causes the path to contract onto the prescribed endpoint at  $t = 1$ .

**Derived Conditional Velocity  $\mathbf{v}_t(\cdot|\mathbf{z})$ .** The conditional velocity  $\mathbf{v}_t(\cdot|\mathbf{z})$  is obtained by  $t$ -differentiating the conditional flow map  $\Psi_{0 \rightarrow t}$ . Following the derivation in Proposition 5.3.1, consider the *conditional* ODE defined by the flow map  $\Psi_{0 \rightarrow t}(\mathbf{y}; \mathbf{z})$  with initial condition  $\mathbf{y}$ , where the goal is to identify the corresponding conditional velocity field  $\mathbf{v}_t(\cdot|\mathbf{z})$ :

$$\frac{d}{dt} \Psi_{0 \rightarrow t}(\mathbf{y}; \mathbf{z}) = \mathbf{v}_t \left( \underbrace{\Psi_{0 \rightarrow t}(\mathbf{y}; \mathbf{z})}_{\mathbf{x}} \middle| \mathbf{z} \right).$$

Since  $\Psi_{0 \rightarrow t}(\cdot; \mathbf{z})$  is invertible for  $t \in (0, 1)$ , we may express  $\mathbf{y}$  in terms of the current state  $\mathbf{x} := \Psi_{0 \rightarrow t}(\mathbf{y}; \mathbf{z})$  as  $\mathbf{y} = \Psi_{0 \rightarrow t}^{-1}(\mathbf{x}; \mathbf{z}) = \Psi_{t \rightarrow 0}(\mathbf{x}; \mathbf{z})$ . Substituting this into the ODE yields the following construction of the conditional velocity field:

$$\mathbf{v}_t(\mathbf{x}|\mathbf{z}) := \frac{d}{dt} \Psi_{0 \rightarrow t}(\Psi_{t \rightarrow 0}(\mathbf{x}; \mathbf{z}); \mathbf{z}),$$

which makes explicit that the derivative must be taken along the trajectory that reaches the spatial point  $\mathbf{x}$  at time  $t$ .

Since  $\mathbf{x}_t = \boldsymbol{\mu}_t(\mathbf{z}) + \mathbf{A}_t(\mathbf{z})\mathbf{x}_0$  and  $\mathbf{A}_t(\mathbf{z})$  is invertible on  $(0, 1)$ , we have  $\mathbf{x}_0 = \mathbf{A}_t(\mathbf{z})^{-1}(\mathbf{x} - \boldsymbol{\mu}_t(\mathbf{z}))$ , giving

$$\mathbf{v}_t(\mathbf{x}|\mathbf{z}) = \boldsymbol{\mu}'_t(\mathbf{z}) + \mathbf{A}'_t(\mathbf{z})\mathbf{A}_t(\mathbf{z})^{-1}(\mathbf{x} - \boldsymbol{\mu}_t(\mathbf{z})).$$

**One-Sided Conditioning ( $\mathbf{z} = \mathbf{x}_1$ ).** Choosing  $\boldsymbol{\mu}_t(\mathbf{z}) = b_t \mathbf{z}$  and  $\mathbf{A}_t(\mathbf{z}) = a_t \mathbf{I}$  with  $a_0 = 1, a_1 = 0$  and  $b_0 = 0, b_1 = 1$  (with  $a_t > 0$  for  $t \in (0, 1)$ ) yields

$$\mathbf{x}_t = a_t \mathbf{x}_0 + b_t \mathbf{x}_1, \quad \mathbf{v}_t(\mathbf{x}|\mathbf{x}_1) = b'_t \mathbf{x}_1 + \frac{a'_t}{a_t} (\mathbf{x} - b_t \mathbf{x}_1).$$

On paired samples  $(\mathbf{x}_0, \mathbf{x}_1)$  (with  $\mathbf{x}_t = a_t \mathbf{x}_0 + b_t \mathbf{x}_1$ ), this simplifies to the usual CFM target:

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_1) = a'_t \mathbf{x}_0 + b'_t \mathbf{x}_1.$$

**Two-Sided Conditioning ( $\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1)$ ).** The same template with  $\boldsymbol{\mu}_t(\mathbf{x}_0, \mathbf{x}_1) = b_t \mathbf{x}_1$  and  $\mathbf{A}_t(\mathbf{x}_0, \mathbf{x}_1) = a_t \mathbf{I}$  makes the conditional path *deterministic*:

$$\mathbf{x}_t = a_t \mathbf{x}_0 + b_t \mathbf{x}_1, \quad p_t(\cdot|\mathbf{x}_0, \mathbf{x}_1) = \delta(\cdot - (a_t \mathbf{x}_0 + b_t \mathbf{x}_1)),$$

and the conditional velocity is

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0, \mathbf{x}_1) = a'_t \mathbf{x}_0 + b'_t \mathbf{x}_1,$$

i.e., the standard two-sided CFM target.

**Unconditional Gaussian Path as a Special Case.** If  $\mu_t$  is independent of  $\mathbf{z}$  (denoted  $\mu(t)$ ) and  $\mathbf{A}_t = \frac{\sigma(t)}{\sigma(0)}\mathbf{I}$ , then

$$\Psi_{0 \rightarrow t}(\mathbf{x}_0) = \mu(t) + \sigma(t) \frac{\mathbf{x}_0 - \mu(0)}{\sigma(0)}, \quad \mathbf{v}_t(\mathbf{x}) = \mu'(t) + \frac{\sigma'(t)}{\sigma(t)}(\mathbf{x} - \mu(t)),$$

which recovers the Gaussian density path and the closed-form velocity in Proposition 5.3.1.

### 5.3.4 Probability-Path-First vs. Flow-First Construction

Both constructions aim to connect a source distribution  $p_{\text{src}}$  and a target distribution  $p_{\text{tgt}}$  through conditional dynamics. The *probability-path-first* (Eulerian) view begins by positing a conditional density path  $p_t(\cdot|\mathbf{z})$ , often chosen from Gaussian or affine families so that the associated velocity  $\mathbf{v}_t(\cdot|\mathbf{z})$  can be solved analytically. The *flow-first* (Lagrangian) view instead specifies a conditional flow map  $\Psi_{0 \rightarrow t}(\cdot|\mathbf{z})$  and obtains the velocity directly by differentiation along particle trajectories. While both yield equivalent transport under regularity, they differ in identifiability, ease of computation, and how endpoint constraints are enforced. The following table summarizes these contrasts. The takeaway: path-first is natural when conditional paths admit closed-form velocities; flow-first is natural when you have strong structural priors on trajectories.

Axis	Conditional Probability-Path-First	Conditional Flow-First
<b>Given</b>	Conditional density path $p_t(\cdot \mathbf{z})$ .	Conditional flow map $\Psi_{0 \rightarrow t}(\cdot \mathbf{z})$ (trajectories, for each fixed $\mathbf{z}$ ).
<b>Get Velocity</b>	For each $\mathbf{z}$ , find $\mathbf{v}_t(\cdot \mathbf{z})$ s.t. $\partial_t p_t(\cdot \mathbf{z}) + \nabla \cdot (p_t(\cdot \mathbf{z}) \mathbf{v}_t(\cdot \mathbf{z})) = 0;$ Non-unique: if $\nabla \cdot (p_t \mathbf{w}_t) = 0$ then $\mathbf{v}_t + \mathbf{w}_t$ yields the same $p_t$ .	Along paths (for each $\mathbf{z}$ ): $\mathbf{v}_t(\Psi_{0 \rightarrow t}(\cdot \mathbf{z}) \mathbf{z}) = \frac{d}{dt} \Psi_{0 \rightarrow t}(\cdot \mathbf{z}).$ When $\Psi_{0 \rightarrow t}$ is invertible, one can solve $\mathbf{v}_t(\mathbf{x} \mathbf{z}) = \frac{d}{dt} \Psi_{0 \rightarrow t}^{-1}(\Psi_{0 \rightarrow t}(\mathbf{x}) \mathbf{z}).$
<b>Closed Form of <math>\mathbf{v}_t(\cdot \mathbf{z})</math></b>	Convenient when $p_t(\cdot \mathbf{z})$ is Gaussian / exponential-family; otherwise obtaining $\mathbf{v}_t(\cdot \mathbf{z})$ is nontrivial.	Convenient when $\Psi_{0 \rightarrow t}(\cdot \mathbf{z})$ has structure (affine/low-rank); avoids density evaluation.
<b>Uniqueness of <math>\mathbf{v}_t(\cdot \mathbf{z})</math></b>	For each $\mathbf{z}$ , $\mathbf{v}_t(\cdot \mathbf{z})$ is underdetermined unless a selection rule (e.g., potential flow / min. kinetic energy) is imposed.	Given $\Psi_{0 \rightarrow t}(\cdot \mathbf{z})$ , both $p_t(\cdot \mathbf{z}) = (\Psi_{0 \rightarrow t}(\cdot \mathbf{z}))_# p_0$ and $\mathbf{v}_t(\cdot \mathbf{z})$ are determined; non-invertible maps still define $\mathbf{v}_t(\cdot \mathbf{z})$ along trajectories, while invertible ones make it unique.
<b>Realizability</b>	Must verify the constructed $\mathbf{v}_t(\cdot \mathbf{z})$ solving the continuity equation on the intended support.	Holds by construction: $p_t(\cdot \mathbf{z}) = (\Psi_{0 \rightarrow t}(\cdot \mathbf{z}))_# p_0(\cdot \mathbf{z}).$
<b>Match <math>(p_{\text{src}}, p_{\text{tgt}})</math></b>	Mix conditionals: $p_{\text{src}} = \int p_0(\cdot \mathbf{z}) \pi(\mathbf{z}) d\mathbf{z},$ $p_{\text{tgt}} = \int p_1(\cdot \mathbf{z}) \pi(\mathbf{z}) d\mathbf{z}.$ Under Gaussian-affine conditional paths with $\mathbf{z}$ -independent coefficients, $p_{\text{src}}$ can be forced to be Gaussian. For a general fixed endpoint $p_{\text{src}}$ (possibly non-Gaussian), the choice of $p_t(\cdot \mathbf{z})$ does not generally pin $p_{\text{src}}$ .	Set $\Psi_{0 \rightarrow 0} = \text{Id}$ and choose boundary condition to hit any $p_{\text{tgt}}$ .
<b>Preferred Scenarios</b>	Diffusion-style constructions; analytic targets via conditional Gaussians $p_t(\cdot \mathbf{z})$ .	Strong structural priors via maps $\Psi_{0 \rightarrow t}(\cdot \mathbf{z})$ ; easy boundary control; accommodates singular/low-dimensional endpoints; natural for map-based regularization/transport costs.

## 5.4 (Optional) Properties of the Canonical Affine Flow

Given two endpoint distributions  $p_0 = p_{\text{src}}$  and  $p_1 = p_{\text{tgt}}$ , a natural and widely used choice for defining the conditional path in flow matching (FM) (Lipman *et al.*, 2022) and rectified flow (RF) (Liu, Gong, *et al.*, 2022) is the linear interpolation

$$a_t = 1 - t, \quad b_t = t,$$

which yields the interpolant

$$\mathbf{x}_t = (1 - t)\mathbf{x}_0 + t\mathbf{x}_1, \quad \mathbf{x}_0 \sim p_{\text{src}}, \quad \mathbf{x}_1 \sim p_{\text{tgt}}.$$

Under this choice, the training objective simplifies to

$$\mathbb{E}_{t \sim \mathcal{U}[0,1]} \mathbb{E}_{\mathbf{x}_0, \mathbf{x}_1} \left[ \|\mathbf{v}_\phi(\mathbf{x}_t, t) - (\mathbf{x}_1 - \mathbf{x}_0)\|_2^2 \right].$$

This linear flow enjoys several appealing properties. In particular, it admits an iterative refinement scheme, known as *Reflow*, which progressively straightens the path between distributions while preserving the marginals.

### 5.4.1 Rectifying Flows: From Noisy Guesses to Structured Pairings

**From Noise to Data via Coherent Paths.** Take the generation task where  $p_{\text{src}}$  is the prior and  $p_{\text{tgt}}$  is the real data. We want a continuous path that transports noise to data. A naive tactic samples  $\mathbf{z}_0 \sim p_{\text{src}}$  and  $\mathbf{x}_1 \sim p_{\text{tgt}}$  independently, interpolates (e.g.  $\mathbf{x}_t = (1 - t)\mathbf{x}_0 + t\mathbf{x}_1$ ), and fits a velocity field to that line. This creates *incoherent pairings*: endpoints are unrelated across iterations, so trajectories fluctuate, variance explodes, convergence slows, and sample quality suffers.

**Why Independent Couplings Fall Short.** Conditional flow matching with independent draws uses

$$\pi(\mathbf{z}) = p_{\text{src}}(\mathbf{x}_0)p_{\text{tgt}}(\mathbf{x}_1),$$

or one-sided variants. Such couplings are sampling-friendly but induce jagged, high-variance paths that a velocity field struggles to model.

**Rectify the Flow via Dependent Coupling.** Rather than relying on arbitrary pairings, we use a pre-trained diffusion model  $\mathbf{v}_{\phi^\times}(\cdot, t)$  as the drift in a PF-ODE to *deterministically* transport each source point. Starting from  $\mathbf{z}(0) = \mathbf{z}_0 \sim p_{\text{src}}$ , we integrate

$$\frac{d\mathbf{z}(t)}{dt} = \mathbf{v}_{\phi^\times}(\mathbf{z}(t), t), \quad t \in [0, 1],$$

to obtain  $\hat{\mathbf{z}}_1 := \mathbf{z}(1)$  positioned near the data space learned from the pre-trained model. The resulting pair  $(\mathbf{z}_0, \hat{\mathbf{z}}_1)$  forms a *dependent coupling*: it follows a structured, model-guided path rather than an arbitrary interpolation. This idea extends naturally to affine reference paths of the form  $\mathbf{x}_t = a_t \mathbf{x}_0 + b_t \mathbf{x}_1$ , where  $\mathbf{x}_0 \sim p_{\text{src}}$  and  $\mathbf{x}_1 \sim p_{\text{tgt}}$ .

---

**Algorithm 2** Rectify Operation

---

**Input:** Reference path  $\{\mathbf{x}_t\}_{t \in [0,1]}$  (e.g.  $\mathbf{x}_t = a_t \mathbf{x}_0 + b_t \mathbf{x}_1$ )

1: **Pre-Train Diffusion.** Fit  $\mathbf{v}_{\phi^x}$  on the chosen path by minimizing

$$\phi^x \in \arg \min_{\phi} \mathbb{E}_{t, \mathbf{x}_0, \mathbf{x}_1} \left[ \left\| \mathbf{v}_{\phi}(\mathbf{x}_t, t) - \frac{d\mathbf{x}_t}{dt} \right\|_2^2 \right].$$

2: **Rectify.** Sample  $\mathbf{z}_0 \sim p_{\text{src}}$  and integrate

$$\frac{d\mathbf{z}(t)}{dt} = \mathbf{v}_{\phi^x}(\mathbf{z}(t), t), \quad \mathbf{z}(0) = \mathbf{z}_0, \quad t \in [0, 1],$$

to obtain  $\hat{\mathbf{z}}_1 = \mathbf{z}(1)$  and the trajectory  $\{\mathbf{z}(t)\}_{t \in [0,1]}$ .

**Output:** Dependent (coherent) pair  $(\mathbf{z}_0, \hat{\mathbf{z}}_1)$  or the full trajectory.

---

**Why It Works: Marginal-Preserving Structure.** Let  $\Phi_{0 \rightarrow t}$  denote the flow map generated by the above ODE defined by the pre-trained diffusion  $\mathbf{v}_{\phi^x}$ ; then  $\mathbf{z}(t) = \Phi_{0 \rightarrow t}(\mathbf{z}_0)$  and  $\hat{\mathbf{z}}_1 = \Phi_{0 \rightarrow 1}(\mathbf{z}_0)$ . The **Rectify** procedure pairs each source point with its flow endpoint, giving the deterministic joint

$$\pi_{\text{Rectify}}(\mathbf{z}_0, \mathbf{z}_1) = p_{\text{src}}(\mathbf{z}_0) \delta(\mathbf{z}_1 - \Phi_{0 \rightarrow 1}(\mathbf{z}_0)).$$

We have two immediate consequences:

- **Source Marginal is Preserved:**  $\int \pi_{\text{Rectify}}(\mathbf{z}_0, \mathbf{z}_1) d\mathbf{z}_1 = p_{\text{src}}(\mathbf{z}_0).$
- **Pushforward Along the Flow:**  $(\Phi_{0 \rightarrow t})_# p_{\text{src}} = \text{Law}(\mathbf{z}(t))$ , i.e., the time- $t$  distribution is the pushforward of  $p_{\text{src}}$  by  $\Phi_{0 \rightarrow t}$ .

If  $\mathbf{v}_{\phi^x}$  matches the oracle drift of a given reference path  $\mathbf{x}_t$ , then all intermediate marginals coincide:

$$\text{Law}(\mathbf{z}(t)) = \text{Law}(\mathbf{x}_t), \quad \text{for all } t \in [0, 1], \quad \text{and} \quad (\Phi_{0 \rightarrow 1})_# p_{\text{src}} = p_{\text{tgt}}.$$

**Summary.** Rectification replaces noisy independent pairings with smooth teacher-guided trajectories, lowering variance, easing optimization, and improving samples. The idea covers canonical linear paths  $\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{x}_1$  and general affine forms  $\mathbf{x}_t = a_t \mathbf{x}_0 + b_t \mathbf{x}_1$ .

For the canonical path, repeatedly applying **Rectify** (“*Reflow*”) further straightens trajectories without increasing transport cost, making training still easier.

### 5.4.2 Reflow: Iteratively Straightening Flows

**Why Reflow?** Independent pairings often induce irregular and meandering ODE trajectories between  $p_{\text{src}}$  and  $p_{\text{tgt}}$ , which increase discretization error and variance during simulation. This raises a natural question:

#### Question 5.4.1

Can we learn couplings that induce transport paths that are closer to straight lines between the two distributions, while still preserving the correct marginals?

This motivates *Reflow*: repeatedly apply **Rectify** to update the coupling so that successive flows become easier to integrate.

**Core Idea: Recursive Straightening via Rectify.** Start from the canonical interpolation on the product coupling  $\pi^{(0)} := p_{\text{src}}(\mathbf{x}_0)p_{\text{tgt}}(\mathbf{x}_1)$ ,

$$\mathbf{x}_t = t\mathbf{x}_0 + (1 - t)\mathbf{x}_1.$$

Applying **Rectify** replaces the independent pairing with a dependent one  $(\mathbf{z}_0, \hat{\mathbf{z}}_1)$ , which empirically induces *lower-curvature* trajectories under the learned field. Iterating this update progressively reduces path curvature (never forcing literal straight lines), improving numerical stability and alignment.

**The Reflow Procedure.** Each iteration performs two steps:

- **Re-Fit Flow:** Train a new velocity field from samples of the current coupling:

$$\phi_{k+1} = \arg \min_{\phi} \mathcal{L}(\phi | \pi^{(k)}), \quad \text{where}$$

$$\mathcal{L}(\phi | \pi^{(k)}) := \mathbb{E}_{t, (\mathbf{z}_0^{(k)}, \hat{\mathbf{z}}_1^{(k)}) \sim \pi^{(k)}} \left[ \left\| \mathbf{v}_{\phi}(\mathbf{z}_t, t) - (\hat{\mathbf{z}}_1^{(k)} - \mathbf{z}_0^{(k)}) \right\|^2 \right] \quad (5.4.1)$$

with  $\mathbf{z}_t = t\mathbf{z}_0^{(k)} + (1 - t)\hat{\mathbf{z}}_1^{(k)}$ .

- **Generate New Coupling:** Solve the learned ODE starting from new source samples  $\mathbf{z}_0^{(k+1)} \sim p_{\text{src}}$ :

$$\hat{\mathbf{z}}_1^{(k+1)} \leftarrow \mathbf{z}_0^{(k+1)} + \int_0^1 \mathbf{v}_{\phi_{k+1}}(\mathbf{z}(t), t) dt,$$

and define the updated coupling:

$$\pi^{(k+1)}(\mathbf{z}_0, \mathbf{z}_1) := p_{\text{src}}(\mathbf{z}_0) \delta \left( \mathbf{z}_1 - \hat{\mathbf{z}}_1^{(k+1)} \right).$$

In other words, `Reflow` can be viewed as repeatedly applying the `Rectify` operator, producing a sequence of progressively refined couplings:

$$\pi^{(k+1)} = \text{Rectify} \left( \pi^{(k)} \right) \quad (5.4.2)$$

so that both the flow and the coupling evolve together, yielding progressively more stable transport paths.

### 5.4.3 Properties of Reflow

Two key theoretical properties drive the usefulness of `Reflow`: it reduces transport cost and it straightens the trajectories.

**I. Reflow Never Increases Transport Cost.** Let  $c(\mathbf{y})$  be a convex cost function (e.g.,  $\|\mathbf{y}\|_2^p$  with  $p \geq 1$ ). Each `Rectify` step forms a new coupling  $(\mathbf{z}_0, \hat{\mathbf{z}}_1)$  whose cost is no worse than the original:

#### Proposition 5.4.1: Rectify May Reduce Transport Costs

Assuming an ideal velocity field  $\mathbf{v}^* = \mathbf{v}_{\phi^x}$ , we have:

$$\mathbb{E} [c(\hat{\mathbf{z}}_1 - \mathbf{z}_0)] \leq \mathbb{E} [c(\mathbf{x}_1 - \mathbf{x}_0)].$$

#### Proof for Proposition.

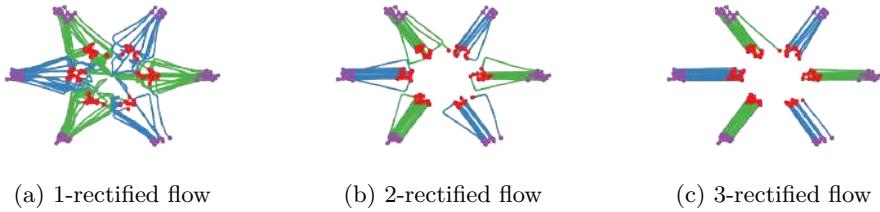
Follows from Jensen's inequality. See Liu, Gong, *et al.* (2022) for a full derivation. ■

Applying this result recursively shows that the `Reflow` process does not increase the transport cost.

**II. Reflow Straightens the Path.** The longer we iterate `Reflow`, the straighter the learned trajectories may become. To measure this, define the *straightness functional* of a path  $\mathbf{Y} = \{\mathbf{y}_t\}_{t \in [0,1]}$  as

$$\mathcal{S}(\mathbf{Y}) := \int_0^1 \mathbb{E} \left[ \left\| \mathbf{y}_1 - \mathbf{y}_0 - \frac{d\mathbf{y}_t}{dt} \right\|_2^2 \right] dt.$$

If  $\mathcal{S}(\mathbf{Y}) = 0$ , then  $\mathbf{Y}$  is exactly a straight line.



**Figure 5.7: Illustration of Reflow from Liu, Gong, et al. (2022).** Paths become progressively straighter with Rectify procedure.

#### Proposition 5.4.2: Reflow Straightens the Stochastic Path

For rectified paths  $\mathbf{Z}^{(k)}$ , we have:

$$\min_{k \in \{0, \dots, K\}} \mathcal{S}(\mathbf{Z}^{(k)}) \leq \frac{\mathbb{E} [\|\mathbf{x}_1 - \mathbf{x}_0\|^2]}{K}.$$

#### Proof for Proposition.

See Theorem 3.7 of Liu (2022).

The FM or RF formulation with linear interpolation kernels, together with the Reflow procedure, provides a simpler training objective and a practical method for refining stochastic couplings. For theoretical details, we refer readers to (Liu, Gong, et al., 2022; Liu, 2022).

**III. Connection to Optimal Transport.** Lastly, we note that straight-line couplings are not necessarily optimal in the sense of optimal transport (OT). This involves some terminology that will be introduced in Section 7.2; we therefore refer readers who are not familiar with OT to that section.

A hallmark of quadratic-cost optimal transport is that particles travel along straight lines: a particle at  $\mathbf{x}_0$  moves to  $\mathbf{T}(\mathbf{x}_0)$  via  $\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{T}(\mathbf{x}_0)$ , where  $\mathbf{T}$  is the optimal transport map. However, not every map  $\mathbf{S}$  generating such straight-line paths, i.e.,  $\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{S}(\mathbf{x}_0)$ , is optimal. The map  $\mathbf{S}$  yields the optimal flow only if it minimizes the Monge cost  $\mathbb{E}[\|\mathbf{x}_0 - \mathbf{S}(\mathbf{x}_0)\|^2]$ . Thus, while straight-line paths are necessary, they are not sufficient; optimality also depends on the correct endpoint map  $\mathbf{T}$ .

#### Example: Straight Couplings Need Not Be Optimal

Let  $p_{\text{src}} = p_{\text{tgt}} = \mathcal{N}(\mathbf{0}, \mathbf{I})$ . For the cost  $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^p$  with  $p > 0$ , the  $c$ -optimal coupling is the identity coupling  $\pi^*$ , where  $\pi^*$  is the law of  $(\mathbf{x}, \mathbf{x})$  with  $\mathbf{x} \sim p_{\text{src}}$ .

Now consider the coupling  $\pi_{\mathbf{A}}$  defined as the law of  $(\mathbf{x}, \mathbf{Ax})$ , where  $\mathbf{x} \sim p_{\text{src}}$  and  $\mathbf{A}$  is a rotation matrix satisfying  $\mathbf{A}^\top \mathbf{A} = \mathbf{I}$ ,  $\det(\mathbf{A}) = 1$ ,  $\mathbf{A} \neq \mathbf{I}$ , and  $-1$  is not an eigenvalue. Then  $\pi_{\mathbf{A}}$  is a valid coupling of  $p_{\text{src}}$  and  $p_{\text{tgt}}$ , and corresponds to straight-line paths between  $\mathbf{x}$  and  $\mathbf{Ax}$ , but it is not  $c$ -optimal for any twice-differentiable strictly convex cost  $c$  with invertible Hessian. The suboptimality arises from the rotational transformation. As discussed in Equation (7.5.2), even removing the rotation may not lead to an optimal coupling. ■

We will continue exploring the connection to OT in Section 7.5.2.

## 5.5 Closing Remarks

This chapter has illuminated the third and final foundational perspective on diffusion models, one rooted in the principles of deterministic flows. Our exploration began with Normalizing Flows (NFs), which leverage the change-of-variables formula to learn an exact, invertible mapping between a simple prior and the data distribution. We then saw this concept evolve into a continuous-time process with Neural ODEs, where a learned velocity field governs the transformation. However, this approach comes with the significant drawback of requiring costly ODE simulations within the training loop.

The modern framework of Flow Matching (FM) was presented as an elegant and efficient solution to this challenge. By pre-defining a probability path  $\{p_t\}_t$  and a corresponding velocity field that satisfies the continuity equation, FM establishes a clear target for the ODE flow. Crucially, just as we saw in the variational and score-based views, FM employs a powerful conditioning trick. This transforms the intractable problem of matching the marginal velocity field into a simple and tractable regression against a known conditional velocity, making training entirely simulation-free. This perspective recasts diffusion models themselves as a special case of learning a deterministic flow to transport a Gaussian prior to the data distribution.

With the introduction of the flow-based view, our survey of the three conceptual pillars of diffusion modeling is now complete. Throughout this journey, a remarkable pattern has emerged: each framework, despite its unique origins in VAEs, EBMs, or NFs, has converged on a continuous-time generative process and has relied on a conditioning strategy to enable tractable learning.

In the next chapter, we will finally synthesize these parallel threads into a single, unified framework. We will:

1. Formally demonstrate that the variational, score-based, and flow-based perspectives are not merely analogous but are mathematically equivalent at a fundamental level.
2. Show how the Fokker-Planck equation serves as the universal law governing density evolution across all three views, revealing that they are simply different lenses for describing the same core generative principle.

This unified lens will provide a complete and systematic understanding of the modern diffusion paradigm.

# 6

---

## A Unified and Systematic Lens on Diffusion Models

---

*Mathematics is the art of giving the same name to different things.*

---

Henri Poincaré

This chapter presents a systematic viewpoint that connects the variational, score based, and flow based perspectives within a coherent picture. While motivated by different intuitions, these approaches converge on the same core mechanism underlying modern diffusion methods. Building on Chapters 2 to 5, we observe a common recipe: define a forward corruption process that traces a path of marginals, then learn a time varying vector field that transports a simple prior to the data distribution along this path.

A key ingredient across all perspectives is the conditioning trick introduced in Section 6.1, which transforms an intractable marginal objective into a tractable conditional one, leading to stable and efficient training.

In Section 6.2 we analyze the training objective in a systematic way, identifying its essential components and clarifying how loss functions are formulated in the variational, score-based, and flow-based viewpoints.

Section 6.3 shows that any affine forward noise injection of the form  $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$  can be equivalently transformed into the standard linear schedule  $\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\boldsymbol{\epsilon}$ . Moreover, common parameterizations such as noise prediction, clean data prediction, score prediction, and velocity prediction are interchangeable at the level of gradients. Thus, the choices of noise schedulers and parameterizations both adhere to the same modeling principle.

Finally, Section 6.4 brings the discussion together and identifies the governing rule: the Fokker–Planck equation. Whether viewed as a variational scheme (discrete time denoising), a score-based method (SDE formulation), or a flow-based method (ODE formulation), each constructs a generator whose marginals follow the same density evolution. The Fokker–Planck equation thus serves as the universal constraint respected by all three viewpoints, with differences arising only in parameterization and training objectives.

## 6.1 Conditional Tricks: The Secret Sauce of Diffusion Models

Until now, we have explored diffusion models from three seemingly distinct origins: variational, score-based, and flow based perspectives. Each was originally motivated by different goals and led to its own training objectives (with a fixed  $t$ ):

- **Variational View:** Learn a parametrized density  $p_\phi(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t)$  to approximate the oracle reverse transition  $p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t)$  by minimizing:

$$\mathcal{J}_{\text{KL}}(\phi) := \mathbb{E}_{p_t(\mathbf{x}_t)} [\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t) \| p_\phi(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t))] ;$$

- **Score-Based View:** Learn a score model  $\mathbf{s}_\phi(\mathbf{x}_t, t)$  to approximate the marginal score  $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)$  via:

$$\mathcal{J}_{\text{SM}}(\phi) := \mathbb{E}_{p_t(\mathbf{x}_t)} \left[ \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\|_2^2 \right] ;$$

- **Flow-Based View:** Learn a velocity model  $\mathbf{v}_\phi(\mathbf{x}_t, t)$  to match the oracle velocity  $\mathbf{v}_t(\mathbf{x}_t)$  (e.g., defined by Equation (5.2.10)) by minimizing:

$$\mathcal{J}_{\text{FM}}(\phi) := \mathbb{E}_{p_t(\mathbf{x}_t)} \left[ \|\mathbf{v}_\phi(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t)\|_2^2 \right] .$$

At first glance, these objectives seem hopelessly intractable, since they all require access to oracle quantities that are fundamentally unknowable in general. But here comes the exciting twist: each method independently arrives at the same elegant solution to this problem: *conditioning on the data  $\mathbf{x}_0$* . This technique transforms each intractable training target into a tractable one.

This elegant “conditioning technique” rewrites the objectives as expectations over the known Gaussian conditionals  $p_t(\mathbf{x}_t|\mathbf{x}_0)$ , yielding gradient-equivalent closed-form regression targets and tractable training objectives:

- **Variational View** (Equation (2.2.3)):

$$\mathcal{J}_{\text{KL}}(\phi) = \underbrace{\mathbb{E}_{\mathbf{x}_0} \mathbb{E}_{p_t(\mathbf{x}_t|\mathbf{x}_0)} [\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t, \mathbf{x}_0) \| p_\phi(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t))] + C}_{\mathcal{J}_{\text{CKL}}(\phi)} ;$$

- **Score-Based View** (Equation (3.3.3)):

$$\mathcal{J}_{\text{SM}}(\phi) = \underbrace{\mathbb{E}_{\mathbf{x}_0} \mathbb{E}_{p_t(\mathbf{x}_t|\mathbf{x}_0)} \left[ \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)\|_2^2 \right]}_{\mathcal{J}_{\text{DSM}}(\phi)} + C ;$$

- **Flow-Based View** (Equation (5.2.9)):

$$\mathcal{J}_{\text{FM}}(\phi) = \underbrace{\mathbb{E}_{\mathbf{x}_0} \mathbb{E}_{p_t(\mathbf{x}_t|\mathbf{x}_0)} [\|\mathbf{v}_\phi(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0)\|^2]}_{\mathcal{J}_{\text{CFM}}(\phi)} + C.$$

To build a unified view, we next revisit the conditional KL, score, and velocity objectives in a systematic manner. Crucially, these objectives are not only tractable but also equivalent to their original forms up to a constant vertical shift. The conditional versions ( $\mathcal{J}_{\text{CKL}}$ ,  $\mathcal{J}_{\text{DSM}}$ ,  $\mathcal{J}_{\text{CFM}}$ ) differ from the originals ( $\mathcal{J}_{\text{KL}}$ ,  $\mathcal{J}_{\text{SM}}$ ,  $\mathcal{J}_{\text{FM}}$ ) only by this shift, which leaves the gradients unchanged and thus preserves the optimization landscape. As a result, the minimizers remain uniquely identified with the true oracle targets, since each reduces to a least-squares regression problem whose solution recovers the corresponding conditional expectation:

$$\begin{aligned} p^*(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t) &= \mathbb{E}_{\mathbf{x}_0 \sim p(\cdot|\mathbf{x}_t)} [p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t, \mathbf{x}_0)] &= p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t), \\ \mathbf{s}^*(\mathbf{x}_t, t) &= \mathbb{E}_{\mathbf{x}_0 \sim p(\cdot|\mathbf{x}_t)} [\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)] &= \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t), \quad (6.1.1) \\ \mathbf{v}^*(\mathbf{x}_t, t) &= \mathbb{E}_{\mathbf{x}_0 \sim p(\cdot|\mathbf{x}_t)} [\mathbf{v}_t(\mathbf{x}_t|\mathbf{x}_0)] &= \mathbf{v}_t(\mathbf{x}_t). \end{aligned}$$

This is no coincidence: by making training tractable, these conditional forms reveal a profound unification. Variational diffusion, score-based SDEs, and flow matching are simply different facets of the same principle. Three perspectives, one insight, elegantly connected.

We will continue to explore their equivalence throughout the rest of this chapter.

## 6.2 A Roadmap for Elucidating Training Losses in Diffusion Models

This section builds a systematic view of training losses in diffusion models. In Section 6.2.1, we extend the standard three objectives to a broader set of four parameterizations, showing how they arise from different modeling perspectives. In Section 6.2.2, we then distill these results into a general framework that disentangles the structure of diffusion objectives, laying the groundwork for the equivalence results in Section 6.3.

### 6.2.1 Four Common Parameterizations in Diffusion Models

Throughout this section, we consider the forward perturbation kernel

$$p_t(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N} \left( \mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I} \right),$$

where  $\mathbf{x}_0 \sim p_{\text{data}}$ , as defined in Equation (4.4.1), unless stated otherwise.

Let  $\omega : [0, T] \rightarrow \mathbb{R}_{>0}$  denote a positive time-weighting function. The four standard parameterizations (noise  $\epsilon_\phi$ , clean  $\mathbf{x}_\phi$ , score  $\mathbf{s}_\phi$ , and velocity  $\mathbf{v}_\phi$ ), together with their respective minimizers  $\epsilon^*$ ,  $\mathbf{x}^*$ ,  $\mathbf{s}^*$ , and  $\mathbf{v}^*$ , are summarized below for clarity and to facilitate further discussion.

**Variational View.** Based on the KL divergence in DDPMs (see Sections 2.2.4 and 4.4.3), this approach reduces to predicting either the expected noise that produces  $\mathbf{x}_t$  or the expected clean signal that  $\mathbf{x}_t$  was perturbed from.

1.  **$\epsilon$ -Prediction (Noise Prediction)** (Ho *et al.*, 2020):

$$\epsilon_\phi(\mathbf{x}_t, t) \approx \mathbb{E}[\epsilon | \mathbf{x}_t] = \epsilon^*(\mathbf{x}_t, t) \quad (6.2.1)$$

with training objective

$$\mathcal{L}_{\text{noise}}(\phi) := \mathbb{E}_t \left[ \omega(t) \mathbb{E}_{\mathbf{x}_0, \epsilon} \|\epsilon_\phi(\mathbf{x}_t, t) - \epsilon\|_2^2 \right].$$

Here,  $\epsilon^*$  means the average noise that was injected to obtain the given  $\mathbf{x}_t$ .

2.  **$\mathbf{x}$ -Prediction (Clean Prediction)** (Kingma *et al.*, 2021; Karras *et al.*, 2022; Song *et al.*, 2023):

$$\mathbf{x}_\phi(\mathbf{x}_t, t) \approx \mathbb{E}[\mathbf{x}_0 | \mathbf{x}_t] = \mathbf{x}^*(\mathbf{x}_t, t) \quad (6.2.2)$$

with training objective

$$\mathcal{L}_{\text{clean}}(\phi) := \mathbb{E}_t \left[ \omega(t) \mathbb{E}_{\mathbf{x}_0, \epsilon} \|\mathbf{x}_\phi(\mathbf{x}_t, t) - \mathbf{x}_0\|_2^2 \right].$$

Here,  $\mathbf{x}^*$  means the average of all plausible clean guesses, given the noisy observation  $\mathbf{x}_t$ .

**Score-Based View.** Predicts the score function at noise level  $t$ , which points in the average direction to denoise  $\mathbf{x}_t$  back toward all possible clean samples that could have generated it:

3. **Score Prediction** (Song and Ermon, 2019; Song *et al.*, 2020c):

$$\mathbf{s}_\phi(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \mathbb{E} [\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0) | \mathbf{x}_t] = \mathbf{s}^*(\mathbf{x}_t, t) \quad (6.2.3)$$

with training objective

$$\mathcal{L}_{\text{score}}(\phi) := \mathbb{E}_t \left[ \omega(t) \mathbb{E}_{\mathbf{x}_0, \epsilon} \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)\|_2^2 \right],$$

where the conditional score satisfies  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0) = -\frac{1}{\sigma_t} \boldsymbol{\epsilon}$ .

**Flow-Based View.** Predicts the instantaneous average velocity of the data as it evolves through  $\mathbf{x}_t$ :

4. **v-Prediction (Velocity Prediction)** (Lipman *et al.*, 2022; Liu, 2022; Salimans and Ho, 2021; Albergo *et al.*, 2023):

$$\mathbf{v}_\phi(\mathbf{x}_t, t) \approx \mathbb{E} \left[ \frac{d\mathbf{x}_t}{dt} \middle| \mathbf{x}_t \right] = \mathbf{v}^*(\mathbf{x}_t, t) \quad (6.2.4)$$

with training objective

$$\mathcal{L}_{\text{velocity}}(\phi) := \mathbb{E}_t \left[ \omega(t) \mathbb{E}_{\mathbf{x}_0, \epsilon} \|\mathbf{v}_\phi(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t | \mathbf{x}_0, \epsilon)\|_2^2 \right],$$

where the conditional velocity is  $\mathbf{v}_t(\mathbf{x}_t | \mathbf{x}_0, \epsilon) = \alpha'_t \mathbf{x}_0 + \sigma'_t \boldsymbol{\epsilon}$ .

Here,  $\mathbf{v}^*$  indicates the average velocity vector passing through the observation point  $\mathbf{x}_t$ .

Building on the insight from Equation (6.1.1), all four prediction types ultimately aim to approximate a conditional expectation in the form of the average noise, clean data, score, or velocity given an observed  $\mathbf{x}_t$ .

## 6.2.2 Disentangling the Training Objective of Diffusion Models

As shown in Section 6.2.1, the objective functions for the four prediction types commonly share the following template form for diffusion model training:

$$\mathcal{L}(\phi) := \mathbb{E}_{\mathbf{x}_0, \epsilon} \underbrace{\mathbb{E}_{p_{\text{time}}(t)}}_{\text{time distribution}} \left[ \underbrace{\omega(t)}_{\text{time weighting}} \underbrace{\left\| \text{NN}_\phi(\mathbf{x}_t, t) - (A_t \mathbf{x}_0 + B_t \boldsymbol{\epsilon}) \right\|_2^2}_{\text{MSE part}} \right]. \quad (6.2.5)$$

Here, to enhance training efficiency and optimize the diffusion model learning pipeline, several key design choices are crucial (Karras *et al.*, 2022; Lu and Song, 2024):

- (A) Noise schedule in the forward process of  $\mathbf{x}_t$  via  $\alpha_t$  and  $\sigma_t$ ;
- (B) Prediction types of  $\text{NN}_\phi$  and their associated regression targets  $(A_t \mathbf{x}_0 + B_t \boldsymbol{\epsilon})$ ;
- (C) Time-weighting function  $\omega(\cdot) : [0, T] \rightarrow \mathbb{R}_{\geq 0}$ ;
- (D) Time distribution  $p_{\text{time}}$ .

We elaborate on these four components here to serve as a roadmap for the discussions in the following sections.

**(A) Noise Schedule  $\alpha_t$  and  $\sigma_t$ .** Users have the flexibility to choose schedules tailored to their applications, with common examples summarized in Table 5.2. Importantly, as we will demonstrate in Equations (6.3.3) and (6.3.5), all affine flows of the form  $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$  are mathematically equivalent. Specifically, any such interpolation can be converted to the canonical linear schedule ( $\alpha_t = 1 - t$ ,  $\sigma_t = t$ ) or to a trigonometric schedule ( $\alpha_t = \cos t$ ,  $\sigma_t = \sin t$ ) by appropriate time reparametrization and spatial rescaling.

**(B) Parameterization  $\text{NN}_\phi$  and Training Target  $A_t \mathbf{x}_0 + B_t \boldsymbol{\epsilon}$ .** Users can flexibly choose the model’s prediction target: the clean signal, noise, score, or velocity prediction. As detailed in Section 6.2.1, all these prediction types share a common regression target of the form

$$\text{Regression Target} = A_t \mathbf{x}_0 + B_t \boldsymbol{\epsilon},$$

where the coefficients  $A_t$  and  $B_t$  depend on both the chosen prediction type and the schedule  $(\alpha_t, \sigma_t)$ . These relationships are summarized in Table 6.1.

Although these four parameterizations appear distinct, we will demonstrate in Equation (6.3.1) that they can be transformed into one another through simple algebraic manipulations. Furthermore, we will also show in Equation (6.3.6) that the squared- $\ell_2$  loss term in Equation (6.2.5) remains gradient-equivalent across all prediction types, differing only by a time-weighting factor (beyond  $\omega(t)p_{\text{time}}(t)$ ) that depends solely on the noise schedule  $(\alpha_t, \sigma_t)$ .

**Table 6.1:** Summary of the Relationships Between Different Parameterizations. All four parameterizations are mathematically equivalent and can be converted into one another through straightforward algebraic transformations.

Regression Target =	$A_t \mathbf{x}_0 + B_t \epsilon$	
	$A_t$	$B_t$
Clean	1	0
Noise	0	1
Conditional Score	0	$-\frac{1}{\sigma_t}$
Conditional Velocity	$\alpha'_t$	$\sigma'_t$

**(C) Time Distribution  $p_{\text{time}}(t)$ .** Since the training loss is an expectation over  $t$ , sampling times from  $p_{\text{time}}(t)$  is mathematically equivalent to weighting the per- $t$  MSE by  $p_{\text{time}}(t)$ ; this factor can be absorbed into the existing time weighting  $\omega(t)$ <sup>1</sup>. However, empirical evidence<sup>2</sup> indicates that different choices of  $p_{\text{time}}(t)$  can affect performance. Therefore, we discuss the time distribution  $p_{\text{time}}(t)$  and the time weighting function  $\omega(t)$  separately.

A common choice for the time distribution is the uniform distribution over  $[0, T]$  (Ho *et al.*, 2020; Song *et al.*, 2020c; Lipman *et al.*, 2022; Liu, 2022). Alternative options include the log-normal distribution (Karras *et al.*, 2022) and adaptive importance sampling methods (Song *et al.*, 2021; Kingma *et al.*, 2021).

**(D) Time-Weighting Function  $\omega(t)$ .** A common choice for the weighting function is the constant weighting  $\omega \equiv 1$  (Ho *et al.*, 2020; Karras *et al.*, 2022; Lipman *et al.*, 2022; Liu, 2022), although adaptive weighting schemes have also been proposed (Karras *et al.*, 2023). Certain choices of  $\omega(t)$  transform

---

<sup>1</sup>Our target population objective over time is an integral of the form

$$\mathcal{L} = \int_0^T \omega(t) \mathbf{mse}(t) dt,$$

where  $\mathbf{mse}(t)$  denotes the per- $t$  MSE-like term. If we draw  $t \sim p_{\text{time}}(t)$  during training, an unbiased Monte Carlo estimator of  $\mathcal{L}$  is obtained by

$$\hat{\mathcal{L}} = \mathbb{E}_{t \sim p_{\text{time}}} \left[ \frac{\omega(t)}{p_{\text{time}}(t)} \mathbf{mse}(t) \right],$$

i.e., sampling and weighting are interchangeable via importance weighting.

<sup>2</sup>In practice, though, we approximate the training objective using minibatch SGD on a discrete set of times. Under this approximation, different choices of  $p_{\text{time}}(t)$  change both the variance of the gradients and the effective weight placed on each time step. For this reason we discuss  $p_{\text{time}}(t)$  (sampling) and  $\omega(t)$  (weighting) separately.

Equation (6.2.5) into a tighter upper bound on the negative log-likelihood, effectively reformulating the objective as maximum likelihood training. Notable weighting schemes for  $\omega(t)$  include setting  $\omega(t) = g^2(t)$  (Song *et al.*, 2021), where  $g$  is the diffusion coefficient from the forward SDE in Equation (4.1.3). Other approaches use signal-to-noise ratio (SNR) weighting (Kingma *et al.*, 2021) or monotonic weighting functions (Kingma and Gao, 2023), where  $\omega(t)$  is a monotone function of time.

Overall, regardless of the choice of noise scheduler, prediction type, or time sampling distribution, these factors theoretically converge to influencing the time-weighting in the objective functions. This time-weighting can impact the practical training landscape and, consequently, the model's performance.

## 6.3 Equivalence in Diffusion Models

The four prediction types introduced in Section 6.2.1 will later be shown (Section 6.3.1) to be equivalent under gradient minimization. We then broaden this view in Section 6.3.3, showing that different forward noise schedules are connected by simple time and space rescalings.

### 6.3.1 Four Prediction Types Are Equivalent

We begin by analyzing the design choices for component (B) in Equation (6.2.5).

We have seen that the four prediction types are not independent choices but different views of the same underlying quantity. For example, noise and clean predictions are directly related (Section 2.2.4), as are score and noise predictions (Section 3.4). This recurring pattern points to a deeper principle: all four parameterizations are algebraically equivalent and can be converted into one another through simple transformations. To make this connection precise, we state the following proposition, illustrated in Figure 6.1, following (Kingma *et al.*, 2021).

#### Proposition 6.3.1: Equivalence of Parametrizations

Let the optimal predictions minimizing their respective objectives be

$$\epsilon^*(\mathbf{x}_t, t), \quad \mathbf{x}^*(\mathbf{x}_t, t), \quad \mathbf{s}^*(\mathbf{x}_t, t), \quad \mathbf{v}^*(\mathbf{x}_t, t),$$

corresponding to noise, clean, score, and velocity parameterizations. These satisfy the following equivalences:

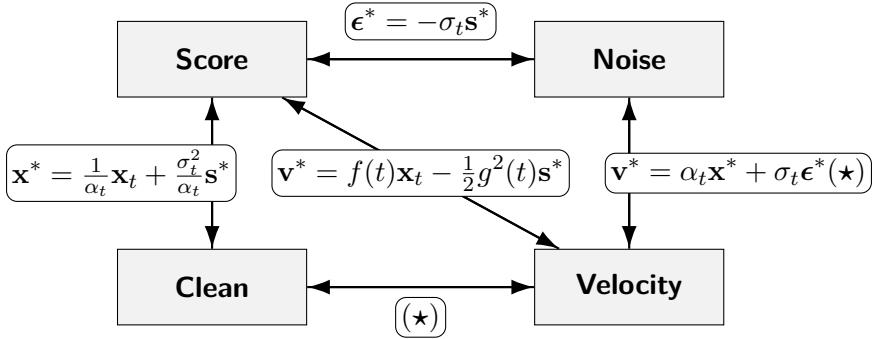
$$\begin{aligned} \epsilon^*(\mathbf{x}_t, t) &= -\sigma_t \mathbf{s}^*(\mathbf{x}_t, t), \\ \mathbf{x}^*(\mathbf{x}_t, t) &= \frac{1}{\alpha_t} \mathbf{x}_t + \frac{\sigma_t^2}{\alpha_t} \mathbf{s}^*(\mathbf{x}_t, t), \\ \mathbf{v}^*(\mathbf{x}_t, t) &= \alpha'_t \mathbf{x}^* + \sigma'_t \epsilon^* = f(t) \mathbf{x}_t - \frac{1}{2} g^2(t) \mathbf{s}^*(\mathbf{x}_t, t). \end{aligned} \tag{6.3.1}$$

Here,  $f(t)$  and  $g(t)$  are related to  $\alpha_t$  and  $\sigma_t$  via Lemma 4.4.1. Moreover, these minimizers satisfy the identities given in Equations (6.2.1) to (6.2.4).

#### Proof for Proposition.

The proof is similar to that of Theorem 4.2.1, which analyzes the global optimum of various matching losses under the DSM objective. See Section D.4

for details.



**Figure 6.1: Equivalent relations among four parameterizations.**  $\mathbf{v}$ -prediction is given by  $\mathbf{v}^* = \alpha_t \mathbf{x}^* + \sigma_t \epsilon^*$ , where clean and  $\epsilon$ -predictions are interchangeable via  $\mathbf{x}_t = \alpha_t \mathbf{x}^* + \sigma_t \epsilon^*$ .

Equation (6.3.1) induces a one-to-one conversion (at each  $t$ , given the forward noising coefficients) between the four parameterizations

$$\epsilon_\phi(\mathbf{x}_t, t), \quad \mathbf{x}_\phi(\mathbf{x}_t, t), \quad \mathbf{s}_\phi(\mathbf{x}_t, t), \quad \mathbf{v}_\phi(\mathbf{x}_t, t).$$

In practice, we train a single network in one parameterization (e.g.,  $\epsilon_\phi$ ). The other quantities are then *defined post hoc* by the conversions in Equation (6.3.1).

### 6.3.2 PF-ODE in Different Parameterizations

The PF-ODE admits several equivalent parameterizations (score, noise, denoised, and velocity). Although interchangeable in principle, the choice has practical consequences: it changes the stiffness of the vector field, the behavior of discretization error, and the ease of optimization. For fast sampling with advanced ODE solvers (see Chapter 9), practitioners often work with  $\epsilon$  or  $\mathbf{x}$  prediction because they align well with solver inputs and reduce error accumulation. For training generators that use only a few function evaluations (see Chapter 11),  $\mathbf{x}$  or  $\mathbf{v}$  prediction often yields smoother objectives and better step to step consistency.

We write the PF-ODE under each parameterization and make the conversions explicit using Equation (6.3.1). The results are collected in the following proposition.

**Proposition 6.3.2: PF-ODE in Different Parameterizations**

Let  $\alpha_t$  and  $\sigma_t$  be the forward perturbation schedules, and denote time derivatives by  $\alpha'_t := \frac{d\alpha_t}{dt}$  and  $\sigma'_t := \frac{d\sigma_t}{dt}$ . Then the empirical PF-ODE admits the equivalent forms

$$\begin{aligned}\frac{dx(t)}{dt} &= \frac{\alpha'_t}{\alpha_t} x(t) - \sigma_t \left( \frac{\alpha'_t}{\alpha_t} - \frac{\sigma'_t}{\sigma_t} \right) \epsilon^*(x(t), t) \\ &= \frac{\sigma'_t}{\sigma_t} x(t) + \alpha_t \left( \frac{\alpha'_t}{\alpha_t} - \frac{\sigma'_t}{\sigma_t} \right) x^*(x(t), t) \\ &= \frac{\alpha'_t}{\alpha_t} x(t) + \sigma_t^2 \left( \frac{\alpha'_t}{\alpha_t} - \frac{\sigma'_t}{\sigma_t} \right) s^*(x(t), t) \\ &= \alpha'_t x^*(x(t), t) + \sigma'_t \epsilon^*(x(t), t) \\ &= v^*(x(t), t).\end{aligned}\tag{6.3.2}$$

To see the Score SDE notation, we recall Lemma 4.4.1. If we set

$$f(t) = \frac{\alpha'_t}{\alpha_t}, \quad g^2(t) = \frac{d}{dt}(\sigma_t^2) - 2\frac{\alpha'_t}{\alpha_t}\sigma_t^2 = 2\sigma_t\sigma'_t - 2\frac{\alpha'_t}{\alpha_t}\sigma_t^2,$$

then the PF-ODE can be written in the familiar Score SDE form:

$$\frac{dx(t)}{dt} = f(t)x(t) - \frac{1}{2}g^2(t)s^*(x(t), t).$$

To give a concrete sense of how the PF-ODE is discretized for sampling, we will present in Section 9.2 the update rule of a widely used diffusion-based ODE sampler, the DDIM scheme. This example will show how an Euler discretization naturally connects with the PF-ODE.

**6.3.3 All Affine Flows Are Equivalent**

We next analyze the design choices for component (A) in Equation (6.2.5).

**State-Level Equivalence.** A convenient canonical interpolation used in FM (Lipman *et al.*, 2022) and RF (Liu, 2022) is

$$x_t^{\text{FM}} = (1-t)x_0 + t\epsilon = x_0 + t(\epsilon - x_0),$$

whose velocity is the constant vector  $\epsilon - x_0$ . The key point of this subsection is that the apparent simplicity of this choice is not essential: any affine interpolation

$$x_t = \alpha_t x_0 + \sigma_t \epsilon$$

can be written as a time-reparameterized and rescaled version of the canonical path. Define

$$c(t) := \alpha_t + \sigma_t, \quad \tau(t) := \frac{\sigma_t}{\alpha_t + \sigma_t} \quad (c(t) \neq 0).$$

A direct algebraic rewrite yields

$$\begin{aligned} \mathbf{x}_t &= \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon} \\ &= (\alpha_t + \sigma_t) \left( \frac{\alpha_t}{\alpha_t + \sigma_t} \mathbf{x}_0 + \frac{\sigma_t}{\alpha_t + \sigma_t} \boldsymbol{\epsilon} \right) \\ &= c(t) ((1 - \tau(t)) \mathbf{x}_0 + \tau(t) \boldsymbol{\epsilon}) = c(t) \mathbf{x}_{\tau(t)}^{\text{FM}}. \end{aligned}$$

Hence every affine path is the image of the canonical FM path under the change of variables  $t \mapsto \tau(t)$  and the spatial rescaling  $\mathbf{x} \mapsto c(t)\mathbf{x}$ . The equality holds pointwise and therefore also in distribution.

For the associated velocities, apply the chain rule to  $\mathbf{x}_t = c(t) \mathbf{x}_{\tau(t)}^{\text{FM}}$ :

$$\begin{aligned} \mathbf{v}(\mathbf{x}_t, t) &:= \frac{d}{dt} (\alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}) \\ &= \frac{d}{dt} (c(t) \mathbf{x}_{\tau(t)}^{\text{FM}}) \\ &= c'(t) \mathbf{x}_{\tau(t)}^{\text{FM}} + c(t) \tau'(t) \frac{d}{ds} \mathbf{x}_s^{\text{FM}} \Big|_{s=\tau(t)} \\ &= c'(t) \mathbf{x}_{\tau(t)}^{\text{FM}} + c(t) \tau'(t) \mathbf{v}^{\text{FM}} \left( \mathbf{x}_{\tau(t)}^{\text{FM}}, \tau(t) \right), \end{aligned}$$

since  $\mathbf{v}^{\text{FM}}(\mathbf{x}_{\tau}^{\text{FM}}, \tau) = -\mathbf{x}_0 + \boldsymbol{\epsilon}$  along the canonical path.

We summarize the above derivation as a formal statement in the following proposition.

### Proposition 6.3.3: Equivalence of Affine Flows

Let  $\mathbf{x}_t^{\text{FM}} = (1 - t) \mathbf{x}_0 + t \boldsymbol{\epsilon}$  and  $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$  with  $c(t) := \alpha_t + \sigma_t \neq 0$  and  $\tau(t) := \sigma_t / (\alpha_t + \sigma_t)$ . Then

$$\begin{aligned} \mathbf{x}_t &= c(t) \mathbf{x}_{\tau(t)}^{\text{FM}}, \\ \mathbf{v}(\mathbf{x}_t, t) &= c'(t) \mathbf{x}_{\tau(t)}^{\text{FM}} + c(t) \tau'(t) \mathbf{v}^{\text{FM}} \left( \mathbf{x}_{\tau(t)}^{\text{FM}}, \tau(t) \right). \end{aligned} \tag{6.3.3}$$

In particular, all affine interpolations are equivalent up to time reparameterization and spatial rescaling.

**Equivalence with Trigonometric Flow.** Another widely used affine flow is the trigonometric interpolation (Salimans and Ho, 2021; Albergo *et al.*, 2023; Lu and Song, 2024). As a concrete example, we also show that *any* affine flow can be expressed in this form. The trigonometric path is defined by

$$\mathbf{x}_u^{\text{Trig}} := \cos(u)\mathbf{x}_0 + \sin(u)\boldsymbol{\epsilon}. \quad (6.3.4)$$

Let  $R_t := \sqrt{\alpha_t^2 + \sigma_t^2}$  and assume  $R_t > 0$ . Choose an angle  $\tau_t$  so that

$$\cos \tau_t = \frac{\alpha_t}{R_t}, \quad \sin \tau_t = \frac{\sigma_t}{R_t}.$$

Then every affine interpolation  $\mathbf{x}_t = \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon}$  is a rescaled and re timed trigonometric path:

$$\mathbf{x}_t = \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon} = R_t \left( \frac{\alpha_t}{R_t}\mathbf{x}_0 + \frac{\sigma_t}{R_t}\boldsymbol{\epsilon} \right) = R_t \mathbf{x}_{\tau_t}^{\text{Trig}}. \quad (6.3.5)$$

The pair  $(\alpha_t, \sigma_t)$  is a point in the plane. Normalizing by  $R_t$  places it on the unit circle, which fixes the angle  $\tau_t$  and hence the state  $\mathbf{x}_{\tau_t}^{\text{Trig}}$ ; the radius  $R_t$  gives the overall scale.

Differentiating  $\mathbf{x}_u^{\text{Trig}}$  with respect to  $u$  gives its velocity,

$$\mathbf{v}_u^{\text{Trig}} = -\sin(u)\mathbf{x}_0 + \cos(u)\boldsymbol{\epsilon}.$$

Through the same change of variables as in equation 6.3.5, this relation provides closed-form conversions for the velocity (and analogously for other parameterizations).

Summarizing the above discussion, we arrive at the following conclusion:

### Conclusion 6.3.1:

Regardless of the schedule  $(\alpha_t, \sigma_t)$ , including VE, VP (such as trigonometric), FM, or RF, affine interpolations are mutually convertible by a suitable change of time variable and a scalar rescaling.

**Training Objectives of Four Parameterizations.** Let  $\mathbf{x}_t = \alpha_t\mathbf{x}_0 + \sigma_t\boldsymbol{\epsilon}$  with  $\sigma_t > 0$  and differentiable  $(\alpha_t, \sigma_t)$  such that  $\alpha'_t\sigma_t - \alpha_t\sigma'_t \neq 0$ . Consider the oracle targets

$$\boldsymbol{\epsilon}^*(\mathbf{x}_t, t) = \mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_t], \quad \mathbf{x}_0^*(\mathbf{x}_t, t) = \mathbb{E}[\mathbf{x}_0|\mathbf{x}_t], \quad \mathbf{v}^*(\mathbf{x}_t, t) = \mathbb{E}[\alpha'_t\mathbf{x}_0 + \sigma'_t\boldsymbol{\epsilon}|\mathbf{x}_t].$$

From Proposition 6.3.1, they satisfy

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = -\frac{1}{\sigma_t} \boldsymbol{\epsilon}^*(\mathbf{x}_t, t) = \frac{\alpha_t}{\sigma_t^2} \left( \mathbf{x}_0^*(\mathbf{x}_t, t) - \frac{\mathbf{x}_t}{\alpha_t} \right), \quad \mathbf{v}^* = \alpha'_t \mathbf{x}_0^* + \sigma'_t \boldsymbol{\epsilon}^*.$$

Under the head conversions

$$\mathbf{s}_\phi \equiv -\frac{1}{\sigma_t} \boldsymbol{\epsilon}_\phi \equiv \frac{\alpha_t}{\sigma_t^2} \left( \mathbf{x}_\phi - \frac{\mathbf{x}_t}{\alpha_t} \right),$$

and the velocity-to-score conversion is

$$\mathbf{s}_\phi = \frac{\alpha_t}{\sigma_t(\alpha'_t \sigma_t - \alpha_t \sigma'_t)} \mathbf{v}_\phi - \frac{\alpha'_t}{\sigma_t(\alpha'_t \sigma_t - \alpha_t \sigma'_t)} \mathbf{x}_t,$$

the per-sample squared losses match up to time-dependent weights:

$$\begin{aligned} \|\mathbf{s}_\phi - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\|_2^2 &= \frac{1}{\sigma_t^2} \|\boldsymbol{\epsilon}_\phi - \boldsymbol{\epsilon}^*\|_2^2 \\ &= \frac{\alpha_t^2}{\sigma_t^4} \|\mathbf{x}_\phi - \mathbf{x}_0^*\|_2^2 \\ &= \left( \frac{\alpha_t}{\sigma_t(\alpha'_t \sigma_t - \alpha_t \sigma'_t)} \right)^2 \|\mathbf{v}_\phi - \mathbf{v}^*\|_2^2. \end{aligned} \quad (6.3.6)$$

By Proposition 6.3.3, any affine flow  $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$  is transferable to the canonical FM path via  $\mathbf{x}_t = c(t) \mathbf{x}_{\tau(t)}^{\text{FM}}$  with  $c(t) = \alpha_t + \sigma_t$  and  $\tau(t) = \sigma_t / (\alpha_t + \sigma_t)$ . Differentiating gives

$$\mathbf{v}_\phi(\mathbf{x}_t, t) = c'(t) \mathbf{x}_{\tau(t)}^{\text{FM}} + c(t) \tau'(t) \mathbf{v}_\phi^{\text{FM}} \left( \mathbf{x}_{\tau(t)}^{\text{FM}}, \tau(t) \right), \quad \mathbf{x}_{\tau(t)}^{\text{FM}} = \frac{\mathbf{x}_t}{c(t)},$$

and the same relation holds for  $\mathbf{v}^*$ . Hence the velocity loss transforms by

$$\begin{aligned} &\|\mathbf{v}_\phi(\mathbf{x}_t, t) - \mathbf{v}^*(\mathbf{x}_t, t)\|_2^2 \\ &= (c(t) \tau'(t))^2 \left\| \mathbf{v}_\phi^{\text{FM}} \left( \frac{\mathbf{x}_t}{c(t)}, \tau(t) \right) - (\mathbf{v}^{\text{FM}})^* \left( \frac{\mathbf{x}_t}{c(t)}, \tau(t) \right) \right\|_2^2. \end{aligned}$$

With the above observation, we arrive at the following conclusion:

### Conclusion 6.3.2:

Score, noise, clean, and velocity training objectives are theoretically equivalent up to time-dependent weights (and, for velocity, an affine head conversion involving  $\mathbf{x}_t$ ) determined by  $(\alpha_t, \sigma_t)$ .

#### 6.3.4 (Optional) Conceptual Analysis of Parametrizations and the Canonical Flow

Even though we have shown in the previous sections that all four parameterizations are mathematically equivalent and can be transformed into one

another, and that the forward affine noise-injection flow is equivalent to the canonical form

$$\mathbf{x}_t^{\text{FM}} = (1 - t)\mathbf{x}_0 + t\boldsymbol{\epsilon},$$

in this subsection we provide further intuition and analyze the potential advantages of using the  $\mathbf{v}$ -prediction parameterization together with this canonical affine flow.

This subsection asks a simple question: *how do different parameterizations and schedules shape what the model learns and how we sample?* We proceed in three steps:

- **Regression Targets and Schedules.** We focus on why combining  $\mathbf{v}$ -prediction with the canonical linear schedule  $(\alpha_t, \sigma_t) = (1 - t, t)$  is natural: it maintains a stable target scale over time and eliminates curvature effects in the dynamics.
- **Solver Implications.** We examine how this parameterization conceptually interacts with numerical integration schemes while deferring concrete examples such as the Euler solver and Heun's method to Sections 9.2.2 and 9.4.5.

Before proceeding, we distinguish between two types of velocity fields to avoid ambiguity. The *conditional velocity*, which serves as a tractable training target, is defined as

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{z}) = \mathbf{x}'_t = \alpha'_t \mathbf{x}_0 + \sigma'_t \boldsymbol{\epsilon}, \quad \text{where } \mathbf{z} = (\mathbf{x}_0, \boldsymbol{\epsilon}),$$

while the *oracle (marginalized) velocity*, used to move samples during inference of PF-ODE solving, is given by

$$\mathbf{v}^*(\mathbf{x}, t) = \mathbb{E}[\mathbf{v}_t(\cdot|\mathbf{z})|\mathbf{x}_t = \mathbf{x}].$$

**Perspective 1: Why  $(\alpha_t, \sigma_t) = (1 - t, t)$  is a Natural Schedule.** Writing  $\sigma_t := \rho(t)$  and  $\alpha_t := 1 - \rho(t)$  for a time-varying  $\rho(t)$ , the conditional velocity becomes

$$\mathbf{v}_t(\mathbf{x}_t|\mathbf{z}) = \rho'(t)(\boldsymbol{\epsilon} - \mathbf{x}_0), \quad \text{where } \mathbf{z} = (\mathbf{x}_0, \boldsymbol{\epsilon}).$$

**Unit-Scale Regression Targets.** For the canonical schedule  $\rho(t) = t$ , the conditional velocity  $\mathbf{v}_t(\cdot|\mathbf{z})$  satisfies

$$\mathbb{E} \left[ \|\mathbf{v}_t(\cdot|\mathbf{z})\|_2^2 \right] = \mathbb{E}_{\boldsymbol{\epsilon}} \|\boldsymbol{\epsilon}\|_2^2 + \mathbb{E}_{\mathbf{x}_0} \|\mathbf{x}_0\|_2^2 = D + \underbrace{\text{Tr Cov}[\mathbf{x}_0]}_{\text{total variance}} + \underbrace{\|\mathbb{E} \mathbf{x}_0\|_2^2}_{\text{mean}}. \quad (6.3.7)$$

Thus the expected target magnitude is constant in  $t$ . After standardizing the data to zero mean and identity covariance (i.e.,  $\text{Cov}[\mathbf{x}_0] = \mathbf{I}$ ), the two components  $\alpha'_t \mathbf{x}_0$  and  $\sigma'_t \boldsymbol{\epsilon}$  contribute comparably for all  $t$ , avoiding gradient explosion/vanishing near the endpoints. To see this, we consider the diffusion's training objective:

$$\mathcal{L}_{\text{velocity}}(\phi) = \mathbb{E}_t \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[ \|\mathbf{v}_\phi(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t | \mathbf{z})\|_2^2 \right].$$

By applying the chain rule, the gradient of this loss with respect to the model parameters  $\phi$  is

$$\nabla_\phi \mathcal{L}_{\text{velocity}}(\phi) = 2 \mathbb{E}_t \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[ \partial_\phi \mathbf{v}_\phi(\mathbf{x}_t, t)^\top (\mathbf{v}_\phi(\mathbf{x}_t, t) - \mathbf{v}_t(\mathbf{x}_t | \mathbf{z})) \right].$$

Thus the scale of the target  $\|\mathbf{v}_t(\mathbf{x}_t | \mathbf{z})\|_2$  influences gradient stability: if it collapses to 0 (or blows up) at some  $t$ , gradients tend to vanish (or explode), all else equal. With the canonical choice  $\rho(t) = t$ , Equation (6.3.7) gives a  $t$ -independent target magnitude, so there is no endpoint ( $t = 0$  or  $t = 1$ ) collapse or blow-up arising from the regression signal (assuming  $\mathbb{E} \|\partial_\phi \mathbf{v}_\phi(\mathbf{x}_t, t)\|^2$  and any time-weights are controlled).

**Interplay of the Canonical Schedule and v-Prediction.** Under the affine path  $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}$ , the oracle velocity decomposes as

$$\mathbf{v}^*(\mathbf{x}, t) = \alpha'_t \mathbf{x}^*(\mathbf{x}, t) + \sigma'_t \boldsymbol{\epsilon}^*(\mathbf{x}, t),$$

with  $\mathbf{x}^* = \mathbb{E}[\mathbf{x}_0 | \mathbf{x}_t = \mathbf{x}]$  and  $\boldsymbol{\epsilon}^* = \mathbb{E}[\boldsymbol{\epsilon} | \mathbf{x}_t = \mathbf{x}]$ . Differentiating at fixed  $\mathbf{x}$  gives

$$\partial_t \mathbf{v}_t^* = \underbrace{\alpha''_t \mathbf{x}^* + \sigma''_t \boldsymbol{\epsilon}^*}_{\text{schedule curvature}} + \alpha'_t \partial_t \mathbf{x}^* + \sigma'_t \partial_t \boldsymbol{\epsilon}^*.$$

With the linear schedule  $\alpha_t = 1 - t$ ,  $\sigma_t = t$ , the curvature terms vanish ( $\alpha''_t = \sigma''_t = 0$ ), so the time-variation of  $\mathbf{v}_t^*$  primarily reflects the posterior evolution ( $\partial_t \mathbf{x}^*$ ,  $\partial_t \boldsymbol{\epsilon}^*$ ) rather than the schedule. This effect is especially clean for v-prediction: the coefficients  $\alpha'_t, \sigma'_t$  are constants ( $-1$  and  $+1$ ), avoiding extra  $t$ -dependent rescaling in the drift. By contrast, score-,  $\mathbf{x}_0$ , or  $\boldsymbol{\epsilon}$ -parameterizations often introduce ratios such as  $\sigma'_t / \sigma_t$  or  $\alpha'_t / \alpha_t$  that can vary sharply near the endpoints, even under a linear schedule. Hence, while not exclusive in principle, the linear  $(1 - t, t)$  schedule combined with v-prediction offers a particularly stable and transparent time dependence for the oracle velocity.

**Minimizing the Conditional Energy.** We next adopt a more theoretical perspective of optimal transport (see Chapter 7). Here, the *conditional kinetic*

*energy* quantifies the total expected motion of the conditional velocity along the forward path, that is, the amount of instantaneous movement (or kinetic effort) required to traverse from  $\mathbf{x}_0$  to  $\epsilon$ :

$$\mathcal{K}[\rho] := \int_0^1 \mathbb{E}_{\mathbf{x}_0, \epsilon} [\|\mathbf{v}_t(\cdot | \mathbf{z})\|_2^2] dt = (D + \text{Tr Cov}[\mathbf{x}_0] + \|\mathbb{E}\mathbf{x}_0\|_2^2) \int_0^1 (\rho'(t))^2 dt.$$

Minimizing  $\mathcal{K}[\rho]$  therefore corresponds to finding the smoothest, least-energy path in expectation. With the boundary conditions  $\rho(0) = 0$  and  $\rho(1) = 1$ , the Euler–Lagrange equation  $\rho''(t) = 0$  gives the minimizer  $\rho(t) = t$ , corresponding to a straight conditional path. This means that, among all smooth interpolations connecting  $\mathbf{x}_0$  and  $\epsilon$ , the canonical flow  $\rho(t) = t$  is the most energy-efficient way to move between them. We will revisit this point in Proposition 7.5.1 for a more detailed treatment.

**Remark on the Oracle Velocity.** If instead we evaluate the energy defined by marginal velocities

$$\int_0^1 \mathbb{E}_{\mathbf{x}_t} [\|\mathbf{v}^*(\mathbf{x}_t, t)\|_2^2] dt,$$

then with  $\mathbf{z} = (\mathbf{x}_0, \epsilon)$  and  $\mathbf{v}_t(\mathbf{x}_t | \mathbf{z}) = \rho'(t)(\epsilon - \mathbf{x}_0)$ ,

$$\mathbf{v}^*(\mathbf{x}, t) = \mathbb{E}[\mathbf{v}_t(\cdot | \mathbf{z}) | \mathbf{x}_t = \mathbf{x}] = \rho'(t) \mathbb{E}[\epsilon - \mathbf{x}_0 | \mathbf{x}_t = \mathbf{x}];$$

and hence, the energy of the marginal velocity becomes

$$\int_0^1 \mathbb{E}_{\mathbf{x}_t \sim p_t} [\|\mathbf{v}^*(\mathbf{x}_t, t)\|_2^2] dt = \int_0^1 \mathbb{E}_{\mathbf{x}_t} [\|\rho'(t) \mathbb{E}[\epsilon - \mathbf{x}_0 | \mathbf{x}_t]\|_2^2] dt = \int_0^1 (\rho'(t))^2 \kappa(t) dt,$$

$$\text{where } \kappa(t) := \mathbb{E}_{\mathbf{x}_t \sim p_t} [\|\mathbb{E}[\epsilon - \mathbf{x}_0 | \mathbf{x}_t]\|_2^2].$$

Consequently, the *marginal*-optimal schedule  $\rho(t)$  need not be linear. It is linear iff  $\kappa(t)$  is constant; in general, the Euler–Lagrange condition

$$(\kappa(t) \rho'(t))' = 0 \Rightarrow \rho'(t) \propto \frac{1}{\kappa(t)}$$

implies that the oracle-optimal schedule re-parameterizes time adaptively. Intuitively,  $\kappa(t)$  quantifies how much of the label  $(\epsilon - \mathbf{x}_0)$  is predictable from  $\mathbf{x}_t \sim p_t$ : the oracle flow slows down where  $\kappa(t)$  is large, reflecting regions where the oracle velocity has high expected magnitude, and speeds up where  $\kappa(t)$  is small. Hence, even though the conditional flow uses the linear schedule  $(1-t, t)$ , the corresponding marginalized (oracle) dynamics are generally nonlinear.

### Perspective 2: Why Velocity Prediction Can Be Considered Natural for Sampling.

**Semilinear Form of the PF–ODE under  $\mathbf{x}$ -,  $\epsilon$ -, and  $\mathbf{s}$ -Predictions.** Under the clean, noise, and score parameterizations, the drift takes a semilinear form (see the first three identities in Equation (6.3.2)):

$$\frac{d\mathbf{x}(t)}{dt} = \underbrace{L(t)\mathbf{x}(t)}_{\text{linear part}} + \underbrace{\mathbf{N}_\phi(\mathbf{x}(t), t)}_{\text{nonlinear part}}, \quad \mathbf{N}_\phi \in \{\mathbf{x}_\phi, \epsilon_\phi, \mathbf{s}_\phi\}.$$

When the linear drift  $L(t)\mathbf{x}(t)$  drives changes in  $\mathbf{x}(t)$  at very different rates in some directions compared with the nonlinear part, the system is *stiff*, meaning that the Jacobian (in  $\mathbf{x}$ ) of the drift

$$\mathbf{J}(\mathbf{x}, t) := L(t) + \nabla_{\mathbf{x}} \mathbf{N}_\phi(\mathbf{x}, t)$$

has eigenvalues whose real parts differ by orders of magnitude (a larger magnitude corresponds to a faster direction)<sup>3</sup>. For instance, the dynamics may involve a “fast linear” change alongside a “slow nonlinear” one in  $\mathbf{x}(t)$ . In such cases, explicit solvers must take very small time steps to remain numerically stable.

To address this imbalance, higher-order stable solvers often apply an *integrating factor* that treats the linear term  $L(t)\mathbf{x}$  analytically and discretizes only the slower nonlinear remainder, albeit at the cost of additional algebraic and implementation complexity. Chapter 9 is dedicated to a detailed discussion of this topic.

**PF–ODE under  $\mathbf{v}$ -Prediction.** With  $\mathbf{v}$ -prediction, the model directly learns the velocity field and integrates

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}_\phi(\mathbf{x}(t), t) \approx \mathbf{v}^*(\mathbf{x}(t), t).$$

In this formulation, the explicit linear term is absorbed into a single learned field, so the dynamics no longer split into separate parts. The step size is

---

<sup>3</sup>Let the PF–ODE drift be  $\mathbf{F}(\mathbf{x}, t) = L(t)\mathbf{x} + \mathbf{N}_\phi(\mathbf{x}, t)$  and assume  $\mathbf{N}_\phi$  is (locally) Lipschitz in  $\mathbf{x}$  with constant  $\text{Lip}_{\mathbf{N}_\phi}(t)$ . For nearby states  $\mathbf{x}, \mathbf{y}$ ,

$$\|\tilde{\mathbf{f}}(\mathbf{x}, t) - \tilde{\mathbf{f}}(\mathbf{y}, t)\| \leq \underbrace{\left( \|L(t)\| + \text{Lip}_{\mathbf{N}_\phi}(t) \right)}_{=: C(t)} \|\mathbf{x} - \mathbf{y}\|.$$

Equivalently, the Jacobian (w.r.t.  $\mathbf{x}$ )

$$\mathbf{J}(\mathbf{x}, t) = L(t) + \nabla_{\mathbf{x}} \mathbf{N}_\phi(\mathbf{x}, t)$$

satisfies  $\|\mathbf{J}(\mathbf{x}, t)\|_{\text{op}} \leq C(t)$  ( i.e., the operator norm induced by the Euclidean norm on  $\mathbb{R}^D$ ). Hence, the real parts of all eigenvalues of  $\mathbf{J}$  are bounded in magnitude by  $C(t)$ . Thus a large  $C(t)$  means fast local rates, so explicit solvers need small steps ( $h \lesssim 1/C(t)$ ).

thus governed by how smoothly the learned field  $\mathbf{v}_\phi(\mathbf{x}, t)$  varies with  $\mathbf{x}$  and  $t$ , rather than by the magnitude of a prescribed scalar coefficient  $L(t)$ . In other words, the potentially rapid linear drift is folded into one coherent velocity field, reducing time-scale disparity and simplifying numerical integration.

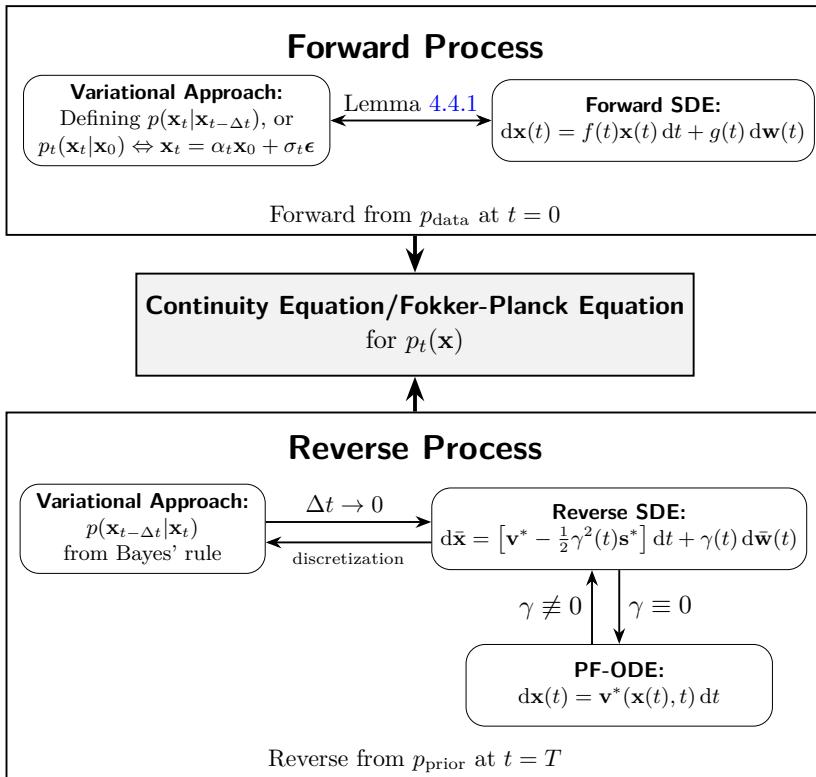
Later in Section 9.2.2, we will illustrate, with a simple example, the structural simplicity of  $\mathbf{v}$ -prediction in the sampling process. To obtain the same discretization update of the PF-ODE as DDIM (Song *et al.*, 2020a) (one of the most widely used fast samplers in diffusion modeling), a plain Euler step under the  $\epsilon$ -,  $\mathbf{x}$ -, or  $\mathbf{s}$ -parameterizations only *approximates* the linear term rather than computing it exactly (see Equation (9.1.8)). Consequently, these parameterizations require a more advanced approach, the *exponential integrator*, to isolate and compute the linear term exactly. In contrast, with  $\mathbf{v}$ -prediction, there is no separate linear term to isolate in the PF-ODE drift, so the plain Euler update naturally coincides with the DDIM formulation. A closely related analogy appears in Section 9.4.5: the second-order DPM-Solver (Lu *et al.*, 2022b) coincides with the classical Heun method: for  $\mathbf{v}$ -prediction this is plain Heun, whereas for  $\epsilon$ -,  $\mathbf{x}$ -, or  $\mathbf{s}$ -prediction it is the exponential Heun. We leave the detailed discussion to their respective sections.

We remark that any improvements in generation (such as achieving higher sample quality with fewer model evaluations in PF-ODE solving) depend on both how accurately  $\mathbf{v}_\phi$  approximates the oracle velocity and how effectively the sampling algorithm (including the numerical integrator, discretization schedule, and step-size control) interacts with it. Thus, adopting the  $\mathbf{v}$ -parameterization does not by itself guarantee better sampling performance.

**Conclusion.** While  $\mathbf{v}$ -prediction combined with the canonical linear schedule offers certain theoretical advantages, such as a constant target magnitude and the absence of schedule curvature, these properties do not necessarily make it universally superior. In practice, model performance depends on a range of interacting factors, including network architecture, normalization schemes, loss weighting over time, the choice of sampler and discretization steps, guidance strength, regularization strategy, data scaling, and overall training budget. Different datasets and objectives may favor alternative parameterizations or schedules, and the optimal configuration is ultimately an empirical question that should be resolved through validation and ablation studies.

## 6.4 Beneath It All: The Fokker–Planck Equation

**Figure 6.2: Unified perspective connecting variational, SDE, and ODE formulations through the continuity equation, where all  $p_t(\mathbf{x})$  evolve under a shared dynamic.** The velocity field  $\mathbf{v}^*(\mathbf{x}, t) = f(t)\mathbf{x} - \frac{1}{2}g^2(t)\mathbf{s}^*(\mathbf{x}, t)$  is governed by the score function  $\mathbf{s}^*(\mathbf{x}, t) := \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ . Coefficients  $f(t)$ ,  $g(t)$ ,  $\sigma_t$ , and  $\alpha_t$  are pre-defined time-dependent functions, and  $\gamma(t)$  is a tunable time-varying hyperparameter.



In this section, we show that the three main perspectives of diffusion models—variational, score-based, and flow-based—are not separate constructions but arise from a single unifying principle: the continuity (Fokker–Planck) equation that governs density evolution under a chosen forward process.

First, we recall that the analysis in Section 4.5 unifies the variational perspective, based on discrete kernels and Bayes’ rule, with the score-based SDE perspective of continuous dynamics. We establish this connection by showing that variational models act as consistent discretizations of the underlying forward and reverse SDEs. Specifically, the marginal densities calculated

step-by-step via the discrete kernels evolve in a manner consistent with the Fokker–Planck equation that governs the continuous-time dynamics. This confirms that the two perspectives are fundamentally equivalent.

We then connect the flow-based and score-based views. In Section 6.4.1, we show that an ODE flow determines a density path whose marginals can always be realized by a family of stochastic processes. This places deterministic flows and stochastic SDEs within the same family.

Together, these results unify the three perspectives under one framework (see Figure 6.2). At last, we conclude this chapter in Section 6.5.

### 6.4.1 Connection of Flow-Based Approach and Score SDE

A remarkable aspect of diffusion model lies in how different dynamic systems, deterministic or stochastic, can trace out the same evolution of probability distributions. In this section, we reveal a natural and elegant connection between ODE-based flows of Section 5.2 and Score SDEs. Specifically, we show that the velocity field defining a generative ODE can be transformed into a stochastic counterpart that follows the same Fokker–Planck dynamics, providing a principled bridge between deterministic interpolation and stochastic sampling. This offers us a continuous family of models, ranging from ODEs to SDEs, that generate the same data distribution path.

We consider the continuous-time setup where the perturbation kernel is given by

$$p_t(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}),$$

where  $\mathbf{x}_0 \sim p_{\text{data}}$ . This conditional distribution induces a marginal density path  $p_t(\mathbf{x}_t) = \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}} [p(\mathbf{x}_t | \mathbf{x}_0)]$  as usual, with  $p_T \approx p_{\text{prior}}$ .

To match this density path, consider the ODE

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}_t(\mathbf{x}(t)), \quad t \in [0, T], \tag{6.4.1}$$

where  $\mathbf{v}_t(\mathbf{x}) = \mathbb{E}[\alpha'_t \mathbf{x}_0 + \sigma'_t \boldsymbol{\epsilon} | \mathbf{x}]$  is the oracle velocity as shown in Equation (5.2.10) (noting that time is flipped to follow the diffusion model convention). Integrating equation 6.4.1 backward from  $\mathbf{x}(T) \sim p_{\text{prior}}$  yields samples from  $p_0$ .

Although this ODE suffices for generating high-quality samples, incorporating stochasticity may improve sample diversity. This motivates the following question:

#### Question 6.4.1

Is there an SDE whose dynamics, starting from  $p_{\text{prior}}$ , yield the same

marginal densities as the ODE in Equation (6.4.1)?

This statement affirms that there exists a family of reverse-time SDEs that induce the same marginal density path as the corresponding PF-ODE. The densities induced by these SDEs satisfy the same Fokker–Planck equation for this path, and therefore their one time marginals coincide with the prescribed interpolation  $\{p_t\}_{t \in [0, T]}$ <sup>4</sup>.

**Proposition 6.4.1: Reverse-Time SDEs Generate the Same Marginals as Interpolations**

Let  $\gamma(t) \geq 0$  be an arbitrary time-dependent coefficient. Consider the reverse-time SDE

$$d\bar{\mathbf{x}}(t) = \left[ \mathbf{v}^*(\bar{\mathbf{x}}(t), t) - \frac{1}{2}\gamma^2(t)\mathbf{s}^*(\bar{\mathbf{x}}(t), t) \right] d\bar{t} + \gamma(t) d\bar{\mathbf{w}}(t), \quad (6.4.2)$$

evolving backward from  $\bar{\mathbf{x}}(T) \sim p_T$  down to  $t = 0$ . Then this process  $\{\bar{\mathbf{x}}(t)\}_{t \in [0, T]}$  matches the prescribed marginals  $\{p_t\}_{t \in [0, T]}$  induced by the ODE's density path. Here,  $\mathbf{s}(\mathbf{x}, t) := \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$  is the score function, and it is related to the velocity field  $\mathbf{v}(\mathbf{x}, t)$  by

$$\mathbf{v}^*(\mathbf{x}, t) = f(t)\mathbf{x} - \frac{1}{2}g^2(t)\mathbf{s}^*(\mathbf{x}, t), \quad \mathbf{s}^*(\mathbf{x}, t) = \frac{1}{\sigma_t} \frac{\alpha_t \mathbf{v}^*(\mathbf{x}, t) - \alpha'_t \mathbf{x}}{\alpha'_t \sigma_t - \alpha_t \sigma'_t}. \quad (6.4.3)$$

**Proof for Proposition.**

The reverse-time Fokker–Planck equation corresponding to Equation (6.4.2) is

$$\partial_{\bar{t}} p = -\nabla \cdot \left( [\mathbf{v}^* - \frac{1}{2}\gamma^2 \mathbf{s}^*] p \right) + \frac{1}{2}\gamma^2 \Delta p.$$

Using the identity  $\nabla \cdot (\mathbf{s}^* p) = \Delta p$  (since  $\mathbf{s}^* = \nabla \log p$ ), the second-order terms cancel, yielding

$$\partial_{\bar{t}} p = -\nabla \cdot (\mathbf{v}^* p),$$

i.e., the first-order (drift-only) Fokker–Planck equation associated with the PF-ODE. Hence the reverse-time SDE and the ODE induce the same marginal density path  $\{p_t\}$ . See Appendix A.2–A.3 of Ma *et al.* (2024). ■

---

<sup>4</sup>For completeness, a forward SDE representation (not needed here) is

$$d\mathbf{x}(t) = f(t)\mathbf{x}(t) dt + g(t) d\mathbf{w}(t),$$

with  $f(t)$  and  $g(t)$  related to  $(\alpha_t, \sigma_t)$  via Equation (4.4.2).

The hyperparameter  $\gamma(t)$  can be chosen arbitrarily, independent of  $\alpha_t$  and  $\sigma_t$ , even after training, as it does not affect the velocity  $\mathbf{v}(\mathbf{x}, t)$  or the score  $\mathbf{s}(\mathbf{x}, t)$ . Below are some examples:

- Setting  $\gamma(t) = 0$  recovers the ODE in Equation (6.4.1).
- When  $\gamma(t) = g(t)$ , Equation (6.4.2) becomes the reverse-time SDE in Equation (4.1.6), since the oracle velocity  $\mathbf{v}(\mathbf{x}, t)$  satisfies (See Proposition 6.3.1):

$$\mathbf{v}^*(\mathbf{x}, t) = f(t)\mathbf{x} - \frac{1}{2}g^2(t)\mathbf{s}^*(\mathbf{x}, t).$$

- Other choices for  $\gamma(t)$  have been explored; e.g., Ma *et al.* (2024) select  $\gamma(t)$  to minimize the KL gap between  $p_{\text{data}}$  and the  $t = 0$  density obtained by solving Equation (6.4.2) from  $t = T$ .

Following Score SDE, the trained velocity field  $\mathbf{v}_{\phi^\times}(\mathbf{x}, t)$  can be converted into a parameterized score function  $\mathbf{s}_{\phi^\times}(\mathbf{x}, t)$  via Equation (6.4.3). Plugging this into Equation (6.4.2) defines an *empirical reverse-time SDE*, which can be sampled by numerically integrating from  $t = T$  with  $\bar{\mathbf{x}}(T) \sim p_{\text{prior}}$ .

This proposition highlights a remarkable flexibility of diffusion models: once a *marginal density path*  $\{p_t\}_{t \in [0, T]}$  is fixed, an entire family of dynamics can reproduce it, including both the PF-ODE and the reverse-time SDEs

$$d\bar{\mathbf{x}}(t) = [\mathbf{v}^*(\bar{\mathbf{x}}, t) - \frac{1}{2}\gamma^2(t)\mathbf{s}^*(\bar{\mathbf{x}}, t)] d\bar{t} + \gamma(t) d\bar{\mathbf{w}}(t), \quad \gamma(t) \geq 0.$$

All such dynamics satisfy the same reverse-time Fokker–Planck equation and hence yield the same marginal evolution. The function  $\gamma(t)$  continuously modulates the level of stochasticity without affecting the one-time distributions, revealing a deep connection between the deterministic flow-based ODE and its stochastic SDE counterpart, as illustrated in Figure 6.2.

## 6.5 Closing Remarks

This chapter has served as the keystone of our theoretical exploration, synthesizing the variational, score-based, and flow-based perspectives into a single, cohesive framework. We have shown that these three seemingly distinct approaches are not merely parallel but are deeply and fundamentally interconnected.

Our unification rests on two core insights. First, we identified the secret sauce common to all frameworks: a conditioning trick that transforms an intractable marginal training objective into a tractable conditional one, enabling stable and efficient learning. Second, we established that the Fokker-Planck equation is the universal law governing the evolution of probability densities. All three perspectives, in their own way, construct a generative process that respects this fundamental dynamic.

Furthermore, we demonstrated that the various model parameterizations, i.e., noise, clean data, score, or velocity prediction, are all interchangeable. This reveals that the choice of prediction target is a matter of implementation and stability rather than a fundamental modeling difference. The ultimate takeaway is that modern diffusion methods, despite their diverse origins, all instantiate the same core principle: they learn a time-dependent vector field to transport a simple prior to the data distribution.

With this unified and principled foundation firmly established, we are now equipped to move from the foundational theory to the practical application and acceleration of diffusion models. The central insight that generation is equivalent to solving a differential equation provides a powerful platform for control and optimization. The subsequent parts of this monograph will leverage this unified understanding to address key practical challenges:

1. Part C will focus on improving the sampling process at inference time. We will explore how to steer the generative trajectory for controllable generation (Chapter 8) and investigate advanced numerical solvers to dramatically accelerate the slow, iterative sampling process (Chapter 9).
2. Part D will then look beyond iterative solvers to learn fast generators directly. We will examine methods that can produce high-quality samples in just one or a few steps, either through distillation from a teacher model (Chapter 10) or by training from scratch (Chapter 11).

Having unified the *what* and *why* of diffusion models, we now turn our attention to the exciting and practical frontiers of *how*.

# 7

---

## (Optional) Diffusion Models and Optimal Transport

---

Mapping one distribution to another (with generation as a special case) is a central challenge. Flow matching addresses this by learning a time dependent velocity field that transports mass from source to target. This naturally connects to transport theory: classical optimal transport seeks the minimal cost path between distributions, while its entropy regularized form, the Schrödinger bridge, selects the most likely controlled diffusion relative to a reference such as Brownian motion.

In this chapter we review the foundations of optimal transport, entropic optimal transport, and Schrödinger bridges as formulations of the distribution-to-distribution problem. This leads to a central question: to what extent do diffusion models realize such optimal transports? They admit two views: as *stochastic processes*, defined through forward and reverse SDEs, and as *deterministic processes*, given by PF-ODEs. The stochastic view aligns directly with entropic optimal transport, while the PF-ODE does not generally correspond to any known transport objective. This gap leaves an open question: under what conditions can diffusion models be regarded as solving an optimal transport problem?

## 7.1 Prologue of Distribution-to-Distribution Translation

Diffusion models fix the terminal distribution to a standard Gaussian,  $p_{\text{prior}}$ . However, many applications require *distribution-to-distribution* translation: transforming a source distribution  $p_{\text{src}}$  into a different target  $p_{\text{tgt}}$ . Examples include converting sketches to photorealistic images or translating between artistic styles.

Modern diffusion methods provide practical ways to achieve this. *One-endpoint* methods such as SDEdit (Meng *et al.*, 2021b) begin with a source image at  $t = 0$ , diffuse it to an intermediate step  $t$ , and then use a pre-trained diffusion model for the target domain to reverse the process. This produces an output that matches the style and content of the target distribution.

*Two-endpoint* methods, like Dual Diffusion Bridge (Su *et al.*, 2022), instead connect the two domains through a shared latent distribution, typically a Gaussian at  $t = 1$ . A forward probability-flow ODE transports samples from  $p_{\text{src}}$  into this latent space, while a reverse ODE trained on the target domain maps them back to  $p_{\text{tgt}}$ . Beyond such sampling-time approaches, the Flow Matching framework described in Section 5.2 offers a training-based alternative: it directly learns an ODE flow that continuously moves mass from  $p_{\text{src}}$  to  $p_{\text{tgt}}$ .

Crucially, transforming between distributions requires more than two separately trained models. It demands a principled mapping that aligns the dynamics at *both* endpoints and does so in the “cheapest” (cost-efficient) way.

In this section, rather than surveying the many diffusion-based translation applications, we shift our focus to the mathematical foundations of this classic distribution-to-distribution problem. In particular, we highlight optimal transport (OT) and its entropic variant, the Schrödinger bridge (SB), which have long been studied in the theoretical community as canonical formulations of cost-efficient (in mathematical sense) distributional transformation.

At its core, the fundamental question is:

### Question 7.1.1

Given two probability distributions, what is the most efficient way to transform one into the other while minimizing the total cost?

Here, the cost,  $c(\mathbf{x}, \mathbf{y})$ , is a non-negative function that assigns a penalty for moving a unit of mass from a point  $\mathbf{x}$  to a point  $\mathbf{y}$ . A common choice is the squared distance,  $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2$ .

This section provides a brief overview to clarify how diffusion-based approaches, including flow matching, connect to classical and regularized optimal transport. The central question we aim to explore is:

**Question 7.1.2**

Is a diffusion model a form of optimal transport connecting  $p_{\text{data}}$  and  $p_{\text{prior}}$ , and in what sense?

To address this question, we first clarify what “optimality” means in Section 7.2. We review classical *optimal transport (OT)* in the static Monge–Kantorovich form (Equation (7.2.1)) and its dynamic Benamou–Brenier formulation (Equation (7.2.3); minimizing kinetic energy subject to the continuity equation), as well as the entropy regularized variant (*entropic OT*) in Equation (7.2.5), which is equivalent to the *Schrödinger Bridge Problem* (Equation (7.2.8)). In the dynamic view, OT induces a deterministic flow satisfying the continuity equation, whereas SB induces a controlled diffusion whose marginals evolve by the Fokker–Planck equation. We provide a high level map between these formulations in Section 7.3.

We then split the discussion into two parts. First, in Section 7.4, we explain that the fixed forward noising SDE used in standard diffusion models is not, by itself, a Schrödinger bridge between arbitrary  $p_{\text{src}}$  and  $p_{\text{tgt}}$ : the forward process is a chosen reference diffusion and both forward-in-time or reverse-time SDE do not, in general, enforce exact endpoint matching to a prescribed target. Hence it is not entropic OT optimal unless one explicitly solves the SB problem with those endpoints; while it is an optimal solution to the half-bridge problem as it is anchored with one starting point.

Second, in Section 7.5, we return to the generative setting with  $p_{\text{src}} = p_{\text{prior}}$  (Gaussian) and  $p_{\text{tgt}} = p_{\text{data}}$ . The PF-ODE defines a deterministic map that transports  $p_{\text{prior}}$  to  $p_{\text{data}}$  by construction. However, this flow is generally not an OT map for a prescribed transport cost (e.g., quadratic  $W_2$ ): it realizes one admissible deterministic coupling among many and does not minimize the Benamou–Brenier action. What follows is we discuss if the “rectify flow” procedure (Section 5.4.1) can lead to a OT map; however, in general, there is no such theoretical guarantee. Therefore, the exact characterization between diffusion model’s PF-ODE map and OT is remaining challenging and unsolved problem.

## 7.2 Taxonomy of the Problem Setups

In this section, we introduce notions of what constitutes the most “efficient” or “optimal” way to transport mass from  $p_{\text{src}}$  to  $p_{\text{tgt}}$ . These include classical optimal transport (OT) and its entropy-regularized variant, which admits an equivalent formulation known as the Schrödinger Bridge. This taxonomy provides background that will later clarify connections with diffusion models.

### 7.2.1 Optimal Transport (OT)

**Monge–Kantorovich (Static) Formulation of OT Problem.** We fix a cost function  $c : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$  that specifies the expense of sending probability mass from  $\mathbf{x}$  to  $\mathbf{y}$ . The goal is to transform the source distribution  $p_{\text{src}}$  into the target distribution  $p_{\text{tgt}}$  as cheaply as possible.

To even define a cost, we must know which pairs  $(\mathbf{x}, \mathbf{y})$  are matched. This role is played by a *coupling*: a joint distribution  $\gamma$  on  $\mathbb{R}^D \times \mathbb{R}^D$  whose marginals are  $p_{\text{src}}$  and  $p_{\text{tgt}}$ . In other words, sampling  $(\mathbf{x}, \mathbf{y}) \sim \gamma$  means we match  $\mathbf{x}$  from the source with  $\mathbf{y}$  from the target. If  $\gamma$  admits a density  $\gamma(\mathbf{x}, \mathbf{y})$  with respect to Lebesgue measure, the marginal constraints read

$$\int_{\mathbb{R}^D} \gamma(\mathbf{x}, \mathbf{y}) \, d\mathbf{y} = p_{\text{src}}(\mathbf{x}), \quad \int_{\mathbb{R}^D} \gamma(\mathbf{x}, \mathbf{y}) \, d\mathbf{x} = p_{\text{tgt}}(\mathbf{y}).$$

That is, integrating out  $\mathbf{y}$  recovers the source density in  $\mathbf{x}$ , while integrating out  $\mathbf{x}$  recovers the target density in  $\mathbf{y}$ .

We give two standard examples for illustration:

1. **Discrete Supports.** If  $p_{\text{src}}$  and  $p_{\text{tgt}}$  are supported on finitely many points, a coupling is represented by a nonnegative matrix  $(\gamma_{ij})$  whose row sums equal  $p_{\text{src}}(i)$  and column sums equal  $p_{\text{tgt}}(j)$ . Each entry  $\gamma_{ij}$  is the amount of mass sent from  $i$  to  $j$ .
2. **Deterministic Map.** If there exists a measurable map  $\mathbf{T}$  with  $\mathbf{T}_\# p_{\text{src}} = p_{\text{tgt}}$ , then  $\gamma = (\mathbf{I}, \mathbf{T})_\# p_{\text{src}}$  is a deterministic coupling that moves each point  $\mathbf{x}$  directly to  $\mathbf{T}(\mathbf{x})$ .

Once a coupling  $\gamma$  is fixed, the transport cost is simply the average unit cost under this plan:

$$\int c(\mathbf{x}, \mathbf{y}) \, d\gamma(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \gamma}[c(\mathbf{x}, \mathbf{y})].$$

In the discrete case, this reduces to  $\sum_{i,j} c_{ij} \gamma_{ij}$ , whereas in the continuous setting it becomes a double integral. In what follows, we will focus only on the continuous case.

The optimal transport problem is then to choose, among all admissible couplings, the one that minimizes this expected cost.

$$\text{OT}(p_{\text{src}}, p_{\text{tgt}}) := \inf_{\gamma \in \Gamma(p_{\text{src}}, p_{\text{tgt}})} \int c(\mathbf{x}, \mathbf{y}) \, d\gamma(\mathbf{x}, \mathbf{y}), \quad (7.2.1)$$

where the feasible set simply enforces the marginal, or mass-conservation, constraints:

$$\begin{aligned} \Gamma(p_{\text{src}}, p_{\text{tgt}}) = & \left\{ \gamma \in \mathcal{P}(\mathbb{R}^D \times \mathbb{R}^D) : \right. \\ & \left. \int \gamma(\mathbf{x}, \mathbf{y}) \, d\mathbf{y} = p_{\text{src}}(\mathbf{x}), \int \gamma(\mathbf{x}, \mathbf{y}) \, d\mathbf{x} = p_{\text{tgt}}(\mathbf{y}) \right\}, \end{aligned}$$

where  $\mathcal{P}(\mathbb{R}^D \times \mathbb{R}^D)$  denotes the set of all probability measures on  $\mathbb{R}^D \times \mathbb{R}^D$ .

**A Special Case: Wasserstein-2 Distance.** The Wasserstein-2 distance is a special case of the Monge–Kantorovich problem with the quadratic cost  $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2$ . It measures the distance between two probability distributions as:

$$\mathcal{W}_2^2(p_{\text{src}}, p_{\text{tgt}}) := \inf_{\gamma \in \Gamma(p_{\text{src}}, p_{\text{tgt}})} \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \gamma} [\|\mathbf{x} - \mathbf{y}\|^2].$$

Under suitable assumptions on  $p_{\text{src}}$  and  $p_{\text{tgt}}$ , Brenier's theorem (see Theorem 7.1)<sup>1</sup> guarantees that the optimal coupling  $\gamma$  for the quadratic cost is concentrated on the graph of a deterministic map  $\mathbf{T} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ . Consequently, the Wasserstein-2 distance can be equivalently expressed as<sup>2</sup>:

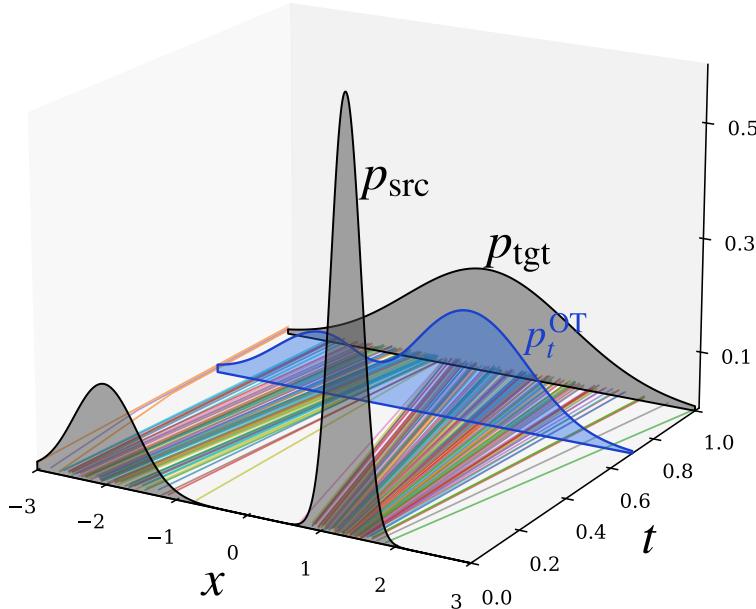
$$\mathcal{W}_2^2(p_{\text{src}}, p_{\text{tgt}}) = \inf_{\substack{\mathbf{T} : \mathbb{R}^D \rightarrow \mathbb{R}^D, \\ \text{s.t. } \mathbf{T} \# p_{\text{src}} = p_{\text{tgt}}}} \mathbb{E}_{\mathbf{x} \sim p_{\text{src}}} [\|\mathbf{T}(\mathbf{x}) - \mathbf{x}\|^2]. \quad (7.2.2)$$

Here,  $\mathbf{T} \# p_{\text{src}} = p_{\text{tgt}}$  means that  $\mathbf{T}$  pushes  $p_{\text{src}}$  forward to  $p_{\text{tgt}}$ , i.e.,  $\mathbf{T}(\mathbf{x}) \sim p_{\text{tgt}}$  for  $\mathbf{x} \sim p_{\text{src}}$ .

Thus, the Wasserstein-2 distance represents the minimal expected squared transport cost among all couplings or transport maps that match the given marginals. The optimal transport map denoted by  $\mathbf{T}^*(\mathbf{x})$ , known as the *Monge map*, yields the most efficient way to transform  $p_{\text{src}}$  into  $p_{\text{tgt}}$ .

<sup>1</sup>Brenier's theorem is about the existence and structure of the optimal transport map for quadratic cost. In particular, if  $p_{\text{src}}$  does not give mass to sets of dimension at most  $D - 1$ , then an optimal transport map  $\mathbf{T}^*$  uniquely exists.

<sup>2</sup>There are three commonly used formulations of the  $\mathcal{W}_2$  distance: the Monge formulation (based on an optimal transport map), the Kantorovich formulation (based on couplings), and the Benamou–Brenier dynamic formulation (see Equation (7.2.3)). These are equivalent under appropriate regularity conditions.



**Figure 7.1: Illustration of dynamic view of OT.** The interpolation  $p_t^{\text{OT}}$  evolves continuously in time, providing the least-cost transport plan that deterministically maps  $p_{\text{src}}$  to  $p_{\text{tgt}}$  (the McCann's displacement interpolation).

**Benamou–Brenier (Dynamic) Formulation of OT.** Instead of mapping distributions directly in a static manner, as in the Monge–Kantorovich formulation, transport can also be modeled as a continuous-time flow:

$$p_0 := p_{\text{src}} \rightarrow p_t \rightarrow p_1 := p_{\text{tgt}}, \quad t \in [0, 1].$$

This dynamic formulation of optimal transport, introduced by Benamou and Brenier (2000), seeks a smooth velocity field  $\mathbf{v}_t(\mathbf{x})$  that describes how mass in  $p_t(\mathbf{x})$  evolves over time.

The Benamou–Brenier formulation<sup>3</sup> shows that, for the quadratic cost  $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2^2$  (i.e., the  $\mathcal{W}_2$  distance), the optimal value of the static OT problem in Equation (7.2.1) is equal to the optimal value of the kinetic energy minimization problem:

---

<sup>3</sup>Benamou–Brenier formulation describes how to compute the  $\mathcal{W}_2$  distance by minimizing kinetic energy over continuous paths of measures and velocities.

$$\mathcal{W}_2^2(p_{\text{src}}, p_{\text{tgt}}) = \min_{\substack{(p_t, \mathbf{v}_t) \text{ s.t. } \partial_t p_t + \nabla \cdot (p_t \mathbf{v}_t) = 0, \\ p_0 = p_{\text{src}}, p_1 = p_{\text{tgt}}}} \int_0^1 \int_{\mathbb{R}^D} \|\mathbf{v}_t(\mathbf{x})\|^2 p_t(\mathbf{x}) d\mathbf{x} dt \quad (7.2.3)$$

where  $p_t$  is a probability distribution on  $\mathbb{R}^D$  for each  $t \in [0, 1]$ . In particular, The optimal transport flow  $p_t(\mathbf{x})$  follows *McCann's displacement interpolation*:

$$\mathbf{T}_t^*(\mathbf{x}) = (1 - t)\mathbf{x} + t\mathbf{T}^*(\mathbf{x}),$$

where  $\mathbf{T}^*(\mathbf{x})$  is the OT map that transports  $p_{\text{src}}$  to  $p_{\text{tgt}}$ . This linear interpolation moves mass along straight lines with constant velocity:  $p_t = \mathbf{T}_t^* \# p_{\text{src}}$  for each  $t \in [0, 1]$ .

The optimal transport map  $\mathbf{T}^*$  satisfies the *Monge–Ampère equation*:

$$p_{\text{tgt}}(\nabla \psi(\mathbf{x})) \det(\nabla^2 \psi(\mathbf{x})) = p_{\text{src}}(\mathbf{x}), \quad (7.2.4)$$

where  $\mathbf{T}^*(\mathbf{x}) = \nabla \psi(\mathbf{x})$  for some convex function  $\psi$  by Brenier's theorem. However, this nonlinear PDE is typically intractable for explicit solutions. Note that this is precisely the change-of-variables relation used by normalizing flows (c.f., Equation (5.0.1)): flows parametrize an invertible transport map with a tractable Jacobian determinant, but do not in general impose the gradient-of-potential structure  $\mathbf{T}^* = \nabla \psi$ ; consequently, a trained flow can differ substantially from the Brenier/OT map.

## 7.2.2 Entropy-Regularized Optimal Transport (EOT)

To motivate EOT concretely, consider empirical distributions built from samples. Suppose  $p_{\text{src}}$  is supported on points  $\{\mathbf{x}^{(i)}\}_{i=1}^n \subset \mathbb{R}^D$  with weights  $a_i$ , and  $p_{\text{tgt}}$  on  $\{\mathbf{y}^{(j)}\}_{j=1}^m \subset \mathbb{R}^D$  with weights  $b_j$ . A coupling is then an  $n \times n$  nonnegative matrix  $\gamma = (\gamma_{ij})$  whose row sums match  $a$  and column sums match  $b$ . Each entry  $\gamma_{ij}$  represents the amount of mass transported from  $\mathbf{x}^{(i)}$  to  $\mathbf{y}^{(j)}$ <sup>4</sup>.

**Why Regularize OT?** Classical OT in this discrete setting (obtained by taking counting measures in the continuous formulation Equation (7.2.1))

---

<sup>4</sup>Empirical (discrete) measures provide a principled proxy for continuous distributions. When the ground cost is  $c(x, y) = d(x, y)^p$  (so the OT value equals  $W_p^p$ ) and the measures have finite  $p$ th moments, the empirical measures converge to the population in  $W_p$  with quantitative rates; see Fournier and Guillin (2015) and the overview in Peyré, Cuturi, *et al.* (2019).

reduces to minimizing

$$\min_{\gamma=(\gamma_{ij})} \sum_{i,j} C_{ij} \gamma_{ij},$$

over all feasible couplings  $\gamma = (\gamma_{ij})$ , where  $C_{ij} = c(\mathbf{x}^{(i)}, \mathbf{y}^{(j)})$  is the cost of moving one unit of mass from source point  $\mathbf{x}^{(i)}$  to target point  $\mathbf{y}^{(j)}$ , for a prescribed ground cost  $c : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}_{\geq 0}$  (e.g.,  $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2^2$ ).

Two main issues arise:

1. **Non-Uniqueness and Instability:** The minimizer  $\gamma^*$  need not be unique. For example, if two transport plans achieve the same minimum cost, the solver may select either one. Consequently, small changes in the inputs  $(a, b, C)$  (such as moving a sample, adjusting weights, or slightly perturbing costs) can cause abrupt jumps in the solution.
2. **High Computational Cost:** The problem is a linear program with  $n^2$  variables and  $2n$  constraints. Practical solvers (e.g., Hungarian algorithm, network simplex (Peyré, Cuturi, *et al.*, 2019)) typically scale as  $\mathcal{O}(n^3)$ , which is infeasible for large  $n$ .

To overcome these bottlenecks, EOT objective function introduces a regularization term to the classical OT problem, controlled by a parameter  $\varepsilon > 0$ :

$$\text{EOT}_\varepsilon(p_{\text{src}}, p_{\text{tgt}}) := \min_{\gamma \in \Gamma(p_{\text{src}}, p_{\text{tgt}})} \int c(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{x}, \mathbf{y}) + \varepsilon \mathcal{D}_{\text{KL}}(\gamma \| M). \quad (7.2.5)$$

The reference measure  $M$  is typically chosen as the product of the marginals,  $p_{\text{src}} \otimes p_{\text{tgt}}$ . The KL divergence term is directly related to the Shannon entropy of the transport plan  $\gamma$ :

$$\mathcal{D}_{\text{KL}}(\gamma \| p_{\text{src}} \otimes p_{\text{tgt}}) = -\mathcal{H}(\gamma) + \text{Constant},$$

where  $\mathcal{H}(\gamma) := - \int \gamma(\mathbf{x}, \mathbf{y}) \log \gamma(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y}$ .

The addition of this regularization term yields several theoretical and practical advantages, which we briefly outline below:

### Why Entropy Regularizer Helps?

1. **Mass Spreading.** Since  $t \mapsto t \log t$  is convex and grows rapidly for large  $t$ , minimizing  $\int \gamma \log \gamma$  penalizes *peaky* couplings (some  $\gamma(\mathbf{x}, \mathbf{y})$  very large, most near zero). For fixed total mass  $\int \gamma = 1$ , it favors plans where  $\gamma(\mathbf{x}, \mathbf{y})$  is more evenly distributed over  $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^D \times \mathbb{R}^D$ . Equivalently, maximizing Shannon entropy promotes higher “uncertainty” (diffuseness).
2. **Strict Convexity and Uniqueness.** Because  $\mathcal{H}$  is strictly concave, the objective in Equation (7.2.5) is strictly convex in  $\gamma$ , yielding a *unique* minimizer  $\gamma_\varepsilon^*$  that depends continuously on  $(p_{\text{src}}, p_{\text{tgt}}, c)$ .
3. **Sinkhorn Form and Positivity.** Under mild conditions<sup>5</sup>, the optimizer has the *Schrödinger/Sinkhorn form*

$$\gamma_\varepsilon^*(\mathbf{x}, \mathbf{y}) = u(\mathbf{x}) \exp\left(-\frac{c(\mathbf{x}, \mathbf{y})}{\varepsilon}\right) v(\mathbf{y}) p_{\text{src}}(\mathbf{x}) p_{\text{tgt}}(\mathbf{y}),$$

for positive scaling functions  $u, v$  (unique up to a global factor). In practice, the continuous formulation is approximated with finite samples, reducing EOT to a finite (sampled) Sinkhorn iteration. The entropic objective is strictly convex, and the scaling (Sinkhorn/IPFP) algorithm solves it efficiently (Sinkhorn, 1964; Cuturi, 2013). For a dense problem with  $n$  support points per marginal (an  $n \times n$  kernel), each Sinkhorn iteration costs  $\mathcal{O}(n^2)$  time and  $\mathcal{O}(n^2)$  memory, making the method more scalable and practical (Altschuler *et al.*, 2017).

4. **Limits in  $\varepsilon$ .** As  $\varepsilon \rightarrow 0$ , the optimal plan  $\gamma_\varepsilon^*$  becomes increasingly concentrated, approaching a (possibly singular) classical OT coupling (we will revisit this connection in Section 7.3.2). As  $\varepsilon$  increases,  $\gamma_\varepsilon^*$  gradually spreads out and approaches the independent coupling  $p_{\text{src}} \otimes p_{\text{tgt}}$ .

### 7.2.3 Schrödinger Bridge (SB)

**KL Formulation of SB.** The Schrödinger Bridge (SB) problem, introduced by Erwin Schrödinger in the 1930s, asks the following question. Suppose particles move according to some simple reference dynamics, such as Brownian motion. Now imagine that we observe the particles at two times: at  $t = 0$  their distribution is  $p_{\text{src}}$ , and at  $t = 1$  it is  $p_{\text{tgt}}$ . Among all possible stochastic processes that connect these two distributions, which one deviates the least from

---

<sup>5</sup>We assume that  $c < \infty$  holds  $p_{\text{src}} \otimes p_{\text{tgt}}$ -almost everywhere, and that the marginal kernel integrals are finite and positive. For simplicity, we focus on the case where  $\gamma_\varepsilon^*$ ,  $p_{\text{src}}$ , and  $p_{\text{tgt}}$  admit densities with respect to the Lebesgue measure.

the reference dynamics? Here “deviation” is measured by the KL divergence, so the solution to the SB problem is the most likely way to deform Brownian motion into a process that satisfies the prescribed boundary conditions.

To make this precise, let  $\mathbf{x}_{0:T} := \{\mathbf{x}_t\}_{t \in [0,T]}$  denote a complete trajectory of the process. We write  $P$  for the *law of trajectories*, that is, the probability distribution over entire sample paths. The time- $t$  marginal of  $P$  is denoted by  $p_t$  (or  $P_t$ ), which describes the distribution of the state  $\mathbf{x}_t$  at a single time. Formally, for a measurable set  $A \subseteq \mathbb{R}^D$ ,

$$p_t(A) = P(\mathbf{x}_t \in A).$$

In other words,  $p_t$  can be viewed as the empirical distribution obtained by sampling many full trajectories from  $P$  and then collecting the states at time  $t$ —for instance, as a histogram if the state is one-dimensional.

Consider a reference diffusion  $\{\mathbf{x}_t\}_{t \in [0,T]}$  governed by the SDE

$$d\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t) dt + g(t) d\mathbf{w}_t, \quad (7.2.6)$$

where  $\mathbf{f}: \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}^D$ ,  $g: [0, T] \rightarrow \mathbb{R}$ , and  $\{\mathbf{w}_t\}_{t \in [0, T]}$  is a standard Brownian motion. Let  $R$  denote the *path law* (joint distribution) of the full trajectory  $\mathbf{x}_{0:T} := \{\mathbf{x}_t\}_{t \in [0, T]}$ ; this  $R$  will serve as the *reference* trajectory distribution.

With this notation, the Schrödinger Bridge (SB) problem seeks a trajectory law  $P$  that is closest to  $R$  in KL divergence while matching the prescribed endpoint marginals:

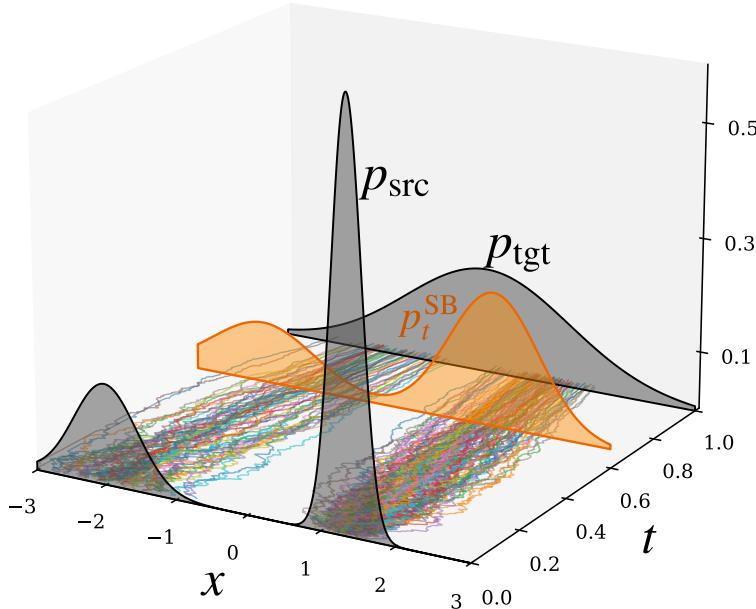
$$\text{SB}(p_{\text{src}}, p_{\text{tgt}}) := \min_P \mathcal{D}_{\text{KL}}(P \| R) \quad \text{s.t.} \quad P_0 = p_{\text{src}}, \quad P_T = p_{\text{tgt}}. \quad (7.2.7)$$

The optimizer  $P^*$  depends on the chosen reference process  $R$ .

**Stochastic control view of SB.** Rather than optimizing over arbitrary path distributions  $P$  in Equation (7.2.7), a more tractable approach is to take the reference dynamics as an anchor and allow it to drift. This is done by introducing a time-dependent drift  $\mathbf{v}_t(\mathbf{x}_t)$ , which perturbs the reference process and generates a family of candidate trajectory distributions. The resulting dynamics take the form of a *controlled diffusion*:

$$d\mathbf{x}_t = [\mathbf{f}(\mathbf{x}_t, t) + \mathbf{v}_t(\mathbf{x}_t)] dt + g(t) d\mathbf{w}_t,$$

where  $\mathbf{v}_t: \mathbb{R}^D \rightarrow \mathbb{R}^D$  is the drift to be optimized later (Equation (7.2.8)). Under standard integrability conditions (e.g., Novikov’s condition) and by Girsanov’s



**Figure 7.2: Illustration of stochastic control view of SB.** The bridge seeks the stochastic path that deviates least from the reference while connecting  $p_{\text{src}}$  and  $p_{\text{tgt}}$ .

theorem (see Section C.2.1), the KL divergence between the controlled law  $P$  and the reference  $R$  admits the dynamic (kinetic) form

$$\mathcal{D}_{\text{KL}}(P\|R) = \mathbb{E}_P \left[ \frac{1}{2} \int_0^T \frac{\|\mathbf{v}_t(\mathbf{x}_t)\|^2}{g^2(t)} dt \right] = \frac{1}{2} \int_0^T \int_{\mathbb{R}^D} \frac{\|\mathbf{v}_t(\mathbf{x})\|^2}{g^2(t)} p_t(\mathbf{x}) d\mathbf{x} dt,$$

where  $p_t$  is the time- $t$  marginal of  $\mathbf{x}_t$  under the controlled process. The second equality follows from the law of total expectation.

Hence, the SB problem can be reformulated as minimizing the expected control energy over all admissible drifts  $\mathbf{v}_t$  that steer the process from  $p_{\text{src}}$  at  $t = 0$  to  $p_{\text{tgt}}$  at  $t = T$  (Dai Pra, 1991; Pra and Pavon, 1990; Pavon and Wakolbinger, 1991; Chen *et al.*, 2016). This leads to the *stochastic control formulation*:

$$\begin{aligned} & \text{SB}_\varepsilon(p_{\text{src}}, p_{\text{tgt}}) \\ = & \min_{\substack{\mathbf{v}_t \text{ s.t. } d\mathbf{x}_t = [\mathbf{f}(\mathbf{x}_t, t) + \mathbf{v}_t(\mathbf{x}_t)] dt + g(t) d\mathbf{w}_t, \\ \mathbf{x}_0 \sim p_{\text{src}}, \mathbf{x}_T \sim p_{\text{tgt}}}} \frac{1}{2} \int_0^T \int_{\mathbb{R}^D} \frac{\|\mathbf{v}_t(\mathbf{x})\|^2}{g^2(t)} p_t(\mathbf{x}) d\mathbf{x} dt, \end{aligned} \quad (7.2.8)$$

Importantly, the endpoint distributions  $p_{\text{src}}$  and  $p_{\text{tgt}}$  are arbitrary; the control  $\mathbf{v}_t$  is chosen precisely to “bridge” the reference dynamics between these marginals while staying as close as possible (in KL divergence) to the reference process  $R$ .

**A Special Brownian Reference.** Equation (7.2.8) resembles the Benamou–Brenier formulation of OT in Equation (7.2.3), especially when the reference process  $R^\varepsilon$  (with  $\varepsilon > 0$ ) is chosen to be a Brownian motion:

$$d\mathbf{x}_t = \sqrt{\varepsilon} d\mathbf{w}_t,$$

so that  $\mathbf{f} \equiv \mathbf{0}$  and  $g(t) \equiv \sqrt{\varepsilon}$ .

In this setting, the SB problem seeks a path distribution  $P$  that stays closest (in KL divergence) to the Brownian reference  $R^\varepsilon$ , while matching the endpoint marginals:

$$\text{SB}_\varepsilon(p_{\text{src}}, p_{\text{tgt}}) := \min_P \mathcal{D}_{\text{KL}}(P \| R^\varepsilon) \quad \text{s.t.} \quad P_0 = p_{\text{src}}, P_T = p_{\text{tgt}}. \quad (7.2.9)$$

The equivalent stochastic control formulation then becomes

$$\text{SB}_\varepsilon(p_{\text{src}}, p_{\text{tgt}}) = \min_{\substack{\mathbf{v}_t \text{ s.t. } d\mathbf{x}_t = \sqrt{\varepsilon} d\mathbf{w}_t, \\ \mathbf{x}_0 \sim p_{\text{src}}, \mathbf{x}_T \sim p_{\text{tgt}}}} \frac{1}{2\varepsilon} \int_0^T \int_{\mathbb{R}^D} \|\mathbf{v}_t(\mathbf{x})\|^2 p_t(\mathbf{x}) d\mathbf{x} dt. \quad (7.2.10)$$

**Why We Need to Specify a Reference Distribution?** Unlike in classical OT, the SB problem requires a reference distribution due to its stochastic nature. In OT, the cost function (e.g.,  $c(\mathbf{x}, \mathbf{y}) \propto \|\mathbf{x} - \mathbf{y}\|^2$ ) implicitly defines a unique, deterministic geodesic path, making a reference unnecessary. In contrast, the SB setting admits infinitely many stochastic processes connecting the marginals, with no intrinsic notion of a “natural” path. The reference measure  $R$  encodes the system’s underlying physics or geometric structure (e.g., Brownian motion) and defines the KL-based optimization objective  $\mathcal{D}_{\text{KL}}(P \| R)$ , without which the notion of optimality is undefined.

**Coupled PDE Characterization.** A convenient way to describe the SB solution is through two space–time potentials  $\Psi(x, t)$  and  $\widehat{\Psi}(x, t)$ . Let  $p_t^{\text{SB}}$  denote the marginal at time  $t \in [0, T]$  of the optimal trajectory law  $P^*$  in Equation (7.2.7). Then one has the symmetric factorization (Dai Pra, 1991)

$$p_t^{\text{SB}}(x) = \Psi(x, t)\widehat{\Psi}(x, t), \quad (7.2.11)$$

where  $\Psi$  and  $\widehat{\Psi}$  solve the (linear) *Schrödinger system* (Caluya and Halder, 2021; Chen *et al.*, 2021; Chen *et al.*, 2022):

$$\begin{aligned} \frac{\partial \Psi}{\partial t}(\mathbf{x}, t) &= -\nabla_{\mathbf{x}}\Psi(\mathbf{x}, t) \cdot \mathbf{f}(\mathbf{x}, t) - \frac{g^2(t)}{2}\Delta_{\mathbf{x}}\Psi(\mathbf{x}, t), \\ \frac{\partial \widehat{\Psi}}{\partial t}(\mathbf{x}, t) &= -\nabla_{\mathbf{x}} \cdot (\widehat{\Psi}(\mathbf{x}, t) \mathbf{f}(\mathbf{x}, t)) + \frac{g^2(t)}{2}\Delta_{\mathbf{x}}\widehat{\Psi}(\mathbf{x}, t) \end{aligned} \quad (7.2.12)$$

subject to

$$\Psi(\mathbf{x}, 0)\widehat{\Psi}(\mathbf{x}, 0) = p_{\text{src}}(\mathbf{x}), \quad \Psi(\mathbf{x}, T)\widehat{\Psi}(\mathbf{x}, T) = p_{\text{tgt}}(\mathbf{x}).$$

**Forward-Time Schrödinger Bridge SDE.** Once  $\Psi$  is known, the optimal dynamics is the reference diffusion tilted by the space–time factor  $\Psi$ :

$$d\mathbf{x}_t = [\mathbf{f}(\mathbf{x}_t, t) + g^2(t)\nabla_{\mathbf{x}}\log\Psi(\mathbf{x}_t, t)]dt + g(t)d\mathbf{w}_t, \quad \mathbf{x}_0 \sim p_{\text{src}}. \quad (7.2.13)$$

Let  $Q$  denote the trajectory law of Equation (7.2.13) (so  $Q_0 = p_{\text{src}}$  and  $Q_T = p_{\text{tgt}}$  by Equations (7.2.11) and (7.2.12)). Then  $Q = P^*$  and the minimizer  $\mathbf{v}^*$  to Equation (7.2.8) is (see (Chen *et al.*, 2021)’s Section 4.6):

$$\mathbf{v}_t^*(\mathbf{x}) = g^2(t)\nabla_{\mathbf{x}}\log\Psi(\mathbf{x}, t).$$

That is, drift correction  $g^2\nabla_{\mathbf{x}}\log\Psi$  is precisely the minimal KL perturbation of the reference needed to match the endpoint marginals.

**Reverse-Time Schrödinger Bridge SDE.** The same optimal path law can be generated reverse in time. A convenient way to see the form of the reverse-time drift is to conceptually use the standard time-reversal identity for diffusions:

$$\mathbf{b}^-(\mathbf{x}, t) = \mathbf{b}^+(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}}\log p_t^{\text{SB}}(\mathbf{x}),$$

where  $\mathbf{b}^+ = \mathbf{f} + g^2\nabla\log\Psi$  and  $p_t = \Psi\widehat{\Psi}$ . This gives

$$\mathbf{b}^-(\mathbf{x}, t) = \mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}}\log\widehat{\Psi}(\mathbf{x}, t).$$

Thus the reverse-time SDE reads

$$d\mathbf{x}_t = \left[ \mathbf{f}(\mathbf{x}_t, t) - g^2(t) \nabla_{\mathbf{x}} \log \hat{\Psi}(\mathbf{x}_t, t) \right] dt + g(t) d\bar{\mathbf{w}}_t, \quad \mathbf{x}_T \sim p_{\text{tgt}}. \quad (7.2.14)$$

Equivalently, reparametrizing time by  $\mathbf{y}_\tau := \mathbf{x}_{T-\tau}$  so that  $\tau$  increases from 0 to  $T$ . Then  $\mathbf{y}_\tau$  evolves forward in  $\tau$  from  $\mathbf{y}_0 \sim p_{\text{tgt}}$  as

$$\begin{aligned} d\mathbf{y}_\tau = & [-\mathbf{f}(\mathbf{y}_\tau, T-\tau) + g^2(T-\tau) \nabla_{\mathbf{y}} \log \hat{\Psi}(\mathbf{y}_\tau, T-\tau)] d\tau \\ & + g(T-\tau) d\mathbf{w}_\tau. \end{aligned} \quad (7.2.15)$$

In the reverse-time stochastic control formulation of Equation (7.2.8)(same quadratic energy with the reversed clock):

$$\min_{\mathbf{u}_\tau \text{ s.t. } d\mathbf{y}_\tau = [-\mathbf{f}(\mathbf{y}_\tau, T-\tau) + \mathbf{u}_\tau(\mathbf{y}_\tau)] d\tau + g(T-\tau) d\mathbf{w}_\tau, \mathbf{y}_0 \sim p_{\text{tgt}}, \mathbf{y}_T \sim p_{\text{src}}} \frac{1}{2} \int_0^T \int_{\mathbb{R}^D} \frac{\|\mathbf{u}_\tau(\mathbf{y})\|^2}{g^2(T-\tau)} p_{T-\tau}(\mathbf{y}) d\mathbf{y} d\tau. \quad (7.2.16)$$

the optimal control is

$$\mathbf{u}_t^*(\mathbf{x}) = -g^2(t) \nabla_{\mathbf{x}} \log \hat{\Psi}(\mathbf{x}, t).$$

Both the forward and reverse descriptions yield the same optimal path law  $P^*$  which are linked by

$$\nabla \log p_t^{\text{SB}} = \nabla \log \Psi + \nabla \log \hat{\Psi}, \quad \mathbf{b}^- = \mathbf{b}^+ - g^2 \nabla \log p_t^{\text{SB}},$$

so their marginals coincide with  $p_t^{\text{SB}}$  at every time. The additional drift terms  $g^2 \nabla \log \Psi$  (forward) and  $-g^2 \nabla \log \hat{\Psi}$  (reverse-time) act as control forces that steer the reference diffusion to match the endpoint marginals while staying closest to the reference in relative entropy.

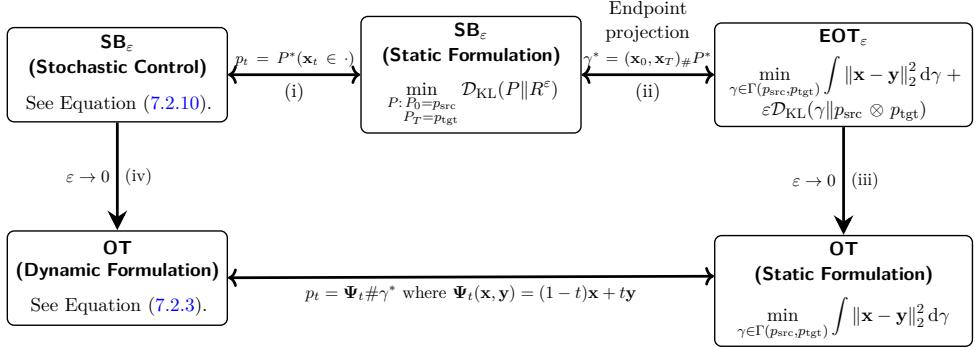
**Practical Obstacles to the Coupled PDE Approach.** To construct the generative process based on Equation (7.2.14), one must solve the coupled PDEs in Equation (7.2.12) to obtain the backward Schrödinger potential  $\hat{\Psi}$ . However, these PDEs are notoriously difficult to solve, even in low-dimensional settings, which makes their direct application in generative modeling challenging. To circumvent this, several works have proposed alternative strategies: leveraging Score SDE techniques to iteratively solve each half-bridge problem ( $p_{\text{tgt}} \leftarrow p_{\text{src}}$  and  $p_{\text{tgt}} \rightarrow p_{\text{src}}$ ) (De Bortoli *et al.*, 2021); optimizing surrogate likelihood bounds (Chen *et al.*, 2022; Liu *et al.*, 2023); or designing simulation-free training based on an analytical solution of the posterior  $\mathbf{x}_t | \mathbf{x}_0, \mathbf{x}_T$  for sample pairs  $(\mathbf{x}_0, \mathbf{x}_T) \sim p_{\text{src}} \otimes p_{\text{tgt}}$  (Liu *et al.*, 2023). We do not delve into the technical details here but briefly discuss the connection between diffusion models and SB in Section 7.4.

### 7.2.4 Global Pushforwards and Local Dynamics: An OT Analogy for DGMs

From the optimal-transport viewpoint (in Equation (7.2.1)), one can leverage deep generative models to learn a transport (pushforward) map from a simple prior to the data, i.e.,  $\mathbf{G}_\phi \# p_{\text{prior}} \approx p_{\text{data}}$ . Although  $\mathbf{G}_\phi$  generally does not coincide with the optimal transport map (except in works (Genevay *et al.*, 2018; Onken *et al.*, 2021) that impose an OT objective under suitable conditions), the Benamou–Brenier formulation (in Equation (7.2.3)) provides a complementary, dynamic perspective. Rather than directly learning a single global map, it describes transport as a continuous flow generated by a time-dependent local vector field, tracing a smooth path between  $p_{\text{prior}}$  and  $p_{\text{data}}$ . This dynamic formulation parallels the relationship between the static Schrödinger Bridge problem (in Equation (7.2.7)) and its stochastic-control counterpart (in Equation (7.2.8)), where the optimal coupling is realized as a controlled diffusion process. A similar analogy emerges in generative modeling: standard DGMs such as GANs or VAEs learn a global pushforward map, whereas diffusion models learn a time-dependent local vector field that drives the generative dynamics.

### 7.3 Relationship of Variant Optimal Transport Formulations

**Figure 7.3: Relationship between variants of optimal transport with  $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2^2$  and Reference  $R^\varepsilon$  in SB.** We summarize the equivalences: (i)  $\text{SB}_\varepsilon$  (stochastic control)  $\Leftrightarrow$   $\text{SB}_\varepsilon$  (Static formulation), where  $p_t$  is the time- $t$  marginal of the path measure  $P$ ; (ii)  $\text{SB}_\varepsilon$  (Static formulation)  $\Leftrightarrow$   $\text{EOT}_\varepsilon$  (see Section 7.3.1); (iii)  $\text{EOT}_\varepsilon$  (Static formulation)  $\Leftrightarrow$   $\text{OT}_\varepsilon$  (static) (see Section 7.3.2); (iv)  $\text{SB}_\varepsilon$  (stochastic control)  $\Leftrightarrow$   $\text{OT}$  (dynamic) (see Section 7.3.3).



Before delving into the technical details, it is helpful to clarify how the different formulations of optimal transport and its entropic regularizations are connected. At a high level, these problems can be viewed as related (see Figure 7.3 for their diagram for connection):

- (i) SB problem  $\text{SB}_\varepsilon$  with the specific reference  $R^\varepsilon$  given by Brownian motion

$$d\mathbf{x}_t = \sqrt{\varepsilon} d\mathbf{w}_t$$

is equivalent to its static formulation: the evolving marginals  $p_t$  are precisely the time- $t$  slices of the optimal path measure  $P$  (see Section 7.2.3);

- (ii) Static formulation of  $\text{SB}_\varepsilon$  connects directly to the entropic OT problem,  $\text{EOT}_\varepsilon$  (see Section 7.3.1);
- (iii)  $\text{EOT}_\varepsilon$ , in turn, can be related back to the static formulation of entropic OT,  $\text{OT}_\varepsilon$  (see Section 7.3.2);
- (iv) Stochastic control perspective of  $\text{SB}_\varepsilon$  can also be linked to the dynamic formulation of classical OT (see Section 7.3.3).

Together, these non-trivial relationships provide a compact view across stochastic control, entropy-regularized, and classical OT frameworks.

### 7.3.1 SB and EOT are (Dual) Equivalent

In this section, we present two complementary perspectives showing that SB are essentially equivalent to EOT. Unlike classical optimal transport, which produces a single deterministic map, SB yields a *stochastic* flow of particles: mass is transported probabilistically, with marginals evolving under diffusion-like dynamics.

From the static viewpoint, SB coincides with EOT, where the goal is to find a coupling between the two endpoint distributions that balances transport cost with entropy. From the dynamic viewpoint, SB describes a controlled diffusion process that remains as close as possible to a simple reference (such as Brownian motion) while still matching the desired endpoints. Each perspective independently establishes the equivalence, providing two consistent ways to understand SB/EOT as canonical formulations of distribution-to-distribution transformation.

**Static Schrödinger Bridge.** Let

$$\tilde{R}^\varepsilon(\mathbf{x}, \mathbf{y}) := \frac{1}{Z_\varepsilon} e^{-c(\mathbf{x}, \mathbf{y})/\varepsilon} p_{\text{src}}(\mathbf{x}) p_{\text{tgt}}(\mathbf{y}),$$

with a normalizing constant:

$$Z_\varepsilon := \iint e^{-c(\mathbf{x}, \mathbf{y})/\varepsilon} p_{\text{src}}(\mathbf{x}) p_{\text{tgt}}(\mathbf{y}) d\mathbf{x} d\mathbf{y}.$$

Then the entropic OT objective

$$\begin{aligned} \min_{\gamma \in \Gamma(p_{\text{src}}, p_{\text{tgt}})} \left\{ \int c d\gamma + \varepsilon \mathcal{D}_{\text{KL}}(\gamma \| p_{\text{src}} \otimes p_{\text{tgt}}) \right\} &= \varepsilon \min_{\gamma \in \Gamma(p_{\text{src}}, p_{\text{tgt}})} \mathcal{D}_{\text{KL}}(\gamma \| \tilde{R}^\varepsilon) \\ &\quad - \varepsilon \log Z_\varepsilon, \end{aligned} \tag{7.3.1}$$

so it is equivalent (up to an additive constant) to the static Schrödinger Bridge (in Equation (7.2.9)):

$$\min_{\gamma \in \Gamma} \mathcal{D}_{\text{KL}}(\gamma \| \tilde{R}^\varepsilon).$$

**Dynamic Equivalence (Brownian Reference).** We can also view the equivalence from the dynamic equivalence which is the A classical result (Mikami and Thieullen, 2006) says that entropic OT with quadratic cost

$$c(\mathbf{x}, \mathbf{y}) = \frac{\|\mathbf{y} - \mathbf{x}\|^2}{2T}$$

is affinely equivalent to the SB problem where the reference path law  $R^\varepsilon$  is Brownian motion on  $[0, T]$ ,

$$d\mathbf{x}_t = \sqrt{\varepsilon} d\mathbf{w}_t.$$

Here, “affinely equivalent” means the optimal values differ by a positive scaling and an additive constant (independent of the decision variable), so the minimizers coincide. In particular, let  $P^*$  be the optimal path distribution for SB and let  $\gamma^*$  be the optimal transport plan for EOT. Then if  $\mathbf{x}_{[0:T]} \sim P^*$ , the pair of endpoints  $(\mathbf{x}_0, \mathbf{x}_T)$  has distribution  $\gamma^*$ :

$$P^* \text{ solves SB} \iff \gamma^* \text{ solves EOT and } (\mathbf{x}_0, \mathbf{x}_T) \sim \gamma^*.$$

In words: the optimal process from the dynamic (SB) problem induces the optimal coupling for the static (EOT) problem. Conversely, (under mild conditions on the heat kernel,) any optimal static coupling can be realized as the endpoints of some optimal SB process.

The key idea to derive this fact is that the KL divergence over paths can be broken down according to the endpoints, which means the Schrödinger bridge problem reduces to a KL divergence just over the joint distribution of  $(\mathbf{x}_0, \mathbf{x}_T)$ . For Brownian motion the transition density between  $\mathbf{x}$  and  $\mathbf{y}$  has a Gaussian form, so its negative log is quadratic:

$$-\varepsilon \log p_T(\mathbf{y} | \mathbf{x}) = \frac{\|\mathbf{y} - \mathbf{x}\|^2}{2T} + \text{const.}$$

This shows that the endpoint KL is exactly the same as the entropic OT objective with quadratic cost, up to an irrelevant constant.

**SB with General Reference Determines the EOT Cost.** As we discussed in Equation (7.2.7), the SB problem is not restricted to Brownian motion; it can be defined with any (well-posed) reference process. This choice uniquely determines the cost function in the corresponding EOT problem. The key connection is that the SB *reference dynamics* induce the EOT *cost function*.

Let the reference process be governed by an SDE over  $[0, T]$ , yielding a transition density  $p_T(\mathbf{y} | \mathbf{x})$ , the likelihood of reaching  $\mathbf{y}$  at time  $T$  from  $\mathbf{x}$  at time 0. Then, the EOT cost function is given (up to a scaling constant) by

$$c(\mathbf{x}, \mathbf{y}) \propto -\log p_T(\mathbf{y} | \mathbf{x}).$$

With this cost, solving the SB problem becomes equivalent to solving an EOT problem. In short, choosing the reference dynamics in SB is mathematically equivalent to specifying the transport cost in EOT. By Equation (7.3.1), the entropic OT objective differs from the static SB objective; hence the two problems are equivalent and have the same minimizer.

### 7.3.2 $\text{EOT}_\varepsilon$ is Reduced to OT as $\varepsilon \rightarrow 0$

Let  $\gamma_\varepsilon^*$  denote the optimal plan for the  $\text{EOT}_\varepsilon$ , and let  $\gamma^*$  be an optimal plan for the unregularized OT problem in Equation (7.2.1). The following result (Mikami and Thieullen, 2008; Peyré, Cuturi, *et al.*, 2019) shows that as  $\varepsilon \rightarrow 0$ , the entropic optimal plan  $\gamma_\varepsilon^*$  converges (in a suitable sense) to the OT plan  $\gamma^*$ , and the EOT cost converges to the OT cost.

This convergence result is both fundamental and practically important. One of the reasons is that the entropy-regularized OT problem  $\text{EOT}_\varepsilon$  admits efficient numerical solutions via algorithms such as Sinkhorn. Thus, the result provides theoretical justification for using  $\text{EOT}_\varepsilon$  with small  $\varepsilon$  as a computationally tractable proxy for the classical OT problem in Equation (7.2.1), even when the cost function  $c(\mathbf{x}, \mathbf{y})$  is more general than the quadratic case.

#### Theorem 7.3.1: (Informal) $\text{EOT}_\varepsilon$ Converges to OT.

As  $\varepsilon \rightarrow 0$ , the optimal values converge:

$$\lim_{\varepsilon \rightarrow 0} \text{EOT}_\varepsilon(p_{\text{src}}, p_{\text{tgt}}) = \text{OT}(p_{\text{src}}, p_{\text{tgt}}).$$

Moreover, the optimal plans  $\gamma_\varepsilon^*$  converge weakly to  $\gamma^*$ . That is,

$$\mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \gamma_\varepsilon^*}[g(\mathbf{x}, \mathbf{y})] \rightarrow \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \gamma^*}[g(\mathbf{x}, \mathbf{y})],$$

for all bounded continuous (test) functions  $g : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ .

#### Proof for Theorem.

For a rigorous proof, we refer to the literature (Mikami and Thieullen, 2008; Peyré, Cuturi, *et al.*, 2019). Below we provide a heuristic derivation of the value convergence.

Let us denote the corresponding optimal values by

$$V_\varepsilon := \text{EOT}_\varepsilon(p_{\text{src}}, p_{\text{tgt}}), \quad V_0 := \text{OT}(p_{\text{src}}, p_{\text{tgt}})$$

for notational simplicity.

**Upper Bound.** By optimality of  $\gamma_\varepsilon^*$ , its value  $V_\varepsilon$  is bounded by the cost of using the plan  $\gamma^*$ :

$$V_\varepsilon \leq \int c \, d\gamma^* + \varepsilon \mathcal{D}_{\text{KL}}(\gamma^* \| p_{\text{src}} \otimes p_{\text{tgt}}).$$

Assuming the KL term is a finite constant  $K$ , we get  $V_\varepsilon \leq V_0 + \varepsilon K$ . Taking the limit superior yields  $\limsup_{\varepsilon \rightarrow 0} V_\varepsilon \leq V_0$ .

**Lower Bound.** Since the KL-divergence is non-negative,  $V_\varepsilon \geq \int c d\gamma_\varepsilon^*$ . By definition of  $V_0$  as the minimal transport cost, any plan's cost is at least  $V_0$ , so  $\int c d\gamma_\varepsilon^* \geq V_0$ . This implies  $V_\varepsilon \geq V_0$  for all  $\varepsilon > 0$ , and thus  $\liminf_{\varepsilon \rightarrow 0} V_\varepsilon \geq V_0$ .

Combining the upper and lower bounds shows the convergence of the optimal value,  $\lim_{\varepsilon \rightarrow 0} V_\varepsilon = V_0$ . The convergence of the optimal plan itself,  $\gamma_\varepsilon^* \rightarrow \gamma^*$  in the weak sense, is a more advanced result from  $\Gamma$ -convergence theory that we omit. ■

### 7.3.3 $\text{SB}_\varepsilon$ is Reduced to OT as $\varepsilon \rightarrow 0$

For each  $\varepsilon > 0$ , let  $\mathbf{v}_t^\varepsilon$  be a minimizer of the SB problem as in Equation (7.2.10), and let  $p_t^\varepsilon$  be the marginal distribution of the controlled SDE  $\mathbf{x}_t$  induced by  $\mathbf{v}_t^\varepsilon$ . Then  $p_t^\varepsilon$  satisfies the associated Fokker–Planck equation. In contrast, denote by  $(p_t^0, \mathbf{v}_t^0)$  a minimizer of the Benamou–Brenier formulation of optimal transport (see Equation (7.2.3)).

The following theorem<sup>6</sup> states that as  $\varepsilon \rightarrow 0$ , the SB problem converges to the OT problem. This result is practically important for reasons similar to those in Theorem 7.3.1. The objective  $\text{SB}_\varepsilon$  can be efficiently solved using Sinkhorn type algorithms, yielding a numerically tractable and differentiable proxy for optimal transport. This is especially valuable in high dimensional or large scale settings, where direct solvers (e.g., based on the Benamou–Brenier formulation) become computationally expensive.

#### Theorem 7.3.2: (Informal) $\text{SB}_\varepsilon$ Converges to OT.

As  $\varepsilon \rightarrow 0$ , we have:

$$\lim_{\varepsilon \rightarrow 0} \text{SB}_\varepsilon(p_{\text{src}}, p_{\text{tgt}}) = \text{OT}(p_{\text{src}}, p_{\text{tgt}}),$$

where OT is of the Benamou–Brenier formulation as in Equation (7.2.3). Moreover,  $p_t^\varepsilon$  converges weakly to  $p_t^0$ , and  $\mathbf{v}_t^\varepsilon$  converges weakly to  $\mathbf{v}_t^0$  in the appropriate function spaces.

#### Proof for Theorem.

---

<sup>6</sup>We remark that the convergence of the optimal values in the theorem is in the sense of  $\Gamma$ -convergence, rather than a classical pointwise limit. Although this requires more technical background, we omit the details here and state only the conceptual result.

A full rigorous proof of the convergence result is beyond our scope; we refer the reader to Léonard (2012) and Léonard (2014) for detailed derivations. Nevertheless, we can heuristically understand why this convergence may hold.

In the stochastic control formulation of the SB problem Equation (7.2.10), the controlled SDE are given by:

$$d\mathbf{x}_t = \mathbf{v}_t^\varepsilon(\mathbf{x}_t)dt + \sqrt{2\varepsilon}d\mathbf{w}_t.$$

As  $\varepsilon \rightarrow 0$ , the noise term vanishes, and the SDE formally approaches a deterministic ODE:

$$d\mathbf{x}_t = \mathbf{v}_t^0(\mathbf{x}_t)dt.$$

This suggests that the optimal value of the SB problem converges to that of the optimal transport problem:

$$\lim_{\varepsilon \rightarrow 0} \text{SB}_\varepsilon(p_{\text{src}}, p_{\text{tgt}}) = \text{OT}(p_{\text{src}}, p_{\text{tgt}}).$$

In parallel, the marginal density  $p_t^\varepsilon$  satisfies the Fokker–Planck equation:

$$\partial_t p_t^\varepsilon + \nabla \cdot (p_t^\varepsilon \mathbf{v}_t^\varepsilon) = \varepsilon \Delta p_t^\varepsilon.$$

Again, as  $\varepsilon \rightarrow 0$ , the diffusion term vanishes, and the equation formally reduces to the continuity equation:

$$\partial_t p_t^0 + \nabla \cdot (p_t^0 \mathbf{v}_t^0) = 0.$$

Until now, we have presented the fundamental equivalences (under their respective assumptions) between EOT and SB, as well as their important connection to OT through a limiting process, illustrated in Figure 7.3. Next, we will explore how diffusion models connect to these concepts.

## 7.4 Is Diffusion Model's SDE Optimal Solution to SB Problem?

### 7.4.1 Diffusion models as a Special Case of Schrödinger Bridges

SB framework extends (score-based) diffusion models by enabling nonlinear interpolation between arbitrary source and target distributions. It achieves this by adding control drift terms derived from scalar potentials  $\Psi(\mathbf{x}, t)$  and  $\hat{\Psi}(\mathbf{x}, t)$ , which guide a reference diffusion process to match prescribed endpoint marginals (see Equation (7.2.12)) and follow the decomposition:

$$\nabla \log \Psi(\mathbf{x}, t) + \nabla \log \hat{\Psi}(\mathbf{x}, t) = \nabla \log p_t^{\text{SB}}(\mathbf{x}).$$

This generalization allows the model to move beyond standard Gaussian priors and generate samples from broader distributions.

**Connection to Diffusion Models.** Diffusion models arise as a special case of the SB framework. Suppose the potential is constant,  $\Psi(\mathbf{x}, t) \equiv 1$ . Under this assumption, the second PDE in Equation (7.2.12) reduces to the standard Fokker–Planck equation, whose solution is the marginal density of the reference process:

$$\hat{\Psi}(\mathbf{x}, t) = p_t^{\text{SB}}(\mathbf{x}). \quad (7.4.1)$$

The corresponding SB forward SDE thus becomes the uncontrolled reference process:

$$d\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t) dt + g(t) d\mathbf{w}_t,$$

and the SB backward SDE simplifies to:

$$d\mathbf{x}_t = \left[ \mathbf{f}(\mathbf{x}_t, t) - g^2(t) \nabla \log p_t^{\text{SB}}(\mathbf{x}_t) \right] dt + g(t) d\bar{\mathbf{w}}_t,$$

which matches Anderson’s reverse-time SDE used in diffusion models. This correspondence shows that diffusion models can be interpreted as the zero-control limit of SB, where no additional drift is introduced by the potentials.

**Boundary Conditions and Generality.** The above reduction is purely formal unless the boundary constraints are compatible. For arbitrary source/target ( $p_{\text{src}}, p_{\text{tgt}}$ ), the PDE boundary conditions in Equation (7.2.12) are generally not satisfied by the choice  $\Psi \equiv 1$ . Full SB resolves this by learning nontrivial potentials that induce a nonlinear control drift, bending the reference dynamics to match any prescribed endpoints. By contrast, diffusion models fix one endpoint to a simple prior (typically Gaussian) and learn only the reverse-time score to reach the data. With this perspective, SB is the more flexible

umbrella: with nontrivial potentials it bridges arbitrary endpoints; with  $\Psi \equiv 1$  it collapses to the diffusion-model case above. We additionally remark that in the standard linear diffusion model,  $p_T \approx p_{\text{prior}}$  holds only as  $T \rightarrow \infty$ , so the match to the prior is merely approximate.

### 7.4.2 Diffusion Models as Schrödinger Half-Bridges

In this section, we explain why diffusion models are not full Schrödinger bridges, but can instead be understood through the relaxed notion of *Schrödinger half-bridges*. A half-bridge enforces only one endpoint constraint (either  $p_{\text{prior}}$  or  $p_{\text{data}}$ ) rather than both, making it a one-sided variant of the full bridge. Before formalizing this connection, we introduce the definition of Schrödinger half-bridges, building on the general formulation in Equation (7.2.7) with arbitrary  $p_{\text{src}}$  and  $p_{\text{tgt}}$ . We will then return to diffusion models and show how the half-bridge viewpoint naturally applies when the endpoints are given by  $p_{\text{prior}}$  and  $p_{\text{data}}$ .

**Schrödinger Half-Bridges** The SB problem asks for a stochastic process whose law is closest (in KL divergence) to a simple reference process, while *matching two endpoint distributions*  $p_{\text{src}}$  and  $p_{\text{tgt}}$ . Solving the full bridge requires enforcing both boundary conditions, which is often computationally difficult. A useful relaxation is the *half-bridge* problem: instead of matching both endpoints, we match only one of them.

Formally, let  $R$  be the reference path distribution. The *forward half-bridge* seeks a path distribution  $P$  minimizing

$$\min_{P: P_0 = p_{\text{src}}} \mathcal{D}_{\text{KL}}(P \| R),$$

subject to the single constraint  $P_0 = p_{\text{src}}$ . Similarly, the *backward half-bridge* constrains only the terminal distribution,

$$\min_{P: P_T = p_{\text{tgt}}} \mathcal{D}_{\text{KL}}(P \| R).$$

In words, the forward half-bridge asks: among all processes starting from the desired initial distribution, which one looks most like the reference? The backward half-bridge asks the same question for processes ending at the desired terminal distribution. By combining these two relaxations iteratively, one can approximate the full SB.

**Diffusion Models Miss Exact Endpoint Matching.** A key difference between diffusion models and the SB framework lies in the treatment of the terminal

distribution  $p_T$ . In standard diffusion models, the forward SDE is typically linear in  $\mathbf{x}_t$  (see Equation (4.3.2)) and designed so that  $p_T$  approximates the prior only as  $T \rightarrow \infty$ :

$$p_T \approx p_{\text{prior}}.$$

At finite time, however,  $p_T$  is a Gaussian whose parameters depend on  $p_{\text{data}}$  (see Section C.1.5). As a result, it generally does not match the desired prior without careful tuning.

In contrast, the SB framework enforces exact marginal matching at a finite time  $T$  by introducing an additional control drift of the form  $g^2(t)\nabla_{\mathbf{x}} \log \Psi(\mathbf{x}, t)$ . This ensures that the terminal distribution precisely satisfies  $p_T = p_{\text{prior}}$ , regardless of the initial data distribution  $p_0 = p_{\text{data}}$ . In summary:

- **Diffusion Models:**  $p_T \approx p_{\text{prior}}$ , asymptotically as  $T \rightarrow \infty$ ,
- **Schrödinger Bridge:**  $p_T = p_{\text{prior}}$  exactly at finite  $T$ , enabled by solving for the control potentials  $\Psi$  and  $\hat{\Psi}$ .

**Diffusion Schrödinger Bridge.** Standard diffusion models do not enforce  $P_T = p_{\text{prior}}$ , and thus only solve a Schrödinger *half-bridge* from  $p_{\text{data}}$  to  $p_{\text{prior}}$ .

To address this, the Diffusion Schrödinger Bridge (DSB) (De Bortoli *et al.*, 2021) alternates between matching both endpoint marginals by following the idea of the Iterative Proportional Fitting (IPF) algorithm, an alternating projection method. This extends diffusion models to solve the full SB problem as follows<sup>7</sup>:

- **Step 0: Reference Process.** Initialize with  $P^{(0)} := R_{\text{fwd}}$ , the reference forward SDE:

$$d\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)dt + g(t)d\mathbf{w}_t, \quad \mathbf{x}_0 \sim p_{\text{data}}.$$

This ensures  $P_0^{(0)} = p_{\text{data}}$ , but typically  $P_T^{(0)} \neq p_{\text{prior}}$ .

- **Step 1: Backward Pass.** Compute the process  $P^{(1)}$  that matches  $p_{\text{prior}}$  at time  $T$  while staying close to  $P^{(0)}$ :

$$P^{(1)} = \underset{P: P_T = p_{\text{prior}}}{\arg \min} \mathcal{D}_{\text{KL}}(P \| P^{(0)}).$$

This is achieved via approximating the oracle score function with a neural network  $\mathbf{s}_{\phi^\times}$ , which results in the reverse-time SDE:

$$d\mathbf{x}_t = \left[ \mathbf{f}(\mathbf{x}_t, t) - g^2(t)\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t) \right] dt + g(t)d\bar{\mathbf{w}}_t,$$

---

<sup>7</sup>Although this description uses  $p_{\text{data}}$  and  $p_{\text{prior}}$ , the DSB framework applies to any pair of endpoint distributions.

simulated backward from  $\mathbf{x}_T \sim p_{\text{prior}}$ .

- **Iteration.** The process  $P^{(1)}$  satisfies  $P_T^{(1)} = p_{\text{prior}}$ , but its initial marginal  $P_0^{(1)}$  typically deviates from  $p_{\text{data}}$ . IPF addresses this by learning a forward SDE to adjust  $P_0^{(1)}$  back to  $p_{\text{data}}$ , followed by another backward pass to enforce  $p_{\text{prior}}$ . This alternation continues, refining the process until convergence to the optimal bridge  $P^*$ , which satisfies both  $P_0^* = p_{\text{data}}$  and  $P_T^* = p_{\text{prior}}$ . De Bortoli *et al.* (2021) prove convergence under mild conditions.

## 7.5 Is Diffusion Model's ODE an Optimal Map to OT Problem?

In this section, we focus on quadratic-cost optimal transport problem.

### 7.5.1 PF-ODE Flow Is Generally Not Optimal Transport

This section presents the result of Lavenant and Santambrogio (2022), which demonstrates that the solution map of the PF-ODE does not generally yield the optimal transport map under a quadratic cost.

**Setup.** We consider a VP SDE, specifically the Ornstein–Uhlenbeck process, which evolves a smooth initial density  $p_0$  toward the standard Gaussian  $\mathcal{N}(\mathbf{0}, \mathbf{I})$ :

$$d\mathbf{x}(t) = -\mathbf{x}(t) dt + \sqrt{2} d\mathbf{w}(t), \quad \mathbf{x}(0) \sim p_0.$$

The associated PF-ODE is given by

$$\frac{d\mathbf{S}_t(\mathbf{x})}{dt} = -\mathbf{S}_t(\mathbf{x}) - \nabla \log p_t(\mathbf{S}_t(\mathbf{x})), \quad \mathbf{S}_0(\mathbf{x}) = \mathbf{x}.$$

Here,  $\mathbf{S}_t$  denotes the flow map pushing forward  $p_0$  to the marginal  $p_t$ :

$$(\mathbf{S}_t) \# p_0 = p_t, \quad \text{that is, } p_t(\mathbf{y}) = \int_{\mathbb{R}^D} \delta(\mathbf{y} - \mathbf{S}_t(\mathbf{x})) p_0(\mathbf{x}) d\mathbf{x}.$$

These densities  $p_t$  evolves via the Fokker-Planck equation:

$$\frac{\partial p_t}{\partial t} = \nabla \cdot (\mathbf{x} p_t) + \Delta p_t.$$

This is equivalent to a continuity equation with velocity field:

$$\mathbf{v}_t(\mathbf{x}) = -\mathbf{x} - \nabla \log p_t(\mathbf{x}),$$

whose flow is given by  $\mathbf{S}_t(\mathbf{x})$ . In other words, the PF-ODE can be written as:

$$\frac{d\mathbf{S}_t(\mathbf{x})}{dt} = \mathbf{v}_t(\mathbf{S}_t(\mathbf{x})).$$

As  $t \rightarrow \infty$ , the map transports the initial distribution to the prior:

$$\mathbf{S}_\infty \# p_0 = \mathcal{N}(\mathbf{0}, \mathbf{I}) =: p_{\text{prior}}.$$

**Objective of Lavenant and Santambrogio (2022)'s Argument.** Lavenant and Santambrogio (2022) do not directly assess whether the terminal map  $\mathbf{S}_\infty$  from  $p_0$  to the Gaussian is optimal. Instead, they construct a specific

initial distribution  $p_0$  and examine the entire PF-ODE trajectory. Their key observation is that optimality may fail at some point along the flow.

They consider the intermediate marginal  $p_t = \mathbf{S}_t \# p_0$  and define the residual transport map from  $p_{t_0}$  to the Gaussian as

$$\mathbf{T}_{t \rightarrow \infty} := \mathbf{S}_\infty \circ \mathbf{S}_t^{-1}, \quad \text{for all } t \geq 0.$$

The core of their argument shows that, for a carefully chosen  $p_0$ , there exists a time  $t_0 \geq 0$  such that  $\mathbf{T}_{t_0 \rightarrow \infty}$  is not the quadratic-cost optimal transport map from the new starting distribution  $p_{t_0}$  to  $\mathcal{N}(\mathbf{0}, \mathbf{I})$ .

This result demonstrates that PF-ODE flows do not, in general, yield optimal transport maps, and that the property of optimality can break down for certain initial distributions.

**Some Tools.** The argument of Lavenant and Santambrogio (2022) crucially relies on the following result, known as *Brenier's theorem*:

**Theorem 7.1** (Informal Brenier's Theorem). Let  $\nu_1, \nu_2$  be two probability distributions on  $\mathbb{R}^D$  with smooth densities. A smooth map  $\mathbf{T} : \mathbb{R}^D \rightarrow \mathbb{R}^D$  is the optimal transport from  $\nu_1$  to  $\nu_2$  (under quadratic cost) if and only if  $\mathbf{T} = \nabla u$  for some convex function  $u$ . In this case,  $D\mathbf{T}$  is symmetric and positive semi-definite, and  $u$  satisfies the Monge–Ampère equation:

$$\det D^2 u(\mathbf{x}) = \frac{\nu_1(\mathbf{x})}{\nu_2(\nabla u(\mathbf{x}))}.$$

The proof also implicitly uses the following fact, which we will not repeat each time: *a map is the optimal transport between two distributions if and only if its inverse is the optimal transport in the reverse direction.*

**Proof Sketch: PF-ODE Is Not an OT Map in General.** Lavenant and Santambrogio (2022) employ a proof by contradiction: they assume that for every  $t \geq 0$ , the map

$$\mathbf{T}_t = \mathbf{S}_t \circ \mathbf{S}_\infty^{-1}$$

is the quadratic-cost optimal transport map from  $\mathcal{N}(\mathbf{0}, \mathbf{I})$  to  $p_t$ .

**Step 1: Brenier's Theorem.** By Brenier's Theorem, the Jacobian of any optimal transport map from Gaussian must be symmetric and positive semi-definite. Thus,

$$D\mathbf{T}_t(\mathbf{x}) = D\mathbf{S}_t(\mathbf{S}_\infty^{-1}(\mathbf{x}))D(\mathbf{S}_\infty^{-1})(\mathbf{x})$$

must be symmetric for all  $t$  and  $\mathbf{x}$ . Here,  $D\mathbf{T}_t(\mathbf{x})$  denotes the total differentiation with respect to  $\mathbf{x}$ .

**Step 2: Time-Differentiating the Symmetry Condition.** Differentiating in time:

$$\frac{\partial}{\partial t} D\mathbf{T}_t(\mathbf{x}) = \left( \frac{\partial}{\partial t} D\mathbf{S}_t \right) (\mathbf{S}_\infty^{-1}(\mathbf{x})) D(\mathbf{S}_\infty^{-1})(\mathbf{x}).$$

Given that the symmetry holds for all  $t$ , it follows that this derivative remains symmetric.

Using the flow ODE (differentiating in  $\mathbf{x}$ ), we obtain:

$$\frac{\partial(D\mathbf{S}_t)}{\partial t} = D\mathbf{v}_t(\mathbf{S}_t) \cdot D\mathbf{S}_t = (-\mathbf{I} - D^2 \log p_t(\mathbf{S}_t)) \cdot D\mathbf{S}_t.$$

Combining the above, we see that

$$(-\mathbf{I} - D^2 \log p_t(\mathbf{S}_t)) \cdot D\mathbf{S}_t \cdot D(\mathbf{S}_\infty^{-1})$$

is symmetric for all  $t \geq 0$ .

At  $t = 0$ , we have  $\mathbf{S}_0 = \mathbf{I}$  and  $D\mathbf{S}_0 = \mathbf{I}$ , yielding:

$$(-\mathbf{I} - D^2 \log p_0(\mathbf{S}_\infty^{-1}(\mathbf{x}))) \cdot D(\mathbf{S}_\infty^{-1})(\mathbf{x}) \text{ is symmetric.}$$

**Step 3: The Commutation Condition.** Since  $\mathbf{T}_0 = \mathbf{S}_\infty^{-1}$  is assumed to be optimal, its Jacobian  $D\mathbf{T}_0 = D(\mathbf{S}_\infty^{-1})$  is symmetric. Moreover, the Hessian  $D^2 \log p_0$  is symmetric. Recall that two symmetric matrices multiply to a symmetric matrix if and only if they commute. Hence, for all  $\mathbf{x} \in \mathbb{R}^D$ ,

$$D^2 \log p_0(\mathbf{S}_\infty^{-1}(\mathbf{x})) \text{ must commute with } D(\mathbf{S}_\infty^{-1})(\mathbf{x}).$$

Setting  $\mathbf{y} = \mathbf{S}_\infty^{-1}(\mathbf{x})$  gives the equivalent condition: for all  $\mathbf{y} \in \mathbb{R}^D$ ,

$$D^2 \log p_0(\mathbf{y}) \text{ must commute with } D\mathbf{S}_\infty(\mathbf{y}).$$

Now, we transform this condition into a more computable form. Since  $\mathbf{S}_\infty$  is optimal between  $p_0$  and  $\mathcal{N}(\mathbf{0}, \mathbf{I})$ , Brenier's theorem guarantees that  $\mathbf{S}_\infty = \nabla u$  for some convex function  $u$ . From the Monge–Ampère equation, it follows that:

$$\log p_0(\mathbf{y}) = \log \det(D^2 u(\mathbf{y})) - \frac{1}{2} \|\nabla u(\mathbf{y})\|^2 + \text{Constant.}$$

The condition becomes (with  $D\mathbf{S}_\infty = D^2 u$ ):

$$D^2 \left( \log \det D^2 u - \frac{1}{2} \|\nabla u\|^2 \right) \quad \text{must commute with } D^2 u. \quad (7.5.1)$$

This yields a necessary condition for  $\mathbf{T}_t$  to be optimal.

**Step 4: Constructing the Counterexample.** Let us show how to leverage this necessary condition to derive a contradiction.

Assume we can construct a convex function  $u$  such that

$$D^2 \left( \log \det D^2 u(\mathbf{x}) - \frac{1}{2} |\nabla u(\mathbf{x})|^2 \right)$$

does not commute with  $D^2 u(\mathbf{x})$  for some  $\mathbf{x} \in \mathbb{R}^D$ . Defining  $p_0 = (\nabla u)^{-1} \# \mathcal{N}(\mathbf{0}, \mathbf{I})$ , Brenier's theorem implies that  $\nabla u$  is the optimal transport from  $p_0$  to  $\mathcal{N}(\mathbf{0}, \mathbf{I})$ . However, the condition in Equation (7.5.1) fails, leading to a contradiction. Thus, our goal is to construct such a function. Consider

$$u(\mathbf{x}) = \frac{1}{2} \|\mathbf{x}\|^2 + \varepsilon \phi(\mathbf{x}), \quad \text{for a small } \varepsilon.$$

Then  $D^2 u(\mathbf{0}) = \mathbf{I} + \varepsilon D^2 \phi(\mathbf{0})$ , and the commutation condition at  $\mathbf{x} = \mathbf{0}$  requires  $D^2 \phi(\mathbf{0})$  to commute with  $D^2(\Delta \phi)(\mathbf{0})$ .

For example, in  $\mathbb{R}^2$ , choosing

$$\phi(x_1, x_2) = x_1 x_2 + x_1^4$$

provides a counterexample where the Hessian  $D^2 \log p_0$  and the Jacobian  $D^2 u$  do not commute.

This contradiction shows that  $\mathbf{T}_t$  cannot be optimal for all  $t \geq 0$ . Therefore, there exists some  $t_0 \geq 0$  such that the map  $\mathbf{T}_{t_0 \rightarrow \infty}$  is not optimal.

### 7.5.2 Can Canonical Linear Flow and Reflow Leads to an OT Map?

We have seen that the PF-ODE (especially in VP type forward kernel) is generally not an OT map. One natural question now is:

#### Question 7.5.1

*Does the linear interpolation flow  $(1 - t)\mathbf{x}_0 + t\mathbf{x}_1$  with  $\mathbf{x}_0 \sim p_{\text{src}}$  and  $\mathbf{x}_1 \sim p_{\text{tgt}}$ , when applied to the independent coupling  $\pi(\mathbf{x}_0, \mathbf{x}_1) = p_{\text{src}}(\mathbf{x}_0)p_{\text{tgt}}(\mathbf{x}_1)$ , recover the OT map?*

The answer to the question is no.

Nevertheless, combining a linear path with a given coupling offers a practical upper bound on the true OT cost. Among all possible paths, linear interpolation provides the tightest such upper bound, as we will see in the following discussion.

**Canonical Linear Flow and Optimal Transport.** Focusing on optimal transport with quadratic cost, we consider the equivalent form of Equation (7.2.1), the Benamou–Brenier formulation in Equation (7.2.3):

$$\mathcal{K}(p_{\text{src}}, p_{\text{tgt}}) := \min_{\substack{(p_t, \mathbf{v}_t) \text{ s.t. } \partial_t p_t + \nabla \cdot (p_t \mathbf{v}_t) = 0, \\ p_0 = p_{\text{src}}, p_1 = p_{\text{tgt}}}} \int_0^1 \int_{\mathbb{R}^D} \|\mathbf{v}_t(\mathbf{x})\|^2 p_t(\mathbf{x}) d\mathbf{x} dt.$$

However, solving this minimization problem directly is typically intractable, as it requires solving a highly nonlinear partial differential equation, namely the Monge–Ampère equation.

While solving the Benamou–Brenier formulation is generally intractable, Liu (2022) and Lipman *et al.* (2024) reveal that its kinetic energy admits a practical upper bound. This is achieved by restricting the search to a simpler family of *conditional flows*, where each path is defined by its fixed endpoints  $(\mathbf{x}_0, \mathbf{x}_1)$  drawn from a coupling  $\pi_{0,1}$  of the source and target distributions. Within this *conditional flow* family, the canonical linear interpolation emerges as the optimal choice, as formalized below.

**Proposition 7.5.1: An Upper Bound on OT Kinetic Energy via Conditional Flows**

Let  $\pi_{0,1}$  be any coupling between  $p_{\text{src}}$  and  $p_{\text{tgt}}$ .

- (1) The kinetic energy is bounded above by the expected path energy of any conditional flow  $\Psi_t(\mathbf{x}_0, \mathbf{x}_1)$  that connects the endpoints:

$$\mathcal{K}(p_{\text{src}}, p_{\text{tgt}}) \leq \mathbb{E}_{(\mathbf{x}_0, \mathbf{x}_1) \sim \pi_{0,1}} \left[ \int_0^1 \|\Psi'_t(\mathbf{x}_0, \mathbf{x}_1)\|^2 dt \right].$$

- (2) The unique conditional flow  $\Psi_t^*$  that minimizes the upper bound on the right-hand side is the linear interpolation path:

$$\Psi_t^*(\mathbf{x}_0, \mathbf{x}_1) = (1-t)\mathbf{x}_0 + t\mathbf{x}_1.$$

Substituting this optimal path yields the tightest version of the bound:

$$\mathcal{K}(p_{\text{src}}, p_{\text{tgt}}) \leq \mathbb{E}_{(\mathbf{x}_0, \mathbf{x}_1) \sim \pi_{0,1}} \|\mathbf{x}_1 - \mathbf{x}_0\|^2.$$

**Proof for Proposition.**

The proof relies on a straightforward application of Jensen's inequality and the tower property of conditional expectations, before solving the simplified variational problem with the Euler-Lagrange equation; we refer to (Lipman *et al.*, 2024)'s Section 4.7 for the complete argument. ■

In other words, the linear interpolation  $\Psi_t^*$  (i.e., the forward kernel used by Flow Matching and Rectified Flow) minimizes an upper bound on the true kinetic energy for any chosen coupling  $\pi_{0,1}$ .

We emphasize that optimality within this class of conditional flows does not guarantee global optimality on the marginal distributions.

**Reflow and Optimal Transport.** The most naive transport plan between two distributions is to connect their samples with straight lines using a simple independent coupling. However, this approach is demonstrably not optimal, as the failure lies not in the straight-line paths themselves, but in the inefficient initial pairing of points.

The Reflow procedure may offer a constructive response. It is an iterative algorithm designed specifically to correct this pairing, and crucially, each step is guaranteed to be cost-non-increasing (Liu, Gong, *et al.*, 2022). This property suggests Reflow systematically pushes the transport plan towards a

more optimal configuration, which naturally motivates the central question of its convergence.

### Question 7.5.2

*What happens if we apply the `Rectify` operator iteratively? Can the resulting sequence of transport plans converge to the optimal one, or does the fixed point of the Reflow process yield the OT map?*

The short answer is no in general. Below, we explain what may go wrong. To recall, the Reflow procedure iteratively refines the coupling between  $p_{\text{src}}$  and  $p_{\text{tgt}}$  via the update:

$$\pi^{(k+1)} = \text{Rectify}(\pi^{(k)}),$$

initialized with the product coupling  $\pi^{(0)} := p_{\text{src}}(\mathbf{x}_0)p_{\text{tgt}}(\mathbf{x}_1)$ . More precisely, `Rectify` output the updated coupling  $\pi^{(k+1)}$  via the following: At each iteration  $k = 0, 1, 2, \dots$ , a velocity field  $\mathbf{v}_t^{(k)}$  is learned via:

$$\mathbf{v}_t^{(k)} \in \arg \min_{\mathbf{u}_t} \mathcal{L}(\mathbf{u}_t | \pi^{(k)}),$$

where  $\mathcal{L}(\mathbf{u}_t | \pi^{(k)})$  is the loss (e.g., RF or FM loss) defined in Equation (5.4.1). Here, for notational simplicity, we adopt a non-parametric formulation for the velocity field, rather than a parameterized form  $\phi$  employed in other contexts. The updated coupling is then given by:

$$\pi^{(k+1)}(\mathbf{x}_0, \mathbf{x}_1) := p_{\text{src}}(\mathbf{x}_0) \delta(\mathbf{x}_1 - \Psi_1^{(k)}(\mathbf{x}_0)),$$

where  $\Psi_1^{(k)}$  denotes the solution map at time  $t = 1$  obtained by integrating  $\mathbf{v}_t^{(k)}$  from initial condition  $\mathbf{x}_0$ .

It has been observed in (Liu, Gong, *et al.*, 2022) that for a coupling  $\pi$  between  $p_{\text{src}}$  and  $p_{\text{tgt}}$ , the existence of a velocity field  $\mathbf{v}_t$  that minimizes the Reflow loss, that is, satisfies  $\mathcal{L}(\mathbf{v}_t | \pi) = 0$ , does not necessarily imply that the transport is optimal.

Motivated by the Benamou–Brenier framework, where the optimal transport velocity is known to be the gradient of a potential function, Liu (2022) proposed an additional constraint: the velocity field  $\mathbf{v}_t$  should be a potential field. Accordingly, the objective in Equation (5.4.1) is modified to restrict  $\mathbf{v}_t$  to the space of gradient vector fields, also known as potential flows:

$$\mathbf{w}_t^{(k)} \in \arg \min_{\substack{\mathbf{u}_t: \mathbf{u}_t = \nabla \varphi \\ \text{for some } \varphi: \mathbb{R}^D \rightarrow \mathbb{R}}} \mathcal{L}(\mathbf{u}_t | \pi^{(k)}), \quad (7.5.2)$$

with the rest of the procedure remaining the same as in `Rectify`. We denote this associated operator as  $\text{Rectify}_\perp$ , emphasizing the projection onto irrotational vector fields.

Let  $\pi$  be a coupling between  $p_{\text{src}}$  and  $p_{\text{tgt}}$ . Liu, Gong, *et al.* (2022) conjecture the following equivalence characterizing optimality:

- (i)  $\pi$  is an optimal transport coupling.
- (ii)  $\pi$  is a fixed point of the potential rectification operator:

$$\pi = \text{Rectify}_\perp(\pi).$$

- (iii) There exists a gradient velocity field  $\mathbf{v}_t = \nabla \varphi_t$  such that the rectify loss vanishes:

$$\mathcal{L}(\mathbf{v}_t | \pi) = 0.$$

However, Hertrich *et al.* (2025) exhibit two types of counterexamples:

1. When the intermediate distributions  $p_t$  have disconnected support, one can find fixed points of  $\text{Rectify}_\perp$  with zero Reflow loss and gradient velocity fields that nonetheless fail to produce the optimal coupling.
2. Even when both endpoint distributions are Gaussian, there exist couplings whose loss is arbitrarily small but whose deviation from the optimal coupling is arbitrarily large.

Therefore, while rectified flows may yield strong generative models, their reliability as optimal transport solvers remains limited. This highlights an important gap between generative modeling and principled optimal transport theory, inviting further research at their intersection.

Finally, we note that transport cost does not always correlate with downstream performance; as such, computing the exact optimal transport map may not necessarily lead to better practical outcomes. Nonetheless, variants of optimal transport remain fundamental to many problems in science and engineering. Diffusion models offer a powerful framework for exploring these challenges.

## **Part C**

# **Sampling of Diffusion Models**

## Chapter 4

**Generation with Diffusion Model  $\mathbf{v}^*(\mathbf{x}, t)$** 

$\iff$  Solve the ODE backward from  $T$  to 0 with  $\mathbf{x}(T) \sim p_{\text{prior}}$  (more generally, from  $s$  to  $t$  with  $s > t$ ):

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}^*(\mathbf{x}(t), t)$$

$$\iff \mathbf{x}(0) = \mathbf{x}(T) + \int_0^T \mathbf{v}^*(\mathbf{x}(t), t) dt$$

**Steering Generation**

$$\mathbf{x}(0) = \mathbf{x}(T) + \int_0^T [\mathbf{v}^*(\mathbf{x}(t), t) + \text{Guidance}] dt$$

Chapter 8

**Fast Generation with Numerical Solvers**

$$\mathbf{x}(0) = \mathbf{x}(T) + \int_0^T \mathbf{v}^*(\mathbf{x}(t), t) dt$$

Estimating the Integration

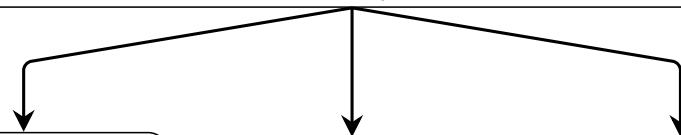
Chapter 9

**Learning a Fast Diffusion-Based Generator**

$$\mathbf{x}(0) = \mathbf{x}(T) + \int_0^T \mathbf{v}^*(\mathbf{x}(t), t) dt$$

Learning the Integration

Chapter 10 and Chapter 11



# 8

---

## Guidance and Controllable Generation

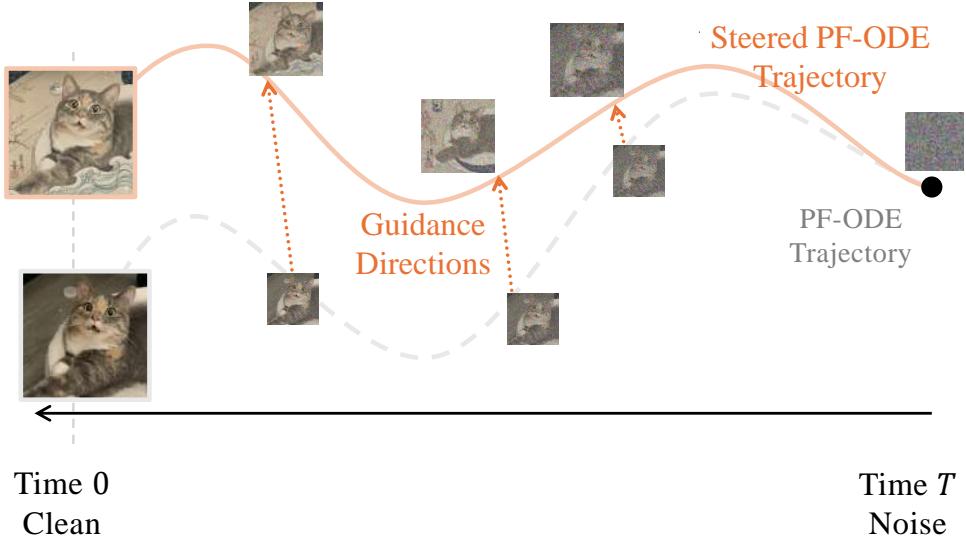
---

Diffusion models are powerful generative frameworks. In the unconditional setting, the goal is to learn  $p_{\text{data}}(\mathbf{x})$  and generate samples without external input.

Many applications, however, require *conditional generation*, where outputs satisfy user-specified criteria. This can be achieved by steering an unconditional model or directly learning the conditional distribution  $p_0(\mathbf{x}|\mathbf{c})$ , with condition  $\mathbf{c}$  (e.g., label, text description, or sketch) guiding the process.

This chapter builds on a principled view of the conditional score, which decomposes into an *unconditional direction* and a *guidance direction* that nudges samples toward the condition while preserving realism. We explain why guidance is essential, show how the conditional score serves as a unifying interface for control, and survey ways to approximate the guidance term. We then distinguish *control* (meeting the condition) from *alignment* (meeting human preference under the condition), and describe how preferences can be incorporated into the same framework. Finally, we discuss direct optimization of preference without additional reward models (i.e., a learned scorer that assigns higher values to outputs better aligned with human preference).

## 8.1 Prologue



**Figure 8.1: Illustration of steered diffusion sampling.** Reverse-time PF-ODE sampling begins from pure noise at the right ( $t = T$ ) and gradually evolves toward a clean sample at the left ( $t = 0$ ). During this process, guidance directions  $\nabla_{\mathbf{x}_t} \log \tilde{p}_t(\mathbf{c}|\mathbf{x}_t)$ , weighted by  $w_t$ , modify the velocity field according to  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + w_t \nabla_{\mathbf{x}_t} \log \tilde{p}_t(\mathbf{c}|\mathbf{x}_t)$ . These additional directions steer the trajectory toward the desired attribute (Japanese painting style) while the sample is progressively refined from coarse to fine detail.

The generation process of diffusion models proceeds in a coarse-to-fine manner, providing a flexible framework for controllable generation. At each step, a small amount of noise is removed and the sample becomes clearer, gradually revealing more structure and detail. This property enables control over the generation process: by adding a guidance term to the learned, time-dependent velocity field, we can steer the generative trajectory to reflect user intent.

A principled foundation for guidance-based sampling in diffusion models is the Bayesian decomposition of the conditional score. For each noise level  $t$ ,

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c}) = \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)}_{\text{unconditional direction}} + \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t)}_{\text{guidance direction}}. \quad (8.1.1)$$

This identity shows that conditional sampling can be implemented by adding a guidance term  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t)$  on top of the unconditional score. A wide

range of controllable generation methods (e.g., classifier guidance (Dhariwal and Nichol, 2021), general training-free guidance (Ye *et al.*, 2024)) can be interpreted as different approximations of this guidance term, since  $p_t(\mathbf{c}|\mathbf{x}_t)$  is generally intractable due to marginalization over  $\mathbf{x}_0$ .

Once such an approximation is available, sampling simply replaces the unconditional score with its conditional counterpart. Using Equation (8.1.1), the PF-ODE becomes

$$\begin{aligned}\frac{d\mathbf{x}(t)}{dt} &= f(t)\mathbf{x}(t) - \frac{1}{2}g^2(t)\underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}(t)|\mathbf{c})}_{\text{conditional score}} \\ &= f(t)\mathbf{x}(t) - \frac{1}{2}g^2(t)\left[\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}(t)) + \nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}(t))\right].\end{aligned}\quad (8.1.2)$$

We highlight that steering these time-dependent vector fields fundamentally relies on their linearity, so the discussion below, formulated in score prediction, naturally extends to  $\mathbf{x}$ -,  $\epsilon$ -, and  $\mathbf{v}$ -prediction through their linear relationships as in Equation (6.3.1).

### Instantiations of the Guidance Direction.

- Classifier Guidance (CG).** In Section 8.2, classifier guidance (CG) trains a time-conditional classifier  $p_\psi(\mathbf{c}|\mathbf{x}_t, t)$  on noised data  $\mathbf{x}_t$  (obtained by corrupting clean labeled samples at level  $t$ ). At sampling time, its input gradient provides the guidance term:

$$\nabla_{\mathbf{x}_t} \log p_\psi(\mathbf{c}|\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t),$$

which is then added to the unconditional score (Dhariwal and Nichol, 2021).

- Classifier-Free Guidance (CFG).** In Section 8.3, CFG directly trains a single conditional model

$$\mathbf{s}_\phi(\mathbf{x}_t, t, \mathbf{c}) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c}),$$

where the unconditional model is learned jointly by randomly replacing the condition with a special null token for a fraction of the training steps.

- Training-Free (Surrogate) Guidance.** The conditional  $p_t(\mathbf{c}|\mathbf{x}_t)$  is generally intractable because it requires marginalizing over the clean latent  $\mathbf{x}_0$ :

$$p_t(\mathbf{c}|\mathbf{x}_t) = \int p(\mathbf{c}|\mathbf{x}_0)p(\mathbf{x}_0|\mathbf{x}_t)d\mathbf{x}_0,$$

and, in typical applications, at least one of these factors is unknown, making the integral intractable.

In Section 8.4.1, training-free (loss-based) guidance avoids evaluating the conditional likelihood  $p_t(\mathbf{c}|\mathbf{x}_t)$  directly. Instead, it introduces an off-the-shelf loss  $\ell(\mathbf{x}_t, \mathbf{c}; t)$  and defines a surrogate conditional distribution  $\tilde{p}_t(\mathbf{c}|\mathbf{x}_t)$  as,

$$\tilde{p}_t(\mathbf{c}|\mathbf{x}_t) \propto \exp(-\tau \ell(\mathbf{x}_t, \mathbf{c}; t)), \quad \tau > 0,$$

which acts as a pseudo-likelihood. This formulation sidesteps the intractability of computing the true conditional likelihood while still enabling guidance through gradients of the chosen loss. Its conditional score is computed solely by the gradient of the loss with  $\tau$ :

$$\nabla_{\mathbf{x}_t} \log \tilde{p}_t(\mathbf{c}|\mathbf{x}_t) = -\tau \nabla_{\mathbf{x}_t} \ell(\mathbf{x}_t, \mathbf{c}; t).$$

This term is added to the unconditional score with a guidance weight  $w_t$ :

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + w_t [-\tau \nabla_{\mathbf{x}_t} \ell(\mathbf{x}_t, \mathbf{c}; t)].$$

which is exactly the score of the *tilted* density  $\tilde{p}_t^{\text{tilt}}(\mathbf{x}_t|\mathbf{c})$  defined as:

$$\tilde{p}_t^{\text{tilt}}(\mathbf{x}_t|\mathbf{c}) \propto p_t(\mathbf{x}_t) \tilde{p}_t(\mathbf{c}|\mathbf{x}_t)^{w_t} \propto p_t(\mathbf{x}_t) \exp(-w_t \tau \ell(\mathbf{x}_t, \mathbf{c}; t)).$$

In practice, we replace the conditional score of sampling in Equation (8.1.2) with this tilted score, and solving the resulting ODE to draw samples.

In view of this, classifier guidance is simply surrogate guidance with a learned classifier  $\tilde{p}_t(\mathbf{c}|\mathbf{x}_t) := p_{\psi^x}(\mathbf{c}|\mathbf{x}_t, t)$  via:

$$\ell(\mathbf{x}_t, \mathbf{c}; t) = -\log p_{\psi^x}(\mathbf{c}|\mathbf{x}_t, t), \quad \tau = 1.$$

The effect of guidance on the sampling trajectory is illustrated in Figure 8.1.

All of these techniques can likewise be applied on top of a conditional model, allowing extra control signals to be injected during generation.

### Remark.

Guided PF-ODE does *not* sample from the tilted family (in general). Even with exact scores and exact ODE integration, replacing the score by the *tilted* score does not make the time- $t$  marginals equal to  $\{\tilde{p}_t^{\text{tilt}}(\cdot|\mathbf{c})\}_{t \in [0,1]}$ , nor the terminal law equal to  $\tilde{p}_0^{\text{tilt}}(\cdot|\mathbf{c})$ .

Define

$$\mathbf{v}_t^{\text{orig}} = \mathbf{f} - \frac{1}{2} g^2(t) \nabla \log p_t, \quad \mathbf{h}_t(\mathbf{x}) = e^{-w_t \tau \ell(\mathbf{x}, \mathbf{c}; t)}, \quad \tilde{p}_t^{\text{tilt}} = \frac{p_t \mathbf{h}_t}{Z_t}.$$

The guided PF-ODE uses

$$\mathbf{v}_t^{\text{tilt}} = \mathbf{f} - \frac{1}{2}g^2(t)\nabla \log \tilde{p}_t^{\text{tilt}} = \mathbf{v}_t^{\text{orig}} - \frac{1}{2}g^2(t)\nabla \log \mathbf{h}_t.$$

If  $\tilde{p}_t^{\text{tilt}}$  were the true marginals, they would satisfy

$$\partial_t \tilde{p}_t^{\text{tilt}} + \nabla \cdot (\tilde{p}_t^{\text{tilt}} \mathbf{v}_t^{\text{tilt}}) = 0.$$

But a direct calculation gives the residual

$$\begin{aligned} & \partial_t \tilde{p}_t^{\text{tilt}} + \nabla \cdot (\tilde{p}_t^{\text{tilt}} \mathbf{v}_t^{\text{tilt}}) \\ &= \tilde{p}_t^{\text{tilt}} \left[ \partial_t \log \mathbf{h}_t + \mathbf{v}_t^{\text{orig}} \cdot \nabla \log \mathbf{h}_t - \frac{1}{2}g^2(t)(\Delta \log \mathbf{h}_t + \|\nabla \log \mathbf{h}_t\|^2) - \frac{Z'_t}{Z_t} \right]. \end{aligned}$$

Thus  $\tilde{p}_t^{\text{tilt}}$  coincides with the PF-ODE marginals if and only if this bracket vanishes for all  $\mathbf{x}$ , i.e.

$$\partial_t \log \mathbf{h}_t + \mathbf{v}_t^{\text{orig}} \cdot \nabla \log \mathbf{h}_t = \frac{1}{2}g^2(t)(\Delta \log \mathbf{h}_t + \|\nabla \log \mathbf{h}_t\|^2) + \frac{Z'_t}{Z_t}.$$

This condition holds trivially when  $\omega_t \equiv 0$  (unconditional generation), but almost never for  $\mathbf{h}_t(\mathbf{x}) = e^{-w_t \tau \ell(\mathbf{x}, \mathbf{c}; t)}$ , except in very special cases of  $w_t$  or  $\ell$ . Therefore, in general,  $\{\tilde{p}_t^{\text{tilt}}\}$  are *not* the PF-ODE marginals, and terminal samples are *not* distributed as  $\tilde{p}_0^{\text{tilt}}(\mathbf{x}_0 | \mathbf{c})$ .

### From Control to Better Alignment with Direct Preference Optimization.

Strong control can be on-condition but off-preference: a sample may satisfy the conditioning signal (e.g., the prompt) yet deviate from what humans actually prefer. We formalize this by *tilting* the conditional target by a preference rating<sup>1</sup>:

$$\tilde{p}_0^{\text{tilt}}(\mathbf{x}_0 | \mathbf{c}) \propto p_0(\mathbf{x}_0 | \mathbf{c}) \exp(\beta r(\mathbf{x}_0, \mathbf{c})),$$

where  $r(\mathbf{x}_0, \mathbf{c})$  is a scalar alignment rating (reward) for a clean sample  $\mathbf{x}_0$  and condition  $\mathbf{c}$  (larger  $r$  indicates better alignment). In practice,  $r$  may be (i) the logit or log-probability of an external reward/classifier, (ii) a similarity measure (e.g., CLIP/perceptual (Radford *et al.*, 2021)), or (iii) a learned preference model.

Existing methods for achieving such steerability typically collect human labels of the relative quality of model generations and fine-tune the conditional diffusion model to align with these preferences, often through reinforcement learning from human feedback (RLHF). However, RLHF is a complex and often

---

<sup>1</sup>We remark that the training-free guidance can also be viewed in the same framework of finding a tilted distribution with a guidance of loss  $\ell(\mathbf{x}_t, \mathbf{c}, t)$

unstable procedure: it first fits a reward model to capture human preferences, and then fine-tunes the conditional diffusion model with reinforcement learning to maximize this estimated reward while constraining policy drift from the original model.

This naturally raises the question: *can we remove the reward model training stage altogether?* We address this with Diffusion-DPO (Wallace *et al.*, 2024), an adaptation of Direct Preference Optimization (Rafailov *et al.*, 2023) originally developed for large language models. As described in Section 8.5, Diffusion-DPO learns the preference tilt directly from pairwise choices, so the conditional diffusion model is fine-tuned to align to preferences without a separate reward model.

## 8.2 Classifier Guidance

### 8.2.1 Foundation of Classifier Guidance

Let  $\mathbf{c}$  denote a conditioning variable drawn from a distribution  $p(\mathbf{c})$ , such as a class label, caption, or other auxiliary information. Our goal is to draw samples from  $p_0(\mathbf{x}|\mathbf{c})$ . In diffusion-based conditional generation, we realize this goal by running the reverse-time dynamics whose time marginals are  $p_t(\cdot|\mathbf{c})$ . The drift of these dynamics depends on the conditional score

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c}), \quad t \in [0, T].$$

Hence a standard and effective route<sup>2</sup> is to estimate this quantity.

A fundamental insight, based on Bayes' rule, is that the conditional score can be decomposed as:

$$\begin{aligned} \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c}) &= \nabla_{\mathbf{x}_t} \log \left( \frac{p_t(\mathbf{x}_t)p_t(\mathbf{c}|\mathbf{x}_t)}{p(\mathbf{c})} \right) \\ &= \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t) - \nabla_{\mathbf{x}_t} \log p(\mathbf{c}) \\ &= \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)}_{\text{unconditional score}} + \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t)}_{\text{classifier gradient}}, \end{aligned} \quad (8.2.1)$$

where  $p_t(\mathbf{c}|\mathbf{x}_t)$  indicates a probability of  $\mathbf{c}$  conditioned on  $\mathbf{x}_t$  which predicts the condition  $\mathbf{c}$  from the noisy input  $\mathbf{x}_t$  at time  $t$ .

This decomposition<sup>3</sup> motivates the *Classifier Guidance* (CG) approach proposed by Dhariwal and Nichol (2021), which leverages a pre-trained time-dependent classifier  $p_t(\mathbf{c}|\mathbf{x}_t)$  to steer the generation process. Specifically, we define a one-parameter family of *guided densities* (tilted conditionals) with guidance scale  $\omega \geq 0$ :

$$p_t(\mathbf{x}_t|\mathbf{c}, \omega) \propto p_t(\mathbf{x}_t)p_t(\mathbf{c}|\mathbf{x}_t)^\omega, \quad (8.2.2)$$

which yields the score function:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c}, \omega) = \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + \omega \nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t). \quad (8.2.3)$$

Geometrically, this tilts the unconditional flow in the direction that increases the class likelihood. When  $\omega = 1$ ,  $p_t(\mathbf{x}_t|\mathbf{c}, \omega)$  coincides with the true conditional

---

<sup>2</sup>One could in principle obtain  $p_0(\mathbf{x}|\mathbf{c})$  from an unconditional generator via rejection or importance sampling if  $p(\mathbf{c}|\mathbf{x})$  were available and well calibrated. This is rarely practical for high-dimensional or rare conditions.

<sup>3</sup>In the last identity, since  $\nabla_{\mathbf{x}_t} \log p(\mathbf{c})$  does not depend on  $\mathbf{x}_t$ , it vanishes under differentiation.

$p_t(\mathbf{x}_t|\mathbf{c})$ ; for  $\omega \neq 1$ , it is a guided (tempered) reweighting rather than the literal conditional.

The scalar  $\omega \geq 0$  modulates the influence of the classifier:

- $\omega = 1$ : recovers the true conditional score  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c})$ .
- $\omega > 1$ : amplifies the classifier signal, typically increasing conditional fidelity (often at the expense of diversity).
- $0 \leq \omega < 1$ : down-weights the classifier signal, typically increasing sample diversity while weakening conditioning.

**Practical Approximation in CG.** In practice, CG is a training-free method (w.r.t. the diffusion model) for steering a pre-trained unconditional diffusion model,

$$\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t).$$

CG is applied only at sampling time, without modifying the diffusion model itself. To enable this, a time-dependent classifier  $p_\psi(\mathbf{c}|\mathbf{x}_t, t)$  is trained separately to predict the condition  $\mathbf{c}$  from noisy inputs  $\mathbf{x}_t$  at different noise levels  $t$ . The classifier is trained in a standard way by minimizing the cross-entropy loss:

$$\mathbb{E}_{t \sim \mathcal{U}[0, T], (\mathbf{x}, \mathbf{c}) \sim p_{\text{data}}, \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[ -\log p_\psi(\mathbf{c}|\mathbf{x}_t, t) \right], \quad (8.2.4)$$

where  $(\mathbf{x}, \mathbf{c}) \sim p_{\text{data}}$  denotes paired labeled data, and  $\mathbf{x}_t = \alpha_t \mathbf{x} + \sigma_t \boldsymbol{\epsilon}$  is the noisy input at time  $t$ . The classifier must be explicitly conditioned on  $t$  (e.g., via time embeddings), since it is expected to operate reliably across all noise levels.

After training, the classifier provides scores that serve as a surrogate for the true likelihood gradient:

$$\nabla_{\mathbf{x}_t} \log p_{\psi^\times}(\mathbf{c}|\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t).$$

### 8.2.2 Inference with CG

At inference time, the classifier gradient  $\nabla_{\mathbf{x}_t} \log p_{\psi^\times}(\mathbf{c}|\mathbf{x}_t, t)$  is added to the unconditional score function and scaled by a guidance weight  $\omega$ , yielding an approximation to the guided score  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c}, \omega)$  from Equation (8.2.3):

$$\begin{aligned} \mathbf{s}^{\text{CG}}(\mathbf{x}_t, t, \mathbf{c}; \omega) &:= \underbrace{\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t)}_{\text{uncond. direction}} + \omega \underbrace{\nabla_{\mathbf{x}_t} \log p_{\psi^\times}(\mathbf{c}|\mathbf{x}_t, t)}_{\text{guidance direction}} \\ &\approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c}, \omega). \end{aligned}$$

Accordingly, one simply replaces the unconditional score function  $\mathbf{s}_{\phi^X}(\mathbf{x}_t, t)$  in the reverse-time SDE or PF-ODE with the guided score  $\mathbf{s}^{CG}(\mathbf{x}_t, t, \mathbf{c}; \omega)$  for a specified  $\omega$  as in Equation (8.1.2), thereby steering the generative trajectory toward samples that align with the condition  $\mathbf{c}$ .

### 8.2.3 Advantages and Limitations

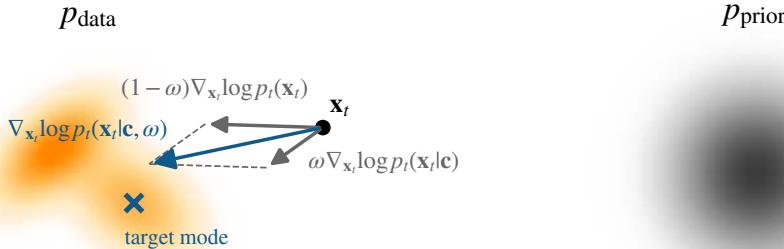
CG provides a simple and flexible mechanism for conditional generation, allowing for explicit control over the strength of conditioning via  $\omega$ . It can be used with any pre-trained unconditional diffusion model, requiring only an additional classifier for conditioning.

However, the approach has notable limitations:

- **Training Cost:** The classifier must be trained to operate across all noise levels, which is computationally expensive.
- **Robustness:** Classifiers must generalize well to severely corrupted inputs  $\mathbf{x}_t$ , especially for large  $t$ , which can be challenging.
- **Separate Training:** Since the classifier is trained independently of the diffusion model, it may not align perfectly with the learned data distribution.

## 8.3 Classifier-Free Guidance

### 8.3.1 Foundation of Classifier-Free Guidance



**Figure 8.2: Illustration of CFG.** The adjusted score  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{c}, \omega)$  is obtained as a linear interpolation between the unconditional score  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$  and the conditional score  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{c})$ , weighted by  $\omega$ . The resulting direction steers samples from the prior toward modes of the data distribution consistent with the target condition.

*Classifier-free guidance* (CFG) (Ho and Salimans, 2021) is a simplified approach to classifier-based guidance that eliminates the need for a separate classifier. The key idea is to modify the gradient of the score function in a way that allows for effective conditioning without explicit classifiers. Specifically, the gradient of the log-probability of the conditional distribution is adjusted as follows:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{c} | \mathbf{x}_t) = \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{c}) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t). \quad (8.3.1)$$

Substituting this expression into Equation (8.2.3) yields the following formulation for the conditioned score:

$$\begin{aligned} \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{c}, \omega) &= \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + \omega (\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{c}) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)) \\ &= \underbrace{\omega \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{c})}_{\text{conditional score}} + \underbrace{(1 - \omega) \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)}_{\text{unconditional score}}. \end{aligned} \quad (8.3.2)$$

The hyperparameter  $\omega$  again plays a critical role in controlling the influence of the conditioning information (we take  $\omega \geq 0$ ):

- At  $\omega = 0$ , the model behaves as an unconditional diffusion model, completely ignoring the conditioning.
- At  $\omega = 1$ , the model uses the conditional score without additional guidance.

- For  $\omega > 1$ , the model places more emphasis on the conditional score and less on the unconditional score, strengthening alignment with  $\mathbf{c}$  but typically reducing diversity.

### 8.3.2 Training and Sampling of CFG

#### Joint Training of Unconditional and Conditional Diffusion Models via CFG.

Unlike CG, CFG requires retraining a diffusion model that explicitly accounts for the conditioning variable  $\mathbf{c}$ . Training two separate models for the conditional and unconditional score functions, however, is often computationally prohibitive. To address this, CFG adopts a single model  $\mathbf{s}_\phi(\mathbf{x}_t, t; \mathbf{c})$  that learns both score functions within a single model by treating  $\mathbf{c}$  as an additional input. The training procedure is defined as follows:

- For unconditional training, a null token  $\emptyset$  is passed in place of the conditioning input, yielding  $\mathbf{s}_\phi(\mathbf{x}_t, t, \emptyset)$ .
- For conditional training, the true conditioning variable  $\mathbf{c}$  is provided as input, resulting in  $\mathbf{s}_\phi(\mathbf{x}_t, t, \mathbf{c})$ .

These two training regimes are unified by randomly replacing  $\mathbf{c}$  with the null input  $\emptyset$  with probability  $p_{\text{uncond}}$  (a user-defined hyperparameter typically set to 0.1). This joint training strategy enables the model to simultaneously learn both conditional and unconditional score functions. The full training algorithm is presented in Algorithm 4, alongside a comparison to standard unconditional training shown in Algorithm 3. We remark that during training, the CFG weight  $\omega$  is not utilized.

**Algorithm 3** Uncond. DM

---

```

1: Repeat
2:    $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$ 
3:    $t \sim \mathcal{U}[0, T]$ 
4:    $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
5:    $\mathbf{x}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$ 
6:   Take gradient step on:
       $\nabla_\phi \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \mathbf{s}\|^2$ 
7: until converged

```

---

**Algorithm 4** CFG for Cond. DM

---

```

Input:  $p_{\text{uncond}}$ : prob. of unconditional
        dropout
1: Repeat
2:    $(\mathbf{x}, \mathbf{c}) \sim p_{\text{data}}(\mathbf{x}, \mathbf{c})$ 
3:    $\mathbf{c} \leftarrow \emptyset$  with prob.  $p_{\text{uncond}}$ 
4:    $t \sim \mathcal{U}[0, T]$ 
5:    $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
6:    $\mathbf{x}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$ 
7:   Take gradient step on:
       $\nabla_\phi \|\mathbf{s}_\phi(\mathbf{x}_t, t, \mathbf{c}) - \mathbf{s}\|^2$ 
8: until converged

```

---

**Conditioned Sampling with CFG.** Once the model  $\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t, \mathbf{c})$  is trained using Algorithm 4, the CFG can be applied during sampling. The gradient of the log-probability is given by:

$$\begin{aligned}\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{c}, \omega) &= \omega \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{c}) + (1 - \omega) \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) \\ &\approx \omega \underbrace{\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t, \mathbf{c})}_{\text{conditional score}} + (1 - \omega) \underbrace{\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t, \emptyset)}_{\text{unconditional score}} \quad (8.3.3) \\ &=: \mathbf{s}_{\phi^\times}^{\text{CFG}}(\mathbf{x}_t, t, \mathbf{c}; \omega).\end{aligned}$$

During sampling, a fixed (or optionally time-dependent) classifier-free guidance weight  $\omega$  is applied. The unconditional score  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$  in the reverse-time SDE (Equation (4.1.6)) or PF-ODE (Equation (4.1.8)) is then replaced by the guided score  $\mathbf{s}_{\phi^\times}^{\text{CFG}}(\mathbf{x}_t, t, \mathbf{c}; \omega)$  as in Equation (8.1.2), which combines the conditional and unconditional scores in a weighted manner.

This formulation enables controllable generation by adjusting  $\omega$ , allowing samples to be guided toward the conditioning signal  $\mathbf{c}$  while retaining diversity. CFG thus offers an effective and computationally efficient way to achieve precise conditional generation, as it requires training only a single diffusion model.

## 8.4 (Optional) Training-Free Guidance

In this section, we present the high-level philosophy underlying a wide range of training-free guidance methods (Chung *et al.*, 2023; Ye *et al.*, 2024; He *et al.*, 2024; Bansal *et al.*, 2023). Despite variations in implementation and application, these methods are unified by the central principle expressed in Equation (8.1.1). We first introduce the high-level approach of training-free guidance in Section 8.4.1 and then extend this idea to training-free inverse problem solving, with a brief overview provided in Section 8.4.2.

**Setup and Notations.** Let  $\mathbf{c}$  denote a conditioning variable. We assume access to a pre-trained diffusion model  $\mathbf{s}_{\phi^\times}(\mathbf{x}_t, t)$  expressed in score prediction<sup>4</sup>. In addition, suppose we are given a non-negative function

$$\ell(\cdot, \mathbf{c}): \mathbb{R}^D \rightarrow \mathbb{R}_{\geq 0}$$

that quantifies how well a sample  $\mathbf{x} \in \mathbb{R}^D$  aligns with the condition  $\mathbf{c}$ , where smaller values of  $\ell(\mathbf{x}, \mathbf{c})$  indicate stronger alignment. Concrete examples of such a function include: (i)  $\mathbf{c}$  is a reference image, and  $\ell(\cdot, \mathbf{c})$  is a similarity score measuring perceptual closeness; (ii)  $\ell(\cdot, \mathbf{c})$  is a feature-based similarity score computed via a pre-trained model such as CLIP (Radford *et al.*, 2021).

Consider the standard linear-Gaussian forward noising kernel  $p_t(\cdot | \mathbf{x}_0) := \mathcal{N}(\cdot; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$ . We recall the DDIM update in Equation (9.2.3) and take it as an example:

$$\mathbf{x}_{t \rightarrow t-1} = \alpha_{t-1} \underbrace{\hat{\mathbf{x}}_0(\mathbf{x}_t)}_{\text{in data space}} - \sigma_{t-1} \sigma_t \underbrace{\hat{\mathbf{s}}(\mathbf{x}_t)}_{\text{in noise space}}, \quad (8.4.1)$$

where  $\hat{\mathbf{x}}_0(\mathbf{x}_t) := \mathbf{x}_{\phi^\times}(\mathbf{x}_t, t)$  is the (clean)  $\mathbf{x}$ -prediction, and  $\hat{\mathbf{s}}(\mathbf{x}_t) := \mathbf{s}_{\phi^\times}(\mathbf{x}_t, t)$  as the score-prediction from  $\mathbf{x}_t$  at time level  $t$ .

### 8.4.1 Conceptual Framework for Training-Free Guidance

Most training-free guidance methods (Ye *et al.*, 2024) introduce corrections either in the *data space* or the *noise space* to steer the DDIM update in Equation (8.4.2) toward satisfying the condition  $\mathbf{c}$ :

$$\mathbf{x}_{t \rightarrow t-1} = \underbrace{\alpha_{t-1} \left( \hat{\mathbf{x}}_0(\mathbf{x}_t) + \eta_t^{\text{data}} \mathcal{G}_0 \right)}_{\text{A. data space}} - \sigma_{t-1} \sigma_t \underbrace{\left( \hat{\mathbf{s}}(\mathbf{x}_t) + \eta_t^{\text{latent}} \mathcal{G}_t \right)}_{\text{B. noise space}}, \quad (8.4.2)$$

---

<sup>4</sup>Here, we adopt the score and  $\mathbf{x}$ -prediction parameterization for simplicity of mathematical expression; other parameterizations (e.g.,  $\epsilon$ -prediction) can be handled analogously.

where  $\eta_t^{\text{data}}, \eta_t^{\text{latent}} \geq 0$  are time-dependent guidance strengths, and  $\mathcal{G}_0, \mathcal{G}_t$  are correction terms defined below.

**A. Guidance in Data Space.** By descending along the negative gradient direction

$$\mathcal{G}_0 := -\nabla_{\mathbf{x}_0} \ell(\mathbf{x}_0, \mathbf{c}),$$

the modified clean estimate in data space,

$$\hat{\mathbf{x}}_0(\mathbf{x}_t) + \eta_t^{\text{data}} \mathcal{G}_0,$$

can be gradually steered toward samples that better satisfy the condition  $\mathbf{c}$ . This gradient-descent scheme can be applied iteratively to progressively improve alignment.

Representative examples include MGPD (He *et al.*, 2023) and UGD (Bansal *et al.*, 2023).

**B. Guidance in Noise Space.** As discussed in Section 8.1, the conditional score  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{c}|\mathbf{x}_t)$  is generally intractable. A practical approximation is to introduce a surrogate likelihood  $\tilde{p}_t(\mathbf{c}|\mathbf{x}_t)$ :

$$\tilde{p}_t(\mathbf{c}|\mathbf{x}_t) \propto \exp(-\eta \ell(\hat{\mathbf{x}}_0(\mathbf{x}_t), \mathbf{c}))$$

with a re-scaling constant  $\eta > 0$  so that

$$\nabla_{\mathbf{x}_t} \log \tilde{p}_t(\mathbf{c}|\mathbf{x}_t) = -\eta \nabla_{\mathbf{x}_t} \ell(\hat{\mathbf{x}}_0(\mathbf{x}_t), \mathbf{c}) =: \mathcal{G}_t,$$

where  $\hat{\mathbf{x}}_0(\mathbf{x}_t)$  is obtained via the diffusion model's prediction. Plugging this into the Bayes rule for conditional scores yields the proxy:

$$\begin{aligned} \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{c}) &\approx \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)}_{\text{unconditional}} + \underbrace{\nabla_{\mathbf{x}_t} \log \tilde{p}_t(\mathbf{c}|\mathbf{x}_t)}_{\text{guidance}} \\ &\approx \hat{\mathbf{s}}(\mathbf{x}_t) + \eta_t^{\text{latent}} \mathcal{G}_t, \end{aligned}$$

which serves as the correction with the guidance in the noise spaces.

However, we note that evaluating  $\mathcal{G}_t$  requires backpropagation through the  $\mathbf{x}$ -prediction, i.e.,

$$\nabla_{\mathbf{x}_t} \hat{\mathbf{x}}_0(\mathbf{x}_t)^\top \cdot \nabla_{\mathbf{x}_0} \log \ell_{\mathbf{c}}(\mathbf{x}_0)|_{\mathbf{x}_0=\hat{\mathbf{x}}_0(\mathbf{x}_t)},$$

which may result in substantial computational cost in practice.

Representative examples include (Yu *et al.*, 2023; Chung *et al.*, 2022; Bansal *et al.*, 2023).

### 8.4.2 Examples of Training-Free Approaches to Inverse Problems

The principle introduced in Section 8.4.1 has important applications in inverse problems. We begin with an overview of the background and then provide several concrete examples illustrating how to leverage pre-trained diffusion models for inference-time inverse problem solving.

**Background on Inverse Problems.** Let  $\mathcal{A}$  be a corruption operator (which may be linear or nonlinear, known or unknown), such as a blurring kernel or inpainting, and let  $\mathbf{y}$  be an observation generated by the following corruption model:

$$\mathbf{y} = \mathcal{A}(\mathbf{x}_0) + \sigma_y \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (8.4.3)$$

The objective of inverse problems is to sample from the posterior distribution  $p_0(\mathbf{x}_0|\mathbf{y})$ , where there may exist infinitely many possible reconstructions  $\mathbf{x}_0$  corresponding to the given observation  $\mathbf{y}$ . The goal is to recover an  $\mathbf{x}_0$  that removes the corruptions in  $\mathbf{y}$  while preserving its faithful and semantic features.

Traditional approaches to solving inverse problems typically follow a supervised framework, which requires collecting paired data of corrupted and restored samples  $(\mathbf{y}, \mathbf{x})$  and relies on optimization methods or supervised training of neural networks. Such approaches can be costly in terms of data preparation and may lack generalization to unseen data.

**Pre-Trained Diffusion Models as Inverse Problems Solvers.** As previously shown, the conditional score can be decomposed via Bayes' rule:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{y}) = \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)}_{\text{data score}} + \underbrace{\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t)}_{\text{measurement alignment}}. \quad (8.4.4)$$

This decomposition separates the data score and a measurement alignment term with  $\mathbf{y}$  specific to the inverse problem. It enables solving Equation (8.4.3) in an unsupervised manner by modeling the clean data distribution  $p_{\text{data}}$  and applying it during inversion. More specifically:

- **Data score**  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ : Approximated using a pre-trained diffusion model  $s_{\phi^x}(\mathbf{x}_t, t)$  trained on clean data.
- **Measurement alignment**  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t)$ : Intractable in closed form, as it involves marginalizing over latent variables.

Consequently, most training-free approaches using pre-trained diffusion models focus on approximating  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t)$ . We adopt a common meta-form summarized in (Daras *et al.*, 2024):

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t) \approx -\frac{\mathcal{P}_t \mathcal{M}_t}{\gamma_t}.$$

Here:

- $\mathcal{M}_t$ : error vector quantifying the mismatch between the observation  $\mathbf{y}$  and the estimated signal,
- $\mathcal{P}_t$ : a mapping that projects  $\mathcal{M}_t$  back to the ambient space of  $\mathbf{x}_t$ ,
- $\gamma_t$ : scalar controlling the guidance strength.

Representative methods instantiate  $\mathcal{M}_t$ ,  $\mathcal{P}_t$ , and  $\gamma_t$  differently, as highlighted below with color-coded components.

**Instantiations of Diffusion-Based Inverse Problem Solvers.** We present representative methods that leverage a pre-trained diffusion model to provide unsupervised approaches (requiring no paired data) that can be flexibly applied to various inverse problems using the same learned proxy for  $p_{\text{data}}$ .

**Score SDE (Song *et al.*, 2020c).** One of the earliest works on diffusion-based inverse problem solvers. It considers a known linear corruption model  $\mathbf{A}$  and focuses on the noiseless setting with  $\sigma_y = 0$ . Since  $\mathbf{A}$  is linear, one can form a noise-level-matched observation

$$\mathbf{y}_t := \alpha_t \mathbf{y} + \sigma_t \boldsymbol{\epsilon},$$

and use the residual  $\mathbf{y}_t - \mathbf{A}\mathbf{x}_t$  (note:  $\mathbf{y}_t \neq \mathbf{A}\mathbf{x}_t$  in general) to drive a likelihood-style correction. A common approximation (dropping the multiplicative constant) is

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t) \approx -\mathbf{A}^\top (\mathbf{y}_t - \mathbf{A}\mathbf{x}_t).$$

**Iterative Latent Variable Refinement (ILVR) (Choi *et al.*, 2021).** Using the same setup as ScoreSDE's case, ILVR estimates:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t) \approx -\mathbf{A}^\dagger (\mathbf{y}_t - \mathbf{A}\mathbf{x}_t) = -(\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top (\mathbf{y}_t - \mathbf{A}\mathbf{x}_t),$$

where  $\mathbf{A}^\dagger$  is the Moore–Penrose pseudoinverse, and  $\mathbf{y}_t = \alpha_t \mathbf{y} + \sigma_t \boldsymbol{\epsilon}_t$ .

**Diffusion Posterior Sampling (DPS) (Chung et al., 2022).** A widely used method for inverse problems with known nonlinear forward operator  $\mathcal{A}$  and additive Gaussian noise level  $\sigma_y \geq 0$  is *Denoising Posterior Score* (DPS), which approximates

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|X_0 = \hat{\mathbf{x}}_0(\mathbf{x}_t)), \quad (8.4.5)$$

where  $\hat{\mathbf{x}}_0(\mathbf{x}_t) := \mathbb{E}[\mathbf{x}_0|\mathbf{x}_t]$  denotes the conditional mean of the clean sample given the noisy observation  $\mathbf{x}_t$  at time  $t$ , and is typically estimated using Tweedie's formula (Equation (3.3.6)) from a pre-trained diffusion model.

This one-point approximation assumes that the conditional distribution  $p(\mathbf{x}_0|\mathbf{x}_t)$  is sharply concentrated, and follows from:

$$\begin{aligned} p_t(\mathbf{y}|\mathbf{x}_t) &= \int p_t(\mathbf{y}|\mathbf{x}_t, \mathbf{x}_0)p(\mathbf{x}_0|\mathbf{x}_t)d\mathbf{x}_0 \\ &= \int p_t(\mathbf{y}|\mathbf{x}_0)p(\mathbf{x}_0|\mathbf{x}_t)d\mathbf{x}_0 \approx p_t(\mathbf{y}|X_0 = \hat{\mathbf{x}}_0(\mathbf{x}_t)), \end{aligned}$$

where we have used that  $\mathbf{y}$  depends only on  $\mathbf{x}_0$  (not on  $\mathbf{x}_t$ ) given  $\mathbf{x}_0$ , and the approximation holds under the assumption that the posterior  $p(\mathbf{x}_0|\mathbf{x}_t)$  is tightly peaked around its mean.

Since

$$p_t(\mathbf{y}|X_0 = \hat{\mathbf{x}}_0(\mathbf{x}_t)) = \mathcal{N}(\mathbf{y}; \mathcal{A}(\hat{\mathbf{x}}_0(\mathbf{x}_t)), \sigma_y^2 \mathbf{I}),$$

we compute

$$\begin{aligned} \nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t) &\approx \nabla_{\mathbf{x}_t} \log \mathcal{N}(\mathbf{y}; \mathcal{A}(\hat{\mathbf{x}}_0), \sigma_y^2 \mathbf{I}) \\ &= -\frac{1}{2\sigma_y^2} \nabla_{\mathbf{x}_t} \|\mathbf{y} - \mathcal{A}(\hat{\mathbf{x}}_0)\|^2 \\ &= \frac{1}{\sigma_y^2} [\mathcal{J}_{\mathcal{A}}(\hat{\mathbf{x}}_0(\mathbf{x}_t)) \cdot \nabla_{\mathbf{x}_t} \hat{\mathbf{x}}_0(\mathbf{x}_t)]^\top (\mathbf{y} - \mathcal{A}(\hat{\mathbf{x}}_0(\mathbf{x}_t))), \end{aligned}$$

where  $\mathcal{J}_{\mathcal{A}}(\hat{\mathbf{x}}_0(\mathbf{x}_t)) := \nabla_{\mathbf{x}_0} \mathcal{A}(\mathbf{x})|_{\mathbf{x}=\hat{\mathbf{x}}_0(\mathbf{x}_t)}$  denotes the Jacobian of the forward operator with respect to its input. This formula propagates the gradient through the score approximation pipeline, reflecting how the measurement likelihood changes with respect to perturbations in the noisy sample  $\mathbf{x}_t$ .

For linear inverse problems, this further simplifies to:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t) \approx \frac{1}{\sigma_y^2} [\mathbf{A} \cdot \nabla_{\mathbf{x}_t} \hat{\mathbf{x}}_0(\mathbf{x}_t)]^\top (\mathbf{y} - \mathbf{A}(\hat{\mathbf{x}}_0(\mathbf{x}_t))).$$

A large body of work explores diffusion-based inverse problem solvers by proposing various approximations for  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{y}|\mathbf{x}_t)$ . For a comprehensive overview, we refer readers to the survey by Daras *et al.* (2024).

## 8.5 From Reinforcement Learning to Direct Preference Optimization for Model Alignment

In the pursuit of aligning generative models with human intent, the prevailing paradigm has been Reinforcement Learning from Human Feedback (RLHF). While effective, RLHF is a complex, multi-stage process that can be unstable. This section introduces *Direct Preference Optimization (DPO)* (Rafailov *et al.*, 2023), a more streamlined and stable method that reaches the same goal without explicit reward modeling or reinforcement learning. We then outline its extension to diffusion models via *Diffusion-DPO* (Wallace *et al.*, 2024).

### 8.5.1 The Motivation: Circumventing the Pitfalls of RLHF

The goal of alignment is to steer a base, pre-trained model (e.g., an SFT model) toward outputs that humans prefer. RLHF proceeds in three stages. First, *supervised fine-tuning (SFT)* trains a base model on prompt–response pairs. Second, *reward modeling (RM)* fits a model on preference data consisting of prompts  $\mathbf{c}$  and paired responses (a preferred “winner”  $\mathbf{x}_w$  and a dispreferred “loser”  $\mathbf{x}_l$ ), learning a scalar  $r(\mathbf{c}, \mathbf{x})$  with  $r(\mathbf{c}, \mathbf{x}_w) > r(\mathbf{c}, \mathbf{x}_l)$ . Third, *RL fine-tuning* optimizes the SFT model (policy  $\pi^5$ ) with an algorithm such as PPO (Schulman *et al.*, 2017), maximizing expected reward from  $r$  while regularizing by a KL penalty that keeps  $\pi$  close to the reference/SFT distribution.

Despite its impact, this pipeline suffers from drawbacks: the RL stage is unstable and computationally expensive because it is on-policy—each update requires freshly generated samples from the current model; it also requires training and hosting multiple large models (SFT, reward, and sometimes a value model); and it optimizes only a proxy for human preferences, so flaws in the reward model can be exploited. This motivates a central question:

#### Question 8.5.1

*Can we eliminate explicit reward modeling and the unstable RL step, directly optimizing the model on preference data?*

Direct Preference Optimization (DPO) streamlines alignment by replacing the multi-stage RLHF pipeline with a single, supervised-style step. Instead of training a separate reward model and running unstable RL algorithms like PPO, DPO directly fits the policy to preference pairs using a simple logistic loss, while staying close to a fixed reference model. The key insight is that the

---

<sup>5</sup>A policy maps a prompt/history (state) to a distribution over responses/actions.

KL-regularized RLHF objective can be rewritten so that the log-likelihood ratio between the policy and the reference acts as an implicit reward. This preserves the same regularization toward the reference policy but avoids costly rollouts and explicit reward modeling.

In Section 8.5.2, we briefly review the RLHF pipeline and its reliance on large reward models and RL fine-tuning. In Section 8.5.3, we present DPO, originally proposed for language models, which circumvents reward model training and simplifies alignment fine-tuning. Finally, in Section 8.5.4, we extend this idea to diffusion models, introducing Diffusion-DPO as a practical and stable alignment method in the generative modeling setting.

### 8.5.2 RLHF: Bradley–Terry View

**Short Introduction to RLHF.** RLHF begins with a learned judge: a reward model  $r_\psi$  that assigns a scalar preference score to candidate responses for the same prompt  $\mathbf{c}$ . The dataset  $\mathcal{D}$  consists of pairs  $(\tilde{\mathbf{x}}, \mathbf{x})$  annotated with a label  $y$  indicating whether  $\tilde{\mathbf{x}}$  is preferred over  $\mathbf{x}$ . The label can be binary  $y \in \{0, 1\}$  or a soft value  $y \in [0, 1]$  obtained by aggregating multiple raters. The training objective is a simple logistic loss

$$\begin{aligned} \mathcal{L}_{\text{RM}}(\psi) = -\mathbb{E}_{(\mathbf{c}, \tilde{\mathbf{x}}, \mathbf{x}, y) \sim \mathcal{D}} & \left[ y \log \sigma(r_\psi(\mathbf{c}, \tilde{\mathbf{x}}) - r_\psi(\mathbf{c}, \mathbf{x})) \right. \\ & \left. + (1 - y) \log (1 - \sigma(r_\psi(\mathbf{c}, \tilde{\mathbf{x}}) - r_\psi(\mathbf{c}, \mathbf{x}))) \right], \end{aligned} \quad (8.5.1)$$

where  $\sigma(u) = 1/(1+e^{-u})$ . In practice, preference pairs in  $\mathcal{D}$  may originate from various sources: curated responses, model snapshots at different checkpoints, or generations from a pre-trained conditional diffusion model. A standard convention is to store them in an ordered format (winner, loser). Under this convention we simply set  $y = 1$ , and Equation (8.5.1) reduces to the special case (with  $\tilde{\mathbf{x}} = \mathbf{x}^w$  and  $\mathbf{x} = \mathbf{x}^l$ ):

$$\mathcal{L}_{\text{RM}}(\psi) = -\mathbb{E}_{(\mathbf{c}, \mathbf{x}_w, \mathbf{x}_l) \sim \mathcal{D}} \left[ \log \sigma(r_\psi(\mathbf{c}, \mathbf{x}_w) - r_\psi(\mathbf{c}, \mathbf{x}_l)) \right]. \quad (8.5.2)$$

**Bradley–Terry View and KL Connection.** It is standard to interpret

$$p_{r_\psi}(\tilde{\mathbf{x}} \succ \mathbf{x} | \mathbf{c}) := \sigma(r_\psi(\mathbf{c}, \tilde{\mathbf{x}}) - r_\psi(\mathbf{c}, \mathbf{x}))$$

through the Bradley–Terry (BT) model (Bradley and Terry, 1952), which converts two scalar scores into a win probability. This formulation highlights two key properties: (i) only the *difference* of scores matters (so  $r_\psi(\mathbf{c}, \cdot)$  is

shift-invariant), and (ii) the loss pushes the predicted winner’s score above the loser’s score. To see (ii) intuitively, consider one pair with label  $y \in \{0, 1\}$  and define

$$\Delta r := r_\psi(\mathbf{c}, \tilde{\mathbf{x}}) - r_\psi(\mathbf{c}, \mathbf{x}), \quad p := \sigma(\Delta r), \quad \sigma(u) = \frac{1}{1+e^{-u}}.$$

The per-example logistic loss is

$$\ell = -[y \log p + (1 - y) \log(1 - p)].$$

Then

$$\frac{\partial \ell}{\partial \Delta r} = \sigma(\Delta r) - y.$$

Under gradient descent with step size  $\eta > 0$ , the score gap updates as

$$\Delta r \leftarrow \Delta r - \eta(\sigma(\Delta r) - y).$$

Hence, if  $y = 1$  (“ $\tilde{\mathbf{x}}$  wins”), then  $\sigma(\Delta r) - 1 \leq 0$ , so  $\Delta r$  increases (winner up, loser down); if  $y = 0$ ,  $\Delta r$  decreases.

Each per-example term in Equation (8.5.1) can be viewed as the cross-entropy between the observed Bernoulli label and the model’s predicted win probability:

$$-[y \log p_{r_\psi} + (1 - y) \log(1 - p_{r_\psi})] = \mathcal{D}_{\text{KL}}(\text{Bern}(y) \parallel \text{Bern}(p_{r_\psi})) + \mathcal{H}(\text{Bern}(y)),$$

where  $\mathcal{H}$  is the entropy of the target Bernoulli distribution. Averaging over the dataset  $\mathcal{D}$  gives

$$\mathcal{L}_{\text{RM}}(\psi) = \mathbb{E}_{\mathcal{D}} \left[ \mathcal{D}_{\text{KL}} \left( \text{Bern}(y) \parallel \text{Bern}(p_{r_\psi}) \right) \right] + \underbrace{\mathbb{E}_{\mathcal{D}} [\mathcal{H}(\text{Bern}(y))]}_{\text{independent of } \psi}. \quad (8.5.3)$$

Thus, minimizing the logistic loss is equivalent to minimizing the KL divergence between the empirical Bernoulli distribution of human labels and the model’s predicted Bernoulli distribution. In the binary case ( $y \in \{0, 1\}$ ), this equivalence is exact; for soft labels ( $y \in [0, 1]$ ), the result holds up to an entropy constant offset. Intuitively, the reward model is trained to adjust its win probabilities until they align with the empirical human win rates observed in the dataset.

From this point onward, we adopt the most common convention where  $\mathcal{D}$  stores pairs in an ordered format:  $(\mathbf{x}^w, \mathbf{x}^l, \mathbf{c}) \sim \mathcal{D}$ . Under this convention, the label is always  $y = 1$ , and the loss simplifies to the ordered form given in Equation (8.5.2), which we will use in the following discussion.

**KL Regularized Policy Optimization (with Fixed Reward).** With the fitted reward  $r := r_{\psi^x}$  trained via Equation (8.5.2), and a conditional pre-trained diffusion model  $p_{\phi^x}(\mathbf{x}|\mathbf{c})$ , RLHF then adjusts a learnable policy  $\pi_{\theta}(\mathbf{x}|\mathbf{c})$ , usually fine-tuned on top of  $p_{\phi^x}(\mathbf{x}|\mathbf{c})$ , toward higher-reward responses. At the same time, the policy is regularized to stay close to a reference model, taken as the pre-trained diffusion model  $\pi_{\text{ref}}(\mathbf{x}|\mathbf{c}) := p_{\phi^x}(\mathbf{x}|\mathbf{c})$ , using a  $\mathcal{D}_{\text{KL}}$  penalty:

$$\max_{\theta} \mathbb{E}_{\mathbf{c} \sim p(\mathbf{c})} \left[ \mathbb{E}_{\mathbf{x} \sim \pi_{\theta}(\cdot|\mathbf{c})} [r_{\psi}(\mathbf{c}, \mathbf{x})] - \beta \mathcal{D}_{\text{KL}}(\pi_{\theta}(\cdot|\mathbf{c}) \parallel \pi_{\text{ref}}(\cdot|\mathbf{c})) \right], \quad (8.5.4)$$

which makes the two forces explicit: seek samples the judge prefers, but stay close to the pre-trained reference.

We remark that the reward objective in Equation (8.5.2) uses only labeled pairs and does not require that  $\mathcal{D}$  be generated by the reference model (i.e., the pre-trained conditional diffusion model). While not required, collecting pairs from models close to the intended policy can reduce distribution shift and make the learned reward more reliable in the region where it will be used.

In summary, RLHF proceeds in two stages: first fit the reward  $r^*$  by minimizing the loss in Equation (8.5.2) (equivalently, the expected binary  $\mathcal{D}_{\text{KL}}$  in Equation (8.5.3)); then optimize the policy  $\pi^*$  by solving Equation (8.5.4).

### 8.5.3 DPO Framework

**The Bridge from RLHF.** The KL-regularized policy objective in Equation (8.5.4) has a simple closed-form solution for each prompt  $\mathbf{c}$ , given the fitted reward  $r := r_{\psi^\times}$ , expressed in the following energy-based form (Peters *et al.*, 2010):

$$\pi^*(\mathbf{x}|\mathbf{c}) = \frac{1}{Z(\mathbf{c})} \pi_{\text{ref}}(\mathbf{x}|\mathbf{c}) \exp(r(\mathbf{c}, \mathbf{x})/\beta), \quad (8.5.5)$$

where  $\pi_{\text{ref}}(\mathbf{x}|\mathbf{c}) := p_{\phi^\times}(\mathbf{x}|\mathbf{c})$ , and  $Z(\mathbf{c})$  is the partition function ensuring  $\int \pi^*(\mathbf{x}|\mathbf{c}) d\mathbf{x} = 1$ .

For smaller  $\beta$ ,  $\exp(r/\beta)$  becomes sharper, so  $\pi^*$  concentrates on high reward regions: reward dominates, the policy moves farther from  $\pi_{\text{ref}}$ , diversity decreases, and training may become unstable or prone to reward hacking. For larger  $\beta$ ,  $\exp(r/\beta)$  flattens, keeping  $\pi^*$  closer to  $\pi_{\text{ref}}$ : the KL term dominates, updates are conservative, diversity follows the reference, but reward gains are limited.

Since our aim is to fine-tune the policy directly (without training a separate reward model), Equation (8.5.5) lets us *define* an *implicit reward* from any policy. We introduce below:

**Defining an Implicit Reward Motivated by Inverting Equation (8.5.5).** Equation (8.5.5) suggests an immediate inversion: for any policy  $\pi$  (with support contained in  $\pi_{\text{ref}}$ ), define

$$r_\pi(\mathbf{c}, \mathbf{x}) = \beta \log \frac{\pi(\mathbf{x}|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}|\mathbf{c})} + \beta \log Z(\mathbf{c}). \quad (8.5.6)$$

Then Equation (8.5.5) holds with  $\pi$  in place of  $\pi^*$ , i.e.,  $\pi$  would be the optimizer of Equation (8.5.4) for the reward function  $r_\pi$ . In this sense,  $r_\pi$  is an *implicit (policy-induced) reward*: it is identified up to the prompt-dependent constant  $\beta \log Z(\mathbf{c})$ , which vanishes in any pairwise comparison such as in the BT model:

$$r_\pi(\mathbf{c}, \mathbf{x}_w) - r_\pi(\mathbf{c}, \mathbf{x}_l) = \beta \left( \log \frac{\pi(\mathbf{x}_w|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_w|\mathbf{c})} - \log \frac{\pi(\mathbf{x}_l|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_l|\mathbf{c})} \right).$$

This cancellation is exactly what makes the constant irrelevant for preference learning and leads directly to the DPO loss on log-probability differences.

**DPO's Training Loss.** Plug the implicit reward Equation (8.5.6) into the BT model of Equation (8.5.2) for a labeled pair  $(\mathbf{x}_w, \mathbf{x}_l)$  under the same prompt

c. The constants  $\log Z(\mathbf{c})$  cancel between winner and loser, yielding a single logistic-loss objective on log-probability differences:

$$\mathcal{L}_{\text{DPO}}(\boldsymbol{\theta}; \pi_{\text{ref}}) = -\mathbb{E}_{(\mathbf{c}, \mathbf{x}_w, \mathbf{x}_l) \sim \mathcal{D}} \left[ \log \sigma \left( \beta \left( \log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_w | \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_w | \mathbf{c})} - \log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{x}_l | \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_l | \mathbf{c})} \right) \right) \right].$$

In words: DPO pushes up the (temperature-scaled) advantage of the winner over the loser, measured as the difference of log-likelihood improvements over the reference:

$$-\log \sigma \left( \beta [\text{log-ratio difference of } \frac{\pi_{\boldsymbol{\theta}}}{\pi_{\text{ref}}} \text{ at } \mathbf{x}_w \text{ vs. } \mathbf{x}_l] \right).$$

This achieves the goal of RLHF in a single, stable maximum-likelihood-style stage, without training an explicit reward model.

### 8.5.4 Diffusion-DPO

**Why Naive DPO Fails for Diffusion Models?** Evaluating the sample likelihood  $\pi_{\boldsymbol{\theta}}(\mathbf{x} | \mathbf{c})$  in diffusion models requires the instantaneous change-of-variables formula (divergence of the drift) of ODE solving (see Equation (4.2.7))<sup>6</sup>, which is computationally intensive. Moreover, differentiating through the entire sampling trajectory can suffer from vanishing or exploding gradients. To avoid these issues, Diffusion-DPO works at the *path* level. We take the discrete-time diffusion model (e.g., DDPM) as an illustrative example; the continuous-time diffusion model is analogous.

**Defining Pathwise Implicit Rewards.** Let a trajectory be  $\mathbf{x}_{0:T} := (\mathbf{x}_T, \dots, \mathbf{x}_0)$  under the reverse-time Markov chain with conditionals  $\pi(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{c})$ . Here,  $\mathbf{x}_T$  denotes a sample from the prior (highest noise), and  $\mathbf{x}_0$  is the clean output in data space. Since generation in diffusion models proceeds along a full denoising path, it is natural to extend preferences from final outputs to the entire trajectory. We therefore assign each trajectory a reward  $R(\mathbf{c}, \mathbf{x}_{0:T})$ , which reduces to an endpoint reward if it depends only on  $\mathbf{x}_0$ , but can also capture cumulative effects along the path.

We replace the sample-level KL in Equation (8.5.4) by a pathwise KL as:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{c} \sim p(\mathbf{c})} \left[ \underbrace{\mathbb{E}_{\mathbf{x}_{0:T} \sim \pi_{\boldsymbol{\theta}}(\cdot | \mathbf{c})} [R(\mathbf{c}, \mathbf{x}_{0:T})]}_{\text{reward over paths}} - \beta \mathcal{D}_{\text{KL}}(\pi_{\boldsymbol{\theta}}(\cdot | \mathbf{c}) \| \pi_{\text{ref}}(\cdot | \mathbf{c})) \right],$$

---

<sup>6</sup>In discrete-time diffusion models (e.g., DDPM), evaluating  $\pi_{\boldsymbol{\theta}}(\mathbf{x}_0 | \mathbf{c})$  requires marginalizing over the latent reverse trajectory  $\mathbf{x}_{1:T}$ .

where  $\pi_\theta(\cdot|\mathbf{c})$  and  $\pi_{\text{ref}}(\cdot|\mathbf{c})$  are the *path* distributions. It aims to maximize the reward for reverse process  $\pi_\theta(\cdot|\mathbf{c})$ , while matching the distribution of the original reference reverse process  $\pi_{\text{ref}}(\cdot|\mathbf{c})$ .

For each prompt  $\mathbf{c}$ , the optimizer has the simple energy-based form

$$\pi^*(\mathbf{x}_{0:T}|\mathbf{c}) = \frac{1}{Z(\mathbf{c})} \pi_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{c}) \exp(R(\mathbf{c}, \mathbf{x}_{0:T})/\beta), \quad (8.5.7)$$

with  $Z(\mathbf{c})$  a normalizer. Inverting Equation (8.5.7) motivates the definition of an *implicit path reward* for any policy  $\pi$ :

$$R_\pi(\mathbf{c}, \mathbf{x}_{0:T}) := \beta \log \frac{\pi(\mathbf{x}_{0:T}|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{c})} + \beta \log Z(\mathbf{c}),$$

whose constant  $\beta \log Z(\mathbf{c})$  is irrelevant for pairwise comparisons.

**From Pathwise Implicit Rewards to DPO.** Apply the Bradley–Terry model to *paths* for a labeled pair  $(\mathbf{x}_0^w, \mathbf{x}_0^l)$  under the same prompt  $\mathbf{c}$ , and use the standard logistic log-loss:

$$\begin{aligned} \mathcal{L}_{\text{Diff-DPO}}(\boldsymbol{\theta}; \pi_{\text{ref}}) &:= -\mathbb{E}_{(\mathbf{c}, \mathbf{x}_0^w, \mathbf{x}_0^l) \sim \mathcal{D}} [\log \sigma(\Delta R(\mathbf{c}; \boldsymbol{\theta}))], \quad \text{where} \\ \Delta R(\mathbf{c}; \boldsymbol{\theta}) &:= \underbrace{\mathbb{E}_{\mathbf{x}_{1:T}^w \sim \pi_\theta(\cdot|\mathbf{x}_0^w, \mathbf{c})} [R_{\pi_\theta}(\mathbf{c}, (\mathbf{x}_0^w, \mathbf{x}_{1:T}^w))]}_{\text{winner path expectation}} \\ &\quad - \underbrace{\mathbb{E}_{\mathbf{x}_{1:T}^l \sim \pi_\theta(\cdot|\mathbf{x}_0^l, \mathbf{c})} [R_{\pi_\theta}(\mathbf{c}, (\mathbf{x}_0^l, \mathbf{x}_{1:T}^l))]}_{\text{loser path expectation}}. \end{aligned} \quad (8.5.8)$$

Here, the expectation  $\mathbb{E}_{\mathbf{x}_{1:T} \sim \pi_\theta(\cdot|\mathbf{x}_0, \mathbf{c})}[\cdot]$  means: given a fixed endpoint  $\mathbf{x}_0$  (e.g., the winner  $\mathbf{x}_0^w$ ) from the dataset, we take an expectation over latent denoising trajectories  $\mathbf{x}_{1:T}$  under the model-induced conditional path distribution (the posterior over reverse-time trajectories) that, with kernels  $\pi_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})$ , could produce  $\mathbf{x}_0$ . Since these intermediate states are unobserved, we average the path reward over all such trajectories.

However, Equation (8.5.8) is impractical for three practical reasons:

1. **Endpoint Conditioning Induces an Intractable Path Posterior.** The term  $\mathbb{E}_{\pi_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0, \mathbf{c})}[\cdot]$  averages over reverse paths constrained to hit  $\mathbf{x}_0$ , whereas the sampler runs  $\mathbf{x}_T \rightarrow \dots \rightarrow \mathbf{x}_0$  without this constraint. Conditioning on the endpoint creates a diffusion-bridge posterior with generally no closed form and costly sampling.
2. **Nested,  $\boldsymbol{\theta}$ -Coupled Expectations.** The loss  $-\log \sigma(\Delta R(\mathbf{c}; \boldsymbol{\theta}))$  with

$$\Delta R = \mathbb{E}_{\text{paths}|\mathbf{x}_0^w, \mathbf{c}} [R_{\pi_\theta}] - \mathbb{E}_{\text{paths}|\mathbf{x}_0^l, \mathbf{c}} [R_{\pi_\theta}]$$

has both the path joint distribution and the integrand  $R_{\pi_\theta}$  depending on  $\theta$ . Thus  $\nabla_\theta$  must differentiate through the sampling distribution, leading to REINFORCE/pathwise couplings and high-variance gradients.

3. **Long Chains, Large Sums, and Expensive Backpropagation.** In  $R_{\pi_\theta}(\mathbf{c}, \mathbf{x}_{0:T})$ , computing

$$\beta [\log \pi_\theta(\mathbf{x}_{0:T} | \mathbf{c}) - \log \pi_{\text{ref}}(\mathbf{x}_{0:T} | \mathbf{c})]$$

requires  $\mathcal{O}(T)$  per-step log-densities with  $T \sim 10^2\text{--}10^3$ , for both policy  $\pi_\theta$  and reference  $\pi_{\text{ref}}$ , and for both winner/loser paths. Backpropagating through these stochastic chains (or bridge samplers) is memory and compute heavy and can be unstable; repeating this over many samples per pair and across all triplets pushes training beyond practical budgets.

**Toward a Tractable Surrogate for Equation (8.5.8).** To make this computable, we apply a key mathematical insight. By leveraging properties of diffusion models and applying Jensen's inequality, we can optimize a tractable upper bound on this loss. This transforms the problem from evaluating an entire path's likelihood to evaluating an expectation over the individual, single-step transitions within the path:

Because  $-\log \sigma(\cdot)$  is convex, Jensen's inequality yields an upper bound by moving the inner expectations outside the log:

$$\begin{aligned} & \mathcal{L}_{\text{Diff-DPO}}(\theta; \pi_{\text{ref}}) \\ & \leq -\mathbb{E}_{(\mathbf{c}, \mathbf{x}_0^w, \mathbf{x}_0^l) \sim \mathcal{D}} \mathbb{E}_{\substack{\mathbf{x}_{1:T}^w \sim \pi_\theta(\cdot | \mathbf{x}_0^w, \mathbf{c}) \\ \mathbf{x}_{1:T}^l \sim \pi_\theta(\cdot | \mathbf{x}_0^l, \mathbf{c})}} [\log \sigma(\beta(R(\mathbf{c}, \mathbf{x}_{0:T}^w) - R(\mathbf{c}, \mathbf{x}_{0:T}^l)))]. \end{aligned}$$

Using the implicit-reward identity  $R_{\pi_\theta} = \beta \log \frac{\pi_\theta}{\pi_{\text{ref}}} + \beta \log Z(\mathbf{c})$  and cancellation of the constant between winner and loser, the bound becomes

$$\mathcal{L}_{\text{Diff-DPO}}(\theta; \pi_{\text{ref}}) \leq -\mathbb{E} \left[ \log \sigma \left( \beta \left( \log \frac{\pi_\theta(\mathbf{x}_{0:T}^w | \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{0:T}^w | \mathbf{c})} - \log \frac{\pi_\theta(\mathbf{x}_{0:T}^l | \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{0:T}^l | \mathbf{c})} \right) \right) \right]. \quad (8.5.9)$$

**A Tractable Surrogate (Stepwise Form).** We now exploit the Markov property of the reverse process to decompose the upper bound of  $\mathcal{L}_{\text{Diff-DPO}}$ . This allows us to express the path-level preference as a sum of per-step contributions, converting the intractable pathwise loss into a tractable single-step estimator. The resulting form reduces to a DSM-style MSE difference.

Concretely, for the reverse chain,

$$\begin{aligned}\pi_{\theta}(\mathbf{x}_{0:T}|\mathbf{c}) &= \pi_{\theta}(\mathbf{x}_T|\mathbf{c}) \prod_{t=1}^T \pi_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}), \\ \pi_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{c}) &= \pi_{\text{ref}}(\mathbf{x}_T|\mathbf{c}) \prod_{t=1}^T \pi_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}).\end{aligned}$$

Hence

$$\frac{\pi_{\theta}(\mathbf{x}_{0:T}|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{c})} = \frac{\pi_{\theta}(\mathbf{x}_T|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_T|\mathbf{c})} \prod_{t=1}^T \frac{\pi_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}.$$

If the prior at time  $T$  is the same for both models,  $\pi_{\theta}(\mathbf{x}_T|\mathbf{c}) = \pi_{\text{ref}}(\mathbf{x}_T|\mathbf{c})$ , then the first factor equals 1, and taking logs yields

$$\log \frac{\pi_{\theta}(\mathbf{x}_{0:T}|\mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{0:T}|\mathbf{c})} = \sum_{t=1}^T \log \frac{\pi_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}.$$

It follows that the bound in Equation (8.5.9) can be written as

$$\mathcal{L}_{\text{Diff-DPO}}(\boldsymbol{\theta}; \pi_{\text{ref}}) \leq -\mathbb{E} \left[ \log \sigma \left( \beta \sum_{t=1}^T \Delta_t \right) \right],$$

where each per-step contribution is

$$\Delta_t = \log \frac{\pi_{\theta}(\mathbf{x}_{t-1}^w|\mathbf{x}_t^w, \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{t-1}^w|\mathbf{x}_t^w, \mathbf{c})} - \log \frac{\pi_{\theta}(\mathbf{x}_{t-1}^l|\mathbf{x}_t^l, \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{t-1}^l|\mathbf{x}_t^l, \mathbf{c})}.$$

To obtain a tractable estimator, we apply a single step Jensen upper bound: sample  $t \sim \mathcal{U}\{1, \dots, T\}$  (one timestep per training pair) and rescale by  $T$ . This yields

$$-\log \sigma \left( \beta \sum_{t=1}^T \Delta_t \right) \leq \mathbb{E}_t [-\log \sigma(\beta T \Delta_t)].$$

Thus the final objective is an expected per-step surrogate,

$$\mathcal{L}_{\text{Diff-DPO}}(\boldsymbol{\theta}; \pi_{\text{ref}}) \leq -\mathbb{E}_{\substack{(\mathbf{c}, \mathbf{x}_0^w, \mathbf{x}_0^l) \sim \mathcal{D} \\ t \sim \mathcal{U}\{1, \dots, T\}}} [\log \sigma(\beta T \Delta_t)],$$

which reduces the original pathwise loss to a tractable single step upper-bound estimator.

For Gaussian reverse conditionals used in diffusion models (take  $\epsilon$ -prediction as an example),

$$\log \frac{\pi_{\theta}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})}{\pi_{\text{ref}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})} = \text{const} - \lambda_t \left( \underbrace{\|\hat{\epsilon}_{\theta}(\mathbf{x}_t, t, \mathbf{c}) - \epsilon_t\|^2}_{\text{policy}} - \underbrace{\|\hat{\epsilon}_{\text{ref}}(\mathbf{x}_t, t, \mathbf{c}) - \epsilon_t\|^2}_{\text{reference}} \right),$$

where  $\lambda_t > 0$  absorbs noise schedule factors. Thus each per-time contribution is proportional to an MSE difference (policy vs. reference) at slice  $t$ .

For notation simplicity, define for any  $\mathbf{x}_t$ :

$$\Delta\text{MSE}(\mathbf{x}_t) := \|\hat{\epsilon}_{\theta}(\mathbf{x}_t, t, \mathbf{c}) - \boldsymbol{\epsilon}\|^2 - \|\hat{\epsilon}_{\text{ref}}(\mathbf{x}_t, t, \mathbf{c}) - \boldsymbol{\epsilon}\|^2.$$

This motivates the following practical surrogate for  $\mathcal{L}_{\text{Diff-DPO}}(\boldsymbol{\theta}; \pi_{\text{ref}})$ :

$$\begin{aligned} \tilde{\mathcal{L}}_{\text{Diff-DPO}}(\boldsymbol{\theta}; \pi_{\text{ref}}) := \\ \mathbb{E}_{\substack{(\mathbf{c}, \mathbf{x}_0^w, \mathbf{x}_0^l) \sim \mathcal{D} \\ t \sim \mathcal{U}\{1, \dots, T\}, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})}} \left[ w(t) (\Delta\text{MSE}(\mathbf{x}_t^w) - \Delta\text{MSE}(\mathbf{x}_t^l)) \right], \end{aligned}$$

where  $\mathbf{x}_t^w = \alpha_t \mathbf{x}_0^w + \sigma_t \boldsymbol{\epsilon}$  and  $\mathbf{x}_t^l = \alpha_t \mathbf{x}_0^l + \sigma_t \boldsymbol{\epsilon}$  share the same noise  $\boldsymbol{\epsilon}$  for variance reduction, and  $w(t) > 0$  collects the time weighting (e.g.,  $w(t) \propto \lambda_t$ ).

Intuitively, minimizing  $\tilde{\mathcal{L}}_{\text{Diff-DPO}}$  increases the model's prediction accuracy on the winner relative to the reference and decreases it on the loser. Because improvements are always measured relative to  $\pi_{\text{ref}}$  at the same time step, the policy is nudged toward winner-like denoising trajectories and away from loser-like ones, while remaining anchored to the reference.

## 8.6 Closing Remarks

This chapter has shifted our focus from foundational principles to the practical challenge of controllable generation. We established a unified framework for guidance based on the Bayesian decomposition of the conditional score, which elegantly separates the generative process into an unconditional direction and a steering term.

We saw this principle manifest in several powerful techniques. We covered methods that require dedicated training, such as Classifier Guidance (CG), which uses an external classifier , and the more efficient Classifier-Free Guidance (CFG), which learns conditional and unconditional scores within a single model. We also explored flexible training-free guidance methods, which can steer a pre-trained model at inference time by defining a surrogate likelihood from an arbitrary loss function, enabling applications from artistic control to solving inverse problems without any retraining.

Beyond simple conditioning, we delved into the nuanced task of aligning model outputs with human preferences. After reviewing the standard but complex RLHF pipeline, we introduced Direct Preference Optimization (DPO) and its novel adaptation, Diffusion-DPO, as a more direct and stable alternative. This approach elegantly bypasses the need for an explicit reward model and reinforcement learning by deriving a loss directly from preference data.

Through these techniques, we have assembled a powerful toolkit for steering the generative process. However, a major practical hurdle remains untouched: the significant computational cost and latency of the iterative sampling process itself. Having addressed what to generate, we now turn to the equally important question of how fast we can generate it. The next chapter will tackle this challenge directly:

1. We will leverage the insight that sampling is equivalent to solving an ODE to explore sophisticated numerical solvers designed to drastically reduce the number of required steps.
2. We will investigate a sequential of influential methods, including DDIM, DEIS, and the DPM-Solver family, which have made diffusion models far more practical by accelerating sampling speed by orders of magnitude.

# 9

---

## Sophisticated Solvers for Fast Sampling

---

The generation process of a diffusion model, which maps noise to data samples, is mathematically equivalent to solving either an SDE or its associated ODE. This procedure is inherently slow, since it relies on numerical solvers that approximate solution trajectories with many small integration steps (see Chapter A for a brief introduction). Accelerating inference has therefore become a central research objective. Broadly, existing approaches fall into two categories:

- **Training-Free Approaches:** The focus of this chapter. These methods develop advanced numerical solvers to improve the efficiency of diffusion sampling without additional training.
- **Training-Based Approaches:** Covered in Chapters 10 and 11. These techniques either distill a pre-trained diffusion model into a fast generator, or directly learn the ODE flow map (solution) so that only a few sampling steps are required.

SDE-based samplers (e.g., Euler–Maruyama) may yield more diverse samples due to stochasticity but typically require more steps (Xu *et al.*, 2023). Here we focus on ODE-based generation, whose principles extend naturally to the SDE setting.

## 9.1 Prologue

### 9.1.1 Advanced Solvers for Diffusion Models

The Score SDE framework (Song *et al.*, 2020c) established a key foundation by rigorously linking the discrete-time diffusion and ELBO formulations (Sohl-Dickstein *et al.*, 2015; Ho *et al.*, 2020) with the continuous-time SDE/ODE perspective of generative modeling. This unification not only provides theoretical clarity but also enables principled development of efficient sampling algorithms based on numerical integration.

Concretely, suppose we have a pre-trained diffusion model  $\mathbf{s}_{\phi^x}(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$  (which admits the other three equivalent expressions as in Section 6.3). In this case, the sampling procedure can be viewed as solving the PF-ODE with initial condition  $\mathbf{x}(T) \sim p_{\text{prior}}$ , integrated backward in time from  $t = T$  down to  $t = 0$ :

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), t) - \frac{1}{2}g^2(t) \underbrace{\nabla_{\mathbf{x}} \log p_t(\mathbf{x}(t))}_{\approx \mathbf{s}_{\phi^x}(\mathbf{x}(t), t)}.$$

This ODE is directly associated with the forward stochastic process

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t) dt + g(t) d\mathbf{w}(t),$$

showing the continuous-time connection between the generative (reverse-time) and noising (forward-time) dynamics.

The exact solution of the PF-ODE can be written equivalently in integral form:

$$\begin{aligned} \Psi_{T \rightarrow 0}(\mathbf{x}(T)) &= \mathbf{x}(T) + \int_T^0 \left[ f(\tau)\mathbf{x}(\tau) - \frac{1}{2}g^2(\tau)\nabla_{\mathbf{x}} \log p_{\tau}(\mathbf{x}(\tau)) \right] d\tau \\ &\approx \mathbf{x}(T) + \int_T^0 \left[ f(\tau)\mathbf{x}(\tau) - \frac{1}{2}g^2(\tau)\mathbf{s}_{\phi^x}(\mathbf{x}(\tau), \tau) \right] d\tau \quad (9.1.1) \\ &=: \tilde{\Psi}_{T \rightarrow 0}(\mathbf{x}(T)). \end{aligned}$$

Here,  $\Psi_{s \rightarrow t}(\mathbf{x})$  denotes the flow map of the *oracle* PF-ODE, mapping a state  $\mathbf{x}$  at time  $s$  to its evolved state at time  $t$  (see Equation (4.1.9)). In contrast,  $\tilde{\Psi}_{s \rightarrow t}(\mathbf{x})$  denotes the flow map of the *empirical* PF-ODE, obtained by replacing the true diffusion model  $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$  with its learned approximation  $\mathbf{s}_{\phi^x}(\mathbf{x}, t)$ . Thus,  $\tilde{\Psi}_{s \rightarrow t} \approx \Psi_{s \rightarrow t}$ .

Since the integral form of  $\tilde{\Psi}_{s \rightarrow t}$  cannot be evaluated in closed form, sampling must rely on *numerical solvers*. These methods approximate the solution by discretizing time and replacing the continuous integral with a

finite sum of local drift evaluations, thereby tracing an approximate trajectory. Such solver-based integral approximations are referred to as *training-free* algorithms for fast diffusion sampling, since they aim to approximate the PF-ODE solution directly from the frozen pre-trained score model  $\mathbf{s}_{\phi^\times}$  without requiring any additional learning.

Below we first detail the common concept of numerical solvers and introduce the notations used later.

**Discretized Approximation of Continuous Trajectories.** Let  $\mathbf{x}_T$  denote the initial state at time  $T$ , and consider a decreasing partition

$$T = t_0 > t_1 > \dots > t_M = 0. \quad (9.1.2)$$

Starting from  $\tilde{\mathbf{x}}_{t_0} = \mathbf{x}_T \sim p_{\text{prior}}$ , the solver produces a sequence  $\{\tilde{\mathbf{x}}_{t_i}\}_{i=0}^M$  that ideally approximates the empirical PF-ODE flow  $\tilde{\Psi}_{T \rightarrow t_i}(\mathbf{x}_T)$ , itself a proxy for the oracle map  $\Psi_{T \rightarrow t_i}(\mathbf{x}_T)$ . Each numerical step advances the state via this empirical velocity field, and the final iterate  $\tilde{\mathbf{x}}_{t_M}$  serves as an estimate of the clean sample  $\mathbf{x}_0$  at  $t = 0$ .

### 9.1.2 A Common Framework for Designing Solvers in Literature

Zhang and Chen (2022) highlighted three practical principles for designing numerical solvers for the PF-ODE associated with diffusion models.

**I. Semilinear Structure.** Although Song *et al.* (2020c) establish the foundation for a general drift  $\mathbf{f}(\mathbf{x}(t), t)$ , in most scheduler formulations the drift is instantiated in a linear form

$$\mathbf{f}(\mathbf{x}, t) := f(t) \mathbf{x}, \quad f : \mathbb{R} \rightarrow \mathbb{R},$$

which induces the PF-ODE in a *semilinear* structure:

$$\frac{d\mathbf{x}(t)}{dt} = \underbrace{f(t)\mathbf{x}(t)}_{\text{linear part}} - \underbrace{\frac{1}{2}g^2(t)\mathbf{s}_{\phi^\times}(\mathbf{x}(t), t)}_{\text{nonlinear part}}. \quad (9.1.3)$$

This linear–nonlinear split in  $\mathbf{x}$  is advantageous for accuracy and stability and motivates specialized integrators (see discussion near Equation (9.1.6) below) (Hochbruck and Ostermann, 2005; Hochbruck and Ostermann, 2010).

**II. Parameterizations beyond the Score.** As  $t \rightarrow 0$ , the true score  $\nabla_{\mathbf{x}} \log p_t(\cdot)$  can change very rapidly (for example, when  $p_{\text{data}}$  is concentrated near a low-dimensional manifold) (Kim *et al.*, 2022). This makes it difficult for a neural

network  $\mathbf{s}_{\phi^\times}$ , which is trained to approximate the score directly, to remain accurate.

To see why, recall the oracle relation (see Equation (6.3.1))

$$\boldsymbol{\epsilon}^*(\mathbf{x}_t, t) = -\sigma_t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t),$$

where  $\boldsymbol{\epsilon}^*(\mathbf{x}_t, t) = \mathbb{E}[\boldsymbol{\epsilon}|\mathbf{x}_t]$  is the oracle noise, and  $(\alpha_t, \sigma_t)$  are the mean and standard deviation of the perturbation kernel  $\mathbf{x}_t|\mathbf{x}_0 \sim \mathcal{N}(\alpha_t \mathbf{x}_0, \sigma_t^2 I)$ , connected to  $f(t), g(t)$  via Equation (4.4.2). From the orthogonality property in  $L^2$ ,

$$\mathbb{E} \|\boldsymbol{\epsilon}\|_2^2 = \mathbb{E} \|\boldsymbol{\epsilon}^*\|_2^2 + \mathbb{E} \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}^*\|_2^2 \Rightarrow \mathbb{E} \|\boldsymbol{\epsilon}^*\|_2^2 \leq \mathbb{E} \|\boldsymbol{\epsilon}\|_2^2 = D.$$

Hence the oracle noise predictor is always bounded, but the score grows like

$$\mathbb{E} \|\mathbf{s}^*(\mathbf{x}_t, t)\|_2^2 = \sigma_t^{-2} \mathbb{E} \|\boldsymbol{\epsilon}^*(\mathbf{x}_t, t)\|_2^2 \leq \frac{D}{\sigma_t^2}.$$

Thus, as  $t \rightarrow 0$ , the score can blow up at the rate  $1/\sigma_t^2$ , while the noise predictor stays bounded. Because neural networks can only approximate smoothly growing functions, score prediction tends to be numerically unstable and less accurate, which in turn can harm numerical PF-ODE solvers when relying on a pre-trained model as a drift.

For this reason, a widely used alternative is to predict the noise  $\boldsymbol{\epsilon}_{\phi^\times}$  (or its variants such as  $\mathbf{x}$ - or  $\mathbf{v}$ -prediction), which is stably bounded and admits a simple closed-form relation to the score:

$$\mathbf{s}_{\phi^\times}(\mathbf{x}, t) = -\frac{1}{\sigma_t} \boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}, t).$$

Substituting this relation into the PF-ODE (cf. Equation (6.3.2)) gives

$$\frac{d\mathbf{x}(t)}{dt} = \underbrace{f(t)\mathbf{x}(t)}_{\text{linear part}} + \underbrace{\frac{1}{2} \frac{g^2(t)}{\sigma_t} \boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}(t), t)}_{\text{nonlinear part}}. \quad (9.1.4)$$

This parameterization is commonly adopted by modern PF-ODE solvers.

**III. Exponential Integrators for semilinear PF-ODEs.** For the semilinear structure in Equation (9.1.4), the *exponential integrator formula* in Equation (9.1.6) provides an exact alternative representation of the solution. To see this, let  $\mathbf{x}_s$  denote the state at start time  $s$ , and let  $t \in [0, s]$  be the terminal time<sup>1</sup>.

---

<sup>1</sup>Here,  $s$  is the start time and  $t$  the terminal time, so sampling integrates backward with  $s > t$ .

For clarity, write the nonlinear part of Equation (9.1.4) as

$$\mathbf{N}(\mathbf{x}(t), t) := \frac{1}{2} \frac{g^2(t)}{\sigma_t} \boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}(t), t).$$

The ODE can then be written as

$$\frac{d\mathbf{x}(t)}{dt} - \underbrace{f(t)\mathbf{x}(t)}_{\text{linear part}} = \underbrace{\mathbf{N}(\mathbf{x}(t), t)}_{\text{nonlinear part}}. \quad (9.1.5)$$

To isolate the linear term, we introduce the *exponential integrator*

$$\mathcal{E}(s \rightarrow t) := \exp\left(\int_s^t f(u) du\right),$$

and multiply both sides of the ODE by its inverse  $\mathcal{E}(t \rightarrow s)$ . By the product rule,

$$\mathcal{E}^{-1}(s \rightarrow t) \left( \frac{d\mathbf{x}(t)}{dt} - f(t)\mathbf{x}(t) \right) = \frac{d}{dt} [\mathcal{E}^{-1}(s \rightarrow t)\mathbf{x}(t)].$$

Hence the equation becomes

$$\frac{d}{dt} [\mathcal{E}^{-1}(s \rightarrow t)\mathbf{x}(t)] = \mathcal{E}^{-1}(s \rightarrow t) \mathbf{N}(\mathbf{x}(t), t).$$

Integrating from  $s$  to  $t$  and then multiplying back by  $\mathcal{E}(s \rightarrow t)$  gives the solution:

$$\tilde{\Psi}_{s \rightarrow t}(\mathbf{x}_s) = \underbrace{\mathcal{E}(s \rightarrow t)\mathbf{x}_s}_{\text{linear part}} + \frac{1}{2} \int_s^t \frac{g^2(\tau)}{\sigma_\tau} \mathcal{E}(\tau \rightarrow t) \boldsymbol{\epsilon}_{\phi^\times}(\mathbf{x}_\tau, \tau) d\tau. \quad (9.1.6)$$

We refer the reader to Section A.1.3 for the full details of the derivation.

To explain why the exponential–integration form in Equation (9.1.6) is preferable to Equation (9.1.4) for few-step sampling (large  $\Delta s$ ), we compare their one-step updates. Using variation of constants,  $\mathcal{E}(s \rightarrow s - \Delta s) = e^{-f(s)\Delta s}$  and freezing  $\mathbf{N}(\mathbf{x}(\tau), \tau) \approx \mathbf{N}(\mathbf{x}_s, s)$  for  $\tau \in [s - \Delta s, s]$ , the exponential–Euler update of Equation (9.1.6) is

$$\mathbf{x}_{s-\Delta s}^{\text{Exp-Euler}} = \underbrace{e^{-f(s)\Delta s} \mathbf{x}_s}_{\text{linear part}} + \underbrace{\frac{e^{-f(s)\Delta s} - 1}{f(s)} \mathbf{N}(\mathbf{x}_s, s)}_{\text{nonlinear part}}, \quad (9.1.7)$$

with the natural limit  $(e^{-f\Delta s} - 1)/f \rightarrow -\Delta s$  as  $f \rightarrow 0$ . Here the linear factor  $e^{-f(s)\Delta s}$  is exactly computed (no approximation).

In contrast, approximating  $f(\tau)\mathbf{x}_\tau - \mathbf{N}(\mathbf{x}_\tau, \tau) \approx f(s)\mathbf{x}_s - \mathbf{N}(\mathbf{x}_s, s)$  for  $\tau \in [s - \Delta s, s]$  yields the plain–Euler step for Equation (9.1.4):

$$\mathbf{x}_{s-\Delta s}^{\text{Euler}} = \mathbf{x}_s - \Delta s [f(s)\mathbf{x}_s + \mathbf{N}(\mathbf{x}_s, s)] = \underbrace{(1 - f(s)\Delta s)\mathbf{x}_s}_{\text{linear part}} - \underbrace{\Delta s \mathbf{N}(\mathbf{x}_s, s)}_{\text{nonlinear part}}. \quad (9.1.8)$$

The linear factor in Equation (9.1.8) is the first–order Taylor approximation of the exponential in Equation (9.1.7):

$$e^a = 1 + a + \frac{a^2}{2} + \frac{a^3}{6} + \dots, \quad a := -f(s)\Delta s,$$

so the gap is  $e^a - (1 + a) = \frac{a^2}{2} + \mathcal{O}(a^3)$ . As soon as  $|f(s)|\Delta s$  is not tiny (i.e., the step size  $\Delta s$  is not sufficiently small), Euler’s linear update  $(1 + a)\mathbf{x}_s$  mis-scales the true factor  $e^a\mathbf{x}_s$  by a relative error of order  $a/2$ . This is purely linear distortion from the discretization. The exponential–Euler step avoids it by applying the exact linear multiplier, which is especially important when taking large steps.

### 9.1.3 Approaches of PF-ODE Numerical Solvers

Numerical solvers for diffusion models can be broadly grouped into two categories.

**Time Stepping Methods.** This class of methods discretizes the time interval  $[0, T]$  and approximates the PF-ODE using various numerical integration schemes designed for efficiency. We present the most fundamental, principled, and widely adopted approaches as representative examples:

**Denoising Diffusion Implicit Model (DDIM).** DDIM, introduced in Section 9.2 (with its update form already appearing in Section 4.1.4), is one of the earliest fast samplers for diffusion models. Originally proposed from a variational perspective, it introduces a non-Markovian forward family whose marginals match those of the original diffusion, thereby enabling a deterministic reverse process and flexible step skipping. From the ODE viewpoint, however, DDIM can be understood more directly: it corresponds to applying a single exponential-Euler step, i.e., approximating the diffusion model term inside the integral as constant, to the exponential-integration formula Equation (9.1.6), which yields the update in Equation (9.1.7).

**Diffusion Exponential Integrator Sampler (DEIS).** DEIS (Zhang and Chen, 2022), introduced in Section 9.3, was the first to exploit the semilinear structure of the PF-ODE by applying exponential integrators. The key idea is to treat the linear part exactly via an integrating factor and approximate only the nonlinear integral term. Unlike the Euler method, which assumes a constant integrand inside the exponential integrator formula, DEIS reuses the history of previously estimated points along the trajectory. Specifically, it fits a higher-order interpolation (a *Lagrange polynomial*) to the past evaluations and uses it to approximate the integral at the next step. Geometrically, this polynomial interpolation captures the curvature of the trajectory much more accurately than a constant approximation, enabling higher-order accuracy and improved stability for large step sizes.

This reuse of past evaluations to anchor the next update (so that each step requires only one new model call) is referred to as a *multistep method*. In contrast, a *single-step method* (e.g., DDIM) relies only on the most recent state for the next update. Such methods are simpler but typically more costly to achieve high accuracy, since they require more function evaluations (or more steps) overall.

**The Diffusion Probabilistic Model (DPM)-Solver Family.** The DPM-Solver family, including DPM-Solver (Lu *et al.*, 2022b) (Section 9.4), DPM-Solver++ (Lu *et al.*, 2022c) (Section 9.5), and DPM-Solver-v3 (Zheng *et al.*, 2023) (Section 9.7), builds on the semilinear structure of the PF-ODE with a crucial time reparameterization, the *half-log signal-to-noise ratio (SNR)*:

$$\lambda_t := \frac{1}{2} \log \frac{\alpha_t^2}{\sigma_t^2} = \log \frac{\alpha_t}{\sigma_t}.$$

This change of variables transforms the nonlinear term into an exponentially weighted integral

$$\int_{\lambda_s}^{\lambda_t} e^{-\lambda} \hat{\epsilon}_{\phi^\times}(\hat{x}_\lambda, \lambda) d\lambda,$$

where  $\hat{\epsilon}_{\phi^\times}$  denotes the model expressed in the reparameterized time  $\lambda$  (details in Equation (9.4.4)). This representation makes higher-order approximations of the integral both more accurate.

DPM-Solver introduced higher-order solvers by using Taylor expansions in  $\lambda$ , tailored to the half-log SNR reparameterization, showing that few NFEs suffice for high-quality samples. DPM-Solver++ adapted the method to classifier-free guidance with  $\mathbf{x}$ -prediction for greater stability. DPM-Solver-v3 further

automated the choice of parameterization by casting it as an optimization problem that minimizes local error in a principled way.

**(Optional) Time Parallel Methods.** A complementary strategy accelerates sampling by parallelizing computations across different time intervals, rather than processing them strictly in sequence.

**ParaDiGMs.** Introduced in Section 9.8, this method (Shih *et al.*, 2023) reformulates the ODE solution as a fixed-point problem. This perspective allows integral terms to be evaluated in parallel, alleviating the sequential bottleneck of standard time-stepping solvers. Importantly, this approach is not limited to the exponential-integrator form; it applies equally to general PF-ODEs with nonlinear drift  $\mathbf{f}(\mathbf{x}, t)$ . Moreover, it is solver-agnostic: the fixed-point formulation wraps any time-stepping rule by replacing the integral with a weighted sum of model evaluations at selected times, so Euler-, DEIS-, or DPM-Solver-style updates can be used while their evaluations are performed in parallel.

**True Computational Cost (NFEs).** In practice, the wall-clock cost is dominated not by the number of discretization steps, but by how many times we must call the model network. We refer to this count as the *number of function evaluations (NFE)*. If a sampler performs  $m$  evaluations per step over  $N$  steps, the cost scales as

$$\text{NFE} = m N.$$

For example, first-order Euler or exponential-Euler schemes have  $m = 1$ , while single-step  $k$ th-order methods typically require  $m \geq k$  (e.g.,  $k$ th order of DPM-Solver). Multistep methods (e.g., DEIS, multistep version of DPM-Solver++) reuse past evaluations so that after a short warm-up phase the average  $m$  is close to 1. Classifier-free guidance effectively doubles the number of calls at each step. Thus, in practice, “faster” sampling means achieving a lower NFE, not simply taking fewer steps.

**A Remark on Using the Equivalent Form of the PF-ODE.** In the discussion below, we will use the results in Section 6.3, which support the interchangeable use of the equivalent parameterizations  $(f(t), g(t))$  and  $(\alpha_t, \sigma_t)$  of the perturbation kernel with  $\mathbf{x}_t | \mathbf{x}_0 \sim \mathcal{N}(\cdot; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$ , related via

$$f(t) = \frac{\alpha'_t}{\alpha_t}, \quad g^2(t) = \frac{d}{dt}(\sigma_t^2) - 2\frac{\alpha'_t}{\alpha_t}\sigma_t^2 = 2\sigma_t\sigma'_t - 2\frac{\alpha'_t}{\alpha_t}\sigma_t^2.$$

Under these relations, the PF-ODE can be written in several equivalent forms (cf. Equation (6.3.2)).

## 9.2 DDIM

In this section, we introduce one of the pioneering approaches for accelerating sampling in diffusion models: *Denoising Diffusion Implicit Models* (DDIM), which is also among the most widely used ODE-based solvers. Although its name suggests a variational origin, as demonstrated in Section 6.3.2 for  $(\mathbf{x}, \epsilon)$ -prediction, we will show that its practical update rule can also be interpreted as a straightforward application of the Euler method to approximate the integral in Equation (9.1.6). This ODE perspective not only provides a principled reinterpretation of DDIM, but also lays a foundation for designing more flexible and efficient fast samplers.

The original variational derivation of DDIM will be revisited in Section 9.2.3. In Section 9.2.4, we establish a clear correspondence between the DDIM update rule and conditional flow matching, showing that the DDIM dynamics can be interpreted as the flow learned by CFM.

### 9.2.1 Interpreting DDIM as an ODE Solver

Let  $s > t$  denote two discrete time steps, with  $s$  being the starting time and  $t$  the target time for the update. To approximate the integral in Equation (9.1.6), a natural choice is to fix the integrand at  $s$  (the start of the step), assuming that

$$\epsilon_{\phi^\times}(\mathbf{x}_\tau, \tau) \approx \epsilon_{\phi^\times}(\mathbf{x}_s, s), \quad \text{for all } \tau \in [t, s].$$

This assumption leads to an Euler update approximation (see also Equation (9.1.7)), which gives rise to the following update rule:

$$\tilde{\mathbf{x}}_t = \mathcal{E}(s \rightarrow t)\tilde{\mathbf{x}}_s + \left( \frac{1}{2} \int_s^t \frac{g^2(\tau)}{\sigma_\tau} \mathcal{E}(\tau \rightarrow t) d\tau \right) \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_s, s), \quad (9.2.1)$$

for an initial point  $\tilde{\mathbf{x}}_s$ . Here, the integral becomes analytically tractable, resulting in the following practical and efficient DDIM update formula:

#### Proposition 9.2.1: DDIM = Euler Method (Exponential Euler)

The update rule in Equation (9.2.1), derived by applying the Euler method to the exponential integrator form in Equation (9.1.6), yields the following DDIM update:

$$\tilde{\mathbf{x}}_t = \frac{\alpha_t}{\alpha_s} \tilde{\mathbf{x}}_s - \alpha_t \left( \frac{\sigma_s}{\alpha_s} - \frac{\sigma_t}{\alpha_t} \right) \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_s, s). \quad (9.2.2)$$

**Proof for Proposition.**

We use Equation (4.4.2) that

$$f(t) = \frac{\alpha'_t}{\alpha_t}, \quad g^2(t) = \frac{d}{dt}(\sigma_t^2) - 2\frac{\alpha'_t}{\alpha_t}\sigma_t^2 = 2\sigma_t\sigma'_t - 2\frac{\alpha'_t}{\alpha_t}\sigma_t^2.$$

With this, we obtain

$$\mathcal{E}(s \rightarrow t) = e^{\int_s^t f(u) du} = e^{\log \alpha_u|_{u=s}^{u=t}} = \frac{\alpha_t}{\alpha_s}.$$

So

$$\begin{aligned} \int_s^t \frac{g^2(\tau)}{2\sigma_\tau} e^{\int_\tau^t f(u) du} d\tau &= \int_s^t \frac{g^2(\tau)}{2\sigma_\tau} \frac{\alpha_t}{\alpha_\tau} d\tau \\ &= \alpha_t \int_s^t \frac{1}{2\sigma_\tau \alpha_\tau} \left( \frac{d\sigma_\tau^2}{d\tau} - 2 \frac{d \log \alpha_\tau}{d\tau} \sigma_\tau^2 \right) d\tau \\ &= \alpha_t \int_s^t \frac{d}{d\tau} \left( \frac{\sigma_\tau}{\alpha_\tau} \right) d\tau \\ &= -\alpha_t \left( \frac{\sigma_s}{\alpha_s} - \frac{\sigma_t}{\alpha_t} \right). \end{aligned}$$

This correspondence reveals that DDIM can be interpreted as a first-order Euler method applied to the exponential-integrator transformed semilinear PF-ODE.

### 9.2.2 Intuition Behind DDIM with Different Parameterizations

DDIM is one of the most widely used methods for accelerating diffusion sampling and usually may take in different parameterizations (see Equation (6.3.1)) other than  $\epsilon$ -prediction. In this subsection, we present reformulation under different parameterizations, with later on provide more intuitive interpretation of DDIM.

**DDIM in Different Parameterizations.** In practice, one uses a pre-trained diffusion model expressed in one of the standard parameterizations and substitutes the corresponding predictor for the oracle target in the DDIM discretization of the PF-ODE. For clarity, we state the oracle version below; the implementable version follows by the replacements

$$\epsilon_{\phi^\times} \approx \epsilon^*, \quad \mathbf{x}_{\phi^\times} \approx \mathbf{x}^*, \quad \mathbf{s}_{\phi^\times} \approx \mathbf{s}^*, \quad \mathbf{v}_{\phi^\times} \approx \mathbf{v}^*.$$

### Corollary 9.2.1: DDIM in Different Parametrizations

Let  $s > t$ . Starting from  $\tilde{\mathbf{x}}_s \sim p_s$  and ending at time  $t$ , the DDIM update in different parametrizations are as:

$$\begin{aligned}
\tilde{\mathbf{x}}_t &= \frac{\alpha_t}{\alpha_s} \tilde{\mathbf{x}}_s + \alpha_t \left( \frac{\sigma_t}{\alpha_t} - \frac{\sigma_s}{\alpha_s} \right) \boldsymbol{\epsilon}^*(\tilde{\mathbf{x}}_s, s) \\
&= \frac{\sigma_t}{\sigma_s} \tilde{\mathbf{x}}_s + \alpha_s \left( \frac{\alpha_t}{\alpha_s} - \frac{\sigma_t}{\sigma_s} \right) \mathbf{x}^*(\tilde{\mathbf{x}}_s, s) \\
&= \frac{\alpha_t}{\alpha_s} \tilde{\mathbf{x}}_s + \sigma_s^2 \left( \frac{\alpha_t}{\alpha_s} - \frac{\sigma_t}{\sigma_s} \right) \mathbf{s}^*(\tilde{\mathbf{x}}_s, s) \\
&= \alpha_t \underbrace{\mathbf{x}^*(\tilde{\mathbf{x}}_s, s)}_{\approx \mathbf{x}_{\phi}^* \atop \text{estimated clean}} + \sigma_t \underbrace{\boldsymbol{\epsilon}^*(\tilde{\mathbf{x}}_s, s)}_{\approx \boldsymbol{\epsilon}_{\phi}^* \atop \text{estimated noise}} .
\end{aligned} \tag{9.2.3}$$

The last identity in Equation (9.2.3) gives a clear view of DDIM: starting from  $\tilde{\mathbf{x}}_s \sim p_s$ , the (estimated) clean part  $\mathbf{x}^*(\tilde{\mathbf{x}}_s, s)$  and (estimated) noise part  $\boldsymbol{\epsilon}^*(\tilde{\mathbf{x}}_s, s)$  act as interpolation endpoints that reconstruct a  $\tilde{\mathbf{x}}_t \sim p_t$  with coefficients  $(\alpha_t, \sigma_t)$ .

Indeed, DDIM can be viewed as an *direct* Euler discretization of the  $\mathbf{v}$ -parametrized PF-ODE without applying exponential integrators. From Proposition 6.3.2, the PF-ODE also takes the following form of  $\mathbf{v}$ -prediction:

$$\frac{d\mathbf{x}(\tau)}{d\tau} = \alpha'_\tau \mathbf{x}^*(\mathbf{x}(\tau), \tau) + \sigma'_\tau \boldsymbol{\epsilon}^*(\mathbf{x}(\tau), \tau), \quad \tau \in [t, s].$$

Starting at  $\tilde{\mathbf{x}}_s$  and integrating over  $[t, s]$ , Euler's method freezes the predictors at the right endpoint:

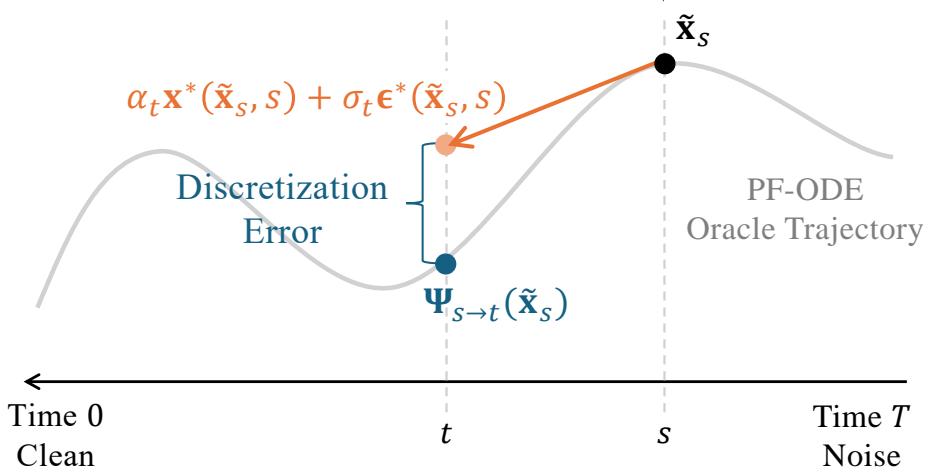
$$\mathbf{x}^*(\mathbf{x}(\tau), \tau) \approx \mathbf{x}^*(\tilde{\mathbf{x}}_s, s), \quad \boldsymbol{\epsilon}^*(\mathbf{x}(\tau), \tau) \approx \boldsymbol{\epsilon}^*(\tilde{\mathbf{x}}_s, s),$$

for all  $\tau \in [t, s]$ . This gives

$$\begin{aligned}
\tilde{\mathbf{x}}_t &= \tilde{\mathbf{x}}_s + \int_s^t (\alpha'_\tau \mathbf{x}^* + \sigma'_\tau \boldsymbol{\epsilon}^*) d\tau \\
&\approx \tilde{\mathbf{x}}_s + (\alpha_t - \alpha_s) \mathbf{x}^*(\tilde{\mathbf{x}}_s, s) + (\sigma_t - \sigma_s) \boldsymbol{\epsilon}^*(\tilde{\mathbf{x}}_s, s) \\
&= \alpha_t \mathbf{x}^*(\tilde{\mathbf{x}}_s, s) + \sigma_t \boldsymbol{\epsilon}^*(\tilde{\mathbf{x}}_s, s),
\end{aligned}$$

where the last identity follows directly from Equation (6.3.1). The derived formula above exactly matches the final identity in the DDIM update (Equation (9.2.3)). See Equation (9.2.3) for illustration.

With velocity prediction, the linear term  $f(t)\mathbf{x}$  in the PF-ODE is absorbed into the target  $\mathbf{v}^*(\mathbf{x}(t), t) = \alpha'_t \mathbf{x}_0 + \sigma'_t \boldsymbol{\epsilon}$ . By the Fundamental Theorem of



**Figure 9.1: Illustration of DDIM as an Euler discretization of the PF-ODE.** Starting from a state  $\tilde{\mathbf{x}}_s$  at time  $s$ , the oracle PF-ODE trajectory (gray curve) deterministically evolves to  $\Psi_{s \rightarrow t}(\tilde{\mathbf{x}}_s)$  at time  $t$ . In contrast, the DDIM update (orange) directly maps  $\tilde{\mathbf{x}}_s$  to  $\alpha_t \mathbf{x}^*(\tilde{\mathbf{x}}_s, s) + \sigma_t \epsilon^*(\tilde{\mathbf{x}}_s, s)$ . The discrepancy between this Euler step and the true PF-ODE trajectory introduces a discretization error, shown in blue. If  $t$  is far from  $s$ , the discrepancy can become large, leading to degraded generation quality.

Calculus, the integrals  $\int_s^t \alpha'_\tau d\tau$  and  $\int_s^t \sigma'_\tau d\tau$  simplify to  $(\alpha_t - \alpha_s)$  and  $(\sigma_t - \sigma_s)$ , so a single Euler step already yields the closed-form DDIM update:

$$\tilde{\mathbf{x}}_t = \alpha_t \tilde{\mathbf{x}}^*(\tilde{\mathbf{x}}_s, s) + \sigma_t \tilde{\epsilon}^*(\tilde{\mathbf{x}}_s, s).$$

That is, with v-prediction, there is no separate linear term to isolate in the PF-ODE drift, so the plain Euler update naturally coincides with the DDIM formulation. In contrast, under the  $\epsilon$ -,  $\mathbf{x}$ -, or  $\mathbf{s}$ -prediction parameterizations, the PF-ODE drift can be decomposed into a *semilinear* form consisting of a linear term and a nonlinear correction, which fits the general template given in Equation (9.1.5). A naïve Euler step then only *approximates* the linear term instead of computing it exactly (see the argument in Equation (9.1.8)). DDIM, on the other hand, corresponds to an *exponential–Euler* (integrating-factor) step that handles this linear component analytically. Therefore, v-prediction leads to the simplest and most direct Euler integration, whereas the other parameterizations require the exponential–Euler form to achieve the same DDIM behavior.

The above discussion also echoes the arguments presented in Section 6.3.4 and leads to the following conclusion:

**Observation 9.2.1: (Exponential) Euler and DDIM Updates**

Given the same schedulers  $(\alpha_t, \sigma_t)$ ,

$\mathbf{v}$ -prediction: Euler = DDIM,

$\epsilon$ -,  $\mathbf{x}$ -, or  $\mathbf{s}$ -prediction: exp-Euler = DDIM  $\neq$  plain Euler,

where, in the  $\epsilon$ -,  $\mathbf{x}$ -, or  $\mathbf{s}$ -prediction cases, the plain Euler step is not equivalent to DDIM, since the linear term is only approximated and may lead to reduced stability.

**Illustrative Example of DDIM Under Different Parameterizations.** We illustrate with a simple example using oracle replacements  $(\epsilon^*, \mathbf{x}^*, \nabla_{\mathbf{x}} \log p_t$ , and  $\mathbf{v}^*$ ), based on Equation (9.2.3). Assume the forward kernel  $\alpha_t = 1$  and  $\sigma_t = t$  (Karras *et al.*, 2022). The DDIM (exp-Euler) update

$$\tilde{\mathbf{x}}_t = \frac{\alpha_t}{\alpha_s} \tilde{\mathbf{x}}_s - \alpha_t \left( \frac{\sigma_s}{\alpha_s} - \frac{\sigma_t}{\alpha_t} \right) \epsilon^*(\tilde{\mathbf{x}}_s, s)$$

reduces to

$$\tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}_s - (s - t) \epsilon^*(\tilde{\mathbf{x}}_s, s).$$

Conceptually, subtracting the time gap  $(s - t)$  multiplied by the oracle noise estimate  $\epsilon^*(\tilde{\mathbf{x}}_s, s)$  pushes the current sample  $\tilde{\mathbf{x}}_s$  toward a cleaner estimate.

Using the  $\mathbf{x}$ -prediction oracle  $\mathbf{x}^*$ , which is related to the noise oracle by

$$\epsilon^*(\tilde{\mathbf{x}}_s, s) = \frac{\tilde{\mathbf{x}}_s - \mathbf{x}^*(\tilde{\mathbf{x}}_s, s)}{s},$$

we obtain

$$\tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}_s - \frac{s - t}{s} (\tilde{\mathbf{x}}_s - \mathbf{x}^*(\tilde{\mathbf{x}}_s, s)) = \frac{t}{s} \tilde{\mathbf{x}}_s + \left(1 - \frac{t}{s}\right) \mathbf{x}^*(\tilde{\mathbf{x}}_s, s). \quad (9.2.4)$$

Thus,  $\tilde{\mathbf{x}}_t$  is a convex combination of the current sample  $\tilde{\mathbf{x}}_s$  and the  $\mathbf{x}$ -prediction  $\mathbf{x}^*(\tilde{\mathbf{x}}_s, s)$ , which serves as the oracle estimate of the clean data. Moreover, we can rewrite this as

$$\tilde{\mathbf{x}}_t - \mathbf{x}^* = \frac{t}{s} (\tilde{\mathbf{x}}_s - \mathbf{x}^*), \quad t < s,$$

which shows that the denoising residual contracts by the factor  $t/s \in (0, 1)$  at each step (so no overshoot occurs when  $t < s$ ).

Using the score oracle, related to the noise oracle by

$$\epsilon^*(\tilde{\mathbf{x}}_s, s) = -\sigma_s \nabla_{\mathbf{x}} \log p_s(\tilde{\mathbf{x}}_s),$$

the DDIM (exp–Euler) update becomes

$$\tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}_s + (s - t) s \nabla_{\mathbf{x}} \log p_s(\tilde{\mathbf{x}}_s).$$

This moves  $\tilde{\mathbf{x}}_s$  uphill along the score field (toward higher likelihood regions), with step size proportional to the time gap  $(s - t)$  and the noise scale  $s$ .

Finally, using the velocity oracle with  $\mathbf{v}^*(\tilde{\mathbf{x}}_s, s) = -\epsilon^*(\tilde{\mathbf{x}}_s, s)$ , the DDIM update can be written as

$$\tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}_s + (t - s) \mathbf{v}^*(\tilde{\mathbf{x}}_s, s),$$

so the secant slope satisfies the finite-difference identity

$$\frac{\tilde{\mathbf{x}}_t - \tilde{\mathbf{x}}_s}{t - s} = \mathbf{v}^*(\tilde{\mathbf{x}}_s, s).$$

Intuitively, this means the update is a straight-line step following the local ODE drift.

**Challenge of DDIM.** However, the first-order Euler discretization has global error  $\mathcal{O}(h)$ , so accuracy degrades as the maximum step size  $h := \max_i |t_i - t_{i-1}|$  grows. To improve accuracy, the literature develops higher-order schemes that raise the global order to  $\mathcal{O}(h^k)$  ( $k \geq 2$ ) through richer local approximations. With suitable timestep allocation, these methods may achieve a target quality in fewer steps. It is important to note, however, that higher order alone does not guarantee fewer steps or lower wall-clock cost, since each step may require multiple model evaluations. In practice, the true measure of efficiency is the number of function evaluations,  $\text{NFE} = m N$ , and “faster” means reaching the desired quality with a smaller NFE, not merely fewer steps.

### 9.2.3 (Optional) A Variational Perspective on DDIM

Indeed, the motivation for DDIM comes from revisiting DDPM through its variational perspective. In DDPM, the reverse process is tied to a particular Markovian forward transition kernel  $p(\mathbf{x}_t | \mathbf{x}_{t-\Delta t})$ , which enforces small step sizes in order to approximate the multi-step posterior correctly. DDIM departs from this restriction by observing that the training objective depends only on the marginal perturbations  $p_t(\mathbf{x}_t | \mathbf{x}_0)$ , not on the specific forward transition. This insight allows one to construct reverse dynamics directly from the marginals, so intermediate steps can be skipped while marginal consistency is preserved. Because the transition is defined to map  $p_s(\mathbf{x}_s | \mathbf{x}_0)$  to  $p_t(\mathbf{x}_t | \mathbf{x}_0)$  for any  $t < s$ , we may use a coarse time grid with far fewer updates, which reduces the number of model evaluations and yields fast few-step sampling.

**Revisiting DDPM’s Variational View.** In DDPM, training fixes a family of marginal perturbation kernels  $p_t(\mathbf{x}_t|\mathbf{x}_0)$  and optimizes a surrogate objective that depends only on these marginals. At sampling time, however, the reverse conditional is the Bayesian posterior under the one-step forward kernel:

$$p(\mathbf{x}_{t-\Delta t}|\mathbf{x}_t, \mathbf{x}_0) = \frac{p(\mathbf{x}_t|\mathbf{x}_{t-\Delta t}) p_{t-\Delta t}(\mathbf{x}_{t-\Delta t}|\mathbf{x}_0)}{p_t(\mathbf{x}_t|\mathbf{x}_0)}.$$

This ties the reverse update to the *particular* forward transition  $p(\mathbf{x}_t|\mathbf{x}_{t-\Delta t})$ . If one tries to skip steps by enlarging  $\Delta t$  while reusing the same one-step kernel, this no longer matches the true multi-step posterior and typically degrades the marginals.

**Original DDIM Motivation.** DDIM observes that the training objective constrains only the marginals  $p_t(\mathbf{x}_t|\mathbf{x}_0)$ , not the intermediate reverse transitions. Hence, one may *specify* a family of reverse conditionals  $\pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)$  for any  $t < s$  that are *one-step marginally consistent*<sup>2</sup>:

---

<sup>2</sup>If we choose the “user-defined” reverse transition kernel  $\pi$  in Equation (9.2.5) to be exactly the same as the “true” conditional distribution:  $\pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) = p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0)$ , then the marginal consistency condition

$$\int \pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) p_s(\mathbf{x}_s|\mathbf{x}_0) d\mathbf{x}_s = p_t(\mathbf{x}_t|\mathbf{x}_0)$$

is simply the consequence of *law of total probability* (also known as the *tower property*) for the conditional joint distribution:

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \int p(\mathbf{x}_t, \mathbf{x}_s|\mathbf{x}_0) d\mathbf{x}_s = \int p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) p_s(\mathbf{x}_s|\mathbf{x}_0) d\mathbf{x}_s.$$

Or equivalently, by explicitly expressing the Bayesian posterior as

$$p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) = \frac{p(\mathbf{x}_s|\mathbf{x}_t, \mathbf{x}_0) p_t(\mathbf{x}_t|\mathbf{x}_0)}{p_s(\mathbf{x}_s|\mathbf{x}_0)},$$

then multiplying by  $p_s(\mathbf{x}_s|\mathbf{x}_0)$  and marginalizing over  $\mathbf{x}_s$ , we recover

$$\int p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) p_s(\mathbf{x}_s|\mathbf{x}_0) d\mathbf{x}_s = p_t(\mathbf{x}_t|\mathbf{x}_0),$$

which is exactly the same marginal-consistency condition.

In the Markov forward case, one further has  $p(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) = p(\mathbf{x}_t|\mathbf{x}_s)$ , reducing the following expression:

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \int p(\mathbf{x}_t|\mathbf{x}_s) p_s(\mathbf{x}_s|\mathbf{x}_0) d\mathbf{x}_s.$$

$$\int \pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0) p_s(\mathbf{x}_s | \mathbf{x}_0) d\mathbf{x}_s = p_t(\mathbf{x}_t | \mathbf{x}_0). \quad (9.2.5)$$

This construction removes any dependence on the forward one-step kernel  $p(\mathbf{x}_t | \mathbf{x}_{t-\Delta t})$  and legitimizes coarse (skipped) time steps.

**Derivation of Discrete-Time DDIM.** Consider the general forward perturbation:

$$p_t(\mathbf{x}_t | \mathbf{x}_0) := \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}),$$

where  $\mathbf{x}_0 \sim p_{\text{data}}$ .

DDIM does not require the reverse update to coincide with the Bayesian posterior tied to the one-step forward kernel. It suffices to *choose* a reverse conditional that preserves the marginals. Concretely, for any  $t < s$  we posit the Gaussian family

$$\pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; a_{t,s} \mathbf{x}_0 + b_{t,s} \mathbf{x}_s, c_{t,s}^2 \mathbf{I}), \quad (9.2.6)$$

with coefficients  $(a_{t,s}, b_{t,s}, c_{t,s})$  to be determined by the marginal-consistency constraint Equation (9.2.5). Since all involved kernels are Gaussian, sampling  $\mathbf{x}_s | \mathbf{x}_0 = \alpha_s \mathbf{x}_0 + \sigma_s \epsilon'$  and then  $\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0$  from Equation (9.2.6) yields

$$\begin{aligned} \mathbf{x}_t &= a_{t,s} \mathbf{x}_0 + b_{t,s} \mathbf{x}_s + c_{t,s} \epsilon \\ &= a_{t,s} \mathbf{x}_0 + b_{t,s} (\alpha_s \mathbf{x}_0 + \sigma_s \epsilon') + c_{t,s} \epsilon \\ &= (a_{t,s} + b_{t,s} \alpha_s) \mathbf{x}_0 + \sqrt{b_{t,s}^2 \sigma_s^2 + c_{t,s}^2} \epsilon'', \end{aligned} \quad (9.2.7)$$

where  $\epsilon, \epsilon', \epsilon'' \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  are independent (Gaussian-sum property). On the other hand,

$$\mathbf{x}_t \sim p_t(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I}).$$

Equating means and variances between this target and Equation (9.2.7) gives

$$\alpha_t = a_{t,s} + b_{t,s} \alpha_s, \quad \sigma_t^2 = b_{t,s}^2 \sigma_s^2 + c_{t,s}^2.$$

This system is underdetermined, so we treat  $c_{t,s}$  as a free parameter with the natural constraint  $0 \leq c_{t,s} \leq \sigma_t$ , and solve for  $a_{t,s}, b_{t,s}$ :

$$b_{t,s} = \frac{\sqrt{\sigma_t^2 - c_{t,s}^2}}{\sigma_s}, \quad a_{t,s} = \alpha_t - \alpha_s b_{t,s}. \quad (9.2.8)$$

Here, we take the nonnegative root for  $b_{t,s}$  without loss of generality.

Substituting Equation (9.2.8) into Equation (9.2.6) yields

$$\pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \underbrace{\alpha_t \mathbf{x}_0 + \frac{\sqrt{\sigma_t^2 - c_{t,s}^2}}{\sigma_s} (\mathbf{x}_s - \alpha_s \mathbf{x}_0)}_{\text{mean}}, c_{t,s}^2 \mathbf{I}\right). \quad (9.2.9)$$

Equivalently, the mean in Equation (9.2.9) expands to

$$\left(\alpha_t - \alpha_s \frac{\sqrt{\sigma_t^2 - c_{t,s}^2}}{\sigma_s}\right) \mathbf{x}_0 + \left(\frac{\sqrt{\sigma_t^2 - c_{t,s}^2}}{\sigma_s}\right) \mathbf{x}_s.$$

### Lemma 9.2.2: DDIM Coefficients

Let  $\pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0)$  be given by Equation (9.2.6). If the marginal-consistency condition Equation (9.2.5) holds, then the coefficients are exactly those in Equation (9.2.8), with  $0 \leq c_{t,s} \leq \sigma_t$ .

#### Remark.

1. In DDIM we *choose* the reverse kernel  $\pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0)$  to satisfy the marginal-consistency constraint, and in general

$$\pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0) \neq p(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0),$$

where  $p(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0)$  is the Bayesian posterior associated with a particular forward one-step kernel. By Bayes' rule,

$$p(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0) \propto p(\mathbf{x}_s | \mathbf{x}_t) p_t(\mathbf{x}_t | \mathbf{x}_0),$$

and this posterior is *not* required for specifying  $\pi$  or for training.

2. Only in the special case where the variance parameter is chosen to match the DDPM posterior variance (the  $\eta = 1$  setting in Equation (9.2.10)) do we have  $\pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0) = p(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0)$ ; otherwise  $\pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0) \neq p(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0)$ .
3. Without imposing a Markov constraint, in general  $p(\mathbf{x}_s | \mathbf{x}_t, \mathbf{x}_0) \neq p(\mathbf{x}_s | \mathbf{x}_t)$ . The equality  $p(\mathbf{x}_s | \mathbf{x}_t, \mathbf{x}_0) = p(\mathbf{x}_s | \mathbf{x}_t)$  is tied to a particular Markov forward model, which DDIM does not assume for its reverse construction.

The forward marginals  $\{p_t(\mathbf{x}_t | \mathbf{x}_0)\}_t$  do not uniquely determine the reverse conditional transitions. There exist infinitely many kernels  $\pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0)$  that

satisfy Equation (9.2.5), any of which can be freely specified. The parameter  $c_{t,s}$  indexes this family and controls the amount of noise injected at each reverse step  $s \rightarrow t$ . Below, we introduce this family of DDIM solvers.

**DDIM Sampler (Step  $s \rightarrow t$ ).** The DDIM sampler follows from the chosen reverse kernel  $\pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0)$  in Equation (9.2.9) by replacing  $\mathbf{x}_0$  with a predictor from a pre-trained model. Using the  $\epsilon$ -prediction network  $\epsilon_{\phi^\times}$  (plug-and-play, no retraining), we set

$$\mathbf{x}_{\phi^\times}(\mathbf{x}_s, s) := \frac{\mathbf{x}_s - \sigma_s \epsilon_{\phi^\times}(\mathbf{x}_s, s)}{\alpha_s}, \quad p_{\phi^\times}(\mathbf{x}_t | \mathbf{x}_s) := \pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_{\phi^\times}(\mathbf{x}_s, s)).$$

Substituting  $\mathbf{x}_{\phi^\times}$  into Equation (9.2.9) yields the update

$$\mathbf{x}_t = \frac{\alpha_t}{\alpha_s} \mathbf{x}_s + \left( \sqrt{\sigma_t^2 - c_{t,s}^2} - \frac{\alpha_t}{\alpha_s} \sigma_s \right) \epsilon_{\phi^\times}(\mathbf{x}_s, s) + c_{t,s} \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

where  $c_{t,s} \in [0, \sigma_t]$  controls stochasticity.

For notational convenience define the forward factors

$$\alpha_{t|s} := \frac{\alpha_t}{\alpha_s}, \quad \sigma_{t|s}^2 := \sigma_t^2 - \alpha_{t|s}^2 \sigma_s^2,$$

so that  $p(\mathbf{x}_t | \mathbf{x}_s) = \mathcal{N}(\alpha_{t|s} \mathbf{x}_s, \sigma_{t|s}^2 \mathbf{I})$ . Then the sampler can be written as

$$\mathbf{x}_t = \alpha_{t|s} \mathbf{x}_s + \left( \sqrt{\sigma_t^2 - c_{t,s}^2} - \alpha_{t|s} \sigma_s \right) \epsilon_{\phi^\times}(\mathbf{x}_s, s) + c_{t,s} \epsilon_t.$$

By varying  $c_{t,s}$ , one obtains a family of samplers that share the same pre-trained diffusion model and do not require retraining:

- **DDPM Step (Posterior Variance):**  $c_{t,s} = \frac{\sigma_s}{\sigma_t} \sigma_{t|s}$  makes  $\pi(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0)$  equal to the Bayesian posterior  $p(\mathbf{x}_t | \mathbf{x}_s, \mathbf{x}_0)$  induced by the one-step forward kernel. Replacing  $\mathbf{x}_0$  with its predictor yields the standard DDPM reverse update  $p_{\phi^\times}(\mathbf{x}_t | \mathbf{x}_s)$ , i.e., the Markov DDPM step with  $\alpha_t^2 + \sigma_t^2 = 1$  (Equation (2.2.14)).

- **Deterministic DDIM ( $\eta = 0$ ):**  $c_{t,s} = 0$  gives

$$\mathbf{x}_t = \alpha_{t|s} \mathbf{x}_s + (\sigma_t - \alpha_{t|s} \sigma_s) \epsilon_{\phi^\times}(\mathbf{x}_s, s),$$

which matches the ODE-view DDIM jump.

- **Interpolation:** Define

$$c_{t,s} = \eta \frac{\sigma_s}{\sigma_t} \sigma_{t|s}, \quad \eta \in [0, 1], \tag{9.2.10}$$

so that  $\eta$  smoothly interpolates between the stochastic DDPM update ( $\eta = 1$ ) and the deterministic DDIM update ( $\eta = 0$ ).

### 9.2.4 DDIM as Conditional Flow Matching

In this subsection, we will see that deterministic DDIM can be understood as searching for a conditional flow map that pushes  $p_s(\cdot|\mathbf{x}_0)$  forward to  $p_t(\cdot|\mathbf{x}_0)$ . The tangent of this conditional flow coincides with the conditional velocity used in conditional flow matching (CFM). Marginalizing this conditional velocity yields the PF-ODE drift, whose plain Euler discretization recovers the marginal DDIM update in  $\mathbf{v}$ -prediction.

We revisit the DDIM one-step conditional marginal-consistency identity (Equation (9.2.5))

$$\int \pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) p_s(\mathbf{x}_s|\mathbf{x}_0) d\mathbf{x}_s = p_t(\mathbf{x}_t|\mathbf{x}_0), \quad t < s,$$

i.e., if  $\mathbf{x}_s \sim p_s(\cdot|\mathbf{x}_0)$  then pushing  $\mathbf{x}_s$  forward by the chosen reverse kernel reproduces  $p_t(\cdot|\mathbf{x}_0)$ . When the reverse kernel is deterministic, it amounts to finding a conditional map  $\Psi_{s \rightarrow t}(\cdot|\mathbf{x}_0)$  that pushes  $p_s(\cdot|\mathbf{x}_0)$  forward to  $p_t(\cdot|\mathbf{x}_0)$ :

$$\pi(\mathbf{x}_t|\mathbf{x}_s, \mathbf{x}_0) = \delta(\mathbf{x}_t - \Psi_{s \rightarrow t}(\mathbf{x}_s|\mathbf{x}_0)), \quad (\Psi_{s \rightarrow t}(\cdot|\mathbf{x}_0))_\# p_s(\cdot|\mathbf{x}_0) = p_t(\cdot|\mathbf{x}_0).$$

Under the linear-Gaussian path  $\mathbf{x}_\tau = \alpha_\tau \mathbf{x}_0 + \sigma_\tau \epsilon$ , similar arguments as in Equations (9.2.6) and (9.2.7) lead to the *conditional map*

$$\Psi_{s \rightarrow t}(\mathbf{x}_s|\mathbf{x}_0) = \frac{\sigma_t}{\sigma_s} \mathbf{x}_s + \left( \alpha_t - \alpha_s \frac{\sigma_t}{\sigma_s} \right) \mathbf{x}_0,$$

whose instantaneous *conditional velocity* is

$$\mathbf{v}_t^*(\mathbf{x}|\mathbf{x}_0) = \partial_h|_{h=0} \Psi_{t \rightarrow t+h}(\mathbf{x}|\mathbf{x}_0) = \frac{\sigma'_t}{\sigma_t} \mathbf{x} + \left( \alpha'_t - \alpha_t \frac{\sigma'_t}{\sigma_t} \right) \mathbf{x}_0.$$

We refer to  $\Psi_{s \rightarrow t}(\cdot|\mathbf{x}_0)$  as the DDIM conditional map.

With  $p_t(\mathbf{x}|\mathbf{x}_0)$ , conditional flow matching fits the time-dependent field to this target velocity,

$$\mathcal{L}_{\text{CFM}}(\phi) = \mathbb{E}_{t, \mathbf{x}_0, \mathbf{x}_t \sim p_t(\cdot|\mathbf{x}_0)} \|\mathbf{v}_\phi(\mathbf{x}_t, t) - \mathbf{v}_t^*(\mathbf{x}_t|\mathbf{x}_0)\|^2,$$

so the CFM regression target equals the conditional velocity of the DDIM conditional map.

#### Observation 9.2.2: Conditional Level

Along the conditional Gaussian path, the DDIM conditional map and the CFM target generate the same conditional flow  $\Psi_{s \rightarrow t}(\cdot|\mathbf{x}_0)$ .

Averaging the conditional velocity over the posterior of  $\mathbf{x}_0$  given  $\mathbf{x}_t = \mathbf{x}$  yields the marginal PF–ODE drift,

$$\mathbf{v}^*(\mathbf{x}, t) = \mathbb{E} [\mathbf{v}_t^*(\mathbf{x}|\mathbf{x}_0)|\mathbf{x}_t = \mathbf{x}],$$

which, under the linear–Gaussian scheduler, takes the separable predictor form

$$\mathbf{v}^*(\mathbf{x}, t) = \alpha'_t \mathbf{x}^*(\mathbf{x}, t) + \sigma'_t \boldsymbol{\epsilon}^*(\mathbf{x}, t), \quad \mathbf{x} = \alpha_t \mathbf{x}^*(\mathbf{x}, t) + \sigma_t \boldsymbol{\epsilon}^*(\mathbf{x}, t).$$

We have seen that the plain Euler step of the PF-ODE with this marginalized  $\mathbf{v}$ -prediction is exactly the DDIM update (the last identity in Equation (9.2.3)).

In short, DDIM is (i) a deterministic conditional transport whose tangent equals the CFM target, and (ii) after marginalizing that tangent, a Euler step of the PF–ODE whose step coincides with the DDIM update.

### 9.3 DEIS

In the exponential–integrator formula (Equation (9.1.6)),

$$\int_s^t \frac{g^2(\tau)}{2\sigma_\tau} \mathcal{E}(\tau \rightarrow t) \epsilon_{\phi^\times}(\mathbf{x}_\tau, \tau) d\tau,$$

the only unknown is the model output  $\epsilon_{\phi^\times}(\mathbf{x}_\tau, \tau)$ ; the schedule terms and the weight  $\mathcal{E}(\tau \rightarrow t)$  are known once  $(\alpha, \sigma, g)$  are fixed. DDIM (Euler’s method) approximates this integral by holding the model output constant:

$$\epsilon_{\phi^\times}(\mathbf{x}_\tau, \tau) \approx \epsilon_{\phi^\times}(\mathbf{x}_s, s), \quad \tau \in [t, s].$$

However, this is only first–order accurate and can fail when the model output changes quickly in time.

A natural question then arises: *can we make better use of the model evaluations already computed?* As in classical *multistep solvers*, instead of treating  $\epsilon_{\phi^\times}(\mathbf{x}_\tau, \tau)$  as constant (Euler), we can reuse previous outputs (anchors) to fit a simple curve in time. Because the weight  $\frac{g^2(\tau)}{2\sigma_\tau} \mathcal{E}(\tau \rightarrow t)$  is known, the integral can then be evaluated exactly for this fitted curve. In effect, the hard integral of an unknown function is replaced by the exact integral of an approximating curve defined by past model calls. This is precisely the principle behind *Diffusion Exponential Integrator Sampler* (DEIS) (Zhang and Chen, 2022).

For readers familiar with classical ODE solvers, DEIS can be viewed as an Adams–Bashforth scheme (Iserles, 2009) applied in the framework of exponential integrators for the semilinear PF-ODE (Equation (9.1.6)): the linear drift is treated exactly via the integrating factor, while the remaining nonlinear term is advanced using multistep polynomial extrapolation.

We begin in Section 9.3.1 by introducing how to construct a smooth curve that passes through a set of anchors. In Section 9.3.2, we then apply this interpolation technique to approximate the PF-ODE integral, leading to the DEIS algorithm. Finally, in Section 9.3.3, we show that DDIM arises as the special case of DEIS with a constant polynomial.

#### 9.3.1 Polynomial Extrapolation

**Anchor Interpolation for Simple Curves.** Assume we know the value of some time–varying quantity at a few recent times

$$(\tau_0, \mathbf{Y}_0), (\tau_1, \mathbf{Y}_1), \dots, (\tau_n, \mathbf{Y}_n), \quad \tau_0 < \tau_1 < \dots < \tau_n,$$

where each  $\mathbf{Y}_j$  may be vector-valued. The most natural way to get a simple curve that exactly matches these anchors is to use the lowest-degree polynomial that passes through them. The easiest way to enforce that is to multiply factors that vanish at the other nodes and then normalize so that the value at  $\tau_j$  becomes 1. Small cases are intuitive:

**Example:**

$n = 0$  (**Constant**): use the last value,

$$\mathbf{Y}(\tau) \equiv \mathbf{Y}_n.$$

$n = 1$  (**Line**): draw the straight line through the last two anchors,

$$\mathbf{Y}(\tau) = \frac{\tau - \tau_n}{\tau_{n-1} - \tau_n} \mathbf{Y}_{n-1} + \frac{\tau - \tau_{n-1}}{\tau_n - \tau_{n-1}} \mathbf{Y}_n.$$

$n = 2$  (**Quadratic; Parabola**): pass a quadratic curve through the last three anchors. For example, if the anchors are

$$(\tau_{n-2}, \mathbf{Y}_{n-2}), \quad (\tau_{n-1}, \mathbf{Y}_{n-1}), \quad (\tau_n, \mathbf{Y}_n),$$

the quadratic interpolant is

$$\mathbf{Y}(\tau) = \mathbf{Y}_{n-2} \ell_{n-2}(\tau) + \mathbf{Y}_{n-1} \ell_{n-1}(\tau) + \mathbf{Y}_n \ell_n(\tau),$$

where the Lagrange basis functions are

$$\begin{aligned}\ell_{n-2}(\tau) &= \frac{(\tau - \tau_{n-1})(\tau - \tau_n)}{(\tau_{n-2} - \tau_{n-1})(\tau_{n-2} - \tau_n)}, \\ \ell_{n-1}(\tau) &= \frac{(\tau - \tau_{n-2})(\tau - \tau_n)}{(\tau_{n-1} - \tau_{n-2})(\tau_{n-1} - \tau_n)}, \\ \ell_n(\tau) &= \frac{(\tau - \tau_{n-2})(\tau - \tau_{n-1})}{(\tau_n - \tau_{n-2})(\tau_n - \tau_{n-1})}.\end{aligned}$$

These satisfy the interpolation conditions

$$\ell_j(\tau_k) = \delta_{jk}, \quad \text{for } j, k \in \{n-2, n-1, n\}$$

and  $\ell_{n-2}(\tau) + \ell_{n-1}(\tau) + \ell_n(\tau) = 1$  for all  $\tau$ . This curve not only matches all three anchors but also bends to reflect the local curvature. ■

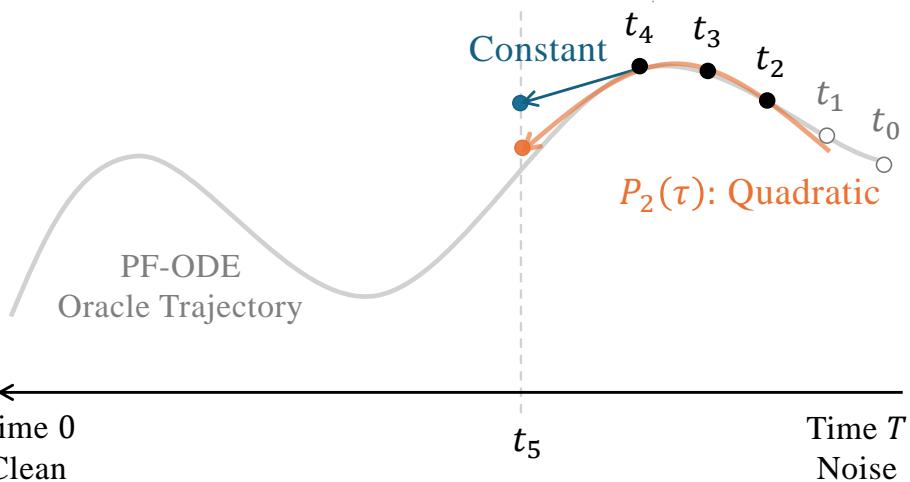
These cases are all part of a single recipe, known as the *Lagrange polynomial*. The idea is simple: we form the curve as a linear blend of the anchors with time-dependent weights,

$$\mathbf{Y}(\tau) = \sum_{j=0}^n \ell_j(\tau) \mathbf{Y}_j, \quad \ell_j(\tau_k) = \delta_{jk}, \quad \sum_{j=0}^n \ell_j(\tau) = 1.$$

Each  $\ell_j(\tau)$  acts like a “spotlight”, taking value 1 at its own anchor ( $\ell_j(\tau_j) = 1$ ) and 0 at the others ( $\ell_j(\tau_k) = 0, k \neq j$ ). In this sense, the Lagrange interpolant is just a linear combination of the anchors with basis functions  $\ell_j(\tau)$ .

### 9.3.2 DEIS: Lagrange Polynomial Approximation of the PF-ODE Integral

Let  $n \geq 0$  be the chosen polynomial degree. At step  $i$ , we approximate the unknown map  $\tau \mapsto \epsilon_{\phi^\times}(\mathbf{x}_\tau, \tau)$  over  $[t_{i-1}, t_i]$  by a degree- $n$  polynomial interpolant built from past model outputs, and substitute this approximation into the exponential–integrator update (Equation (9.1.6)) to obtain  $\tilde{\mathbf{x}}_{t_i}$ . By fitting a polynomial that bends to capture short-term trends of the trajectory, the update intuitively follows the curved behavior of the true ODE solution more closely, especially for larger step sizes.



**Figure 9.2: Illustration of DEIS as a multistep method.** With three past anchors at  $t_2, t_3, t_4$ , DEIS builds a quadratic curve through the model outputs and analytically integrates it to step from  $t_4$  to  $t_5$  (extrapolation). This higher-order update reduces discretization error compared to first-order methods like DDIM, which only use the value at  $t_4$  (constant approximation of the integral).

A degree- $n$  update needs  $n+1$  anchors. When they are available (*sufficient history*,  $i \geq n+1$ ), we use the full degree- $n$  scheme. In the early steps (*insufficient history*,  $i \leq n$ ), we apply the same construction at the highest feasible degree,  $i-1$ , and increase the degree as more anchors accumulate. Below we treat these two scenarios in turn.

**Case I:  $i = n + 1, \dots, M$  (Sufficient History).** Instead of relying solely on the most recent estimate  $\epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1})$ , DEIS reuses the last  $n+1$  model evaluations as anchors,

$$(\tau_j, \mathbf{Y}_j) := (t_{i-1-j}, \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j})), \quad j = 0, \dots, n.$$

as anchors. Viewing  $\tau \mapsto \epsilon_{\phi^\times}(\mathbf{x}_\tau, \tau)$  as a smooth function of time along the trajectory, we construct the degree- $n$  polynomial (Lagrange interpolant)

$$P_n(\tau) = \sum_{j=0}^n \underbrace{\left[ \prod_{\substack{k=0 \\ k \neq j}}^n \frac{\tau - t_{i-1-k}}{t_{i-1-j} - t_{i-1-k}} \right]}_{=: \ell_j^{(i)}(\tau)} \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j})$$

which by construction satisfies  $P_n(\tau_j) = \mathbf{Y}_j$  for each anchor:

$$P_n(\tau_j) = \mathbf{Y}_j = \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j}), \quad j = 0, \dots, n.$$

Each  $\ell_j^{(i)}$  satisfies

$$\ell_j^{(i)}(t_{i-1-m}) = \begin{cases} 1, & m = j, \\ 0, & m \neq j. \end{cases}$$

The Lagrange polynomial provides a smooth extrapolation over the new step:

$$\epsilon_{\phi^\times}(\mathbf{x}_\tau, \tau) \approx P_n(\tau) = \sum_{j=0}^n \ell_j^{(i)}(\tau) \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j}), \quad \tau \in [t_{i-1}, t_i].$$

We then substitute  $P_r(\tau)$  for  $\epsilon_{\phi^\times}(\mathbf{x}_\tau, \tau)$  in the exponential–integrator formula (Equation (9.1.6)):

$$\begin{aligned} & \int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{2\sigma_\tau} \mathcal{E}(\tau \rightarrow t_i) \epsilon_{\phi^\times}(\mathbf{x}_\tau, \tau) d\tau \\ & \approx \sum_{j=0}^r \underbrace{\int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{2\sigma_\tau} \mathcal{E}(\tau \rightarrow t_i) \ell_j^{(i)}(\tau) d\tau}_{=: C_{i,j}} \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j}). \end{aligned}$$

The weights  $C_{i,j}$  are given by

$$C_{i,j} := \frac{1}{2} \int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{\sigma_\tau} \mathcal{E}(\tau \rightarrow t_i) \ell_j^{(i)}(\tau) d\tau,$$

depending only on the schedule  $(\alpha_\tau, \sigma_\tau)$  and the grid  $\{t_i\}$ . Hence, they can be precomputed exactly in closed form once the steps are fixed.

Integrating the linear part exactly with  $\mathcal{E}(t_{i-1} \rightarrow t_i)$ , this leads to the *AB-DEIS-r* update rule<sup>3</sup>,

$$\tilde{\mathbf{x}}_{t_i} = \mathcal{E}(t_{i-1} \rightarrow t_i) \tilde{\mathbf{x}}_{t_{i-1}} + \sum_{j=0}^r C_{i,j} \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j}).$$

It yields a local truncation error of order  $r+1$  under standard smoothness assumptions.

**Case II:  $i = 1, \dots, n$  (Insufficient History).** For the initial steps, only  $i$  past points are available. We therefore set the degree to  $i-1$  and define

$$P_{i-1}(\tau) = \sum_{j=0}^{i-1} \ell_j^{(i)}(\tau) \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j}),$$

where  $\ell_j^{(i)}$  is the Lagrange basis of degree  $i-1$  built on the nodes at time  $\{t_{i-1}, t_{i-2}, \dots, t_0\}$ . This matches all available anchors and seamlessly transitions into the full-history formula once  $i \geq n+1$ .

This is a standard “warm start” in multistep solvers. When history is short, we fit the richest polynomial the data allow: with one anchor ( $i=1$ ), use degree 0 (constant); with two anchors ( $i=2$ ), use degree 1 (linear); with three anchors ( $i=3$ ), use degree 2 (quadratic); and so on, until we reach the target degree  $n$ . In effect, we gradually ramp up from a one-step forecast to a true  $(n+1)$ -step forecast as more history becomes available.

### Example: Special Cases of Lagrange Polynomials

**When  $r = 0$  (one anchor):**

$$P_0(\tau) = \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}).$$

This uses only the most recent value, so the approximation is flat in  $\tau$ . It corresponds to a left-endpoint of the integrand.

**When  $r = 1$  (two anchors):** the Lagrange polynomial is a linear map passing through the two pre-specified anchors.

$$P_1(\tau) = \underbrace{\frac{\tau - t_{i-2}}{t_{i-1} - t_{i-2}}}_{\ell_{i-1}(\tau)} \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}) + \underbrace{\frac{\tau - t_{i-1}}{t_{i-2} - t_{i-1}}}_{\ell_{i-2}(\tau)} \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-2}}, t_{i-2}).$$

Here  $\ell_{i-1}(\tau)$  and  $\ell_{i-2}(\tau)$  are the Lagrange basis weights. They satisfy the

---

<sup>3</sup>“AB” refers to the classical Adams–Bashforth family and exponential time–differencing multistep methods (Hochbruck and Ostermann, 2010).

interpolation (nodal) conditions  $P_1(t_{i-1}) = \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1})$  and  $P_1(t_{i-2}) = \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-2}}, t_{i-2})$ , and with  $\ell_{i-1}(\tau) + \ell_{i-2}(\tau) = 1$ . ■

**Summary of AB-DEIS- $n$  Update.** Combining the two cases, sufficient history and warm start (insufficient history), yields the *AB-DEIS- $n$*  update<sup>4</sup> where  $n$  is the polynomial degree (using up to  $n+1$  past evaluations) as follows:

$$\tilde{\mathbf{x}}_{t_i} = \mathcal{E}(t_{i-1} \rightarrow t_i) \tilde{\mathbf{x}}_{t_{i-1}} + \sum_{j=0}^{\min\{n, i-1\}} C_{i,j} \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1-j}}, t_{i-1-j}),$$

with coefficients

$$C_{i,j} := \frac{1}{2} \int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{\sigma_\tau} \mathcal{E}(\tau \rightarrow t_i) \left[ \prod_{\substack{k=0 \\ k \neq j}}^{\min\{n, i-1\}} \frac{\tau - t_{i-1-k}}{t_{i-1-j} - t_{i-1-k}} \right] d\tau.$$

When  $i \geq n+1$  (sufficient history),  $\min\{n, i-1\} = n$  and the step attains local truncation error  $\mathcal{O}(h^{n+1})$  under standard smoothness assumptions. During warm start ( $i \leq n$ ),  $\min\{n, i-1\} = i-1$  and the per-step order is  $\mathcal{O}(h^{\min\{n, i-1\}+1})$ , ramping up until full order is reached.

However, very large  $n$  often degrades performance due to interpolation ill-conditioning, noise amplification, and tighter stability constraints; small degrees (e.g.,  $n \in \{1, 2, 3\}$ ) usually provide the best accuracy–stability trade-off.

As we will see in the following subsection, the special case  $n=0$  reduces to exponential Euler/DDIM.

### 9.3.3 DDIM = AB-DEIS-0

We observe that when  $n = 0$  (i.e., constant polynomial), the coefficient simplifies to:

$$C_{i0} = \frac{1}{2} \int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{\sigma_\tau} \mathcal{E}(\tau \rightarrow t_i) d\tau.$$

Substituting into the update formula yields the zeroth-order AB-DEIS scheme:

$$\begin{aligned} \tilde{\mathbf{x}}_{t_i} &= \mathcal{E}(t_{i-1} \rightarrow t_i) \tilde{\mathbf{x}}_{t_{i-1}} + C_{i0} \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}) \\ &= e^{\int_{t_{i-1}}^{t_i} f(u) du} \tilde{\mathbf{x}}_{t_{i-1}} + \left( \int_{t_{i-1}}^{t_i} \frac{g^2(\tau)}{2\sigma_\tau} e^{\int_\tau^{t_i} f(u) du} d\tau \right) \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}). \end{aligned} \quad (9.3.1)$$

<sup>4</sup>“AB” refers to the Adams–Bashforth family of exponential time–differencing multistep methods (Hochbruck and Ostermann, 2010).

This is exactly the exponential–Euler step (constant-in-time  $\epsilon_{\phi^x}$  over  $[t_{i-1}, t_i]$ ), which coincides with the deterministic DDIM update. We state this correspondence formally below.

**Proposition 9.3.1: DDIM = AB-DEIS-0**

Equation (9.3.1) is identical to the DDIM update in Equation (9.2.2).

## 9.4 DPM-Solver

The DPM-Solver family, including DPM-Solver (Lu *et al.*, 2022b), DPM-Solver++ (Lu *et al.*, 2022c), and DPM-Solver-v3 (Zheng *et al.*, 2023), represents a major advance in solvers for the PF-ODE. The goal is simple: achieve similar sample quality with far fewer steps. In practice, these methods reduce the steps required by DDIM from more than 50 to about 10-15, which makes generation much more efficient. In addition, DPM-Solver++ and DPM-Solver-v3 are designed to handle classifier free guidance (CFG) (see Section 8.3) for conditional generation. In this section, we first explain the core DPM-Solver (Lu *et al.*, 2022b); its extensions appear in Section 9.5 and Section 9.7.

**High-Level Idea of DPM-Solver.** Like DEIS, DPM-Solver starts from the semilinear form of the PF-ODE and works in the  $\epsilon$ -prediction parameterization, using the exponential integrator (variation of constants) representation in Equation (A.1.2):

$$\frac{d\mathbf{x}_t}{dt} = \frac{\alpha'_t}{\alpha_t} \mathbf{x}_t - \sigma_t \left( \frac{\alpha'_t}{\alpha_t} - \frac{\sigma'_t}{\sigma_t} \right) \epsilon_{\phi^x}(\mathbf{x}_t, t). \quad (9.4.1)$$

The key idea is to reparameterize time by the half-log signal-to-noise ratio, so that the nonlinear term in the exponential integrator formula becomes an exponentially weighted integral. This representation admits low-cost Taylor expansions in  $\lambda$ , which naturally yield higher-order update rules. We will shortly provide an intuitive explanation for why this reparameterization is effective.

### 9.4.1 DPM-Solver’s Insight: Time Reparameterization via Log-SNR

On top of the semilinear structure, a key insight from of DPM-solver is that the standard time parameterization  $t$  is suboptimal for numerical integration in diffusion models. They instead propose reparameterizing time using the *half-log signal-to-noise ratio* (half-log SNR)

$$\lambda_t := \frac{1}{2} \log \frac{\alpha_t^2}{\sigma_t^2} = \log \frac{\alpha_t}{\sigma_t}, \quad (9.4.2)$$

following the log-SNR parameterization of VDM (Kingma *et al.*, 2021). This change-of-variables simplifies the nonlinear integrand, thereby enabling more tractable and accurate higher-order model estimation.

**Change-of-Variable to Log-SNR in PF-ODE.** We now reparametrize time using the half-log SNR,  $\lambda_t := \log(\alpha_t/\sigma_t)$ . For common noise schedules,  $\lambda_t$  is strictly decreasing in  $t$ . Under this assumption, it has an inverse function  $t_\lambda(\cdot)$  that maps  $\lambda$  to  $t$ , satisfying

$$t = t_\lambda(\lambda(t)).$$

We then change the subscripts of  $\mathbf{x}$  and  $\epsilon_{\phi^\times}$  from  $t$  to  $\lambda$ . A hat ( $\hat{\cdot}$ ) indicates that the quantity is expressed in  $\lambda$ . More precisely, we define:

$$\begin{aligned}\hat{\mathbf{x}}_\lambda &:= \mathbf{x}_{t_\lambda(\lambda)}, \\ \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) &:= \epsilon_{\phi^\times}(\mathbf{x}_{t_\lambda(\lambda)}, t_\lambda(\lambda)).\end{aligned}\tag{9.4.3}$$

With this change of variables from  $t$  to  $\lambda_t$ , the exact solution  $\tilde{\Psi}_{s \rightarrow t}$  of the PF-ODE in Equation (9.4.1) becomes:

#### Proposition 9.4.1: Exponentially Weighted Exact Solution

Given an initial value  $\mathbf{x}_s$  at time  $s > 0$ , the exact solution  $\tilde{\Psi}_{s \rightarrow t}(\mathbf{x}_s)$  at time  $t \in [0, s]$  of the PF-ODE can be re-expressed as:

$$\tilde{\Psi}_{s \rightarrow t}(\mathbf{x}_s) = \frac{\alpha_t}{\alpha_s} \mathbf{x}_s - \alpha_t \int_{\lambda_s}^{\lambda_t} e^{-\lambda} \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) d\lambda.\tag{9.4.4}$$

#### Proof for Proposition.

While one may directly apply the change of variables to Equation (9.4.1) to obtain the result, we provide an alternative derivation below for clarity and completeness. Using the relation  $g^2(t) = -2\sigma_t^2 \frac{d\lambda_t}{dt}$ , Equation (9.4.1) can be rewritten as:

$$\frac{d\mathbf{x}_t}{dt} = \frac{d \log \alpha_t}{dt} \mathbf{x}_t - \sigma_t \frac{d\lambda_t}{dt} \epsilon_{\phi^\times}(\mathbf{x}_t, t).$$

Applying the chain rule:

$$\frac{d\mathbf{x}_t}{dt} = \frac{d\hat{\mathbf{x}}_\lambda}{d\lambda} \frac{d\lambda_t}{dt} \quad \text{and} \quad \frac{d \log \alpha_t}{dt} = \frac{d \log \alpha_\lambda}{d\lambda} \frac{d\lambda_t}{dt},$$

the ODE in  $t$  is transformed into an ODE in  $\lambda$  as follows:

$$\begin{aligned}\frac{d\hat{\mathbf{x}}_\lambda}{d\lambda} &= \left( \frac{d\lambda_t}{dt} \right)^{-1} \frac{d\mathbf{x}_t}{dt} \\ &= \left( \frac{d\lambda_t}{dt} \right)^{-1} \left[ \frac{d \log \alpha_t}{dt} \mathbf{x}_t - \sigma_t \frac{d\lambda_t}{dt} \epsilon_{\phi^\times}(\mathbf{x}_t, t) \right] \\ &= \left( \frac{d\lambda_t}{dt} \right)^{-1} \left[ \frac{d \log \alpha_\lambda}{d\lambda} \frac{d\lambda_t}{dt} \hat{\mathbf{x}}_\lambda - \sigma_\lambda \frac{d\lambda_t}{dt} \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) \right] \\ &= \frac{d \log \alpha_\lambda}{d\lambda} \hat{\mathbf{x}}_\lambda - \sigma_\lambda \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda).\end{aligned}$$

Thus, the transformed ODE becomes Equation (9.4.5). We can then apply the same “Exponential Integrator (EI)” technique to Equation (9.4.5) to derive Equation (9.4.4).

In  $\lambda$ -time, the model appears inside an exponentially weighted integral,

$$\int_{\lambda_s}^{\lambda_t} e^{-\lambda} \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) d\lambda,$$

where the  $e^{-\lambda}$  factor produces closed-form coefficients and smooths the integrand, exactly what high-order local approximations require.

Equivalently, changing variables from  $t$  to  $\lambda$  transforms the PF-ODE into the differential form below (see the derivation in the previous proposition):

$$\frac{d\hat{\mathbf{x}}_\lambda}{d\lambda} = \frac{\alpha'_\lambda}{\alpha_\lambda} \hat{\mathbf{x}}_\lambda - \sigma_\lambda \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda). \quad (9.4.5)$$

**Intuition of Why Reparameterize Time?** For strictly monotone  $\lambda(t)$ , the first-order change of variables gives

$$\Delta t \approx \frac{\Delta\lambda}{|\lambda'(t)|}.$$

Thus, for fixed  $\Delta\lambda$ , the induced  $\Delta t$  is smaller where  $|\lambda'(t)|$  is larger (i.e. where  $\lambda$  changes rapidly with  $t$ ), and larger where  $|\lambda'(t)|$  is smaller. This reparameterization does not alter the PF-ODE solution path, only the speed:

$$\frac{d\hat{\mathbf{x}}_\lambda}{d\lambda} = \frac{1}{\lambda'(t)} \frac{d\mathbf{x}_t}{dt}.$$

Consequently, in regions with large  $|\lambda'(t)|$ , the  $\lambda$ -domain derivative is scaled by  $1/|\lambda'(t)|$ , often making the integrand smoother to approximate on a uniform  $\lambda$  grid. (The precise location of large  $|\lambda'(t)|$  depends on the chosen schedule.)

Conceptually, we may want to allocate more timesteps when the process gets closer to the complicated (data) distribution. Below are two simple schedules that illustrate this effect:

- $(\alpha_t, \sigma_t) = (1-t, t)$ : This corresponds to the FM scheduler. Then

$$\lambda(t) = \log \frac{1-t}{t}, \quad \lambda'(t) = -\frac{1}{t(1-t)}, \quad \Delta t \approx \Delta\lambda t(1-t).$$

Hence steps are tiny near both ends ( $t \rightarrow 0, 1$ ) and largest around mid-time.

- $(\alpha_t, \sigma_t) = (1, t)$ : This is the EDM scheduler (Karras *et al.*, 2022), introduced in Section D.6. If we take the independent variable directly as the noise level  $t = \sigma_t$ , then

$$\lambda(t) = \log \frac{1}{t}, \quad \lambda'(t) = -\frac{1}{t}, \quad \Delta t \approx \Delta \lambda t.$$

Uniform spacing in  $\lambda$  is geometric in  $t$ , or equivalently in the variance (many small steps at small  $t$ /high SNR, coarser at large  $t$ ).

### 9.4.2 Estimating the Integral with Taylor Expansion

DEIS fits the integrand by Lagrange interpolation across past evaluations. DPM-Solver instead uses a local Taylor expansion in  $\lambda$ : it is cheaper to evaluate, aligns with the smoothness induced by the  $\lambda$ -parametrization, and yields closed-form step coefficients. We present the details below.

From Equation (9.4.4), starting with the previous point  $\tilde{\mathbf{x}}_s$  at time  $s$ , the solution  $\tilde{\mathbf{x}}_t$  at time  $t$  is given by

$$\tilde{\mathbf{x}}_t = \frac{\alpha_t}{\alpha_s} \tilde{\mathbf{x}}_s - \alpha_t \int_{\lambda_s}^{\lambda_t} e^{-\lambda} \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) d\lambda. \quad (9.4.6)$$

Therefore, we are led to approximate integrals of the form:

$$\int_{\lambda_s}^{\lambda_t} e^{-\lambda} \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) d\lambda.$$

On the interval  $\lambda \in [\lambda_s, \lambda_{t_i}]$ , we approximate the integrand  $\hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda)$  in Equation (9.4.6) by a Taylor expansion with respect to  $\lambda$ . For  $n \geq 1$ , the  $(n-1)$ -th order Taylor expansion about  $\lambda_s$  is given by

$$\hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) = \sum_{k=0}^{n-1} \frac{(\lambda - \lambda_s)^k}{k!} \hat{\epsilon}_{\phi^\times}^{(k)}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}((\lambda - \lambda_s)^n),$$

where the  $k$ -th total derivative with respect to  $\lambda$  is denoted by

$$\hat{\epsilon}_{\phi^\times}^{(k)}(\hat{\mathbf{x}}_\lambda, \lambda) := \frac{d^k}{d\lambda^k} \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda).$$

Substituting this expansion into the integral in Equation (9.4.6) yields a closed-form approximation, which defines the  $n$ -th order solver, referred to as *DPM-Solver- $n$* .

Starting from the previous step estimation  $\tilde{\mathbf{x}}_s$ ,

$$\tilde{\mathbf{x}}_t = \frac{\alpha_t}{\alpha_s} \tilde{\mathbf{x}}_s - \alpha_t \sum_{k=0}^{n-1} \hat{\epsilon}_{\phi^\times}^{(k)}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) \mathcal{C}_k + \mathcal{O}(h^{n+1}), \quad (9.4.7)$$

Here, we denote  $h := \lambda_t - \lambda_s$ , and define:

$$\mathcal{C}_k := \int_{\lambda_s}^{\lambda_t} e^{-\lambda} \frac{(\lambda - \lambda_s)^k}{k!} d\lambda.$$

$\mathcal{C}_k$  can be precomputed analytically by applying integration by parts  $k$  times.

We note that the change of variables  $t \mapsto \lambda$  is used to smooth the integrand and derive coefficients, whereas the solver returns estimates  $\tilde{\mathbf{x}}_t$  on the  $t$ -grid.

Below, we illustrate DPM-Solver-1 as an example.

### Example: DPM-Solver-1

Consider  $n = 1$  (first order) for demonstration . Starting from the previous estimated point  $\tilde{\mathbf{x}}_s$ , Equation (9.4.7) simplifies to:

$$\begin{aligned} \tilde{\mathbf{x}}_t &= \frac{\alpha_t}{\alpha_s} \tilde{\mathbf{x}}_s - \alpha_t \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_s, s) \int_{\lambda_s}^{\lambda_t} e^{-\lambda} d\lambda + \mathcal{O}(h^2) \\ &= \frac{\alpha_t}{\alpha_s} \tilde{\mathbf{x}}_s - \sigma_t (e^h - 1) \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_s, s) + \mathcal{O}(h^2). \end{aligned} \quad (9.4.8)$$

The above formula is exactly the DDIM update; we prove the equivalence in Proposition 9.4.2. ■

DPM-Solver- $n$  with  $n \geq 2$  requires evaluating the  $k^{\text{th}}$ -derivative  $\hat{\epsilon}_{\phi^\times}^{(k)}(\hat{\mathbf{x}}_\lambda, \lambda)$  for  $k \leq n - 1$ . However, directly computing higher-order derivatives is computationally expensive in practice. Lu *et al.* (2022b) also propose efficient approximation methods for these derivatives, which will be detailed in the next subsection.

### 9.4.3 Implementation of DPM-Solver- $n$

**DPM-Solver- $n$  with  $n \geq 2$ .** In practice, implementing a higher-order DPM-Solver- $n$  entails the following:

- Precomputing the coefficients  $\mathcal{C}_k$ ;

- Approximating the  $k^{\text{th}}$  derivative  $\hat{\epsilon}_{\phi^\times}^{(k)}(\hat{\mathbf{x}}_\lambda, \lambda)$  for  $k \leq n - 1$  to circumvent the costly computation of exact higher-order derivatives—a challenge well-studied in the ODE literature (Hochbruck and Ostermann, 2005; Luan, 2021). One common strategy is finite difference approximation.

We now elaborate on the first two points.

**Precomputing  $C_k$ .** Let  $s$  and  $t$  denote the start and end times, respectively, and define  $h := \lambda_t - \lambda_s$ . Starting from  $\mathbf{x}_s$ , the analytical expansion of the exact solution to Equation (9.1.6) reads:

$$\mathbf{x}_t = \frac{\alpha_t}{\alpha_s} \mathbf{x}_s - \sigma_t \sum_{k=0}^{n-1} h^{k+1} \varphi_{k+1}(h) \hat{\epsilon}_{\phi^\times}^{(k)}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) + \mathcal{O}(h^{n+1}), \quad (9.4.9)$$

where each  $\varphi_{k+1}(\cdot)$  admits a closed-form. For  $k = 0, 1, 2$ , they are:

$$\varphi_1(h) = \frac{e^h - 1}{h}, \quad \varphi_2(h) = \frac{e^h - h - 1}{h^2}, \quad \varphi_3(h) = \frac{e^h - \frac{h^2}{2} - h - 1}{h^3}.$$

### Example: DPM-Solver-2/3 with Exact Derivatives

For  $n = 3$  and discrete time steps with  $h := \lambda_t - \lambda_s$ , the expansion becomes:

$$\begin{aligned} \mathbf{x}_t = & \frac{\alpha_t}{\alpha_s} \mathbf{x}_s - \sigma_t \left( e^h - 1 \right) \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) \\ & - \sigma_t \left( e^h - h - 1 \right) \hat{\epsilon}_{\phi^\times}^{(1)}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) \\ & - \sigma_t \left( e^h - \frac{h^2}{2} - h - 1 \right) \hat{\epsilon}_{\phi^\times}^{(2)}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) \\ & + \mathcal{O}(h^4). \end{aligned} \quad (9.4.10)$$

**Approximating  $\hat{\epsilon}_{\phi^\times}^{(k)}(\hat{\mathbf{x}}_\lambda, \lambda)$  for  $k \leq n - 1$ .** For  $n \geq 2$ , following the standard approach of single-step ODE solvers (Atkinson *et al.*, 2009), Lu *et al.* (2022b) introduce an intermediate timestep  $s^{\text{mid}}$  between  $s$  and  $t$  to approximate higher-order derivatives using function evaluations at  $s$  and  $s^{\text{mid}}$ . We illustrate this with the case of  $n = 2$ .

Let  $\gamma \in (0, 1]$  be a hyperparameter specifying an interpolation point within the log-SNR interval  $[\lambda_s, \lambda_t]$ . Given an estimate  $\hat{\mathbf{x}}_s$  at  $s$ , define

$$s^{\text{mid}} = t_\lambda (\lambda_s + \gamma h), \quad \text{where } h := \lambda_t - \lambda_s,$$

The intermediate estimate is given by:

$$\mathbf{x}^{\text{mid}} = \frac{\alpha_s^{\text{mid}}}{\alpha_s} \tilde{\mathbf{x}}_s - \sigma_{s^{\text{mid}}} (e^{\gamma h} - 1) \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_s, s).$$

This yields the following second-order approximation:

$$\begin{aligned} \tilde{\mathbf{x}}_t &= \frac{\alpha_t}{\alpha_s} \tilde{\mathbf{x}}_s - \sigma_t (e^h - 1) \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_s, s) \\ &\quad - \frac{\sigma_t}{\gamma h} (e^h - h - 1) (\epsilon_{\phi^\times}(\mathbf{x}^{\text{mid}}, s^{\text{mid}}) - \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_s, s)) + \mathcal{O}(h^3). \end{aligned} \quad (9.4.11)$$

With  $\gamma = \frac{1}{2}$ , the two-stage update in Algorithm 5 is equivalent to Equation (9.4.11) up to  $\mathcal{O}(h^3)$  (local truncation error).

---

**Algorithm 5** DPM-Solver-2 (with  $\gamma = \frac{1}{2}$ ).

---

**Input:** initial value  $\mathbf{x}_T$ , time steps  $\{t_i\}_{i=0}^M$ , model  $\epsilon_{\phi^\times}$

- 1:  $\tilde{\mathbf{x}}_{t_0} \leftarrow \mathbf{x}_T$
  - 2: **for**  $i \leftarrow 1$  to  $M$  **do**
  - 3:      $h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$
  - 4:      $s_i^{\text{mid}} \leftarrow t_\lambda \left( \frac{\lambda_{t_{i-1}} + \lambda_{t_i}}{2} \right)$
  - 5:      $\mathbf{x}_i^{\text{mid}} \leftarrow \frac{\alpha_{s_i^{\text{mid}}}}{\alpha_{t_{i-1}}} \tilde{\mathbf{x}}_{t_{i-1}} - \sigma_{s_i^{\text{mid}}} (e^{\frac{h_i}{2}} - 1) \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1})$
  - 6:      $\tilde{\mathbf{x}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}} \tilde{\mathbf{x}}_{t_{i-1}} - \sigma_{t_i} (e^{h_i} - 1) \epsilon_{\phi^\times}(\mathbf{x}_i^{\text{mid}}, s_i^{\text{mid}})$
  - 7: **end for**
  - 8: **return**  $\tilde{\mathbf{x}}_{t_M}$
- 

### Remark.

In Equation (9.4.11), the difference quotient

$$\epsilon_{\phi^\times}^{(1)}(\hat{\mathbf{x}}_{\lambda_s}, \lambda_s) \approx \frac{\epsilon_{\phi^\times}(\mathbf{x}^{\text{mid}}, s^{\text{mid}}) - \epsilon_{\phi^\times}(\tilde{\mathbf{x}}_s, s)}{\gamma h}$$

approximates the total  $\lambda$ -derivative of the model along the trajectory. This approximation is accurate up to  $\mathcal{O}(h)$ , and in Equation (9.4.11) it is multiplied by the exact  $\varphi_2$  coefficient  $e^h - h - 1 = \mathcal{O}(h^2)$ . Hence, the resulting contribution is only  $\mathcal{O}(h^3)$ , so the overall scheme achieves second-order accuracy for any  $\gamma \in (0, 1]$ .

Each step requires exactly two model evaluations: one at  $(\tilde{\mathbf{x}}_s, s)$  and one at the predicted midpoint  $(\mathbf{x}^{\text{mid}}, s^{\text{mid}})$ . The interpolation parameter  $\gamma$  does not affect the order of accuracy, but it changes the error constant:

setting  $\gamma = \frac{1}{2}$  symmetrizes the stencil and typically minimizes the constant, which is why the midpoint version is preferred in practice.

For higher-order DPM-Solver- $n$  with  $n \geq 3$ , a similar approach is employed, utilizing intermediate timesteps to approximate higher-order derivatives in a finite difference manner. The detailed methodology is deferred to the original DPM paper.

For readers familiar with numerical ODE solvers, DPM-SOLVER can be viewed as a one-step *exponential integrator* for the semilinear PF-ODE, combined with a change of time variable to the (half-)log-SNR. Its second- and third-order variants are exponential Runge–Kutta-type schemes that use a few staged model evaluations within each step.

**Implementation Detail: Selection of Sampling Timesteps.** To perform sampling, solvers must first predefine a sequence of timesteps  $\{t_i\}_{i=0}^M$ . Lu *et al.* (2022b) propose selecting these steps based on *uniform spacing in log-SNR time*  $\lambda_t$ , where

$$\lambda_{t_i} = \lambda_T + \frac{i}{M}(\lambda_0 - \lambda_T), \quad i = 0, \dots, M.$$

This differs from earlier approaches (Ho *et al.*, 2020; Song *et al.*, 2020c) that use uniform spacing directly in the physical time variable  $t$ . Empirically, DPM-Solver achieves high-quality samples even with very few steps when using uniform  $\lambda$  spacing<sup>5</sup>.

Conceptually, this can be understood geometrically: the accuracy of the local Taylor approximation depends on how smoothly the dynamics evolve in  $\lambda$ . Uniform spacing in  $\lambda$  therefore yields approximately uniform local error across the trajectory, resulting in finer (denser) steps in  $t$  where the signal dominates (high SNR), and coarser (sparser) steps in the noise-dominated regime.

Although the derivation operates in  $\lambda$ -space and the PF-ODE is formulated in a convenient semilinear form in that domain, the pre-trained model and noise schedules  $(\alpha_t, \sigma_t)$  are usually defined with respect to the original time variable  $t$ . During sampling, the solver selects nodes that are uniformly spaced in  $\lambda$  for numerical stability, but all update equations are expressed in  $t$ . Whenever it needs to evaluate the model or retrieve schedule values, the chosen  $\lambda$  node is mapped back to the corresponding time variable, such as the

---

<sup>5</sup>Alternatively, adaptive step-size strategies dynamically adjust the timesteps by combining solvers of different orders; see Appendix C of Lu *et al.* (2022b).

physical time  $t = t_\lambda(\lambda)$  or the variance parameter  $\sigma_t$ , depending on how the model is parameterized (see, for instance, Algorithm 5).

#### 9.4.4 DDIM = DPM-Solver-1

For a fixed schedule  $(\alpha_t, \sigma_t)$ , the DPM-Solver-1 step coincides with the deterministic DDIM ( $\eta = 0$ ) update, independent of the time parameterization (physical time  $t$  or log-SNR time  $\lambda$ ); see the formal statement below.

**Proposition 9.4.2: DDIM is DPM-Solver-1**

The update rule of DDIM, given in Equation (9.2.2), is identical to that of DPM-Solver-1, given in Equation (9.4.8).

**Proof for Proposition.**

By the definition of  $\lambda$ , we have

$$\frac{\sigma_s}{\alpha_s} = e^{-\lambda_s} \quad \text{and} \quad \frac{\sigma_t}{\alpha_t} = e^{-\lambda_t}. \quad (9.4.12)$$

Substituting these expressions, along with  $h = \lambda_t - \lambda_s$ , into Equation (9.2.2) recovers the update rule in Equation (9.4.8), completing the equivalence.

The above proposition may explain why DDIM outperforms traditional Euler methods in  $t$ -parametrization: it effectively exploits the semilinearity of the diffusion ODE under a more suitable  $\lambda$ -reparametrization.

**Remark.**

When the Score SDE paper appeared, Runge–Kutta (RK45) was commonly used to solve the vanilla PF-ODE in Equation (4.2.5), but the semilinearity of its drift remained unexploited. Although DPM-Solver- $k$  ( $k \geq 2$ ) is related to Runge–Kutta methods, it explicitly leverages this semilinearity via a time reparameterization. This explains why DPM-Solver attains higher-order accuracy with far fewer function evaluations, reducing a typical DDIM schedule of several hundred steps to about 10–15 steps while preserving high sample quality.

### 9.4.5 Discussion on DPM-Solver-2 and Classic Heun updates

In Section 9.2.2, we saw that different parameterizations of the PF–ODE lead to different interpretations of classical Euler–type updates:

**v**-prediction: Euler = DDIM,

**ε**-, **x**-, or **s**-prediction: exp–Euler = DDIM  $\neq$  plain Euler.

In this subsection, we further illustrate the connection by examining the analogous relationship between the classic *Heun’s method* and the 2nd–order DPM–Solver across the four parameterizations.

To set the stage, we briefly recall Heun’s method (see also Section A.1.4). Heun’s method is a 2nd–order solver that refines Euler’s method using a predictor–corrector scheme: it first makes an Euler prediction to the end of the step, evaluates the slope there, and then updates using the average of the starting and predicted slopes. Intuitively, it advances along the curve by following the mean slope over the interval (the area of a trapezoid), achieving much higher accuracy than plain Euler.

We work in the log–SNR time  $\lambda$ , where the PF–ODE can be expressed in a simple “linear + nonlinear” form:

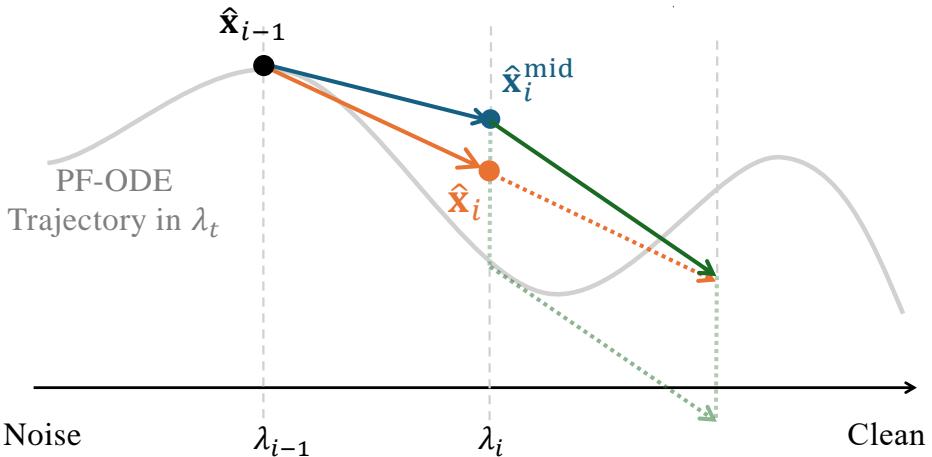
$$\frac{d\hat{\mathbf{x}}(\lambda)}{d\lambda} = \underbrace{L(\lambda)\hat{\mathbf{x}}(\lambda)}_{\text{linear part}} + \underbrace{\mathbf{N}(\hat{\mathbf{x}}(\lambda), \lambda)}_{\text{nonlinear part}},$$

where the scalar  $L(\lambda)$  is determined by the noise schedule and  $\mathbf{N}(\cdot, \lambda)$  collects the nonlinear part. This structure naturally arises from Equation (6.3.2): the **ε**-, **x**-, and **s**-prediction parameterizations yield nonzero  $L(\lambda)$ , resulting in a semilinear form. In contrast, **v**-prediction corresponds to  $L(\lambda) \equiv 0$  (so  $\mathbf{N} = \mathbf{v}$ ), leaving no explicit linear term.

In the remainder of our discussion, we first recall the plain Heun update without considering any semilinear structure, and then introduce the exponential Heun update, which is designed for semilinear ODEs and treats the linear part exactly, analogous to the exponential Euler step in Equations (9.1.7) and (9.1.8). Finally, we relate both Heun updates to DPM-Solver-2 under the four parameterizations and conclude:

**v**-prediction: Heun = DPM-Solver-2,

**ε**-, **x**-, or **s**-prediction: exp–Heun = DPM-Solver-2  $\neq$  plain Heun.



**Figure 9.3: Plain Heun update in log-SNR time.** Starting from the previous state  $\hat{\mathbf{x}}_{i-1}$  at  $\lambda_{i-1}$ , the *predictor* step (blue arrow) performs an explicit Euler move  $h\mathbf{F}(\hat{\mathbf{x}}_{i-1}, \lambda_{i-1})$  to obtain the intermediate estimate  $\hat{\mathbf{x}}_i^{\text{mid}}$ . At this predicted point, the *corrector* step evaluates the new slope  $h\mathbf{F}(\hat{\mathbf{x}}_i^{\text{mid}}, \lambda_i)$  (green arrow) and combines both slopes through a parallelogram construction: the dashed orange diagonal represents the vector sum  $h(\mathbf{F}(\hat{\mathbf{x}}_{i-1}, \lambda_{i-1}) + \mathbf{F}(\hat{\mathbf{x}}_i^{\text{mid}}, \lambda_i))$  starting from  $\hat{\mathbf{x}}_{i-1}$ , and the solid orange arrow is its half-diagonal, having the same direction but half the length. This procedure realizes the plain Heun integration of the PF-ODE trajectory in log-SNR time.

**Plain Heun update.** Denote  $\lambda_i := \lambda_{t_i}$ , then  $\{\lambda_i\}_{i=0}^M$  is an increasing grid in the log-SNR domain, and set  $h := \lambda_i - \lambda_{i-1} > 0$ . Let  $\hat{\mathbf{x}}_{i-1}$  denote the previous iterate in log-SNR time. Applied directly to the full drift

$$\mathbf{F}(\hat{\mathbf{x}}, \lambda) := L(\lambda)\hat{\mathbf{x}} + \mathbf{N}(\hat{\mathbf{x}}, \lambda),$$

the plain Heun update in log-SNR-time is given by

$$\begin{aligned} \text{Predict: } \hat{\mathbf{x}}_i^{\text{mid}} &= \hat{\mathbf{x}}_{i-1} + h\mathbf{F}(\hat{\mathbf{x}}_{i-1}, \lambda_{i-1}), \\ \text{Correct: } \hat{\mathbf{x}}_i &= \hat{\mathbf{x}}_{i-1} + \frac{h}{2} \left( \mathbf{F}(\hat{\mathbf{x}}_{i-1}, \lambda_{i-1}) + \mathbf{F}(\hat{\mathbf{x}}_i^{\text{mid}}, \lambda_i) \right). \end{aligned} \tag{9.4.13}$$

**Exponential Heun update (for Semilinear PF-ODE).** With the *exponential integrator* technique, the idea is to treat the linear and nonlinear parts of the ODE differently. The linear term  $L(\lambda)\hat{\mathbf{x}}$  is integrated exactly over the step, while the nonlinear term  $\mathbf{N}(\hat{\mathbf{x}}, \lambda)$  is only approximated by averaging its effect across the step.

To express this neatly, we introduce the quantity

$$\mathcal{E} := \int_{\lambda_{i-1}}^{\lambda_i} L(\tau) d\tau,$$

which represents the total contribution of the linear coefficient  $L(\lambda)$  over the interval  $[\lambda_{i-1}, \lambda_i]$ . Using  $\mathcal{E}$ , we define two helper coefficients  $c_1(\mathcal{E})$  and  $c_2(\mathcal{E})$  that handle both cases: when  $\mathcal{E}$  is nonzero and when it vanishes:

$$c_1(\mathcal{E}) = \begin{cases} \frac{e^\mathcal{E} - 1}{\mathcal{E}}, & \text{if } \mathcal{E} \neq 0, \\ 1, & \text{if } \mathcal{E} = 0, \end{cases} \quad c_2(\mathcal{E}) = \begin{cases} \frac{e^\mathcal{E} - 1 - \mathcal{E}}{\mathcal{E}^2}, & \text{if } \mathcal{E} \neq 0, \\ \frac{1}{2}, & \text{if } \mathcal{E} = 0. \end{cases}$$

The second case simply ensures continuity when the linear term disappears ( $L(\lambda) = 0$ ), so that the formulas remain valid and reduce smoothly to the standard Heun update as in Equation (9.4.13).

With these coefficients, one update step of the exponential–Heun scheme can be written as:

$$\begin{aligned} \text{Predict: } \hat{\mathbf{x}}_i^{\text{mid}} &= e^\mathcal{E} \hat{\mathbf{x}}_{i-1} + hc_1(\mathcal{E}) \mathbf{N}(\hat{\mathbf{x}}_{i-1}, \lambda_{i-1}), \\ \text{Correct: } \hat{\mathbf{x}}_i &= e^\mathcal{E} \hat{\mathbf{x}}_{i-1} + hc_1(\mathcal{E}) \mathbf{N}(\hat{\mathbf{x}}_{i-1}, \lambda_{i-1}) \\ &\quad + hc_2(\mathcal{E}) (\mathbf{N}(\hat{\mathbf{x}}_i^{\text{mid}}, \lambda_i) - \mathbf{N}(\hat{\mathbf{x}}_{i-1}, \lambda_{i-1})). \end{aligned} \tag{9.4.14}$$

When  $L(\lambda) \equiv 0$ , the coefficients simplify to  $c_1 = 1$  and  $c_2 = \frac{1}{2}$ , and the method reduces to the plain Heun solver in Equation (9.4.13).

When  $L(\lambda) \neq 0$ , the exponential–integrator form of the update integrates the linear term exactly, while the plain Heun method only provides an approximation. To see this, expand the exponential term for a small stepsize  $h = \lambda_i - \lambda_{i-1} > 0$ . Since

$$\mathcal{E} = \int_{\lambda_{i-1}}^{\lambda_i} L(\tau) d\tau = hL(\lambda_{i-1}) + \mathcal{O}(h^2),$$

we can treat  $\mathcal{E}$  as a small quantity of order  $\mathcal{O}(h)$ . The Taylor expansions give:

$$e^\mathcal{E} = 1 + \mathcal{E} + \frac{\mathcal{E}^2}{2} + \mathcal{O}(\mathcal{E}^3), \quad c_1(\mathcal{E}) = 1 + \frac{\mathcal{E}}{2} + \frac{\mathcal{E}^2}{6} + \mathcal{O}(\mathcal{E}^3), \quad c_2(\mathcal{E}) = \frac{1}{2} + \frac{\mathcal{E}}{6} + \frac{\mathcal{E}^2}{24} + \mathcal{O}(\mathcal{E}^3).$$

Substituting these approximations into Equation (9.4.14) and keeping terms up to  $\mathcal{E}^2$  (that is, up to order  $h^2$  since  $\mathcal{E} = \mathcal{O}(h)$ ), the update simplifies exactly to the plain Heun form (Equation (9.4.13)). The remaining difference between the two schemes appears only in higher-order terms of size  $\mathcal{O}(\mathcal{E}^3) = \mathcal{O}(h^3)$ . Intuitively, when the step size  $h$  is small,  $\mathcal{E}$  is also small, so the exponential factors reduce to

$$e^\mathcal{E} \approx 1 + \mathcal{E}, \quad c_1(\mathcal{E}) \approx 1, \quad c_2(\mathcal{E}) \approx \frac{1}{2}.$$

The “linear–handled” exponential–Heun update thus collapses to the plain Heun step.

**Connection of Heun’s Updates to DPM–Solver-2 Under the Four Predictions.** We highlight that, in the  $\epsilon$ -prediction form of the PF-ODE (see Equation (9.4.5)), the dynamics in log–SNR time  $\lambda$  naturally take the required semilinear form:

$$\frac{d\hat{\mathbf{x}}_\lambda}{d\lambda} = \underbrace{\frac{\alpha'_\lambda}{\alpha_\lambda} \hat{\mathbf{x}}_\lambda}_{=:L(\lambda)} + \underbrace{(-\sigma_\lambda \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda))}_{=:N(\hat{\mathbf{x}}_\lambda, \lambda)}.$$

Consequently, for  $\epsilon$ -prediction in log–SNR time  $\lambda$ , the exponential–Heun update in Equation (9.4.14) is *exactly equivalent* to DPM-Solver-2 (with midpoint parameter  $\gamma = \frac{1}{2}$ ; see Algorithm 5).

Similarly, under the  $\mathbf{x}$ - and  $\mathbf{s}$ -prediction parameterizations in log–SNR time, their PF-ODEs also take the same semilinear structure. Hence, the DPM-Solver-2 under the  $\epsilon$ -,  $\mathbf{x}$ -, or  $\mathbf{s}$ -prediction is *identical* to the exponential–Heun update in Equation (9.4.14). In contrast, the  $\mathbf{v}$ -prediction form naturally removes the linear term, so its PF-ODE does not require an exponential integrator; the plain Heun method in log–SNR time already provides the correct 2nd–order update.

Similar to the case of Euler versus exponential Euler in DDIM, we therefore conclude the following:

#### Observation 9.4.1: Heun and DPM-Solver-2 Updates

Given the PF-ODEs in log–SNR time  $\lambda$ ,

$\mathbf{v}$ -prediction: Heun = DPM-Solver-2,

$\epsilon$ -,  $\mathbf{x}$ -, or  $\mathbf{s}$ -prediction: exp–Heun = DPM-Solver-2  $\neq$  plain Heun,

where, in the  $\epsilon$ -,  $\mathbf{x}$ -, or  $\mathbf{s}$ -prediction cases, the plain Heun step is not equivalent to DPM-Solver-2, since the linear term is only approximated instead of being integrated exactly.

## 9.5 DPM-Solver++

### 9.5.1 From DPM-Solver to DPM-Solver++ for Guidance

High-order solvers enable faster sampling without guidance. However, diffusion models are prized for their controllable and flexible generation, typically achieved via guidance (see Chapter 8 for details).

DPM-Solver++ (Lu *et al.*, 2022c) identifies a key limitation of prior high-order solvers: they suffer from stability issues and may become slower than DDIM under large guidance scales (stronger condition). The authors attribute this instability to the amplification of both the output and its derivatives by large guidance scales. Since high-order solvers depend on higher-order derivatives, they are especially sensitive to this effect, resulting in diminished efficiency and stability.

### 9.5.2 DPM-Solver++'s Methodology

To address the aforementioned issues, DPM-Solver++ proposes:

1. adopting  $\mathbf{x}$ -prediction parameterization instead of  $\epsilon$ -prediction;
2. applying thresholding methods (e.g., dynamic thresholding (Saharia *et al.*, 2022)) to keep the predicted data within the training data bounds (mitigating the train-test mismatch at large guidance scales).

We elaborate on the first point. Recall from Equation (6.3.1) that the data and noise parameterizations are linearly related:

$$\epsilon_{\phi^\times}(\mathbf{x}_t, t) = \frac{\mathbf{x}_t - \alpha_t \mathbf{x}_{\phi^\times}(\mathbf{x}_t, t)}{\sigma_t}.$$

Using this relation, DPM-Solver++ rewrites the exact solution  $\tilde{\Psi}_{s \rightarrow t}(\mathbf{x}_s)$  of the empirical PF-ODE (originally expressed in the noise parameterization in Equation (9.4.4)), starting from any  $\mathbf{x}_s$ :

$$\tilde{\Psi}_{s \rightarrow t}(\mathbf{x}_s) = \frac{\alpha_t}{\alpha_s} \mathbf{x}_s - \alpha_t \int_{\lambda_s}^{\lambda_t} e^{-\lambda} \hat{\epsilon}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) d\lambda,$$

into the data parameterization as

$$\tilde{\Psi}_{s \rightarrow t}(\mathbf{x}_s) = \frac{\sigma_t}{\sigma_s} \mathbf{x}_s + \sigma_t \int_{\lambda_s}^{\lambda_t} e^\lambda \hat{\mathbf{x}}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) d\lambda,$$

where we follow the notations in Equation (9.4.3) and further denote:

$$\begin{aligned}\hat{\mathbf{x}}_\lambda &:= \mathbf{x}_{t_\lambda(\lambda)}, \\ \hat{\mathbf{x}}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) &:= \mathbf{x}_{\phi^\times}(\mathbf{x}_{t_\lambda(\lambda)}, t_\lambda(\lambda)).\end{aligned}$$

Based on the  $\mathbf{x}$ -prediction, DPM-Solver++ provides two solver variants:

- **Higher-Order Single-Step Solver:** Introduced in Section 9.5.3. This approach is analogous to that in DPM-Solver, which leverages higher-order Taylor expansions to approximate the integration, but here formulated with the  $\mathbf{x}$ -prediction. The update uses only one previous point to estimate the next step.
- **Multistep (Two-Step) Solver:** Introduced in Section 9.5.4. The design philosophy is similar to DEIS (also multistep); however, DPM-Solver++ specifically reuses two previous points (whereas DEIS allows a general order) to estimate the next step. Each update requires only a single new diffusion model evaluation.

### 9.5.3 DPM-Solver++ Single-Step by Taylor Expansion

Following a similar approach to Section 9.4.3, DPM-Solver++ derives higher-order solvers in the  $\mathbf{x}$ -parameterization. For  $n \geq 0$ , denote the  $n$ -th total derivative of  $\hat{\mathbf{x}}_{\phi^\times}$  with respect to  $\lambda$ , evaluated at  $\lambda_{i-1}$ , by

$$\hat{\mathbf{x}}_{\phi^\times}^{(n)}(\hat{\mathbf{x}}_{\lambda_{i-1}}, \lambda_{i-1}) := \left. \frac{d^n}{d\lambda^n} \hat{\mathbf{x}}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) \right|_{\lambda=\lambda_{i-1}}.$$

Given the previous estimate  $\tilde{\mathbf{x}}_{t_{i-1}}$  at time  $t_{i-1}$ , using the  $(n-1)$ -th Taylor expansion at  $\lambda_{t_{i-1}}$  to approximate  $\hat{\mathbf{x}}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda)$  for  $\lambda \in [\lambda_{t_{i-1}}, \lambda_{t_i}]$  (with  $s = t_{i-1}$  and  $t = t_i$ ) yields the following approximation of  $\tilde{\Psi}_{s \rightarrow t}(\mathbf{x}_s)$ :

$$\begin{aligned} \tilde{\mathbf{x}}_{t_i} &= \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}} \tilde{\mathbf{x}}_{t_{i-1}} + \sigma_{t_i} \underbrace{\sum_{k=0}^{n-1} \hat{\mathbf{x}}_{\phi^\times}^{(k)}(\hat{\mathbf{x}}_{\lambda_{i-1}}, \lambda_{i-1})}_{\text{estimated via finite difference}} \underbrace{\int_{\lambda_{t_{i-1}}}^{\lambda_{t_i}} e^\lambda \frac{(\lambda - \lambda_{t_{i-1}})^k}{k!} d\lambda}_{\text{analytically computable}} \\ &\quad + \mathcal{O}(h_i^{n+1}). \end{aligned}$$

where  $h_i := \lambda_{t_i} - \lambda_{t_{i-1}} > 0$ . As in Equation (9.4.9), the integral admits the closed form

$$\int_{\lambda_{t_{i-1}}}^{\lambda_{t_i}} e^\lambda \frac{(\lambda - \lambda_{t_{i-1}})^k}{k!} d\lambda = e^{\lambda_{t_{i-1}}} h_i^{k+1} \varphi_{k+1}(h_i), \quad \varphi_m(h) := \frac{e^h - \sum_{j=0}^{m-1} \frac{h^j}{j!}}{h^m}.$$

This yields the DPM-Solver++'s single-step update (one previous point to estimate the next). When  $n = 1$ , it reduces to the DDIM update. When  $n = 2$  and  $\hat{\mathbf{x}}_{\phi^\times}^{(1)}(\hat{\mathbf{x}}_{\lambda_{i-1}}, \lambda_{i-1})$  is approximated via a finite difference, it gives *DPM-Solver++(2S)*, an update analogous to DPM-Solver-2 in Algorithm 5 but using the  $\mathbf{x}$ -prediction. DPM-Solver++(2S)'s algorithm is shown in Algorithm 6.

**Algorithm 6** DPM-Solver++(2S): a midpoint special case.

---

**Input:** initial value  $\mathbf{x}_T$ , time steps  $\{t_i\}_{i=0}^M$ , data-prediction model  $\hat{\mathbf{x}}_{\phi^\times}$

- 1:  $\tilde{\mathbf{x}}_{t_0} \leftarrow \mathbf{x}_T; \quad \lambda_{t_i} \leftarrow \log(\alpha_{t_i}/\sigma_{t_i})$   $\triangleright$  log-SNR at the grid
- 2:  $\hat{\mathbf{x}}_{t_0} \leftarrow \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_0}, t_0)$   $\triangleright$  cache at start
- 3: **for**  $i \leftarrow 1$  **to**  $M$  **do**
- 4:    $h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}; \quad s_i^{\text{mid}} \leftarrow t_\lambda\left(\frac{\lambda_{t_{i-1}} + \lambda_{t_i}}{2}\right)$
- 5:    $\mathbf{u}_i \leftarrow \frac{\sigma_{s_i^{\text{mid}}}}{\sigma_{t_{i-1}}} \tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{s_i^{\text{mid}}} (1 - e^{-h_i/2}) \hat{\mathbf{x}}_{t_{i-1}}$   $\triangleright$  forecast to midpoint
- 6:    $\mathbf{D}_i^{\text{mid}} \leftarrow \hat{\mathbf{x}}_{\phi^\times}(\mathbf{u}_i, s_i^{\text{mid}})$   $\triangleright$  one new model call at the midpoint
- 7:    $\tilde{\mathbf{x}}_{t_i} \leftarrow \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}} \tilde{\mathbf{x}}_{t_{i-1}} - \alpha_{t_i} (e^{-h_i} - 1) \mathbf{D}_i^{\text{mid}}$
- 8:    $\hat{\mathbf{x}}_{t_i} \leftarrow \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_i}, t_i)$   $\triangleright$  cache for next step
- 9: **end for**
- 10: **return**  $\tilde{\mathbf{x}}_{t_M}$

---

**9.5.4 DPM-Solver++ Multistep by Recycling History**

High-order single-step solvers rely (explicitly or implicitly) on higher derivatives of the model output; under strong CFG these derivatives can be strongly amplified and destabilize the update. DPM-Solver++ mitigates this with a *multistep* (Adams-type) strategy in log-SNR time  $\lambda$ : it reuses a short history of past data-prediction evaluations along the trajectory to approximate the needed derivatives via finite differences. This reuse requires only one new model call per step. As with DEIS, we separate the presentation into: Case 1. the warm start with no history (first step); Case 2. subsequent steps with two history anchors.

**Case I. DPM-Solver++ with One History Anchor ( $i = 1$ ).** For the first step ( $i = 1$ ; no history), use the first-order DPM-style update (which matches the deterministic DDIM step in data prediction). Let  $h_1 = \lambda_1 - \lambda_0$ .

$$\tilde{\mathbf{x}}_{t_1} = \frac{\sigma_{t_1}}{\sigma_{t_0}} \tilde{\mathbf{x}}_{t_0} + \sigma_{t_1} e^{\lambda_0} (e^{h_1} - 1) \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_0}, t_0)$$

**Case II. DPM-Solver++ with Two History Anchors ( $i \geq 2$ ).** After the warm start, the two-step multistep update reuses the estimations at time  $t_{i-2}$  with  $\tilde{\mathbf{x}}_{t_{i-2}}$  and at time  $t_{i-1}$  with  $\tilde{\mathbf{x}}_{t_{i-1}}$ . At each step  $i \geq 2$ , these provide the two most recent anchors, equivalently in  $\lambda$ -time:

$$(\lambda_{i-1}, \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1})) \quad \text{and} \quad (\lambda_{i-2}, \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-2}}, t_{i-2})),$$

to compute the update  $\tilde{\mathbf{x}}_{t_i}$  using only these cached anchors (no fresh model call is needed to form the update). After obtaining  $\tilde{\mathbf{x}}_{t_i}$ , we evaluate the model once at  $(\tilde{\mathbf{x}}_{t_i}, t_i)$  and cache  $\hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_i}, t_i)$ . This evaluation is performed during step  $i$  and is used as an anchor in the subsequent step  $i+1$ . Namely, we aim for a one-call-per-step update that remains stable under large guidance by discretizing the exact  $\mathbf{x}$ -prediction form

$$\tilde{\Psi}_{s \rightarrow t}(\mathbf{x}_s) = \frac{\sigma_t}{\sigma_s} \mathbf{x}_s + \sigma_t \int_{\lambda_s}^{\lambda_t} e^\lambda \hat{\mathbf{x}}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) d\lambda.$$

Over a single step  $[\lambda_{i-1}, \lambda_i]$ , we treat the linear ODE part exactly and approximate the residual integral by approximating the integrand as a function linear in  $\lambda$  (since there are two anchor points). Concretely, we approximate

$$\lambda \mapsto \hat{\mathbf{x}}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda)$$

on  $[\lambda_{i-1}, \lambda_i]$  by the affine model

$$\hat{\mathbf{x}}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) \approx \mathbf{L}(\lambda) := \mathbf{a}_0 + \mathbf{a}_1(\lambda - \lambda_{i-1}), \quad \lambda \in [\lambda_{i-1}, \lambda_i],$$

where  $\lambda_i = \lambda_{t_i}$ ,  $h_i = \lambda_i - \lambda_{i-1} > 0$ , and the coefficients  $\mathbf{a}_0$  and  $\mathbf{a}_1$  are uniquely specified by the straight line passing through the two most recent anchors:

$$\mathbf{a}_0 = \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}), \quad \mathbf{a}_1 = \frac{\hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}) - \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-2}}, t_{i-2})}{h_{i-1}}.$$

Substituting  $\mathbf{L}(\lambda)$  into the integral thus yields<sup>6</sup>

$$\begin{aligned} \sigma_{t_i} \int_{\lambda_{i-1}}^{\lambda_i} e^\lambda \hat{\mathbf{x}}_{\phi^\times}(\hat{\mathbf{x}}_\lambda, \lambda) d\lambda &\approx \sigma_{t_i} \int_{\lambda_{i-1}}^{\lambda_i} e^\lambda \mathbf{L}(\lambda) d\lambda \\ &= \left( \sigma_{t_i} \int_{\lambda_{i-1}}^{\lambda_i} e^\lambda d\lambda \right) \mathbf{a}_0 + \left( \sigma_{t_i} \int_{\lambda_{i-1}}^{\lambda_i} e^\lambda (\lambda - \lambda_{i-1}) d\lambda \right) \mathbf{a}_1 \\ &= \left( \alpha_{t_i} (1 - e^{-h_i}) \right) \mathbf{a}_0 + \left( \alpha_{t_i} (h_i - 1 + e^{-h_i}) \right) \mathbf{a}_1 \\ &= \alpha_{t_i} (1 - e^{-h_i}) (\mathbf{a}_0 + \beta(h_i) \mathbf{a}_1), \end{aligned}$$

---

<sup>6</sup>The second identity follows from a straightforward algebra. The two needed exponential moments are

$$\int_{\lambda_{i-1}}^{\lambda_i} e^\lambda d\lambda = e^{\lambda_{i-1}} (e^{h_i} - 1), \quad \int_{\lambda_{i-1}}^{\lambda_i} e^\lambda (\lambda - \lambda_{i-1}) d\lambda = e^{\lambda_{i-1}} (h_i e^{h_i} - e^{h_i} + 1).$$

Multiplying by the prefactor  $\sigma_{t_i}$  from the exact form and using  $\alpha_t = \sigma_t e^{\lambda_t}$  (so  $\sigma_{t_i} e^{\lambda_{i-1}} = \alpha_{t_i} e^{-h_i}$ ) gives the convenient coefficients

$$\sigma_{t_i} \int_{\lambda_{i-1}}^{\lambda_i} e^\lambda d\lambda = \alpha_{t_i} (1 - e^{-h_i}), \quad \sigma_{t_i} \int_{\lambda_{i-1}}^{\lambda_i} e^\lambda (\lambda - \lambda_{i-1}) d\lambda = \alpha_{t_i} (h_i - 1 + e^{-h_i}).$$

where  $\beta(h) := \frac{h-1+e^{-h}}{1-e^{-h}}$ . Until this point, we have already reached a valid estimate for  $\tilde{\mathbf{x}}_{t_i}$  as:

$$\tilde{\mathbf{x}}_{t_i} = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}} \tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i} (1 - e^{-h_i}) \mathbf{D}_i, \quad \text{with } \mathbf{D}_i = \mathbf{a}_0 + \beta(h_i) \mathbf{a}_1.$$

In practice, we can obtain a simplified update rule with the same local truncation error (provided the step ratios are bounded) as the above one:

$$\tilde{\mathbf{x}}_{t_i} = \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}} \tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i} (1 - e^{-h_i}) \mathbf{D}_i^{\text{sim}}(\tilde{\mathbf{x}}_{t_{i-1}}, \tilde{\mathbf{x}}_{t_{i-2}}).$$

Here, we define the step ratio  $r_i = h_i/h_{i-1}$ , and

$$\mathbf{D}_i^{\text{sim}}(\tilde{\mathbf{x}}_{t_{i-1}}, \tilde{\mathbf{x}}_{t_{i-2}}) := \left(1 + \frac{1}{2}r_i\right) \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}) - \frac{1}{2}r_i \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-2}}, t_{i-2}).$$

with local error  $\mathcal{O}(h_i^3)$  under standard smoothness assumptions.

To see why, for notational simplicity, we write

$$\mathbf{a}_0 = \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}) =: \hat{\mathbf{x}}_{i-1}, \quad \mathbf{a}_1 = \frac{\hat{\mathbf{x}}_{i-1} - \hat{\mathbf{x}}_{i-2}}{h_{i-1}}.$$

Then

$$\begin{aligned} \mathbf{D}_i &:= \mathbf{a}_0 + \beta(h_i) \mathbf{a}_1 \\ &= \hat{\mathbf{x}}_{i-1} + \frac{\beta(h_i)}{h_{i-1}} (\hat{\mathbf{x}}_{i-1} - \hat{\mathbf{x}}_{i-2}) \\ &= \left(1 + \frac{r_i}{2}\right) \hat{\mathbf{x}}_{i-1} - \frac{r_i}{2} \hat{\mathbf{x}}_{i-2} + \left(\frac{\beta(h_i)}{h_{i-1}} - \frac{r_i}{2}\right) (\hat{\mathbf{x}}_{i-1} - \hat{\mathbf{x}}_{i-2}) \\ &= \left[\left(1 + \frac{1}{2}r_i\right) \hat{\mathbf{x}}_{i-1} - \frac{1}{2}r_i \hat{\mathbf{x}}_{i-2}\right] + \mathcal{O}(h_i^2) \\ &= \mathbf{D}_i^{\text{sim}} + \mathcal{O}(h_i^2) \end{aligned}$$

Here, we use that for small steps, a Taylor expansion of  $\beta(h)$  at  $h = 0$  gives

$$\beta(h) = \frac{h}{2} + \mathcal{O}(h^2) \implies \frac{\beta(h_i)}{h_{i-1}} = \frac{h_i}{2h_{i-1}} + \mathcal{O}(h_i^2/h_{i-1}) = \frac{r_i}{2} + \mathcal{O}(h_i^2/h_{i-1}),$$

and that  $\hat{\mathbf{x}}_{i-1} - \hat{\mathbf{x}}_{i-2} = \mathcal{O}(h_{i-1})$  under some smoothness assumption.

### Remark.

If the log-SNR steps are uniform (every step has the same size  $h$ , so  $h_i \equiv h$

and  $r_i = h_i/h_{i-1} = 1$ ), then the two-anchor blend

$$\mathbf{D}_i^{\text{sim}} = \left(1 + \frac{1}{2}r_i\right)\hat{\mathbf{x}}_{i-1} - \frac{1}{2}r_i\hat{\mathbf{x}}_{i-2}$$

reduces to the classic constants

$$\mathbf{D}_i^{\text{sim}} = \left(1 + \frac{1}{2} \cdot 1\right)\hat{\mathbf{x}}_{i-1} - \frac{1}{2} \cdot 1\hat{\mathbf{x}}_{i-2} = \frac{3}{2}\hat{\mathbf{x}}_{i-1} - \frac{1}{2}\hat{\mathbf{x}}_{i-2}.$$

Those  $(\frac{3}{2}, -\frac{1}{2})$  are exactly the Adams-Bashforth 2 weights for uniform steps, i.e., the standard two-step linear multistep coefficients.

---

**Algorithm 7** DPM-Solver++(2M).

---

**Input:** initial value  $\mathbf{x}_T$ , time steps  $\{t_i\}_{i=0}^M$ , model  $\hat{\mathbf{x}}_{\phi^\times}$

1:  $\tilde{\mathbf{x}}_{t_0} \leftarrow \mathbf{x}_T; \quad \lambda_{t_i} \leftarrow \log(\alpha_{t_i}/\sigma_{t_i}); \quad h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$

2:  $\hat{\mathbf{x}}_0 \leftarrow \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_0}, t_0)$  ▷ cache at start

**Case I. Warm start ( $i = 1$ ) with one anchor (DDIM in x-pred.)**

3:  $\tilde{\mathbf{x}}_{t_1} \leftarrow \frac{\sigma_{t_1}}{\sigma_{t_0}}\tilde{\mathbf{x}}_{t_0} - \alpha_{t_1}(e^{-h_1} - 1)\hat{\mathbf{x}}_0$

4:  $\hat{\mathbf{x}}_1 \leftarrow \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_1}, t_1)$  ▷ One model call & cache

**Case II. Using two history cached anchors (multistep)**

5: **for**  $i \leftarrow 2$  to  $M$  **do**

6:      $r_i \leftarrow h_i/h_{i-1}$  ▷ step ratio

7:      $\mathbf{D}_i^{\text{sim}} \leftarrow \left(1 + \frac{1}{2}r_i\right)\hat{\mathbf{x}}_{i-1} - \frac{1}{2}r_i\hat{\mathbf{x}}_{i-2}$

8:      $\tilde{\mathbf{x}}_{t_i} \leftarrow \frac{\sigma_{t_i}}{\sigma_{t_{i-1}}}\tilde{\mathbf{x}}_{t_{i-1}} + \alpha_{t_i}(1 - e^{-h_i})\mathbf{D}_i^{\text{sim}}$

9:      $\hat{\mathbf{x}}_i \leftarrow \hat{\mathbf{x}}_{\phi^\times}(\tilde{\mathbf{x}}_{t_i}, t_i)$  ▷ One model call & cache

10: **end for**

11: **return**  $\tilde{\mathbf{x}}_{t_M}$

---

## 9.6 PF-ODE Solver Families and Their Numerical Analogues

In this section, we first place the PF-ODE solvers introduced so far (DDIM, DEIS, DPM-Solver, DPM-Solver++) into the context of classical numerical integration methods. We then turn to a closer examination of two representative higher-order solvers, DEIS and DPM-Solver++, and compare their respective designs.

### 9.6.1 PF-ODE Solver Families and Classical Counterparts

The diverse families of PF-ODE samplers can be understood through the lens of classical numerical analysis. Once the linear drift is treated by an integrating factor, each sampler aligns naturally with an established time-stepping scheme: Euler-type methods, Adams–Bashforth (AB) multistep schemes, or Runge–Kutta (RK) single-step integrators. We summarize these correspondences in Table 9.1.

**Table 9.1:** PF-ODE samplers and their numerical-analysis analogues. “exp.” denotes integrating-factor (semilinear) treatment of the linear term (see Equation (9.1.6)). AB = Adams–Bashforth, RK = Runge–Kutta. See Algorithm 5 for *DPM-Solver-2*.

PF-ODE Solver	Type	Classical Numerical Analogue
DDIM	single step	v-prediction: plain Euler; $\epsilon/x/s$ -prediction: exp. Euler
DEIS	multistep	exp. AB ( $n^{\text{th}}$ -order)
DPM-Solver-n	single step	exp. RK ( $n^{\text{th}}$ -order) in log-SNR
DPM-Solver-2	single step	v-prediction: plain Heun in log-SNR (2 <sup>nd</sup> -order); $\epsilon/x/s$ -prediction: exp. Heun in log-SNR (2 <sup>nd</sup> -order)
DPM-Solver++ 2S	single step	exp. RK (2 <sup>nd</sup> -order)
DPM-Solver++ 2M	multistep	exp. AB (2 <sup>nd</sup> -order)

We highlight two representative examples in Table 9.1: the DDIM and DPM–Solver–2 cases. With a fixed scheduler  $(\alpha_t, \sigma_t)$ , we emphasize the illustrative results from Sections 9.2.2, 9.3.3 and 9.4.4: regardless of whether we use log–SNR time or the original physical time,

v-prediction: DDIM = DPM-Solver-1 = DEIS-1 = Euler,

$\epsilon$ -,  $x$ -, or  $s$ -prediction: DDIM = DPM-Solver-1 = DEIS-1 = exp Euler.

In Section 9.4.5, we extended this analogy by examining how DPM-Solver-2

relates to the classic Heun solver under the four parameterizations:

$v$ -prediction: DPM-Solver-2 = Heun,

$\epsilon$ -,  $x$ -, or  $s$ -prediction: DPM-Solver-2 = exp-Heun  $\neq$  plain Heun.

A more general correspondence between DPM-Solver- $n$  and classical RK methods can be understood in the same way.

### 9.6.2 Discussion on DEIS and DPM-Solver++

Aspect	DEIS	DPM++
Core Viewpoint	Exponential-integrator: integrates the linear term exactly; approximates the nonlinear residual by a polynomial over past nodes.	Same integrator idea; formulated in log-SNR time $\lambda$ with data prediction.
Step type	Multistep only	Single-step (2S) and Multistep (2M)
Polynomial Basis	Lagrange interpolation across past anchors (high-order multistep).	Backward divided differences (Newton/Adams-type) in $\lambda$ -time for 2M; algebraically spans the same polynomial space as Lagrange, but not presented as a Lagrange fit.
Solvers Order	High-order multistep (general $r$ ).	Higher-order single-step methods exist (though 2S is the main focus), and a 2nd-order multistep (2M) scheme is provided; higher-order multistep variants are not covered.
History Use	Uses $r+1$ past evaluations to build a high-order update.	2S: one intermediate eval (single-step). 2M: reuses two anchors; after warm start, one model call per step.

**DEIS vs. DPM-Solver++.** Both DEIS and DPM++ are exponential integrator samplers that integrate the linear part exactly and approximate the residual integral by a low-degree polynomial. In unconditional generation, both can achieve high fidelity with as few as 10–20 ODE steps. For conditional generation with CFG, however, DPM++ is often preferable due to its stability under large guidance scales. We summarize the comparison between DEIS and DPM++, and provide further discussion below.

**DEIS.** It is a multistep method obtained by fitting a polynomial to the nonlinear term across past nodes in the Lagrange basis (interpolation through anchors).

**DPM-Solver++.** It works with data prediction in log-SNR time: its single-step (2S) variant uses a Taylor/exponential-integrator step with one

intermediate evaluation, while its multistep (2M) variant reuses history via backward divided differences, which produce the same interpolating polynomial but expressed in the Newton (finite-difference) basis.

In other words, for the same anchor points and function values, the Lagrange and Newton forms are two different coordinate systems for the same polynomial interpolant: Lagrange expresses it as a sum of function values times cardinal basis polynomials, whereas Newton expresses it as a product expansion with coefficients given by divided differences (finite-difference ratios that are easy to update in multistep schemes). The DPM++ paper emphasizes second-order (2S/2M); higher-order multistep extensions can, in principle, be constructed using higher-order Newton bases.

## 9.7 (Optional) DPM-Solver-v3

Both DPM-Solver and DPM-Solver++ design their solvers based on specific parameterizations of the diffusion model ( $\epsilon$ -/ $\mathbf{x}$ -prediction), which lacks a principled approach for selecting the parametrization and may not represent the optimal choice.

In this section, we introduce DPM-Solver-v3 (Zheng *et al.*, 2023), which addresses this issue and enhances sample quality with fewer timesteps or at large guidance scales. DPM-Solver-v3 can be regarded as the culmination of insights of the entire DPM-Solver family (Lu *et al.*, 2022b; Lu *et al.*, 2022c).

**High-Level Overview of DPM-Solver-v3.** We continue to focus on the key principle of DPM-Solver (Lu *et al.*, 2022b), namely the time reparametrization in SRN as expressed in Equation (9.4.5):

$$\frac{d\mathbf{x}_\lambda}{d\lambda} = \frac{\alpha'_\lambda}{\alpha_\lambda} \mathbf{x}_\lambda - \sigma_\lambda \hat{\epsilon}_{\phi^\times}(\mathbf{x}_\lambda, \lambda).$$

The core idea of DPM-Solver-v3 (Zheng *et al.*, 2023) is to introduce three additional underdetermined/free variables into Equation (9.4.5), enabling the original ODE solution to be reformulated equivalently with a new model parameterization. An efficient search method is then proposed to identify an optimal set of these variables, computed on the pre-trained model, with the objective of minimizing discretization errors.

### 9.7.1 Insight 1: Adjusting the Linear Term in Equation (9.4.5)

The PF-ODE is a stiff ODE with distinct timescales in each direction of temporal evolution, complicating its solution with fewer timesteps. Drawing from classic stiff ODE theory (Hochbruck and Ostermann, 2010), Zheng *et al.* (2023) propose modifying the linear part of the ODE to better handle these stiff dynamics. We begin by motivating this approach from classic numerical ODE theory.

**Motivation: From Classic Stiff ODE Solvers.** We begin by considering an abstract form of Equation (9.4.5):

$$\frac{d\mathbf{x}_\lambda}{d\lambda} = \mathbf{v}(\mathbf{x}_\lambda, \lambda),$$

where  $\mathbf{v}(\mathbf{x}, \lambda)$  represents the vector field.

“Rosenbrock-type exponential integrators” are a class of methods developed to efficiently solve stiff ODEs. The key feature of these methods is the flexibility in selecting the linear operator  $\mathbf{L}$ , which decomposes the vector field as:

$$\mathbf{v}(\mathbf{x}, \lambda) = \mathbf{L}\mathbf{x} + \mathbf{N}(\mathbf{x}, \lambda),$$

where  $\mathbf{N}(\mathbf{x}, \lambda)$  denotes the nonlinear remainder. This leads to the following update, starting from  $\mathbf{x}_{\lambda_s}$ , by applying the exponential integrator technique (as usual):

$$\tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s}) = e^{h\mathbf{L}}\mathbf{x}_{\lambda_s} + \int_{\lambda_s}^{\lambda_t} e^{(\lambda_t - \tau)\mathbf{L}}\mathbf{N}(\mathbf{x}_\tau, \tau) d\tau, \quad \text{with } \Delta h := \lambda_t - \lambda_s.$$

We observe that the linear part is in exponential form. The choice of  $\mathbf{L}$  is typically made by utilizing preconditioning information to handle stiffness more efficiently, with the goal of (1) ensuring the stability of the method, (2) improving the convergence rate of the numerical solution, and (3) ensuring that  $e^{\Delta\lambda\mathbf{L}}$  remains computationally efficient.

**Applying the Above Idea to PF-ODE in Equation (9.4.5).** We apply the introduced concept to Equation (9.4.5):

$$\frac{d\mathbf{x}_\lambda}{d\lambda} = \underbrace{\frac{\alpha'_\lambda}{\alpha_\lambda} \mathbf{x}_\lambda - \sigma_\lambda \hat{\epsilon}_{\phi^\times}(\mathbf{x}_\lambda, \lambda)}_{\mathbf{v}(\mathbf{x}_\lambda, \lambda)},$$

which we rewrite as:

$$\frac{d\mathbf{x}_\lambda}{d\lambda} = \underbrace{\left( \frac{\alpha'_\lambda}{\alpha_\lambda} - \ell_\lambda \right) \mathbf{x}_\lambda}_{\text{linear part}} - \underbrace{(\sigma_\lambda \hat{\epsilon}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) - \ell_\lambda \mathbf{x}_\lambda)}_{\text{nonlinear part}}. \quad (9.7.1)$$

Here,  $\ell_\lambda$  is a  $D$ -dimensional free/undetermined variable that depends solely on  $\lambda$ . For notational convenience, we denote the linear and nonlinear parts as

$$\begin{aligned} \mathbf{L}(\lambda) &:= \frac{\alpha'_\lambda}{\alpha_\lambda} - \ell_\lambda, \\ \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) &:= \sigma_\lambda \hat{\epsilon}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) - \ell_\lambda \mathbf{x}_\lambda. \end{aligned} \quad (9.7.2)$$

Zheng *et al.* (2023) propose selecting  $\ell_\lambda$  by solving the following simple least-squares problem:

$$\ell_\lambda^* = \arg \min_{\ell_\lambda} \mathbb{E}_{\mathbf{x}_\lambda \sim p_\lambda^{\phi^\times}(\mathbf{x}_\lambda)} \|\nabla_{\mathbf{x}} \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda)\|_F^2, \quad (9.7.3)$$

where  $\|\cdot\|_F$  denotes the Frobenius norm, and  $p_\lambda^{\phi^\times}$  represents the marginal distribution of samples along the ODE trajectory in Equation (9.7.4) at  $\lambda$ .

We note that  $\ell_\lambda = \ell_\lambda^*$  can be solved analytically. This selection leverages preconditioning information from pre-trained models, conceptually making  $\mathbf{N}_{\phi^\times}$  less sensitive to errors in  $\mathbf{x}$  (as the Lipschitzness of  $\mathbf{N}_{\phi^\times}$ , which is approximately the  $\mathbf{x}$ -gradient, is reduced), and cancels the “linearity” of  $\mathbf{N}_{\phi^\times}$ .

### 9.7.2 Insight 2: Introducing Free Variables in Model Parameterization to Further Minimize Discretization Error

The PF-ODE exhibits a semilinear structure (see Equation (9.7.1) and Equation (9.7.2)). For the sake of notational clarity, we consider the following (abstract) formulation of the empirical PF-ODE:

$$\frac{d\mathbf{x}_\lambda}{d\lambda} = \mathbf{L}(\lambda)\mathbf{x}_\lambda + \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda). \quad (9.7.4)$$

**Motivation: Understanding Discretization Errors and Strategies for Their Minimization.** As usual, the exact solution to this empirical ODE over the interval  $[\lambda_s, \lambda_t]$  can be expressed using the variation-of-parameters formula:

$$\tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s}) = \mathcal{E}(\lambda_s \rightarrow \lambda_t)\mathbf{x}_{\lambda_s} + \int_{\lambda_s}^{\lambda_t} \mathcal{E}(\lambda \rightarrow \lambda_s)\mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) d\lambda, \quad (9.7.5)$$

where  $\mathcal{E}(s \rightarrow t) := e^{-\int_s^t \mathbf{L}(u) du}$ . By using the estimation

$$\mathbf{N}_{\phi^\times}(\mathbf{x}_{\lambda_s}, \lambda_s) \approx \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) \quad \text{for } \lambda \in [\lambda_s, \lambda_t],$$

we can obtain an approximate solution which is given by:

$$\tilde{\mathbf{x}}_{\lambda_t} = \mathcal{E}(\lambda_s \rightarrow \lambda_t)\mathbf{x}_{\lambda_s} + \int_{\lambda_s}^{\lambda_t} \mathcal{E}(\lambda \rightarrow \lambda_s)\mathbf{N}_{\phi^\times}(\mathbf{x}_{\lambda_s}, \lambda_s) d\lambda. \quad (9.7.6)$$

Subtracting Equation (9.7.5) and Equation (9.7.6), and expanding  $\mathbf{N}_{\phi^\times}(\mathbf{x}_{\lambda_s}, \lambda_s)$  in a Taylor series as:

$$\mathbf{N}_{\phi^\times}(\mathbf{x}_{\lambda_s}, \lambda_s) = \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) + (\lambda_s - \lambda)\mathbf{N}_{\phi^\times}^{(1)}(\mathbf{x}_\lambda, \lambda) + \mathcal{O}((\lambda_s - \lambda)^2),$$

we can quantify the first-order discretization error:

$$\tilde{\mathbf{x}}_{\lambda_t} - \tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s}) = \int_{\lambda_s}^{\lambda_t} \mathcal{E}(\lambda \rightarrow \lambda_s)(\lambda_s - \lambda)\mathbf{N}_{\phi^\times}^{(1)}(\mathbf{x}_\lambda, \lambda) d\lambda + \mathcal{O}(h^3),$$

where  $h := \lambda_t - \lambda_s$ . This observation reveals that the discretization error depends on  $\mathbf{N}_{\phi^\times}^{(1)}(\mathbf{x}_\lambda, \lambda)$ .

To reduce this error, Zheng *et al.* (2023) propose rewriting Equation (9.7.5) into an equivalent form using a new parameterization,  $\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda)$ , such that the error term retains the similar structure:

$$\tilde{\mathbf{x}}_{\lambda_t} - \tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s}) = \int_{\lambda_s}^{\lambda_t} \mathcal{E}(\lambda \rightarrow \lambda_s)(\lambda_s - \lambda) \mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_\lambda, \lambda) d\lambda + \mathcal{O}(h^3).$$

Furthermore, the  $\lambda$ -derivative of  $\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda)$  satisfies:

$$\mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_\lambda, \lambda) \propto \mathbf{N}_{\phi^\times}^{(1)}(\mathbf{x}_\lambda, \lambda) - (\mathbf{a}_\lambda \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) + \mathbf{b}_\lambda). \quad (9.7.7)$$

Here,  $\mathbf{a}_\lambda$  and  $\mathbf{b}_\lambda$  are free/undetermined variables.

The goal is then to determine the optimal values of  $\mathbf{a}_\lambda$  and  $\mathbf{b}_\lambda$  that minimize the discretization error by reducing  $\mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_\lambda, \lambda)$ . This can be accomplished by solving the following least squares optimization problem:

$$(\mathbf{a}_\lambda^*, \mathbf{b}_\lambda^*) = \arg \min_{\mathbf{a}_\lambda, \mathbf{b}_\lambda} \mathbb{E}_{\mathbf{x}_\lambda \sim p_\lambda^{\phi^\times}(\mathbf{x}_\lambda)} \left[ \|\mathbf{N}_{\phi^\times}^{(1)}(\mathbf{x}_\lambda, \lambda) - (\mathbf{a}_\lambda \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) + \mathbf{b}_\lambda)\|_2^2 \right]. \quad (9.7.8)$$

Notably, Equation (9.7.8) admits an analytical solution, depending on the pre-trained diffusion model, which can be precomputed.

**Realizing the Strategy for Minimizing Discretization Error.** We begin by considering a linearly transformed version of  $\mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda)$ , defined as:

$$\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda) := e^{-\int_{\lambda_s}^{\lambda} \mathbf{a}_u du} \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) - \int_{\lambda_s}^{\lambda} e^{-\int_{\lambda_s}^r \mathbf{a}_u du} \mathbf{b}_r dr. \quad (9.7.9)$$

We can then easily compute its  $\lambda$ -derivative given by:

$$\mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_\lambda, \lambda) = e^{-\int_{\lambda_s}^{\lambda} \mathbf{a}_u du} \left[ \mathbf{N}_{\phi^\times}^{(1)}(\mathbf{x}_\lambda, \lambda) - (\mathbf{a}_\lambda \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) + \mathbf{b}_\lambda) \right], \quad (9.7.10)$$

which takes the desired form as in Equation (9.7.7).

Using this,  $\mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda)$  can be rewritten as:

$$\begin{aligned} \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) &= e^{\int_{\lambda_s}^{\lambda} \mathbf{a}_u du} \underbrace{\left[ e^{-\int_{\lambda_s}^{\lambda} \mathbf{a}_u du} \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) - \int_{\lambda_s}^{\lambda} e^{-\int_{\lambda_s}^r \mathbf{a}_u du} \mathbf{b}_r dr \right]}_{\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda)} \\ &\quad + \int_{\lambda_s}^{\lambda} e^{-\int_{\lambda_s}^r \mathbf{a}_u du} \mathbf{b}_r dr \\ &= e^{\int_{\lambda_s}^{\lambda} \mathbf{a}_u du} \left[ \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda) + \int_{\lambda_s}^{\lambda} e^{-\int_{\lambda_s}^r \mathbf{a}_u du} \mathbf{b}_r dr \right] \end{aligned}$$

With this reformulation, we can rewrite Equation (9.7.5) and Equation (9.7.6) as:

$$\begin{aligned}\tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s}) &= \mathcal{E}(\lambda_s \rightarrow \lambda_t)\mathbf{x}_{\lambda_s} + \int_{\lambda_s}^{\lambda_t} \mathcal{E}(\lambda \rightarrow \lambda_s) e^{\int_{\lambda_s}^{\lambda} \mathbf{a}_u du} \\ &\quad \left[ \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda) + \int_{\lambda_s}^{\lambda} e^{-\int_{\lambda_s}^r \mathbf{a}_u du} \mathbf{b}_r dr \right] d\lambda\end{aligned}\tag{9.7.11}$$

$$\begin{aligned}\tilde{\mathbf{x}}_{\lambda_t} &= \mathcal{E}(\lambda_s \rightarrow \lambda_t)\mathbf{x}_{\lambda_s} + \int_{\lambda_s}^{\lambda_t} \mathcal{E}(\lambda \rightarrow \lambda_s) e^{\int_{\lambda_s}^{\lambda} \mathbf{a}_u du} \\ &\quad \left[ \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) + \int_{\lambda_s}^{\lambda} e^{-\int_{\lambda_s}^r \mathbf{a}_u du} \mathbf{b}_r dr \right] d\lambda\end{aligned}\tag{9.7.12}$$

Subtracting these two equations and employing the Taylor expansion:

$$\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) = \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda) + (\lambda_s - \lambda) \mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_\lambda, \lambda) + \mathcal{O}((\lambda_s - \lambda)^2),$$

we arrive at:

$$\begin{aligned}\tilde{\mathbf{x}}_{\lambda_t} - \tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s}) &= \int_{\lambda_s}^{\lambda_t} \mathcal{E}(\lambda \rightarrow \lambda_s)(\lambda_s - \lambda) e^{\int_{\lambda_s}^{\lambda} \mathbf{a}_u du} \mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_\lambda, \lambda) d\lambda + \mathcal{O}(h^3) \\ &= \int_{\lambda_s}^{\lambda_t} \mathcal{E}(\lambda \rightarrow \lambda_s)(\lambda_s - \lambda) \left[ \mathbf{N}_{\phi^\times}^{(1)}(\mathbf{x}_\lambda, \lambda) - (\mathbf{a}_\lambda \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) + \mathbf{b}_\lambda) \right] d\lambda + \mathcal{O}(h^3)\end{aligned}$$

Here, the last equality follows from Equation (9.7.10), which is central to our design, and it cancels out the factor  $e^{\int_{\lambda_s}^{\lambda} \mathbf{a}_u du}$ .

Thus, by solving Equation (9.7.8), we can determine the optimal coefficients  $(\mathbf{a}_\lambda^*, \mathbf{b}_\lambda^*)$ , effectively minimizing the discretization error.

### 9.7.3 Combining Both Insights.

We now summarize the discussion so far.

**Procedure in Theory.** For any  $\lambda$ , we first compute  $\ell_\lambda^*$  analytically by solving the least squares problem in Equation (9.7.3):

$$\ell_\lambda^* = \arg \min_{\ell_\lambda} \mathbb{E}_{\mathbf{x}_\lambda \sim p_\lambda^{\phi^\times}(\mathbf{x}_\lambda)} \|\nabla_{\mathbf{x}} \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda)\|_F^2,$$

where  $\mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda)$  is defined in Equation (9.7.2). Next, we compute  $(\mathbf{a}_\lambda^*, \mathbf{b}_\lambda^*)$  analytically by solving the least squares problem in Equation (9.7.8), with

$\ell_\lambda = \ell_\lambda^*$  fixed:

$$(\mathbf{a}_\lambda^*, \mathbf{b}_\lambda^*) = \arg \min_{\mathbf{a}_\lambda, \mathbf{b}_\lambda} \mathbb{E}_{\mathbf{x}_\lambda \sim p_\lambda^{\phi^\times}(\mathbf{x}_\lambda)} \left[ \|\mathbf{N}_{\phi^\times}^{(1)}(\mathbf{x}_\lambda, \lambda) - (\mathbf{a}_\lambda \mathbf{N}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) + \mathbf{b}_\lambda) \|_2^2 \right].$$

Consequently, the resulting  $\tilde{\mathbf{x}}_{\lambda_t}$ , as defined in Equation (9.7.12) (with  $\ell_\lambda$ ,  $\mathbf{a}_\lambda$ , and  $\mathbf{b}_\lambda$  replaced by  $\ell_\lambda^*$ ,  $\mathbf{a}_\lambda^*$ , and  $\mathbf{b}_\lambda^*$  in Equation (9.7.9)), serves as the desired estimation of  $\tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s})$ .

**Implementation Considerations.** Although  $\ell_\lambda^*$ ,  $\mathbf{a}_\lambda^*$ , and  $\mathbf{b}_\lambda^*$  have analytical solutions involving the Jacobian-vector product of the pre-trained diffusion model  $\epsilon_{\phi^\times}$  (as detailed in Appendix C.1.1 of Zheng *et al.* (2023)), their computation requires evaluating expectations over  $p_\lambda^{\phi^\times}$ .

In practice, these quantities are estimated via a Monte Carlo (MCMC) approach. Specifically, a batch of datapoints  $\mathbf{x}_\lambda \sim p_\lambda^{\phi^\times}$  (roughly 1K-4K samples) is drawn by applying an alternative solver (e.g., the 200-step DPM-Solver++ (Lu *et al.*, 2022c)) to Equation (9.4.5), after which the relevant terms related to  $\epsilon_{\phi^\times}$  are computed analytically. Importantly, these statistics can all be pre-computed, ensuring that when DPM-Solver-v3 is applied, the computational overhead associated with these calculations is avoided.

#### 9.7.4 Higher-Order DPM-Solver-v3

The precomputed statistics  $\ell_\lambda^*$ ,  $\mathbf{a}_\lambda^*$ , and  $\mathbf{b}_\lambda^*$ , which are derived by analyzing the first-order discretization error, can also be employed to construct higher-order solvers (see also Section 9.7.5 for further interpretation).

To obtain the  $(n+1)$ -th order approximation of Equation (9.7.11), we utilize the  $n$ -th order Taylor expansion of  $\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda)$  with respect to  $\lambda$  at  $\lambda_s$ , neglecting terms of order  $\mathcal{O}((\lambda_s - \lambda)^{(n+1)})$ . This allows us to approximate  $\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda)$  for  $\lambda \in [\lambda_s, \lambda_t]$ :

$$\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda) \approx \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) + \sum_{k=1}^n \frac{(\lambda - \lambda_s)^k}{k!} \mathbf{N}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s),$$

where this approximation leads to the estimated solution to Equation (9.7.11):

$$\begin{aligned}
\tilde{\mathbf{x}}_{\lambda_t} &= \mathcal{E}(\lambda_s \rightarrow \lambda_t) \mathbf{x}_{\lambda_s} + \int_{\lambda_s}^{\lambda_t} \underbrace{\mathcal{E}(\lambda \rightarrow \lambda_s) e^{\int_{\lambda_s}^{\lambda} \mathbf{a}_u^* du}}_{=: \mathbf{E}(\lambda_s \rightarrow \lambda)} \\
&\quad \cdot \left[ \sum_{k=0}^n \frac{(\lambda - \lambda_s)^k}{k!} \mathbf{N}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s) + \underbrace{\int_{\lambda_s}^{\lambda} e^{-\int_{\lambda_s}^r \mathbf{a}_u^* du} \mathbf{b}_r^* dr}_{=: \mathbf{B}(\lambda_s \rightarrow \lambda)} \right] d\lambda. \\
&= \mathcal{E}(\lambda_s \rightarrow \lambda_t) \mathbf{x}_{\lambda_s} + \int_{\lambda_s}^{\lambda_t} \mathbf{E}(\lambda_s \rightarrow \lambda) \mathbf{B}(\lambda_s \rightarrow \lambda) d\lambda \tag{9.7.13} \\
&\quad + \sum_{k=0}^n \mathbf{N}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s) \int_{\lambda_s}^{\lambda_t} \mathbf{E}(\lambda_s \rightarrow \lambda) \frac{(\lambda - \lambda_s)^k}{k!} d\lambda.
\end{aligned}$$

In a manner similar to the derivation of higher-order DPM in Section 9.4.3, for the  $(n+1)$ -th order approximation, we utilize the finite difference of  $\mathbf{N}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_\lambda, \lambda)$  at the previous  $n+1$  steps,  $\lambda_{i_n}, \dots, \lambda_{i_1}, \lambda_{i_0} := \lambda_s$ , to estimate each  $\mathbf{N}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s)$ . This approach is designed to match the coefficients in the Taylor expansions.

Below, we demonstrate this method with an example for  $n = 2$ :

### Example: Estimating Higher-Order Derivatives ( $n = 2$ Case).

When  $n = 2$ , we aim to estimate the derivatives  $\mathbf{N}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_\lambda, \lambda)$  with  $k = 1, 2$  with previous 3 timesteps  $\lambda_{i_2}, \lambda_{i_1}, \lambda_s$ .

**Linear System for Approximated Derivatives.** Let  $\delta_k = \lambda_{i_k} - \lambda_s$  ( $k = 1, 2$ ). We expand  $\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda)$  around  $\lambda_s$  using a Taylor series. Evaluating at  $\lambda = \lambda_{i_1}$  and  $\lambda = \lambda_{i_2}$ , and rearranging to isolate the derivative terms, we obtain:

$$\begin{aligned}
&\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_{i_1}}, \lambda_{i_1}) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) \\
&\approx \delta_1 \mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_{\lambda_s}, \lambda_s) + \frac{\delta_1^2}{2!} \mathbf{N}_{\phi^\times}^{\text{new},(2)}(\mathbf{x}_{\lambda_s}, \lambda_s), \\
&\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_{i_2}}, \lambda_{i_2}) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) \\
&\approx \delta_2 \mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_{\lambda_s}, \lambda_s) + \frac{\delta_2^2}{2!} \mathbf{N}_{\phi^\times}^{\text{new},(2)}(\mathbf{x}_{\lambda_s}, \lambda_s).
\end{aligned}$$

Here, higher-order terms  $\mathcal{O}(\delta_1^3)$  and  $\mathcal{O}(\delta_2^3)$  are neglected, respectively. This

forms the linear system:

$$\begin{bmatrix} \delta_1 & \delta_1^2 \\ \delta_2 & \delta_2^2 \end{bmatrix} \begin{bmatrix} \frac{\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_{\lambda_s}, \lambda_s)}{1!} \\ \frac{\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(2)}(\mathbf{x}_{\lambda_s}, \lambda_s)}{2!} \end{bmatrix} = \begin{bmatrix} \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_{i_1}}, \lambda_{i_1}) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) \\ \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_{i_2}}, \lambda_{i_2}) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) \end{bmatrix}$$

with the approximated derivatives  $\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s)$  to be solved; hence,

$$\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s) \approx \mathbf{N}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s).$$

**Solving for the Approximated Derivatives  $\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(k)}$ .** Let:

$$\mathbf{R}_2 = \begin{bmatrix} \delta_1 & \delta_1^2 \\ \delta_2 & \delta_2^2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda_{i_2}) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda_s) \\ \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda_{i_1}) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda_s) \end{bmatrix}.$$

The approximated derivatives are:

$$\begin{bmatrix} \frac{\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_{\lambda_s}, \lambda_s)}{1!} \\ \frac{\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(2)}(\mathbf{x}_{\lambda_s}, \lambda_s)}{2!} \end{bmatrix} = \mathbf{R}_2^{-1} \mathbf{b},$$

which can be computed explicitly. ■

Building upon the spirit of the illustrative example, we can easily extend it to the case of the  $k$ -th derivatives for  $k \leq n$  with a general  $n$ :

$$\underbrace{\begin{bmatrix} \delta_1 & \delta_1^2 & \cdots & \delta_1^n \\ \delta_2 & \delta_2^2 & \cdots & \delta_2^n \\ \vdots & \vdots & \ddots & \vdots \\ \delta_n & \delta_n^2 & \cdots & \delta_n^n \end{bmatrix}}_{\mathbf{R}_n} \begin{bmatrix} \frac{\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_{\lambda_s}, \lambda_s)}{1!} \\ \frac{\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(2)}(\mathbf{x}_{\lambda_s}, \lambda_s)}{2!} \\ \vdots \\ \frac{\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(n)}(\mathbf{x}_{\lambda_s}, \lambda_s)}{n!} \end{bmatrix} = \begin{bmatrix} \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_{i_1}}, \lambda_{i_1}) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) \\ \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_{i_2}}, \lambda_{i_2}) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) \\ \vdots \\ \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_{i_n}}, \lambda_{i_n}) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) \end{bmatrix}. \tag{9.7.14}$$

By inverting the Vandermonde matrix  $\mathbf{R}_n$ , we can analytically solve for the approximated derivatives  $\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s)$  for  $k \leq n$ . Therefore, by replacing  $\mathbf{N}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s)$  in Equation (9.7.13) with  $\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s)$ , we obtain an approximated solution, which we still denote as  $\tilde{\mathbf{x}}_{\lambda_t}$ :

$$\begin{aligned}\tilde{\mathbf{x}}_{\lambda_t} &= \mathcal{E}(\lambda_s \rightarrow \lambda_t) \mathbf{x}_{\lambda_s} + \int_{\lambda_s}^{\lambda_t} \mathbf{E}(\lambda_s \rightarrow \lambda) \mathbf{B}(\lambda_s \rightarrow \lambda) d\lambda \\ &\quad + \sum_{k=0}^n \tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s) \int_{\lambda_s}^{\lambda_t} \mathbf{E}(\lambda_s \rightarrow \lambda) \frac{(\lambda - \lambda_s)^k}{k!} d\lambda.\end{aligned}\tag{9.7.15}$$

### 9.7.5 More Interpretations of DPM-Solver-v3

**Minimizing First-order Discretization Error Can Help Higher-order Solvers.**  $\mathbf{a}_\lambda^*$  and  $\mathbf{b}_\lambda^*$  are derived by reducing the first-order discretization error; however, in theory, they can also contribute to controlling errors in higher-order solvers. This result is summarized in the following proposition.

**Proposition 9.7.1: Reducing First-Order Discretization Error Helps Higher-Order Solvers**

Starting from the same initial condition  $\mathbf{x}_{\lambda_s}$ , let the approximated solution  $\tilde{\mathbf{x}}_{\lambda_t}$  be defined as in Equation (9.7.15), and the exact solution  $\tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s})$  as in Equation (9.7.11). Then the discretization error is given by:

$$\begin{aligned}\tilde{\mathbf{x}}_{\lambda_t} - \tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s}) &= \int_{\lambda_s}^{\lambda_t} \left( \int_{\lambda_s}^{\lambda} \mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_u, u) du \right) \mathbf{E}(\lambda_s \rightarrow \lambda) d\lambda \\ &\quad + \sum_{k=1}^n \left( \sum_{j=1}^n (\mathbf{R}_n^{-1})_{kj} \int_{\lambda_s}^{\lambda_{ij}} \mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_\lambda, \lambda) d\lambda \right) \\ &\quad \cdot \int_{\lambda_s}^{\lambda_t} \mathbf{E}(\lambda_s \rightarrow \lambda) \frac{(\lambda - \lambda_s)^k}{k!} d\lambda.\end{aligned}$$

**Proof for Proposition.**

$\tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s})$  can be rewritten into the following expression:

$$\begin{aligned}\tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s}) &= \mathcal{E}(\lambda_s \rightarrow \lambda_t) \mathbf{x}_{\lambda_s} + \int_{\lambda_s}^{\lambda_t} \mathbf{E}(\lambda_s \rightarrow \lambda) \mathbf{B}(\lambda_s \rightarrow \lambda) d\lambda \\ &\quad + \int_{\lambda_s}^{\lambda_t} \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda) \mathbf{E}(\lambda_s \rightarrow \lambda) d\lambda.\end{aligned}$$

Subtracting Equation (9.7.15) by  $\tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s})$ :

$$\begin{aligned}\tilde{\mathbf{x}}_{\lambda_t} - \tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s}) &= \int_{\lambda_s}^{\lambda_t} \left( \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) \right) \mathbf{E}(\lambda_s \rightarrow \lambda) d\lambda \\ &\quad + \sum_{k=1}^n \tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s) \int_{\lambda_s}^{\lambda_t} \mathbf{E}(\lambda_s \rightarrow \lambda) \frac{(\lambda - \lambda_s)^k}{k!} d\lambda.\end{aligned}$$

By inverting the matrix  $\mathbf{R}_n$  in Equation (9.7.14), the solution for any  $k = 1, \dots, n$  is given by:

$$\tilde{\mathbf{N}}_{\phi^\times}^{\text{new},(k)}(\mathbf{x}_{\lambda_s}, \lambda_s) = \sum_{j=1}^n (\mathbf{R}_n^{-1})_{kj} \left( \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_{i_j}}, \lambda_{i_j}) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) \right).$$

The results are obtained by applying the Fundamental Theorem of Calculus, yielding

$$\begin{aligned}\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) &= \int_{\lambda_s}^{\lambda} \mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_u, u) du \\ \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_{i_j}}, \lambda_{i_j}) - \mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_{\lambda_s}, \lambda_s) &= \int_{\lambda_s}^{\lambda_{i_j}} \mathbf{N}_{\phi^\times}^{\text{new},(1)}(\mathbf{x}_\lambda, \lambda) d\lambda.\end{aligned}$$

From the above proposition and Equation (9.7.10), controlling  $\|\mathbf{N}_{\phi^\times}^{\text{new},(1)}\|_2$  reduces  $\|\tilde{\mathbf{x}}_{\lambda_t} - \tilde{\Psi}_{\lambda_s \rightarrow \lambda_t}(\mathbf{x}_{\lambda_s})\|_2$ , assuming sufficient smoothness. ■

**Expressive Power of the Generalized Parameterization  $\mathbf{N}_{\phi^\times}^{\text{new}}$ .** Utilizing Equation (9.7.2) and Equation (9.7.9), we can rewrite  $\mathbf{N}_{\phi^\times}^{\text{new}}$  in the following form:

$$\begin{aligned}\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda) &= \sigma_\lambda e^{-\int_{\lambda_s}^{\lambda} \mathbf{a}_u du} \hat{\epsilon}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) - \ell_\lambda e^{-\int_{\lambda_s}^{\lambda} \mathbf{a}_u du} \mathbf{x}_\lambda \\ &\quad - \int_{\lambda_s}^{\lambda} e^{-\int_{\lambda_s}^r \mathbf{a}_u du} \mathbf{b}_r dr,\end{aligned}\tag{9.7.16}$$

which is conceptually of the following form:

$$\mathbf{T}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) := \boldsymbol{\alpha}(\lambda) \hat{\epsilon}_{\phi^\times}(\mathbf{x}_\lambda, \lambda) + \beta(\lambda) \mathbf{x}_\lambda + \gamma(\lambda).\tag{9.7.17}$$

Indeed, for a fixed  $\lambda$ ,  $\Psi_{\phi^\times}(\mathbf{x}_\lambda, \lambda)$  can be expressed in terms of  $\mathbf{N}_{\phi^\times}^{\text{new}}(\mathbf{x}_\lambda, \lambda)$  through a linear transformation depending on  $\lambda_s$  (see (Zheng *et al.*, 2023)'s Appendix I.1 for more details).

### 9.7.6 Connection of DPM-Solver-v3 to Other Methods

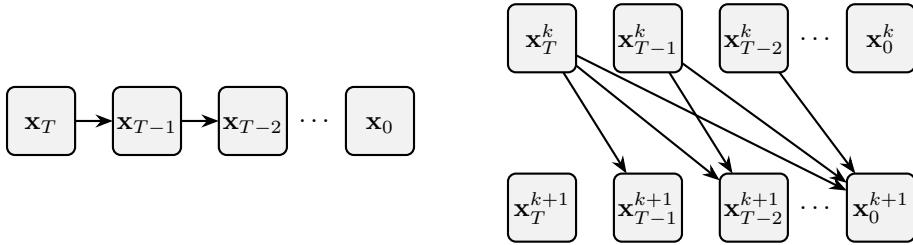
**DPM-Solver-v3's  $\mathbf{N}_{\phi^\times}^{\text{new}}$  is a General Parametrization.** By comparing DPM-Solver-v3 with previous ODE formulations and their corresponding  $\epsilon$ -/ $\mathbf{x}$ -prediction, we can easily identify that they are special cases of our approach by setting  $\ell_\lambda$ ,  $\mathbf{a}_\lambda$ , and  $\mathbf{b}_\lambda$  to specific values:

- $\epsilon$ -prediction:  $(\ell_\lambda, \mathbf{a}_\lambda, \mathbf{b}_\lambda) = (\mathbf{0}_D, -\mathbf{1}_D, \mathbf{0}_D)$
- $\mathbf{x}$ -prediction:  $(\ell_\lambda, \mathbf{a}_\lambda, \mathbf{b}_\lambda) = (\mathbf{1}_D, \mathbf{0}_D, \mathbf{0}_D)$

**First-Order Discretization as an Improved DDIM.** Since  $\mathbf{N}_{\phi^\times}^{\text{new}}$  represents neither noise nor data parameterization but an improved parameterization aimed at minimizing the first-order discretization error, the first-order DPM-Solver-v3 update in Equation (9.7.12) differs from the DDIM update in Equation (9.2.2):

$$\mathbf{x}_{t_{i-1} \rightarrow t_i} = \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}} \tilde{\mathbf{x}}_{t_{i-1}} - \alpha_{t_i} \left( \frac{\sigma_{t_{i-1}}}{\alpha_{t_{i-1}}} - \frac{\sigma_{t_i}}{\alpha_{t_i}} \right) \boldsymbol{\epsilon}_{\phi^\times}(\tilde{\mathbf{x}}_{t_{i-1}}, t_{i-1}).$$

## 9.8 (Optional) ParaDiGMs



(a) Sequential sampling by time-stepping estimation  
in generation process.

(b) Picard iterations with skip dependencies.

**Figure 9.4: Comparisons of two computation graphs.** Left: conventional time-stepping ODE solving, where the solution is propagated sequentially across time. Right: Picard iteration, which enables parallel computation by updating all time nodes simultaneously using the results from the previous iteration, thereby avoiding the strictly sequential nature of time-stepping.

### 9.8.1 From Time-Stepping to Time-Parallel Solver

In the previous sections, we focused on the time-stepping approach, which estimates the trajectory by evolving from the prior time  $T$  toward an arbitrary  $t \in [0, T]$ . Let

$$\mathbf{v}_{\phi^\times}(\mathbf{x}, t) := \mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g^2(t)\mathbf{s}_{\phi^\times}(\mathbf{x}, t)$$

denote the empirical PF-ODE drift from a pre-trained diffusion model. The exact evolution from  $T$  to any intermediate time  $t$  is:

$$\tilde{\Psi}_{T \rightarrow t}(\mathbf{x}(T)) = \mathbf{x}(T) + \int_T^t \mathbf{v}_{\phi^\times}(\mathbf{x}(\tau), \tau) d\tau, \quad \mathbf{x}(T) \sim p_{\text{prior}}. \quad (9.8.1)$$

Time-stepping schemes approximate this integral using discrete updates based on past timesteps.

In this section, we turn to the *time-parallel* approach, exemplified by *ParaDiGMS*, which builds on classical Picard iteration to enable parallel integration across time. The key idea behind ParaDiGMS is to *trade computational resources for faster simulation*.

### 9.8.2 Methodology of ParaDiGMS

**From Trajectories to Picard Iteration as a Fixed-Point Update.** The integral expression in Equation (9.8.1) can be understood as a map that takes in an entire trajectory and produces a new one. Formally, given any candidate trajectory  $\{\mathbf{y}(\tau)\}_{\tau \in [0, T]}$ , we define the operator  $\mathcal{L}$  by

$$(\mathcal{L}[\mathbf{y}(\cdot)])(t) = \mathbf{y}(T) + \int_T^t \mathbf{v}_{\phi^\times}(\mathbf{y}(\tau), \tau) d\tau, \quad t \in [0, T].$$

That is,  $\mathcal{L}$  takes the terminal point  $\mathbf{y}(T)$  and extends it backward in time by integrating the prescribed velocity field  $\mathbf{v}_{\phi^\times}$  along the path.

A true solution trajectory  $\mathbf{x}^*(\cdot)$  of the PF-ODE is precisely one that remains unchanged under this mapping. In other words,  $\mathbf{x}^*(\cdot)$  is a *fixed point* of  $\mathcal{L}$ :

$$\mathbf{x}^*(t) = \mathcal{L}[\mathbf{x}^*(\cdot)](t) \iff \mathbf{x}^*(t) = \mathbf{x}^*(T) + \int_T^t \mathbf{v}_{\phi^\times}(\mathbf{x}^*(\tau), \tau) d\tau.$$

This reformulation shifts the problem from solving an ODE step by step to finding a trajectory that is consistent with the operator  $\mathcal{L}$ .

Building on the operator view above, once we have the trajectory-to-trajectory map  $\mathcal{L}$ , a natural way to find its fixed point is by successive substitution (Picard iteration): apply  $\mathcal{L}$  repeatedly on while evaluating the integral using the trajectory from the previous iterate. More precisely, starting from any initial path  $\mathbf{x}^{(0)}(\cdot)$  (in practice, a constant path  $\mathbf{x}^{(0)}(t) \equiv \mathbf{x}^{(0)}(T)$  with a fixed  $\mathbf{x}^{(0)}(T) \sim p_{\text{prior}}$ ), the update reads

$$\begin{aligned} \mathbf{x}^{(k+1)}(t) &:= \mathcal{L}^{(k)}[\mathbf{x}^{(0)}(\cdot)](t) \\ &= \mathbf{x}^{(k)}(T) + \int_T^t \mathbf{v}_{\phi^\times}(\mathbf{x}^{(k)}(\tau), \tau) d\tau, \quad k = 0, 1, 2, \dots \end{aligned} \tag{9.8.2}$$

This formula preserves the correct time  $T$  anchoring: the iterate always starts from the prior-drawn state  $\mathbf{x}^{(k)}(T)$ , and then accumulates the drift as time decreases from  $T$  down to  $t$ .

**Discrete Picard on a  $T$  to 0 Grid.** To turn Equation (9.8.2) into a practical algorithm, we place a uniform, decreasing grid on  $[0, T]$  by choosing a step count  $N$ , setting  $\Delta t := T/M$ , and defining

$$t_j := T - j\Delta t, \quad j = 0, 1, \dots, M,$$

so  $t_0 = T$  and  $t_M = 0$ . Denote sampled iterates by

$$\mathbf{x}_j^{(k)} := \mathbf{x}^{(k)}(t_j).$$

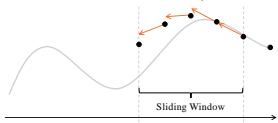
Because the grid runs reversely in time, the integral from  $T$  to  $t_j$  has negative orientation. Approximating it by left endpoints on the partition  $\{[t_{i+1}, t_i]\}_{i=0}^{j-1}$  gives

$$\int_T^{t_j} \mathbf{v}_{\phi^\times}(\mathbf{x}^{(k)}(\tau), \tau) d\tau \approx -\Delta t \sum_{i=0}^{j-1} \mathbf{v}_{\phi^\times}(\mathbf{x}_{t_i}^{(k)}, t_i),$$

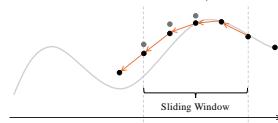
since each small integral over  $[t_{i+1}, t_i]$  equals  $-\int_{t_i}^{t_{i+1}} \cdot d\tau$ . Substituting this approximation into Equation (9.8.2) yields the discrete Picard update

$$\mathbf{x}_j^{(k+1)} = \mathbf{x}_0^{(k)} - \underbrace{\Delta t \sum_{i=0}^{j-1} \mathbf{v}_{\phi^\times}(\mathbf{x}_i^{(k)}, t_i)}_{\text{cumulative sum of drifts}}, \quad j = 1, \dots, M. \quad (9.8.3)$$

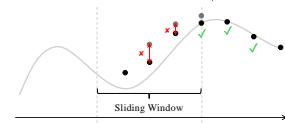
This scheme is simple and parallel-friendly: each drift evaluation  $\mathbf{v}_{\phi^\times}(\mathbf{x}_i^{(k)}, t_i)$  depends only on the previous iterate at the same time node  $t_i$ , so all  $i = 0, \dots, j-1$  evaluations can be computed independently across the grid. The integral is then recovered by a cumulative sum, performed either serially or via a parallel *prefix-sum* (*scan/sliding windows*).



**Figure 9.5:** Compute the drift of  $\mathbf{x}_{\ell:\ell+p}^{(k)}$  on a batch window of size  $p = 4$ , in parallel



**Figure 9.6:** Update the values to  $\mathbf{x}_{\ell:\ell+p}^{(k+1)}$  using the cumulative drift of points in the window



**Figure 9.7:** Determine how far to slide the window forward, based on the error  $\|\mathbf{x}_i^{(k+1)} - \mathbf{x}_i^{(k)}\|^2$ .

**Sliding Windows and Parallel Evaluation.** The discrete Picard update Equation (9.8.3) expresses each  $\mathbf{x}_j^{(k+1)}$  as the left-anchored value  $\mathbf{x}_0^{(k)}$  minus a cumulative sum of drifts. To limit memory and exploit parallel hardware, it is convenient to apply the same idea locally on short, sliding blocks of indices.

Fix a window length  $p$  and a left index  $\ell$ ; the window then covers  $j = \ell, \dots, \ell + p$  with  $t_\ell > t_{\ell+1} > \dots > t_{\ell+p}$ . During iteration  $k$ :

**Step 1. Parallel Drift Evaluation on the Window.** Compute, in parallel and using only the previous iterate,

$$\mathbf{v}_{\phi^\times}(\mathbf{x}_{\ell+i}^{(k)}, t_{\ell+i}), \quad i = 0, 1, \dots, p-1.$$

These are the  $p$  local increments needed to advance from the left edge  $t_\ell$  across the window.

**Step 2. Left-Anchored Cumulative Updates.** Form the windowed updates by anchoring at  $j = \ell$  and accumulating the drift across subintervals:

$$\mathbf{x}_{\ell+j+1}^{(k+1)} = \mathbf{x}_\ell^{(k)} - \Delta t \sum_{i=0}^j \mathbf{v}_{\phi^\times}(\mathbf{x}_{\ell+i}^{(k)}, t_{\ell+i}), \quad j = 0, 1, \dots, p-1. \quad (9.8.4)$$

This is precisely Equation (9.8.3) restricted to the window, with the minus sign reflecting the decreasing time direction. The inner sum is a prefix-sum (scan) over the windowed drifts, so all partial sums can be produced efficiently on parallel hardware.

**Step 3. Progress Control and Window Advance.** Having formed the left-anchored cumulative updates on the current window (Step 2), we now decide how far to slide that window. We measure local convergence by the pointwise Picard change

$$\text{error}_j := \|\mathbf{x}_{\ell+j}^{(k+1)} - \mathbf{x}_{\ell+j}^{(k)}\|^2, \quad j = 1, \dots, p-1,$$

and compare it against prescribed tolerances  $\text{tol}_{\ell+j}$ . That is,  $\text{error}_j$  measures how much the iterate at node  $\ell + j$  changed during the last Picard update. If this number is small, it indicates local agreement between the two successive approximations and hence local convergence of the fixed-point iteration at that node. If it is large, that node has not settled yet and needs more Picard smoothing.

The `stride` is chosen as the first index in the window that fails this test (or the full window length  $p$  if none fail):

$$\text{stride} := \min \left( \{ j \geq 1 : \text{error}_j > \text{tol}_{\ell+j} \} \cup \{p\} \right).$$

We then slide the window by setting  $\ell \leftarrow \ell + \text{stride}$ . In words: we accept all nodes from the left edge up to (but not including) the first one that has not converged; if all nodes have converged, we accept the entire window. We then slide the window by that many accepted nodes,  $\ell \leftarrow \ell + \text{stride}$ , and continue. This advances by at most the window length  $p$ , never skipping any node that has not met its tolerance. If sliding would overrun the grid end  $M$ , we truncate the window to  $p \leftarrow \min\{p, M - \ell\}$  and proceed.

When the window moves forward it uncovers new time nodes that have no values yet. To start Picard iteration there, we simply copy the value from

**Algorithm 8** ParaDiGMS with Sliding Windows

---

**Input:** Drift  $\mathbf{v}_{\phi^\times}(\mathbf{x}, t)$ ;  $\{t_j\}_{j=0}^M$ ; window length  $p$ ;  $\{\text{tol}_j\}_{j=1}^M$

**Output:** Approximate trajectory  $\{\mathbf{x}_j^{(k)}\}_{j=0}^M$  with  $\mathbf{x}_M^{(k)}$  at  $t = 0$

- 1:  $k \leftarrow 0$ ,  $\ell \leftarrow 0$
- 2: Sample  $\mathbf{x}_0^{(0)} \sim p_{\text{prior}}$ ; set  $\mathbf{x}_j^{(0)} \leftarrow \mathbf{x}_0^{(0)}$  for  $j = 1, \dots, \min(p, M)$  ▷ constant extrapolation
- 3: **while**  $\ell < M$  **do**
- 4:    $J \leftarrow \min(p, M - \ell)$  ▷ current window length
- 5:   **Step 1: Parallel**
- 6:   For  $i = 0, \dots, J - 1$ :  $g_i \leftarrow \mathbf{v}_{\phi^\times}(\mathbf{x}_{\ell+i}^{(k)}, t_{\ell+i})$  ▷ drifts from previous iterate (Picard freezing)
- 7:   Compute prefix sums  $S_j \leftarrow \sum_{i=0}^j g_i$  for  $j = 0, \dots, J - 1$  ▷ scan over windowed drifts
- 8:   **Step 2: Cumulative Updates**
- 9:   For  $j = 0, \dots, J - 1$ :  $\mathbf{x}_{\ell+j+1}^{(k+1)} \leftarrow \mathbf{x}_{\ell+j}^{(k)} - \Delta t S_j$  ▷ left-anchored update; cf. Equation (9.8.4)
- 10:   **Step 3: Progress Control and Window Advance**
- 11:   For  $j = 1, \dots, J - 1$ :  $\text{error}_j \leftarrow \|\mathbf{x}_{\ell+j}^{(k+1)} - \mathbf{x}_{\ell+j}^{(k)}\|^2$  ▷ pointwise Picard change
- 12:    $\text{stride} \leftarrow \min \left( \{ j \in \{1, \dots, J - 1\} : \text{error}_j > \text{tol}_{\ell+j} \} \cup \{J\} \right)$
- 13:   **Initialize New Nodes**
- 14:   For  $r = 1, \dots, \text{stride}$ :  $\mathbf{x}_{\ell+J+r}^{(k+1)} \leftarrow \mathbf{x}_{\ell+J}^{(k+1)}$  ▷ constant extrapolation into newly exposed indices
- 15:    $\ell \leftarrow \ell + \text{stride}$ ;  $k \leftarrow k + 1$
- 16: **end while**
- 17: **return**  $\{\mathbf{x}_j^{(k)}\}_{j=0}^M$

---

the left boundary of the window and use it as an initial guess. This “constant extrapolation” is cheap and stable, and will be corrected by later updates. If desired, one can replace it by more accurate guesses, such as linear or polynomial extrapolations from past points.

This completes the procedure of ParaDiGMS. We summarize the algorithm in Algorithm 8.

### 9.8.3 Relation to Time-Stepping Solvers

**Selection of Sliding Window Size.** To place the sliding window scheme in context, note first what happens at the smallest window size. When  $p = 1$ , the

window contains a single step, so Equation (9.8.4) collapses to a first-order time-stepping update of the PF-ODE. The method reduces to, for instance, DDIM, if we use the same way of writing the ODE (e.g. data vs. noise prediction) and choose the same schedule of discrete timesteps as in DDIM.

Increasing  $p$  expands parallelism (more nodes advanced per window) without changing the overall step count  $N$ . Consequently, sample quality continues to be determined by the base discretization (choice of grid/parameterization and per-step formula) together with Picard convergence on each window, which we monitor via the local tolerances.

**Compatibility with Higher-Order Solvers (e.g., DPM).** The sliding-window Picard structure controls *how* increments are computed (in parallel and accumulated by a scan), not *which* local formula defines those increments. Consequently, one may replace the left-endpoint rule by any consistent higher-order quadrature without changing the parallel layout. For example, a trapezoidal variant of Equation (9.8.4) reads

$$\mathbf{x}_{\ell+j+1}^{(k+1)} = \mathbf{x}_\ell^{(k)} - \Delta t \left[ \frac{1}{2} \mathbf{v}_{\phi^\times}(\mathbf{x}_\ell^{(k)}, t_\ell) + \sum_{i=1}^{j-1} \mathbf{v}_{\phi^\times}(\mathbf{x}_{\ell+i}^{(k)}, t_{\ell+i}) + \frac{1}{2} \mathbf{v}_{\phi^\times}(\mathbf{x}_{\ell+j}^{(k)}, t_{\ell+j}) \right],$$

where all drifts are still taken from the previous Picard iterate, so the per-node evaluations remain independent and the inner sum remains a prefix-sum.

Likewise, multistep or exponential-integrator updates used by DPM solvers family (e.g., DPM-Solver++ 2M in log-SNR time) can be inserted by replacing each windowed increment with the corresponding higher-order linear combination of past model evaluations ( $\mathbf{x}$ - or  $\epsilon$ -predictions with precomputed coefficients). The scan then accumulates those weighted increments across the window exactly as before. In short: the parallel scheme is independent of the solver (discretization) choice to approximate the integral. Accuracy comes from the base solver; the windowed prefix-sum just makes it fast.

## 9.9 Closing Remarks

This chapter has confronted one of the most significant practical limitations of diffusion models: their slow, iterative sampling process. We have explored a powerful class of training-free solutions that accelerate generation by leveraging the rich field of numerical methods for differential equations. The core strategy has been to more efficiently solve the PF-ODE, which defines the deterministic generative trajectory from noise to data:

1. We began with the foundational DDIM, which can be understood as a first-order exponential Euler method.
2. We then moved to higher-order multi-step methods like DEIS, which improve accuracy by using a history of past evaluations.
3. Finally, we examined the highly efficient DPM-Solver family, which achieves remarkable performance by introducing a crucial log-SNR time reparameterization.

Through these sophisticated solvers, the number of function evaluations (NFEs) required for high-quality generation has been dramatically reduced from hundreds or thousands to as few as 10-20, making diffusion models significantly more practical.

However, these training-free methods are still fundamentally iterative. They approximate a continuous path step-by-step. This raises a natural and ambitious question: *can we achieve high-quality generation in just one or a very few discrete steps?*

The final part D of this monograph will explore this question through training-based acceleration. We will investigate two main strategies:

1. First, in Chapter 10, we will examine distillation-based methods, where a fast *student* generator is trained to replicate the output of a slow, pre-trained *teacher* diffusion model in far fewer steps.
2. Then, in Chapter 11, we will push this idea further by exploring methods that learn fast, few-step generators from scratch, such as Consistency Training, which define a standalone training principle without relying on any pre-trained model.

This shift from improving the solver to learning the solution map itself represents the frontier of efficient generative modeling, aiming to combine the quality of diffusion models with the speed of one-step generators.

## **Part D**

# **Toward Learning Fast Diffusion-Based Generators**

# 10

---

## Distillation-Based Methods for Fast Sampling

---

This chapter introduces training-based approaches that accelerate diffusion model sampling by teaching new generators to produce samples in only one or a few steps. The central idea, called *distillation*, is to let a fast student model learn from a slow, pre-trained diffusion model (teacher) sampler. While the teacher may require hundreds of steps, the student can achieve comparable quality in only a few steps<sup>1</sup>. Unlike solver-based acceleration, which improves the numerical integration scheme, distillation directly trains a generator to take efficient shortcuts. We highlight two main paradigms: *distribution level distillation*, which skips simulating the full trajectory and instead aligns the student’s output distribution with the teacher’s, and *flow map level distillation*, which trains the student to reproduce the teacher’s sampling path in a faster and more compact way.

---

<sup>1</sup>Here, distillation refers to reducing the number of sampling steps, not to shrinking the model size.

## 10.1 Prologue

A central bottleneck of diffusion models is their slow sampling speed.

As shown through Tweedie’s formula (Section 6.3.1), a diffusion model can be interpreted as an “ $\mathbf{x}$ -prediction” model,  $\mathbf{x}_{\phi^\times}(\mathbf{x}_t, t)$ , trained to recover the expected clean data from a noisy input  $\mathbf{x}_t$  at noise level  $t$ :

$$\mathbf{x}_{\phi^\times}(\mathbf{x}_t, t) \approx \mathbb{E}[\mathbf{x}_0 | \mathbf{x}_t],$$

where the expectation is taken with respect to  $p(\mathbf{x}_0 | \mathbf{x}_t)$ , representing all plausible clean data corresponding to  $\mathbf{x}_t$ . A natural idea is to use  $\mathbf{x}_{\phi^\times}(\mathbf{x}_t, t)$  for one step generation. Yet because this denoiser averages over many plausible outcomes, the prediction becomes overly smooth, and generation with only a few denoising steps leads to blurry, low quality samples.

On the other hand, as discussed in Section 4.2.2, diffusion sampling follows an ODE or SDE trajectory through a long sequence of iterative steps. This produces high fidelity samples, but the large number of steps required makes the process inherently slow. Reducing the NFE (i.e., the number of sampling steps times model calls) speeds up generation but inevitably reduces fidelity. Each solver step introduces an integration error of order  $\mathcal{O}(h^n)$ , where  $n$  is the solver order and  $h = \max_i |t_i - t_{i-1}|$  is the step size. Fewer steps imply a larger time increment  $h$ , which in turn increases the accumulated sampling error and leads to a less accurate trajectory. This creates a fundamental trade off between quality and efficiency in diffusion sampling.

To overcome this bottleneck, a major line of research is *distillation*, which assumes access to a well trained diffusion model (the *teacher*) and trains a generator (the *student*) to reproduce its behavior through a single feed forward or few step computation. This compresses the teacher’s many sampling steps into a fast process, effectively bypassing slow iterative solvers while maintaining high sample fidelity.

Below, we introduce two perspectives on distillation: *distribution level distillation* and *flow map level distillation*<sup>2</sup>.

---

<sup>2</sup>Chronologically, flow map level distillation, represented by *Knowledge Distillation* (KD) (Luhman and Luhman, 2021) and *Progressive Distillation* (PD) (Ho et al., 2020), was proposed earlier in 2021, preceding the family of distribution level distillation approaches that emerged around 2023. For smoother exposition and connection to the next chapter, however, we present distribution level distillation first.

### 10.1.1 Distribution Level Distillation

The goal of distribution based distillation is to train a one-step generator  $\mathbf{G}_\theta(\mathbf{z})$  that maps noise  $\mathbf{z} \sim p_{\text{prior}}$  to a sample  $\hat{\mathbf{x}} = \mathbf{G}_\theta(\mathbf{z})$ , inducing a distribution  $p_\theta(\hat{\mathbf{x}})$  that approximates the target data distribution  $p_{\text{data}}(\mathbf{x})$ . This is typically achieved by minimizing a statistical divergence

$$\min_{\theta} \mathcal{D}(p_\theta(\hat{\mathbf{x}}), p_{\text{data}}(\hat{\mathbf{x}})),$$

where  $\mathcal{D}$  denotes a suitable divergence measurement such as KL.

In practice, distribution based methods align the generator's distribution with the empirical distribution  $p_{\phi^\times}(\mathbf{x})$  produced by a pre-trained diffusion model:

$$\min_{\theta} \mathcal{D}(p_\theta(\hat{\mathbf{x}}), p_{\phi^\times}(\hat{\mathbf{x}})),$$

where  $p_{\phi^\times}$  serves as a surrogate for  $p_{\text{data}}$ . Rather than evaluating this divergence explicitly, these methods approximate its gradient, which can be computed directly from the pre-trained teacher model. This enables the student to align its distribution with the teacher's without requiring full divergence evaluation.

This formulation distills multi-step generative processes of diffusion models into a single step model through distributional alignment. We detail this approach in Section 10.2.

### 10.1.2 Flow Map Level Distillation

We consider the PF-ODE, which can be expressed for any prediction model (see Equation (6.3.1)):

$$\frac{d\mathbf{x}(\tau)}{d\tau} = f(\tau)\mathbf{x}(\tau) - \frac{1}{2}g^2(\tau)\nabla_{\mathbf{x}} \log p_\tau(\mathbf{x}(\tau)) =: \mathbf{v}^*(\mathbf{x}(\tau), \tau). \quad (10.1.1)$$

Its solution map, starting from  $\mathbf{x}_s$  at time  $s$  and evolving reversely to time  $t \leq s$ , is denoted by  $\Psi_{s \rightarrow t}(\mathbf{x}_s)$ ; that is,

$$\Psi_{s \rightarrow t}(\mathbf{x}_s) := \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}(\tau), \tau) d\tau, \quad (10.1.2)$$

where the integral solves the PF-ODE. Intuitively,  $\Psi_{s \rightarrow t}$  transports,  $\mathbf{x}_s$ , noise at time  $s$  to less noisy states at time  $t$  (ultimately data at  $t = 0$ ).

Sampling from a diffusion model corresponds to evaluating  $\Psi_{T \rightarrow 0}(\mathbf{x}_T)$  for  $\mathbf{x}_T \sim p_{\text{prior}}$ . Typically, this integral is approximated by iterative numerical solvers leveraging the velocity field  $\mathbf{v}$  (see Chapter 9), but requires many steps (e.g., at least 10 steps even in DPM-Solver), making sampling slower than

classic one-step generative models such as GAN. This motivates a natural question:

**Question 10.1.1**

Can we learn the solution map  $\Psi_{s \rightarrow t}(\mathbf{x}_s)$  directly?

In particular, learning a map  $\Psi_{T \rightarrow 0}(\mathbf{x}_T)$  with  $\mathbf{x}_T \sim p_{\text{prior}}$  enables one-step generation.

**Trajectory Distillation.** Trajectory distillation seeks to train a neural generator that approximates the solution map at the instance level. Since the PF-ODE integral rarely admits a closed form, it must be approximated numerically during training. To formalize, we introduce the general solver notation

$$\text{Solver}_{s \rightarrow t}(\mathbf{x}_s; \phi^\times) \quad \text{or simply} \quad \text{Solver}_{s \rightarrow t}(\mathbf{x}_s), \quad (10.1.3)$$

denoting numerical integration of the empirical PF-ODE from  $s$  to  $t$  starting at  $\mathbf{x}_s$ , with teacher parameters  $\phi^\times$  (omitted when clear from context).

**An Early Approach: Direct Knowledge Distillation.** To enable few step or even one step generation, a direct approach is to train a generator  $\mathbf{G}_\theta(\mathbf{x}_T, T, 0)$  to imitate the output of a numerical solver evaluated along the full trajectory:

$$\mathbf{G}_\theta(\mathbf{x}_T, T, 0) \approx \text{Solver}_{T \rightarrow 0}(\mathbf{x}_T), \quad \mathbf{x}_T \sim p_{\text{prior}}.$$

This idea underlies one of the earliest trajectory distillation methods, *Knowledge Distillation* (Luhman and Luhman, 2021), which uses the regression loss

$$\mathcal{L}_{\text{KD}}(\theta) := \mathbb{E}_{\mathbf{x}_T \sim p_{\text{prior}}} \|\mathbf{G}_\theta(\mathbf{x}_T, T, 0) - \text{Solver}_{T \rightarrow 0}(\mathbf{x}_T)\|_2^2.$$

While this approach provides direct supervision from the pre-trained teacher, it cannot leverage the strong supervision available in the original training data. In addition, it is computationally expensive if ODE integration is invoked within the training loop, since each parameter update requires solving the ODE to form targets. Finally, because the generator learns only a global mapping from  $T$  to 0, it may lose controllability for steering the generation process from intermediate states. Consequently, most controllable generation techniques introduced in Chapter 8 cannot be directly applied.

**Preface to Progressive Distillation.** *Progressive Distillation* (PD) (Salimans and Ho, 2021) trains a time-conditional Student using *local* supervision from

**Teacher** fragments. Let  $t_0 = T > t_1 > \dots > t_N = 0$  be a fixed time grid. The **Teacher** provides time-stepping maps  $\text{Teacher}_{t_k \rightarrow t_{k+1}}$  for  $k = 0, \dots, N - 1$ .

Rather than supervising only the one-jump  $T \rightarrow 0$ , PD trains the **Student** two-step skip map to match two consecutive **Teacher** steps:

$$\text{Student}_{t_k \rightarrow t_{k+2}} \approx \text{Teacher}_{t_{k+1} \rightarrow t_{k+2}} \circ \text{Teacher}_{t_k \rightarrow t_{k+1}},$$

for  $k = 0, 2, 4, \dots$ . The matching is performed using a simple regression loss (e.g., mean squared error).

After training on locally paired fragments, the **Student** no longer follows every time interval of the original grid. Instead, it advances on every other time point,

$$t_0 \rightarrow t_2 \rightarrow t_4 \rightarrow \dots \rightarrow t_N,$$

which means that each **Student** step effectively covers two consecutive **Teacher** steps. Consequently, the **Student** completes the same overall time span  $[0, T]$  using only  $N/2$  transitions.

After this stage, the trained **Student** replaces the **Teacher** to serve as the new reference model. The entire procedure is then repeated on the coarser grid, where the time step doubles ( $N \rightarrow N/2 \rightarrow N/4 \rightarrow \dots$ ), progressively distilling the trajectory into fewer and fewer steps until the desired number of inference steps is reached. This iterative halving preserves the global time horizon while continually compressing the temporal resolution of the generative process.

**A Unified Perspective of Flow Map Learning.** Various methods, including KD and PD, can be expressed within a unified loss framework:

$$\mathcal{L}_{\text{oracle}}(\boldsymbol{\theta}) := \mathbb{E}_{s,t} \mathbb{E}_{\mathbf{x}_s \sim p_s} [w(s,t) d(\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t), \Psi_{s \rightarrow t}(\mathbf{x}_s))], \quad (10.1.4)$$

where  $\Psi_{s \rightarrow t}$  is the oracle flow map,  $w(s,t) \geq 0$  specifies how different time pairs  $(s,t)$  are weighted,  $d(\cdot, \cdot)$  is a discrepancy measure such as  $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2^2$  or  $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_1$ , and  $p_s$  denotes the forward noised marginal at time  $s$ . Because  $\Psi_{s \rightarrow t}$  is not available in closed form, one must rely on approximations, typically through a pre-trained diffusion model (teacher) or another tractable surrogate.

KD appears as a simple instance of Equation (10.1.4). Selecting a degenerate weighting  $w(s,t) = \delta(s-T) \delta(t-0)$  and using the prior distribution  $p_T = p_{\text{prior}}$ <sup>3</sup>, the oracle loss  $\mathcal{L}_{\text{oracle}}(\boldsymbol{\theta})$  reduces to:

$$\mathbb{E}_{\mathbf{x}_T \sim p_T} \|\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_T, T, 0) - \Psi_{T \rightarrow 0}(\mathbf{x}_T)\|_2^2 \approx \mathcal{L}_{\text{KD}}(\boldsymbol{\theta}),$$

---

<sup>3</sup>This assumption holds for large enough  $T$  or with appropriate noise schedules  $(\alpha_t, \sigma_t)$ .

with  $\text{Solver}_{T \rightarrow 0} \approx \Psi_{T \rightarrow 0}$ . An alternative perspective on this formulation is presented in Section D.5.

PD also fits this template, but instead of supervising only with the single extreme pair  $(T, 0)$ , it uses many *nearby* time pairs and enforces a simple *local consistency* rule: a short step followed by another short step should match the direct two-step move. We return to this in Equation (10.3.3).

In practice, the main challenge is that the oracle flow map  $\Psi_{s \rightarrow t}$  generally has no closed-form expression, making direct supervision infeasible. A range of methods have been developed to approximate this target efficiently, but their success often hinges on the quality of the teacher model. We will return to Equation (10.1.4) in Chapter 11, presenting a principled framework for training-from-scratch methods that eliminate the teacher from the learning loop.

## 10.2 Distribution-Based Distillation

Several works have pursued this distribution-based distillation concurrently under different names, including Distributional Matching Distillation (DMD) (Yin *et al.*, 2024b; Yin *et al.*, 2024a), Variational Score Distillation (VSD) (Poole *et al.*, 2023; Wang *et al.*, 2023; Luo *et al.*, 2023; Lu and Song, 2024), and Score Identity Distillation (SiD) (Zhou *et al.*, 2024). Despite technical differences, they share the same principle: train a generator whose forward-noised marginals match those of the teacher. We focus on VSD as a representative formulation, since the others follow similar principles.

### 10.2.1 Formulation of VSD as a Representative Approach

**Forward Process.** Let  $\{p_t\}_{t \in [0, T]}$  denote the marginal densities of a forward diffusion process induced by

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

with initial distribution  $p_0 = p_{\text{data}}$ . In contrast, let  $p_0^\theta$  denote the distribution of synthetic samples generated by a deterministic one-step generator  $\mathbf{G}_\theta(\mathbf{z})$  from latent variables  $\mathbf{z} \sim p_{\text{prior}}(\mathbf{z})$ . Define  $\{p_t^\theta\}_{t \in [0, T]}$  as the marginal densities obtained by applying the same forward diffusion process to  $p_0^\theta$ , that is,

$$\mathbf{x}_t^\theta := \alpha_t \mathbf{G}_\theta(\mathbf{z}) + \sigma_t \boldsymbol{\epsilon}, \tag{10.2.1}$$

where  $\mathbf{z} \sim p_{\text{prior}}$  and  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . Thus, both  $p_t$  and  $p_t^\theta$  share the same Gaussian diffusion kernel  $p_t(\mathbf{x}_t | \mathbf{x}_0)$  but differ in their starting distributions ( $p_{\text{data}}$  vs.  $p_0^\theta$  of one-step synthetic samples).

**Training Objective and Gradient.** The literature typically adopts the KL divergence to match the distributions  $p_t$  and  $p_t^\theta$ , commonly by minimizing

$$\begin{aligned} \mathcal{L}_{\text{VSD}}(\boldsymbol{\theta}) &:= \mathbb{E}_t \left[ \omega(t) \mathcal{D}_{\text{KL}}(p_t^\theta \| p_t) \right] \\ &= \mathbb{E}_{t, \mathbf{z}, \boldsymbol{\epsilon}} \left[ \omega(t) \left( \log p_t^\theta(\mathbf{x}_t^\theta) - \log p_t(\mathbf{x}_t^\theta) \right) \right], \end{aligned}$$

where  $\omega(t)$  is a time-dependent weighting function. We will discuss in Section 10.2.3 why the KL divergence plays a special role in distribution-level distillation.

As shown in (Wang *et al.*, 2023), the optimum is achieved when  $p_0^{\theta^*} = p_{\text{data}}$ , indicating that the generator's distribution matches the data distribution, and the training objective serves as a valid loss for learning the data distribution.

However, the density-based formulation of the objective lacks an efficient training mechanism. Fortunately, by taking the gradient with respect to  $\theta$ , we arrive at the expression in Equation (10.2.2), which is summarized in the following proposition. For notational simplicity, we denote  $\hat{\mathbf{x}}_t := \mathbf{x}_t^\theta$  as defined in Equation (10.2.1).

### Proposition 10.2.1: $\theta$ -Gradient of $\mathcal{L}_{\text{VSD}}$

We have

$$\begin{aligned} & \nabla_{\theta} \mathcal{L}_{\text{VSD}}(\theta) \\ &= \mathbb{E}_{t, \mathbf{z}, \epsilon} \left[ \omega(t) \alpha_t \left( \nabla_{\mathbf{x}} \log p_t^\theta(\hat{\mathbf{x}}_t) - \nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t) \right) \cdot \partial_{\theta} \mathbf{G}_{\theta}(\mathbf{z}) \right]. \end{aligned} \quad (10.2.2)$$

### **Proof for Proposition.**

The derivation applies the chain rule:

$$\begin{aligned} & \nabla_{\theta} \mathbb{E}_t \left[ \mathcal{D}_{\text{KL}}(p_t^\theta \| p_t) \right] \\ &= \mathbb{E}_{t, \mathbf{z}, \epsilon} \left[ \partial_{\theta} (\log p_t^\theta(\hat{\mathbf{x}}_t) - \log p_t(\hat{\mathbf{x}}_t)) \right] \\ &= \mathbb{E}_{t, \mathbf{z}, \epsilon} \left[ \underbrace{\partial_{\theta} \log p_t^\theta(\hat{\mathbf{x}}_t)}_{\text{first}} + (\nabla_{\mathbf{x}} \log p_t^\theta(\hat{\mathbf{x}}_t))^{\top} \partial_{\theta} \hat{\mathbf{x}}_t - (\nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t))^{\top} \partial_{\theta} \hat{\mathbf{x}}_t \right]. \end{aligned}$$

The first term vanishes by the score-function identity:

$$\mathbb{E}_{\hat{\mathbf{x}}_t \sim p_t^\theta} \left[ \partial_{\theta} \log p_t^\theta(\hat{\mathbf{x}}_t) \right] = \int \partial_{\theta} p_t^\theta(\mathbf{x}) d\mathbf{x} = \partial_{\theta} \int p_t^\theta(\mathbf{x}) d\mathbf{x} = \partial_{\theta}(1) = 0.$$

Using the reparameterization  $\hat{\mathbf{x}}_t = \alpha_t \mathbf{G}_{\theta}(\mathbf{z}) + \sigma_t \epsilon$  gives  $\partial_{\theta} \hat{\mathbf{x}}_t = \alpha_t \partial_{\theta} \mathbf{G}_{\theta}(\mathbf{z})$ , hence

$$\nabla_{\theta} \mathcal{L}_{\text{VSD}}(\theta) = \mathbb{E}_{t, \mathbf{z}, \epsilon} \left[ \omega(t) \alpha_t (\nabla_{\mathbf{x}} \log p_t^\theta(\hat{\mathbf{x}}_t) - \nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t))^{\top} \partial_{\theta} \mathbf{G}_{\theta}(\mathbf{z}) \right].$$

This proves Equation (10.2.2). See Section D.5 for details. ■

We observe that the score functions naturally emerge when taking the gradient with respect to  $\theta$ . Consequently, we require approximations of the score  $\nabla_{\mathbf{x}} \log p_t^\theta(\hat{\mathbf{x}}_t)$  for the one-step generator and  $\nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t)$  for the data distribution, as will be detailed in the following subsection.

### 10.2.2 Training Pipeline of VSD

Existing works (Yin *et al.*, 2024b; Yin *et al.*, 2024a; Poole *et al.*, 2023; Wang *et al.*, 2023; Luo *et al.*, 2023; Lu and Song, 2024) typically address this via a bi-level optimization approach: training a new diffusion model on samples from  $\mathbf{G}_\theta(\mathbf{z})$  to approximate  $\nabla_{\mathbf{x}} \log p_t^\theta(\hat{\mathbf{x}}_t)$ , and employing a pre-trained diffusion model as a proxy for the intractable oracle score function  $\nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t)$  on synthetic samples  $\hat{\mathbf{x}}_t$  (*i.e.*, *the teacher's score*). More precisely, training proceeds by alternating between two phases:

- **Score Estimation Phase.** Fix  $\theta$ . Let  $\hat{\mathbf{x}}_0 = \mathbf{G}_\theta(\mathbf{z})$  and  $\hat{\mathbf{x}}_t = \alpha_t \hat{\mathbf{x}}_0 + \sigma_t \epsilon$  with  $\mathbf{z} \sim p_{\text{prior}}$ ,  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . Train  $s_\zeta$  by DSM using the known Gaussian diffusion kernel  $p_t(\mathbf{x}_t | \mathbf{x}_0)$ :

$$\mathcal{L}_{\text{DSM}}(\zeta; \theta) = \mathbb{E}_{t, \mathbf{z}, \epsilon} \left\| s_\zeta(\hat{\mathbf{x}}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\hat{\mathbf{x}}_t | \hat{\mathbf{x}}_0) \right\|^2,$$

which yields  $s_\zeta(\cdot, t) \approx \nabla_{\mathbf{x}} \log p_t^\theta(\cdot)$  at optimum (for fixed  $\theta$ ).

- **Generator Update Phase.** With  $s_\zeta$  frozen (stop-grad),  $\theta$  is updated by using the gradient in Equation (10.2.2), replacing the individual score terms by their respective proxies:

$$s_\zeta(\hat{\mathbf{x}}_t, t) \approx \nabla_{\mathbf{x}} \log p_t^\theta(\hat{\mathbf{x}}_t), \text{ and } s_{\phi^\times}(\hat{\mathbf{x}}_t, t) \approx \nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t) \quad (\text{teacher}).$$

Equation (10.2.2) then approximately becomes:

$$\nabla_\theta \mathcal{L}_{\text{VSD}}(\theta) \approx \mathbb{E}_{t, \mathbf{z}, \epsilon} \left[ \omega(t) \alpha_t (s_\zeta(\hat{\mathbf{x}}_t, t) - s_{\phi^\times}(\hat{\mathbf{x}}_t, t))^\top \partial_\theta \mathbf{G}_\theta(\mathbf{z}) \right].$$

These two phases repeat until, for all  $t$ ,  $s_\zeta(\cdot, t) \approx s_{\phi^\times}(\cdot, t)$  on the support of  $p_t^\theta$ , so the plug-in gradient in Equation (10.2.2) vanishes. In this convergence regime, we have  $p_t^\theta \approx p_t^{\phi^\times}$  (*the teacher's marginal*) for all  $t > 0$ . Since the forward noising operator (Gaussian convolution) is injective for any fixed  $t > 0$ , it follows that  $p_0^\theta \approx p_0^{\phi^\times}$  (*the teacher's  $t = 0$  distribution*). Thus, the learned one-step generator  $\mathbf{G}_\theta$  matches the teacher's distribution at  $t = 0$ ; when the teacher closely approximates  $p_{\text{data}}$ , this further implies  $p_0^\theta \approx p_{\text{data}}$ .

### 10.2.3 Additional Discussion: Divergence Choices and VSD Applications

**Beyond KL: Can We Use General Divergences?** In principle, one may replace the forward KL term  $\mathcal{D}_{\text{KL}}(p_t^\theta \| p_t)$  in VSD with a more general divergence family, such as the  $f$ -divergence (see Equation (1.1.4)):

$$\mathcal{D}_f(p_t^\theta \| p_t) = \int p_t(\mathbf{x}) f \left( \frac{p_t^\theta(\mathbf{x})}{p_t(\mathbf{x})} \right) d\mathbf{x}.$$

However, the gradient  $\nabla_{\theta} \mathcal{D}_f(p_t^{\theta} \| p_t)$  depends on the *density ratio*

$$r_t(\mathbf{x}) = \frac{p_t^{\theta}(\mathbf{x})}{p_t(\mathbf{x})},$$

through  $f'(r_t)$ , which is intractable for an *implicit student generator*. Here the student is called *implicit* because it can produce samples  $\hat{\mathbf{x}}_t$  through a stochastic mapping  $\hat{\mathbf{x}}_t = \alpha_t \mathbf{G}_{\theta}(\mathbf{z}) + \sigma_t \epsilon$ , but it does not provide a closed-form expression or likelihood for its induced density  $p_t^{\theta}(\mathbf{x})$ . Consequently, computing the functional derivative of  $\mathcal{D}_f$  requires pointwise access to  $r_t(\mathbf{x})$  or its log-gradient, both of which cannot be evaluated in this setting. A common workaround is to introduce an auxiliary critic or discriminator that approximates the density ratio via the variational formulation of  $f$ -divergences, as in  $f$ -GAN (Nowozin *et al.*, 2016), although this introduces an extra network and a nested minimax optimization.

By contrast, for the forward KL, the pathwise gradient simplifies neatly to a score-difference form (Equation (10.2.2)):

$$\nabla_{\theta} \mathcal{D}_{\text{KL}}(p_t^{\theta} \| p_t) = \mathbb{E} \left[ (\nabla_{\mathbf{x}} \log p_t^{\theta}(\hat{\mathbf{x}}_t) - \nabla_{\mathbf{x}} \log p_t(\hat{\mathbf{x}}_t))^{\top} \partial_{\theta} \hat{\mathbf{x}}_t \right].$$

This structure enables a tractable score-only update. The teacher’s pre-trained diffusion model already provides  $\nabla_{\mathbf{x}} \log p_t(\cdot)$ , so we can reuse it directly without learning an auxiliary density-ratio estimator. This formulation yields a non-adversarial training objective that remains fully differentiable and computationally efficient.

**VSD for 3D Generation Using Only a 2D pre-trained Diffusion Model.** VSD (Wang *et al.*, 2023), together with its earlier special case SDS (Poole *et al.*, 2023) where the generator is a Dirac parameterized by  $\theta$ , was originally introduced for 3D scenarios without paired supervision between 3D and 2D data (that is, without ground-truth 3D labels). Let  $\theta \in \mathbb{R}^d$  denote the parameters of a 3D scene, and let  $\mathbf{R}(\theta)$  be a differentiable renderer that produces an image  $\hat{\mathbf{x}}_0 := \mathbf{R}(\theta)$ . The forward noising process is defined as

$$\hat{\mathbf{x}}_t = \alpha_t \mathbf{R}(\theta) + \sigma_t \epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

A pre-trained 2D (image) diffusion teacher provides scores

$$\mathbf{s}_{\phi^x}(\hat{\mathbf{x}}_t, t | \mathbf{c}) \approx \nabla_{\hat{\mathbf{x}}_t} \log p_t(\hat{\mathbf{x}}_t | \mathbf{c}),$$

optionally conditioned on text  $\mathbf{c}$ . The goal is to align the distribution of noisy renderings with the teacher’s marginals at each  $t$ . A minimal formulation is

the score-alignment (VSD) objective under the rendering distribution:

$$\mathcal{L}_{\text{VSD}}^{\text{3D}}(\boldsymbol{\theta}) := \mathbb{E}_{t,\epsilon} \left[ \omega(t) \|\mathbf{s}_\zeta(\hat{\mathbf{x}}_t, t) - \mathbf{s}_{\phi^\times}(\hat{\mathbf{x}}_t, t | \mathbf{c})\|_2^2 \right], \quad \hat{\mathbf{x}}_t = \alpha_t \mathbf{R}(\boldsymbol{\theta}) + \sigma_t \boldsymbol{\epsilon},$$

which transfers image-space score guidance to the 3D parameters through the renderer. Treating both scores as stop gradients with respect to  $\hat{\mathbf{x}}_t$  during the update of  $\boldsymbol{\theta}$  yields

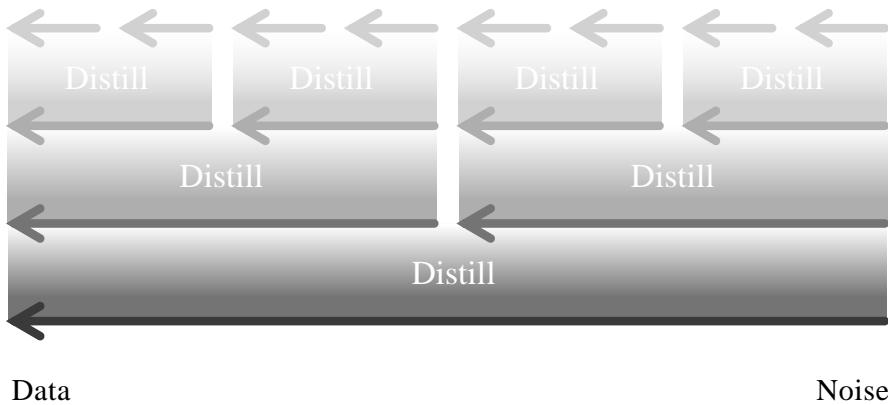
$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{VSD}}^{\text{3D}}(\boldsymbol{\theta}) = \mathbb{E}_{t,\epsilon} \left[ \omega(t) \alpha_t (\mathbf{s}_\zeta - \mathbf{s}_{\phi^\times})^\top \frac{\partial \mathbf{R}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}) \right].$$

When the student score  $\mathbf{s}_\zeta$  is suppressed (Dirac generator), the formulation reduces to SDS (Poole *et al.*, 2023). In practice, optimization alternates exactly as described in Section 10.2.2: first updating the student score on noisy renderings, and then updating  $\boldsymbol{\theta}$  with stop gradients through both scores. Further mathematical details are omitted here for brevity.

### 10.3 Progressive Distillation

Progressive Distillation (PD) (Salimans and Ho, 2021) consists of two procedures that together enable a diffusion model to learn the PF-ODE trajectory more efficiently. The key idea is to progressively reduce the number of integration steps required for high-quality sampling while retaining fidelity to the teacher trajectory.

- **Distillation Operation:** Distills a deterministic sampler (e.g., DDIM) based on a pre-trained teacher model (initially a diffusion model) into a student model that reproduces the same trajectory using only half as many sampling steps.
- **Progressive Operation:** Repeats this distillation process iteratively, each time halving the number of steps, until the student can generate high-quality samples within a small fixed budget (typically 1–4 steps).



**Figure 10.1: Illustration of Progressive Distillation (PD).** At each round, the student model is trained so that a single step reproduces the effect of two adjacent teacher steps. This process distills  $N$  teacher steps into  $N/2$  student steps, and repeating the procedure progressively halves the trajectory length until the desired step count is reached. The arrows indicate how multi-step teacher transitions are compressed into fewer student steps, moving from data to noise.

We first introduce the distillation operation of PD in Section 10.3.1, and then summarize the entire training pipeline in Section 10.3.2. Section 10.3.4 presents an extension for CFG guidance.

### 10.3.1 Distillation Operation in PD

In this section, we fix DDIM in the  $\mathbf{x}$ -prediction parameterization as the time-stepping rule and still write  $\text{Solver}_{s \rightarrow t}$  for the deterministic map obtained by plugging the current teacher's  $\mathbf{x}$ -denoiser into DDIM.

In the first PD round (teacher = pre-trained diffusion model), this coincides with integrating the diffusion PF-ODE via DDIM; in later rounds (teacher = previous student),  $\text{Solver}_{s \rightarrow t}$  is simply the DDIM transition induced by the current teacher, not the original diffusion PF-ODE.

The distillation step is as follows: starting from a noisy input  $\mathbf{x}_s$  (a perturbed version of clean data,  $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$ ), the student is trained to predict a target  $\tilde{\mathbf{x}}$  so that a single student step  $s \rightarrow t$  reproduces the teacher's two consecutive steps  $s \rightarrow u \rightarrow t$ . Let  $\mathbf{x}_{\phi^x}(\mathbf{x}, \tau)$  denote the teacher's  $\mathbf{x}$ -prediction denoiser in this round. Applying the teacher-induced DDIM transition twice gives

$$\tilde{\mathbf{x}}_u := \text{Solver}_{s \rightarrow u}(\mathbf{x}_s; \mathbf{x}_{\phi^x}), \quad \tilde{\mathbf{x}}_t := \text{Solver}_{u \rightarrow t}(\tilde{\mathbf{x}}_u; \mathbf{x}_{\phi^x}).$$

Here, we use the notation of Equation (10.1.3) to denote the deterministic transition map from  $s$  to  $t$  (starting at  $\mathbf{x}_s$ ) induced by plugging  $\mathbf{x}_{\phi^x}$  into DDIM.

#### Question 10.3.1

What is the pseudo-clean  $\tilde{\mathbf{x}}$  at time  $s$  such that the solver produces the same output  $\tilde{\mathbf{x}}_t$  when stepping directly  $s \rightarrow t$  as it does via  $s \rightarrow u \rightarrow t$ ? Specifically, determine  $\tilde{\mathbf{x}}$  satisfying:

$$\tilde{\mathbf{x}}_t = \text{Solver}_{s \rightarrow t}(\mathbf{x}_s; \tilde{\mathbf{x}}).$$

Once a closed-form expression for  $\tilde{\mathbf{x}}$  is obtained, we train a student model  $\mathbf{f}_{\theta}(\mathbf{x}_s, s)$  (also an  $\mathbf{x}$ -prediction model here) to approximate the “two-steps-in-one” target  $\tilde{\mathbf{x}}$  by minimizing

$$\min_{\theta} \mathbb{E}_s \mathbb{E}_{\mathbf{x}_s \sim p_s} \left[ w(\lambda_s) \|\mathbf{f}_{\theta}(\mathbf{x}_s, s) - \tilde{\mathbf{x}}\|_2^2 \right]. \quad (10.3.1)$$

In the following, we show that the DDIM rule yields  $\tilde{\mathbf{x}}$  in closed form through elementary algebra (note that the result holds for both discrete and continuous time):

**Lemma 10.3.1: Two-Steps-in-One Target  $\tilde{\mathbf{x}}$  of DDIM**

Starting from an initial condition  $\mathbf{x}_s$ , if the solver is taken as DDIM, then the “two-step-in-one” target  $\tilde{\mathbf{x}}$  can be computed as

$$\tilde{\mathbf{x}} = \frac{\sigma_s}{\alpha_t \sigma_s - \alpha_s \sigma_t} \tilde{\mathbf{x}}_t - \frac{\sigma_t}{\alpha_t \sigma_s - \alpha_s \sigma_t} \mathbf{x}_s.$$

Here,  $\tilde{\mathbf{x}}_t$  is obtained by applying DDIM (in Equation (9.2.3)) twice, from  $s \rightarrow u \rightarrow t$ :

$$\begin{aligned} s \rightarrow u : \quad \tilde{\mathbf{x}}_u &= \frac{\sigma_u}{\sigma_s} \mathbf{x}_s + \alpha_s \left( \frac{\alpha_u}{\alpha_s} - \frac{\sigma_u}{\sigma_s} \right) \mathbf{x}_{\phi^x}(\mathbf{x}_s, s) \\ u \rightarrow t : \quad \tilde{\mathbf{x}}_t &= \frac{\sigma_t}{\sigma_u} \tilde{\mathbf{x}}_u + \alpha_u \left( \frac{\alpha_t}{\alpha_u} - \frac{\sigma_t}{\sigma_u} \right) \mathbf{x}_{\phi^x}(\tilde{\mathbf{x}}_u, u). \end{aligned}$$

**Proof for Lemma.**

$\tilde{\mathbf{x}}_t$  must be matched with the one-step DDIM from  $s$  to  $t$ ,  $\tilde{\mathbf{x}}'_t$ , expressed as:

$$s \rightarrow t : \quad \tilde{\mathbf{x}}'_t = \frac{\sigma_t}{\sigma_s} \mathbf{x}_s + \alpha_s \left( \frac{\alpha_t}{\alpha_s} - \frac{\sigma_t}{\sigma_s} \right) \tilde{\mathbf{x}}.$$

By equating  $\tilde{\mathbf{x}}'_t$  and  $\tilde{\mathbf{x}}_t$ , we can solve for  $\tilde{\mathbf{x}}$  in terms of  $\tilde{\mathbf{x}}_t$ ,  $s$ , and  $t$ :

$$\begin{aligned} \tilde{\mathbf{x}}_t &= \tilde{\mathbf{x}}'_t \\ \iff \tilde{\mathbf{x}}_t &= \frac{\sigma_t}{\sigma_s} \mathbf{x}_s + \alpha_s \left( \frac{\alpha_t}{\alpha_s} - \frac{\sigma_t}{\sigma_s} \right) \tilde{\mathbf{x}} \\ \iff \tilde{\mathbf{x}} &= \frac{\tilde{\mathbf{x}}_t - \frac{\sigma_t}{\sigma_s} \mathbf{x}_s}{\alpha_s \left( \frac{\alpha_t}{\alpha_s} - \frac{\sigma_t}{\sigma_s} \right)} \\ \iff \tilde{\mathbf{x}} &= \frac{\sigma_s}{\alpha_t \sigma_s - \alpha_s \sigma_t} \tilde{\mathbf{x}}_t - \frac{\sigma_t}{\alpha_t \sigma_s - \alpha_s \sigma_t} \mathbf{x}_s. \end{aligned} \tag{10.3.2}$$

With this formula, PD computes the pseudo-clean target at time  $s$  whose single DDIM step  $s \rightarrow t$  lands exactly at the two-step output  $\tilde{\mathbf{x}}_t$ .

**Practical Discrete Time Grids and Loss.** In practice, we fix a decreasing grid  $t_0 = T > t_1 > \dots > t_N = 0$  and, for brevity, write  $s := t_k$ ,  $u := t_{k+1}$ ,  $t := t_{k+2}$ . The teacher provides one step maps  $\text{Teacher}_{t_k \rightarrow t_{k+1}}$ , and the student learns a two step skip map that matches the teacher composition:

$$\text{Student}_{t_k \rightarrow t_{k+2}} \approx \text{Teacher}_{t_{k+1} \rightarrow t_{k+2}} \circ \text{Teacher}_{t_k \rightarrow t_{k+1}}.$$

We sample triplets  $(s, u, t) = (t_k, t_{k+1}, t_{k+2})$  with  $k \in \{0, \dots, N-2\}$ . The objective Equation (10.3.1) becomes

$$\min_{\theta} \mathbb{E}_{k \sim \mathcal{U}[0, N-2]} \mathbb{E}_{\mathbf{x}_{t_k} \sim p_{t_k}} \left[ w(\lambda_{t_k}) \|\mathbf{f}_{\theta}(\mathbf{x}_{t_k}, t_k) - \tilde{\mathbf{x}}^{(k)}\|_2^2 \right],$$

where the teacher two-step target  $\tilde{\mathbf{x}}^{(k)}$  is computed via Lemma 10.3.1. If the grid is uniform, one may write  $t_k = T(1 - k/N)$  so that

$$s = T\left(1 - \frac{k}{N}\right), \quad u = T\left(1 - \frac{k+1}{N}\right), \quad t = T\left(1 - \frac{k+2}{N}\right),$$

corresponding to evenly spaced time steps of size  $\Delta s = T/N$ .

### 10.3.2 Entire Training Pipeline of PD and Its Sampling

After training on locally paired fragments via Equation (10.3.1), the **Student** no longer follows every interval of the original grid. Instead, each learned step covers two consecutive **Teacher** steps, so the **Student** advances on every other time point,

$$t_0 \rightarrow t_2 \rightarrow t_4 \rightarrow \dots \rightarrow t_N,$$

and thus traverses the same horizon  $[0, T]$  using only  $N/2$  transitions. After this stage, the trained **Student** replaces the **Teacher** as the new denoiser model. The procedure is then repeated on the coarser grid (the time step doubles), yielding the progression

$$N \rightarrow N/2 \rightarrow N/4 \rightarrow \dots,$$

until the desired number of inference steps is reached. At each iteration, the new **Student** is initialized from the updated **Teacher**. This iterative halving preserves the global time horizon while progressively compressing the temporal resolution of the generative process.

**Sampling.** At inference time, using the (DDIM) solver with the current **Student** as the denoiser, the sampler advances on the coarser grid induced by training. After the first round it takes “skip-2” jumps ( $t_0 \rightarrow t_2 \rightarrow \dots \rightarrow t_N$ ), after the next round “skip-4” ( $t_0 \rightarrow t_4 \rightarrow \dots \rightarrow t_N$ ), and so on, halving the number of sampling steps at each iteration while keeping the same start and end times.

### 10.3.3 Additional Discussion: Local Semigroup Matching and the Possibility of Generalized Solvers

**Progressive Distillation as Local Semigroup Matching.** Within the unified objective Equation (10.1.4), the intractable oracle target  $\Psi_{s \rightarrow 0}$  is replaced

by a teacher-induced surrogate that uses the *semigroup property* of the ODE flow (see more details later in Equation (11.2.1)): evolving from  $s$  to  $t$  should be equivalent to going from  $s$  to any intermediate  $u$  and then from  $u$  to  $t$ ,

$$\Psi_{s \rightarrow t} = \Psi_{u \rightarrow t} \circ \Psi_{s \rightarrow u}.$$

PD enforces this locally by training the student's one-step map to match the teacher's composition of two adjacent one-step fragments:

$$\mathbb{E}_s \mathbb{E}_{\mathbf{x}_s \sim p_s} \| \underbrace{\mathbf{G}_\theta(\mathbf{x}_s, s, s - 2\Delta s)}_{\text{student one-step}} - \underbrace{\text{Solver}_{s - \Delta s \rightarrow s - 2\Delta s}(\text{Solver}_{s \rightarrow s - \Delta s}(\mathbf{x}_s))}_{\text{teacher two-step composition}} \|_2^2. \quad (10.3.3)$$

Minimizing Equation (10.3.3) instantiates the semigroup identity on a short decreasing grid (take  $s > u > t$  with  $u = s - \Delta s$  and  $t = s - 2\Delta s$ ):

$$\begin{aligned} \Psi_{s \rightarrow s - 2\Delta s} &= \Psi_{s - \Delta s \rightarrow s - 2\Delta s} \circ \Psi_{s \rightarrow s - \Delta s} \\ &\approx \text{Solver}_{s - \Delta s \rightarrow s - 2\Delta s} \circ \text{Solver}_{s \rightarrow s - \Delta s}, \end{aligned}$$

so training only requires short teacher fragments, rather than a full rollout from time  $s$  all the way to 0.

To connect back to the few-step denoiser view in Equation (10.1.4), define the student's few-step map as a composition of learned jumps:

$$\underbrace{\mathbf{G}_\theta(\mathbf{x}_s, s, 0)}_{\text{few-step denoiser}} = (\mathbf{G}_\theta(\cdot, 2\Delta s, 0) \circ \cdots \circ \mathbf{G}_\theta(\cdot, s, s - 2\Delta s))(\mathbf{x}_s).$$

Conceptually, Equation (10.3.3) provides an efficient *local surrogate* for the global regression

$$\mathbb{E}_{s, \mathbf{x}_s} \| \mathbf{G}_\theta(\mathbf{x}_s, s, 0) - (\text{Solver})_{s \rightarrow 0}^\circ(\mathbf{x}_s) \|_2^2,$$

where  $(\text{Solver})_{s \rightarrow 0}^\circ$  denotes the teacher's full composition from  $s$  to 0 on a grid with step size  $\Delta s$ , serving as a proxy for  $\Psi_{s \rightarrow 0}$ .

**Can we Use Other Solvers?** In the PD introduction above, we focused on DDIM in the  $\mathbf{x}$ -prediction parameterization as a concrete PF-ODE sampler. The local semigroup matching with grid halving is solver-agnostic at the level of deterministic state-to-state maps and extends to the time-stepping methods in Chapter 9 after standard conversions between parameterizations ( $\mathbf{x}$ ,  $\epsilon$ ,  $\mathbf{v}$ , score). However, the closed-form pseudo-target here relies on a *single-step, explicit* update whose one-step map is *affine* in the regression target (as with

DDIM and explicit one-step schemes such as exponential–Euler or explicit RK applied to the PF–ODE). For *multi-step* or *implicit* solvers, which require step history or inner solves, one should instead match the corresponding transition map directly (cf. Equation (10.3.3)) and provide the necessary history or a warm start; a comparable closed-form inversion generally does not exist.

If the sampler is stochastic, freeze the noise sequence per example to obtain a deterministic transition  $\text{Teacher}_{s \rightarrow t}^{(\omega)}$  (with  $\omega$  the fixed noise seed). In that case, PD regresses to a fixed transition map; closed-form pseudo-targets generally require a single step explicit affine update; otherwise, use direct matching as in Equation (10.3.3).

#### 10.3.4 PD with Guidance

Meng *et al.* (2023) proposed a two-stage pipeline for distilling classifier-free guided (CFG) diffusion models: (1) *distill the guidance* into a single network that takes the guidance weight as input, and (2) apply *progressive distillation (PD)* to reduce the sampling steps. They demonstrated this both in pixel space and in latent space (e.g., Stable Diffusion).

**Stage-One Distillation: Distilling Guidance.** Let  $\mathbf{x}_{\phi^\times}(\mathbf{x}_s, s, \mathbf{c})$  denote the (pre-trained) conditional diffusion model output in the “ $\mathbf{x}$ -prediction” parameterization (i.e., a clean estimate) at time  $s$  and condition  $\mathbf{c}$ ; the condition can also be null,  $\mathbf{c} = \emptyset$  (unconditional branch). The  $\omega$ -weighted CFG combination in Equation (8.3.3) can be written as

$$\mathbf{x}_{\phi^\times}^{\omega}(\mathbf{x}_s, s, \mathbf{c}) := (1 + \omega) \mathbf{x}_{\phi^\times}(\mathbf{x}_s, s, \mathbf{c}) - \omega \mathbf{x}_{\phi^\times}(\mathbf{x}_s, s, \emptyset), \quad (10.3.4)$$

where  $\omega \sim p_\omega(\omega)$  for some CFG weighting distribution  $p_\omega$ , typically  $p_\omega(\omega) = \mathcal{U}[\omega_{\min}, \omega_{\max}]$ .

Stage-one introduces a new model  $\mathbf{x}_{\theta_1}(\mathbf{x}_s, s, \mathbf{c}, \omega)$  that directly takes  $\omega$  as input and learns to reproduce the CFG output  $\mathbf{x}_{\phi^\times}^{\omega}(\mathbf{x}_s, s, \mathbf{c})$  by supervised regression:

$$\min_{\theta_1} \mathbb{E}_{\omega \sim p_\omega, s, \mathbf{x} \sim p_{\text{data}}, \mathbf{x}_s \sim p(\mathbf{x}_s | \mathbf{x})} \lambda(s) \|\mathbf{x}_{\theta_1}(\mathbf{x}_s, s, \mathbf{c}, \omega) - \mathbf{x}_{\phi^\times}^{\omega}(\mathbf{x}_s, s, \mathbf{c})\|_2^2.$$

Here  $\lambda(s)$  is a standard schedule-dependent weighting; sampling  $\omega$  each iteration teaches a single network to emulate CFG at arbitrary guidance strengths.

**Stage-Two Distillation: PD.** The stage-one model  $\mathbf{x}_{\theta_1}(\mathbf{x}_s, s, \mathbf{c}, \omega)$  serves as the teacher in PD and is progressively distilled into a student  $\mathbf{x}_{\theta_2}(\mathbf{x}_s, s, \mathbf{c}, \omega)$  with fewer sampling steps, following Section 10.3.2. At each iteration, the number of steps is halved (e.g.,  $N \rightarrow N/2 \rightarrow N/4 \rightarrow \dots$ ).

## 10.4 Closing Remarks

This chapter has introduced our first major paradigm for training-based acceleration. Having exhausted training-free improvements via numerical solvers, we shifted our focus to a new strategy: training a fast *student* generator that learns to replicate the behavior of a slow, pre-trained *teacher* diffusion model.

We explored two primary distillation philosophies. First, in distribution-based distillation, represented by methods like Variational Score Distillation (VSD), the student's output distribution is trained to match the teacher's. This is achieved by aligning their respective score functions across different noise levels, providing a stable, non-adversarial objective. Second, in flow map distillation, we saw how methods like Progressive Distillation (PD) train the student to directly approximate the teacher's solution trajectory. PD's iterative approach, where each round halves the number of sampling steps, proved to be a powerful and practical method for compressing a long iterative process into just a few steps.

These distillation techniques successfully bridge the gap between the high sample quality of iterative diffusion models and the inference speed of one-step generators, offering a compelling pathway to efficient, high-fidelity synthesis.

However, the reliance on a pre-trained teacher model introduces a two-stage pipeline: first train a slow but powerful teacher, then distill it into a fast student. This raises a fundamental question at the forefront of generative modeling research: Can we bypass the teacher entirely?

Is it possible to design a standalone training principle that learns these fast, few-step generators directly from data? The final chapter of this monograph will address this question.

1. We will explore pioneering methods such as Consistency Models that learn the mapping from any point on an ODE trajectory to its destination point.
2. We will delve into generalized concepts of Consistency Models which learn to map any point on an ODE trajectory to another point in a single step.

This shift from improving the solver or distilling a solution to learning the solution map itself represents a significant step toward a new class of generative models that are both principled and highly efficient by design.

# 11

---

## Learning Fast Generators from Scratch

---

*Truth is ever to be found in simplicity, and not in the multiplicity and confusion of things.*

---

Isaac Newton

In Chapter 10 we saw that slow iterative samplers in diffusion models can be compressed into few step generators through distillation. From an engineering perspective, two-stage pipelines are practical because they divide a complex generative training task into clear, independent objectives. The first stage learns the data distribution, while the second accelerates sampling or enhances quality. This separation allows each stage to be optimized independently, making the overall system easier to manage, more stable, and more reliable.

In this chapter, however, the focus shifts to a central question driving the progress of deep generative modeling:

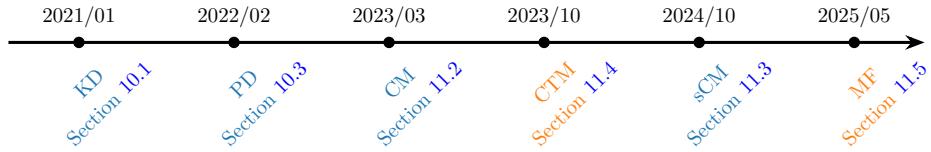
### Question 11.0.1

*Can we design a standalone generative principle that trains in a stable and efficient way, fast sampling, and allows users to easily guide or control what is produced?*

In this chapter we pursue this direction and discuss an alternative approach: training few-step diffusion-based generators *without* relying on a pre-trained model. Our focus is the *flow map model*, which learns a direct transformation that moves samples across time by approximating the oracle flow map of the

PF-ODE. This formulation provides a principled way to transport probability mass from the prior distribution  $p_{\text{prior}}$  to the data distribution  $p_{\text{data}}$ , while preserving the marginal distributions  $p_t$  specified by the forward diffusion process at each intermediate time.

## 11.1 Prologue



**Figure 11.1: Timeline of Flow Map Modeling.** We use blue for the special case  $\Psi_{s \rightarrow 0}$  and orange for the general map  $\Psi_{s \rightarrow t}$ .

**Motivation of Flow Map Models.** In Chapter 10 we showed how the inaccessible regression target in the oracle flow-map loss  $\mathcal{L}_{\text{oracle}}(\boldsymbol{\theta})$  (see Equation (10.1.4)) can be estimated by distilling knowledge from a pre-trained diffusion model to obtain few-step generators. This route is effective and practical: a two-stage pipeline can be engineered for robustness and often remains competitive in both data and compute efficiency.

In this chapter, we shift focus to a broader challenge at the core of deep generative modeling: *Can we establish a standalone generative principle that enables stable, scalable, and efficient training, fast sampling, and generation that can be easily steered by user intentions, without relying on a pre-trained model?* Designing such standalone principles lies at the center of generative modeling.

Diffusion models offer a useful design principle: start with a continuous-time forward process that gradually transforms data into a simple prior (noise) as a reference, and frame the modeling task as learning the reverse-time transport that restores this process to match the desired marginal distributions. This time-dependent formulation also makes it easier to steer the generation process at intermediate steps, compared to one-shot generative maps. Specializing to diffusion-motivated methods, this leads to the question:

### Question 11.1.1

Can we learn the flow map  $\Psi_{s \rightarrow t}(\cdot)$  with a network  $\mathbf{G}_{\boldsymbol{\theta}}(\cdot, s, t)$  (a flow map model) without access to pre-trained models, while maintaining high-fidelity generation?

This chapter develops methods toward this goal, organized around a single objective that also underlies distillation and provides a unified view of flow-map formulations (Boffi *et al.*, 2024; Hu *et al.*, 2025):

$$\mathcal{L}_{\text{oracle}}(\theta) := \mathbb{E}_{s,t} \mathbb{E}_{\mathbf{x}_s \sim p_s} [w(s,t) d(\mathbf{G}_\theta(\mathbf{x}_s, s, t), \Psi_{s \rightarrow t}(\mathbf{x}_s))]. \quad (10.1.4)$$

Here  $s, t$  are sampled from some time distribution (e.g., uniform),  $w(s, t) \geq 0$  assigns weights to the time pairs  $(s, t)$ , and  $d(\cdot, \cdot)$  is a discrepancy measure such as the squared  $\ell_2$  norm. The oracle flow map  $\Psi_{s \rightarrow t}$  represents the ideal transformation that takes a state  $\mathbf{x}_s$  at time  $s$  and transports it directly to time  $t$ :

$$\Psi_{s \rightarrow t}(\mathbf{x}_s) = \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du,$$

where the oracle drift is given as

$$\mathbf{v}^*(\mathbf{x}_u, u) = \mathbb{E} [\alpha'_u \mathbf{x}_0 + \sigma'_u \epsilon | \mathbf{x}_u],$$

while equivalent parametrizations are also possible (see Chapter 6), with common choices including the  $\mathbf{x}$ -prediction and  $\mathbf{v}$ -prediction forms.

At the optimum of the oracle loss, the learned model recovers the true flow map exactly:

$$\mathbf{G}^*(\mathbf{x}_s, s, t) = \Psi_{s \rightarrow t}(\mathbf{x}_s), \quad \text{for all } s, t, \text{ and } \mathbf{x}_s \sim p_s.$$

Because the flow map  $\Psi_{s \rightarrow t}$  cannot be expressed in closed form, it must be approximated. One option, discussed in Chapter 10, is to rely on a pre-trained diffusion model. Alternatively, as we will see in this chapter, new and more tractable surrogates can be introduced. For clarity, existing approaches can be broadly categorized according to whether the training procedure queries a teacher during the loop: *distillation*, which explicitly calls a teacher model, and *training from scratch*, which avoids teacher calls by constructing self-contained surrogates.

Building on this principled objective, we now turn to systematic approaches for learning flow map models, with the aim of developing methods that are practical while also producing generations that more accurately reflect the true data distribution and are computationally efficient. We begin with a high level introduction to this paradigm.

**Special Flow Map: Consistency Functions.** *Consistency Models* (Song *et al.*, 2023) represent one of the earliest pioneering approaches to flow-map learning. They learn a few-step denoiser  $\mathbf{f}_\theta(\cdot, s)$  that approximates the special case of the flow map to the origin:

$$\Psi_{s \rightarrow 0}(\cdot), \quad s \in (0, T].$$

The key idea is that every noisy sample  $\mathbf{x}_s$  should be mapped back to the clean data point  $\mathbf{x}_0$  at the end of its trajectory. Formally, the oracle training objective for the CM family (Song *et al.*, 2023; Song and Dhariwal, 2024; Geng *et al.*, 2024; Lu and Song, 2024) is

$$\mathcal{L}_{\text{oracle-CM}}(\boldsymbol{\theta}) := \mathbb{E}_s \mathbb{E}_{\mathbf{x}_s \sim p_s} [w(s) d(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \Psi_{s \rightarrow 0}(\mathbf{x}_s))]. \quad (11.1.0)$$

In practice, however, the oracle  $\Psi_{s \rightarrow 0}(\mathbf{x}_s)$  is unavailable. It is therefore replaced by a *stop-gradient* target, denoted as  $\mathbf{f}_{\boldsymbol{\theta}-}$ , taken from a slightly earlier step  $\Psi_{s \rightarrow s - \Delta s}(\mathbf{x}_s)$  on the same trajectory:

$$\Psi_{s \rightarrow 0}(\mathbf{x}_s) \approx \mathbf{f}_{\boldsymbol{\theta}-}(\Psi_{s \rightarrow s - \Delta s}(\mathbf{x}_s), s - \Delta s), \quad \Delta s > 0,$$

where  $\Psi_{s \rightarrow s - \Delta s}(\mathbf{x}_s)$  itself must also be approximated. Two practical strategies are available: (i) *distillation*, which relies on a pre-trained diffusion model, and (ii) *training from scratch*, which uses a one-point estimate without teacher guidance.

**General Flow Map.** Two representative approaches are the *Consistency Trajectory Model* (CTM) and *Mean Flow* (MF).

**Consistency Trajectory Models.** *Consistency Trajectory Model* (CTM) (Kim *et al.*, 2024a) is the first work to learn the general flow map  $\Psi_{s \rightarrow t}$  for arbitrary start and end times, and can be viewed as a concrete instance under the unified objective of Equation (10.1.4). CTM adopts an Euler-inspired parametrization by expressing the oracle flow map as

$$\Psi_{s \rightarrow t}(\mathbf{x}_s) := \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du = \frac{t}{s} \mathbf{x}_s + \underbrace{\frac{s-t}{s} \left[ \mathbf{x}_s + \frac{s}{s-t} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du \right]}_{\approx \mathbf{g}_{\boldsymbol{\theta}}},$$

which motivates the neural parameterization

$$\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) := \frac{t}{s} \mathbf{x}_s + \frac{s-t}{s} \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t),$$

where  $\mathbf{g}_{\boldsymbol{\theta}}$  is a neural network trained so that  $\Psi_{s \rightarrow t}(\mathbf{x}_s) \approx \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t)$ .

Since the oracle  $\Psi_{s \rightarrow t}(\mathbf{x}_s)$  is inaccessible, CTM trains against a *stop-gradient* target evaluated at an intermediate time  $u$ :

$$\Psi_{s \rightarrow t}(\mathbf{x}_s) \approx \mathbf{G}_{\boldsymbol{\theta}-}(\Psi_{s \rightarrow u}(\mathbf{x}_s), u, t), \quad u \in [t, s],$$

where the intermediate state  $\Psi_{s \rightarrow u}(\mathbf{x}_s)$  is approximated in one of two ways: (i) *distillation*, which uses a few-step solver applied to a pre-trained diffusion teacher, or (ii) *training from scratch*, which constructs a self-induced teacher directly through the  $\mathbf{G}_{\boldsymbol{\theta}}$  parametrization.

**Mean Flow.** Mean Flow (MF) (Geng *et al.*, 2025a) builds on flow matching by modeling the *average drift* over an interval  $[t, s]$  (with  $t \leq s$ ):

$$\mathbf{h}_\theta(\mathbf{x}_s, s, t) \approx \mathbf{h}^*(\mathbf{x}_s, s, t) := \frac{1}{t-s} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) \, du,$$

also aligning with Equation (10.1.4). Differentiating the identity

$$(t-s) \mathbf{h}^*(\mathbf{x}_s, s, t) = \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) \, du$$

with respect to  $s$  yields a self-referential relation that motivates the MF objective

$$\mathcal{L}_{\text{MF}}(\theta) := \mathbb{E}_s \mathbb{E}_{\mathbf{x}_s \sim p_s} \left[ w(s) \|\mathbf{h}_\theta(\mathbf{x}_s, s, t) - \mathbf{h}_{\theta^-}^{\text{tgt}}(\mathbf{x}_s, s, t)\|_2^2 \right],$$

with stop-gradient target

$$\mathbf{h}_{\theta^-}^{\text{tgt}}(\mathbf{x}_s, s, t) := \mathbf{v}^*(\mathbf{x}_s, s) - (s-t)(\mathbf{v}^*(\mathbf{x}_s, s) \partial_{\mathbf{x}} \mathbf{h}_{\theta^-} + \partial_s \mathbf{h}_{\theta^-}).$$

In practice, the oracle velocity  $\mathbf{v}^*(\mathbf{x}_s, s)$  must also be approximated. Two common strategies are: (i) *distillation*, which leverages a pre-trained diffusion model trained with flow matching, or (ii) *training from scratch*, which uses the one-point conditional velocity  $\alpha'_s \mathbf{x}_0 + \sigma'_s \boldsymbol{\epsilon}$  derived from the forward corruption process  $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$ .

**Relationship Between CTM and MF.** CTM and MF approximate the same path integral but parameterize different surrogates of it:

$$\begin{aligned} \Psi_{s \rightarrow t}(\mathbf{x}_s) &:= \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) \, du \\ &= \frac{t}{s} \mathbf{x}_s + \frac{s-t}{s} \underbrace{\left[ \mathbf{x}_s + \frac{s}{s-t} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) \, du \right]}_{\approx \mathbf{g}_\theta} \\ &= \mathbf{x}_s + (t-s) \underbrace{\left[ \frac{1}{t-s} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) \, du \right]}_{\approx \mathbf{h}_\theta}. \end{aligned}$$

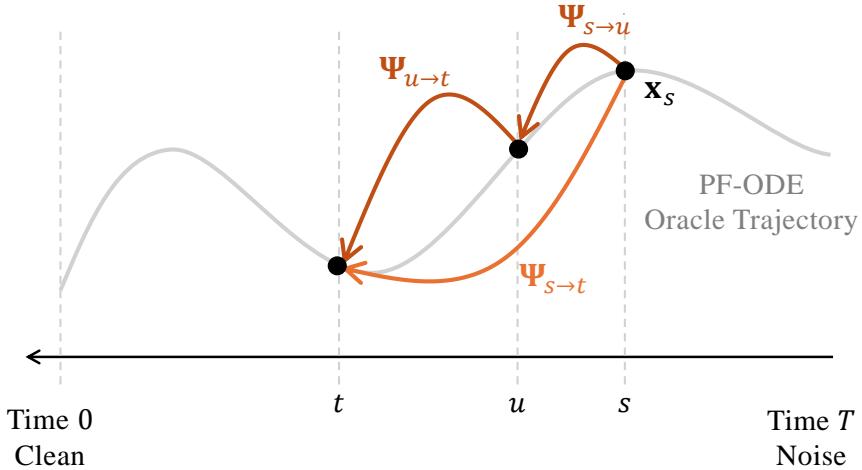
In words, CTM learns an *slope displacement* through  $\mathbf{g}_\theta$ , while MF learns the *average drift*  $\mathbf{h}_\theta$ ; both are consistent ways to approximate the same integral that defines  $\Psi_{s \rightarrow t}$ .

**What Happens Next?** We begin with the CM family, which focuses on the specific flow map  $\Psi_{s \rightarrow 0}$ . This part covers both its discrete time origin in Section 11.2 and its continuous time extension in Section 11.3. We then move on to the general flow map and provide a detailed discussion of two key representatives, CTM and MF. Their parameterizations, training strategies, and practical approximations are presented in Section 11.4 and Section 11.5, respectively.

We remark that the *Elucidating Diffusion Model* (EDM) introduced in Section D.6 offers systematic guidelines for designing the network parameterization of the  $\mathbf{x}$ -prediction model and has demonstrated strong empirical performance. Although this section can be considered optional, the EDM formulation serves as a valuable foundation for CM-style models.

For clarity of exposition later on, we do not strictly follow the chronological order in which these approaches appeared. Instead, we organize the discussion by conceptual relationships. Nevertheless, to acknowledge originality and respect chronology, we provide the historical timeline in Figure 11.1.

## 11.2 Special Flow Map: Consistency Model in Discrete Time



**Figure 11.2: Illustration of the flow map semigroup property.** This property states that transitioning from  $s$  to  $u$  and then from  $u$  to  $t$  is equivalent to transitioning directly from  $s$  to  $t$ .

**An Important Principle of Flow Maps: The Semigroup Property.** Consistency Models (introduced in Sections 11.2 and 11.3) and their generalization, the Consistency Trajectory Model (Section 11.4), define their regression targets by exploiting a key mathematical structure of flow maps. This structure is the fundamental *semigroup property*:

$$\Psi_{u \rightarrow t} \circ \Psi_{s \rightarrow u} = \Psi_{s \rightarrow t}, \quad \Psi_{s \rightarrow s} = \mathbf{I}, \quad \text{for all } s, u, t \in [0, T]. \quad (11.2.1)$$

Intuitively, this means that if we first evolve a state from  $s$  to  $u$  (through  $\Psi_{s \rightarrow u}$ ) and then from  $u$  to  $t$  (through  $\Psi_{u \rightarrow t}$ ), we end up at exactly the same point as if we had evolved directly from  $s$  to  $t$ . This is nothing more than the basic principle of ODE solving<sup>1</sup>: once the starting point of a flow is specified, its future evolution is completely determined, and it follows a single well-defined path. Whether we follow this path in one long step or divide it into smaller intervals, we still move along the same trajectory and arrive at the same final state.

<sup>1</sup>The semigroup property follows from the uniqueness theorem for ODE initial value problems (see Chapter A).

To build further intuition for the semigroup property, consider the solution trajectory  $\{\mathbf{x}(s)\}_{s \in [0, T]}$  of the PF-ODE

$$\frac{d\mathbf{x}(\tau)}{d\tau} = \mathbf{v}^*(\mathbf{x}(\tau), \tau),$$

with a fixed initial condition  $\mathbf{x}(T)$  at time  $T$ , solved backward in time. If we fix the terminal time at  $t = 0$ , the corresponding flow map can be written more simply as

$$\mathbf{f}^*(\cdot, s) := \Psi_{s \rightarrow 0}(\cdot),$$

which is referred to as the *consistency function*. By construction, this function inherits several fundamental properties directly from the semigroup identity of Equation (11.2.1) with  $t = 0$ :

- (i) **Global Consistency:** every point along the trajectory maps to the same clean endpoint,

$$\mathbf{f}^*(\mathbf{x}(s), s) = \mathbf{x}(0), \quad \text{for all } s \in [0, T].$$

This is because

$$\begin{aligned} \mathbf{f}^*(\mathbf{x}(s), s) &= \Psi_{s \rightarrow 0}(\Psi_{0 \rightarrow s}(\mathbf{x}(0))) = (\Psi_{s \rightarrow 0} \circ \Psi_{0 \rightarrow s})(\mathbf{x}(0)) \\ &= \Psi_{0 \rightarrow 0}(\mathbf{x}(0)) = \mathbf{x}(0). \end{aligned}$$

- (ii) **Self Consistency:** any two points along the same trajectory must give identical outputs,

$$\mathbf{f}^*(\mathbf{x}(s), s) = \mathbf{f}^*(\mathbf{x}(u), u), \quad \text{for all } s, u \in [0, T]. \quad (11.2.2)$$

This is a direct re-interpretation of the semigroup identity:  $\Psi_{s \rightarrow 0} \circ \Psi_{0 \rightarrow s} = \Psi_{u \rightarrow 0} \circ \Psi_{0 \rightarrow u}$ .

- (iii) **Local Consistency:** the consistency function is invariant with respect to  $s$  when evaluated along the trajectory,

$$\frac{d}{ds} \mathbf{f}^*(\mathbf{x}(s), s) = 0, \quad \mathbf{f}^*(\mathbf{x}(0), 0) = \mathbf{x}(0). \quad (11.2.3)$$

This follows from global consistency, which states that  $\mathbf{f}^*(\mathbf{x}(s), s)$  does not change with  $s$  along the trajectory.

The three properties are all equivalent. Each states that along any solution trajectory  $s \mapsto \mathbf{x}(s)$ , the flow-to-origin/consistency map  $\mathbf{f}^*(\mathbf{x}(s), s) = \Psi_{s \rightarrow 0}(\mathbf{x}(s))$  yields the same terminal point  $\mathbf{x}(0)$ , independent of the starting time.

**Goal of Consistency Models.** A CM aims to train a neural network  $\mathbf{f}_\theta : \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}^D$  to approximate the special flow map  $\Psi_{s \rightarrow 0}$ , i.e., consistency function<sup>2</sup>. The key idea is to enforce the semigroup property across multiple trajectories of the PF-ODE, ensuring that different noisy versions of the same data point consistently map back to the same clean origin (more precisely, this corresponds to the special case  $t = 0$  and  $u = s - \Delta s$  in Equation (11.2.1)).

There are, however, multiple ways to realize this goal. The choice depends on whether a pre-trained diffusion model is available and whether training is carried out in a discrete-time or continuous-time regime. We begin by summarizing these variants in Table 11.1 and illustrating their objectives in Figure 11.3. The subsequent sections (Sections 11.2 and 11.3) then gradually develop the details of each approach.

**Table 11.1:** Training Objectives of Consistency Models

	<b>Distillation</b>	<b>From Scratch</b>
<b>Discrete-time</b>	Equation (11.2.4)	Equation (11.2.6)
<b>Continuous-time</b>	Equation (11.3.5)	Equation (11.3.6)

### 11.2.1 Discrete-Time Approximations for Learning a Consistency Function

In principle, a consistency function can be learned by minimizing the oracle loss Equation (11.1.0):

$$\mathcal{L}_{\text{oracle-CM}}(\boldsymbol{\theta}) := \mathbb{E}_s \mathbb{E}_{\mathbf{x}_s \sim p_s} [w(s) d(\mathbf{f}_\theta(\mathbf{x}_s, s), \Psi_{s \rightarrow 0}(\mathbf{x}_s))].$$

This objective enforces that every noisy sample  $\mathbf{x}_s$  is mapped back to its clean endpoint  $\Psi_{s \rightarrow 0}(\mathbf{x}_s)$ .

The challenge is that the oracle map  $\Psi_{s \rightarrow 0}(\mathbf{x}_s)$  is not available in practice. To overcome this, Song *et al.* (2023) exploit the *semigroup property*: any noisy state and its consecutive step along the same PF-ODE trajectory must map to the same clean endpoint. Concretely, the oracle target is replaced by a

---

<sup>2</sup>The concept of a consistency function for an ODE generalizes to a function  $\mathbf{f}(\mathbf{x}, t)$  for an SDE such that  $\mathbf{f}(\mathbf{x}_t, t)$  is a (local) martingale with respect to the SDE's natural filtration, i.e.

$$\mathbb{E}[\mathbf{f}(\mathbf{x}_t, t) | \mathbf{x}_s] = \mathbf{f}(\mathbf{x}_s, s), \quad \text{for all } t \geq s.$$

This generalization was proposed/observed in (Daras *et al.*, 2023; Lai *et al.*, 2023a), and the theoretical connections are summarized in (Lai *et al.*, 2023b).

*stop-gradient* target taken from a slightly earlier point on the trajectory:

$$\begin{aligned}\Psi_{s \rightarrow 0}(\mathbf{x}_s) &= \Psi_{s - \Delta s \rightarrow 0}(\Psi_{s \rightarrow s - \Delta s}(\mathbf{x}_s)) \\ &\approx \mathbf{f}_{\theta^-}(\Psi_{s \rightarrow s - \Delta s}(\mathbf{x}_s), s - \Delta s), \quad \Delta s > 0,\end{aligned}$$

where  $\theta^-$  are parameters under the stop-gradient operator. A further difficulty is that the intermediate state  $\Psi_{s \rightarrow s - \Delta s}(\mathbf{x}_s)$  has no closed form either and must itself be approximated. Two practical regimes have been proposed:

**With Pre-trained Diffusion Model (Consistency Distillation).** Suppose we have access to a pre-trained diffusion model. Consistency Distillation (CD) leverages the teacher model to approximate the intermediate state  $\Psi_{s \rightarrow s - \Delta s}(\mathbf{x}_s)$  by simulating only a single backward ODE step:

$$\Psi_{s \rightarrow s - \Delta s}(\mathbf{x}_s) \approx \text{Solver}_{s \rightarrow s - \Delta s}(\mathbf{x}_s).$$

More concretely, a pre-trained diffusion model provides an estimate of the score function  $\mathbf{s}_{\phi^\times}(\mathbf{x}_s, s) \approx \nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s)$ . Using this, one can perform a one-step DDIM update from  $\mathbf{x}_s$  to obtain an approximation of the state at  $s' = s - \Delta s$ :

$$\begin{aligned}\Psi_{s \rightarrow s - \Delta s}(\mathbf{x}_s) &\approx \frac{\alpha_{s'}}{\alpha_s} \mathbf{x}_s + \sigma_s^2 \left( \frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s} \right) \nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s) \\ &\approx \frac{\alpha_{s'}}{\alpha_s} \mathbf{x}_s + \sigma_s^2 \left( \frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s} \right) \mathbf{s}_{\phi^\times}(\mathbf{x}_s, s) \\ &:= \tilde{\mathbf{x}}_{s'}^{\phi^\times}.\end{aligned}$$

Combining this construction with the stop-gradient target yields a practical *discrete-time* proxy for the oracle loss  $\mathcal{L}_{\text{oracle-CM}}(\theta)$ . Formally, over a partition  $0 = s_1 < s_2 < \dots < s_N = T$ , the CD training objective is given by

$$\mathcal{L}_{\text{CD}}^N(\theta, \theta^-; \phi^\times) := \mathbb{E}_{\mathbf{x}_0, \epsilon, i} \left[ \omega(s_i) d(\mathbf{f}_\theta(\mathbf{x}_{s_{i+1}}, s_{i+1}), \mathbf{f}_{\theta^-}(\tilde{\mathbf{x}}_{s_i}^{\phi^\times}, s_i)) \right]. \quad (11.2.4)$$

Here,  $\omega(\cdot)$  is a time-dependent weight,  $d(\cdot, \cdot)$  is a distance measurement, and  $\theta^-$  indicates stop-gradient parameters, which prevent collapse to trivial solutions (e.g., constant predictions).

**Without Pre-Trained Diffusion Model (Consistency Training).** When no pre-trained diffusion model is available, the oracle score  $\nabla_{\mathbf{x}} \log p_s(\mathbf{x}_s)$  can still be estimated directly using a simple one-point approximation (albeit with high variance). Recall that it admits the conditional expectation form:

$$\begin{aligned}\nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s) &= \mathbb{E}_{\mathbf{x}_0 \sim p(\mathbf{x}_0 | \mathbf{x}_s)} [\nabla_{\mathbf{x}_s} \log p(\mathbf{x}_s | \mathbf{x}_0)] \\ &= \mathbb{E}_{\mathbf{x}_0 \sim p(\mathbf{x}_0 | \mathbf{x}_s)} \left[ -\frac{\mathbf{x}_s - \alpha_s \mathbf{x}_0}{\sigma_s^2} \right].\end{aligned}$$

The identity above suggests a simple one-sample estimator. If  $\mathbf{x}_s$  is obtained from a paired sample  $(\mathbf{x}_0, \boldsymbol{\epsilon})$  via  $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$ , then

$$\widehat{\nabla_{\mathbf{x}} \log p_s(\mathbf{x}_s)} := -\frac{\boldsymbol{\epsilon}}{\sigma_s} = -\frac{\mathbf{x}_s - \alpha_s \mathbf{x}_0}{\sigma_s^2}$$

serves as an *unbiased* estimator of the score at  $\mathbf{x}_s$  (conditionally unbiased with respect to  $p(\mathbf{x}_0 | \mathbf{x}_s)$ ). It corresponds exactly to the *conditional score* used as the regression target in denoising score matching.

Plugging this estimate into the DDIM one-step update from  $s$  to  $s' = s - \Delta s$  (see Equation (9.2.3)) yields

$$\begin{aligned}\Psi_{s \rightarrow s'}(\mathbf{x}_s) &\approx \frac{\alpha_{s'}}{\alpha_s} \mathbf{x}_s + \sigma_s^2 \left( \frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s} \right) \nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s) \quad (\text{DDIM}) \\ &\approx \frac{\alpha_{s'}}{\alpha_s} \mathbf{x}_s + \sigma_s^2 \left( \frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s} \right) \widehat{\nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s)} \quad (\text{1-pt score}) \quad (11.2.5) \\ &= \frac{\alpha_{s'}}{\alpha_s} \mathbf{x}_s - \left( \frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s} \right) (\mathbf{x}_s - \alpha_s \mathbf{x}_0) \\ &= \alpha_{s'} \mathbf{x}_0 + \sigma_{s'} \boldsymbol{\epsilon},\end{aligned}$$

where<sup>3</sup>  $\mathbf{x}_0$  is the same data sample and  $\boldsymbol{\epsilon}$  is the same Gaussian noise used to construct  $\mathbf{x}_s$ .

This leads to a teacher-free discrete-time surrogate of the oracle objective  $\mathcal{L}_{\text{oracle-CM}}$ , written as

$$\mathcal{L}_{\text{CT}}^N(\boldsymbol{\theta}, \boldsymbol{\theta}^-) := \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}, i} [\omega(s_i) d(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_{s_{i+1}}, s_{i+1}), \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_{s_i}, s_i))], \quad (11.2.6)$$

with  $\mathbf{x}_{s_i} = \alpha_{s_i} \mathbf{x}_0 + \sigma_{s_i} \boldsymbol{\epsilon}$  and  $\mathbf{x}_{s_{i+1}} = \alpha_{s_{i+1}} \mathbf{x}_0 + \sigma_{s_{i+1}} \boldsymbol{\epsilon}$ .

---

<sup>3</sup>The last identity follows directly from the forward corruption process  $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$  by elementary algebra.

Using  $\alpha_{s'}\mathbf{x}_0 + \sigma_{s'}\boldsymbol{\epsilon}$  directly as an approximation of  $\Psi_{s \rightarrow s'}(\mathbf{x}_s)$  without expectation introduces high variance<sup>4</sup>. Recall, however, the analogous case in denoising score matching (see Section 6.1), where a single conditional score sample serves as the training target yet becomes unbiased once averaged over  $\mathbf{x}_0, \boldsymbol{\epsilon}$  in the loss. By the same reasoning, the expectations over  $\mathbf{x}_0$  and  $\boldsymbol{\epsilon}$  in  $\mathcal{L}_{\text{CT}}^N$  average out this sampling noise, yielding an unbiased loss-level approximation. The following theorem formalizes this expectation-level justification of the one-point estimator.

**Theorem 11.2.1: CM-CT Equivalence up to Error  $\mathcal{O}(\Delta s^2)$**

Let  $s' := s - \Delta s$ , and define

$$\begin{aligned}\mathcal{L}_{\text{CM}}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) &:= \mathbb{E}_{s, \mathbf{x}_0, \boldsymbol{\epsilon}} [w(s) d(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_{s'}, s'))], \\ \mathcal{L}_{\text{CT}}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) &:= \mathbb{E}_{s, \mathbf{x}_0, \boldsymbol{\epsilon}} [w(s) d(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_{s'}, s'))],\end{aligned}$$

where

$$\mathbf{x}_{s'}^{\text{DDIM}} := \frac{\alpha_{s'}}{\alpha_s} \mathbf{x}_s + \sigma_s^2 \left( \frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s} \right) \nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s)$$

is the oracle DDIM update. Both  $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$  and  $\mathbf{x}_{s'} = \alpha_{s'} \mathbf{x}_0 + \sigma_{s'} \boldsymbol{\epsilon}$  share the same pair  $(\mathbf{x}_0, \boldsymbol{\epsilon})$  with  $\mathbf{x}_0 \sim p_{\text{data}}$  and  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . Then,

$$\mathcal{L}_{\text{CM}}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) = \mathcal{L}_{\text{CT}}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) + \mathcal{O}(\Delta s^2).$$

**Proof for Theorem.**

First, note that the DDIM update with the oracle score equals the conditional mean,

$$\mathbf{x}_{s'}^{\text{DDIM}} = \mathbb{E}[\mathbf{x}_{s'} | \mathbf{x}_s],$$

which can also be verified from Equation (11.2.5) by taking the expectation over  $p(\cdot | \mathbf{x}_s)$ . Next, perform a Taylor expansion of

$$d(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s), \mathbf{f}_{\boldsymbol{\theta}^-}(\cdot, s'))$$

around  $\mathbf{x}_{s'}^{\text{DDIM}} = \mathbb{E}[\mathbf{x}_{s'} | \mathbf{x}_s]$ . The linear term of Taylor expansion vanishes because the inner expectation is taken over  $\mathbf{x}_{s'} | \mathbf{x}_s$ , satisfying  $\mathbb{E}[\mathbf{x}_{s'} -$

---

<sup>4</sup>The one-point (conditional) score estimate  $\widehat{\nabla_{\mathbf{x}} \log p_s}(\mathbf{x}_s)$  can be viewed as a one-sample Monte Carlo estimator, which is *conditionally unbiased* given  $\mathbf{x}_s$ : averaging this estimator over the (generally intractable) clean posterior  $p(\cdot | \mathbf{x}_s)$  recovers the true score as

$$\nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s) = \mathbb{E}_{\mathbf{x}_0 \sim p(\cdot | \mathbf{x}_s)} \left[ \widehat{\nabla_{\mathbf{x}} \log p_s}(\mathbf{x}_s) \right].$$

$\mathbf{x}_{s'}^{\text{DDIM}} | \mathbf{x}_s] = 0$ . This shows that, by reparameterizing the conditional as  $\mathbb{E}_{\mathbf{x}_0, \epsilon | \mathbf{x}_s}[\cdot]$  with  $\mathbf{x}_{s'} = \alpha_{s'} \mathbf{x}_0 + \sigma_{s'} \epsilon$ , the DDIM update using the 1-pt score exactly recovers  $\mathbf{x}_{s'}$  *pathwise* for the same  $(\mathbf{x}_0, \epsilon)$  and therefore leaves the inner expectation unchanged. The remaining term is quadratic,  $\mathcal{O}(\Delta s^2)$ , hence  $\mathcal{L}_{\text{CT}} = \mathcal{L}_{\text{CM}} + \mathcal{O}(\Delta s^2)$ . A detailed derivation is provided in Section D.5.

In summary, CD leverages a teacher model for initialization and guidance, which often stabilizes optimization and reduces variance. In contrast, Consistency Training (CT) requires no pre-trained model and can therefore be trained entirely from scratch. Despite this difference, CT serves as a fully standalone generative model.

**Practical Considerations.** In practice, Song *et al.* (2023) adopt the EDM formulation of Karras *et al.* (2022) (see Section D.6) with the forward corruption kernel

$$\mathbf{x}_s = \mathbf{x}_0 + s\epsilon,$$

and use the neural network parameterization proposed therein (cf. Equation (D.6.1)):

$$\mathbf{f}_{\theta}(\mathbf{x}, s) = c_{\text{skip}}(s)\mathbf{x} + c_{\text{out}}(s) \mathbf{F}_{\theta}(c_{\text{in}}(s)\mathbf{x}, c_{\text{noise}}(s)),$$

where  $\mathbf{F}_{\theta}$  is a neural network and the coefficients follow Equation (D.6.5). This parameterization has the important boundary property

$$\mathbf{f}_{\theta}(\mathbf{x}, 0) = \mathbf{x},$$

which enforces consistency at time zero and ensures the network output matches its input when no noise is present.

### 11.2.2 Sampling with Consistency Model

Once a consistency model  $\mathbf{f}_{\theta^{\times}}$  is trained, either in continuous or discrete time, it can be used to generate samples in a single step or a few steps. The algorithm is summarized in Algorithm 9.

**One-Step Generation.** Given an initial latent  $\hat{\mathbf{x}}_T$  sampled from the prior distribution (in practice,  $\mathcal{N}(\mathbf{0}, T^2 \mathbf{I})$ ), a clean sample can be generated via a single function evaluation:

$$\mathbf{f}_{\theta^{\times}}(\hat{\mathbf{x}}_T, T).$$

**Multi-Step Generation.** With pre-selected timesteps

$$T > \tau_1 > \tau_2 > \dots > \tau_{M-1} = 0,$$

start from initial noise  $\hat{\mathbf{x}}_T$  and alternate between noise injection and large clean jumps via the consistency model at earlier time points, gradually refining the sample:

$$\hat{\mathbf{x}}_T \xrightarrow[\text{get a clean}]{\text{long jump}} \mathbf{f}_{\theta^\times}(\hat{\mathbf{x}}_T, T) \xrightarrow[\text{to level } \tau_1]{\text{add noise}} \hat{\mathbf{x}}_{\tau_1} \xrightarrow[\text{get a clean}]{\text{long jump}} \mathbf{f}_{\theta^\times}(\hat{\mathbf{x}}_{\tau_1}, \tau_1) \xrightarrow[\text{to level } \tau_2]{\text{add noise}} \dots$$

---

### Algorithm 9 CM's Sampling with One-Step or Multi-Step Generation

---

**Input:** Consistency model  $\mathbf{f}_{\theta^\times}(\cdot, \cdot)$ , sequence of time points  $T > \tau_1 > \tau_2 > \dots > \tau_{M-1} = 0$ , initial noise  $\hat{\mathbf{x}}_T$

```

1: if one-step then
2:    $\mathbf{x} \leftarrow \mathbf{f}_{\theta^\times}(\hat{\mathbf{x}}_T, T)$ 
3: else
4:    $\mathbf{x} \leftarrow \mathbf{f}_{\theta^\times}(\hat{\mathbf{x}}_T, T)$ 
5:   for  $m = 1$  to  $M - 1$  do
6:     Sample  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
7:      $\hat{\mathbf{x}}_{\tau_m} \leftarrow \alpha_{\tau_m} \mathbf{x} + \sigma_{\tau_m} \epsilon$ 
8:      $\mathbf{x} \leftarrow \mathbf{f}_{\theta^\times}(\hat{\mathbf{x}}_{\tau_m}, \tau_m)$ 
9:   end for
10: end if
Output:  $\mathbf{x}$ 

```

---

### 11.3 Special Flow Map: Consistency Model in Continuous Time

We now move beyond the discrete-time setting of consistency models and consider a continuous-time perspective. Unlike the discrete approach, which fixes a time grid and trains only on those sampled points, the continuous formulation treats the flow map as defined for all times. This shift eliminates the dependence on an arbitrary discretization and provides a more principled alignment with the underlying dynamics. It also helps reduce the approximation errors that naturally arise from discretized integration, and ensures consistency is enforced globally rather than only at selected steps.

#### 11.3.1 Continuous-Time Consistency Model

To motivate the continuous time formulation, we first revisit Equation (11.2.3), which describes the condition under which time derivatives can be taken. Using the chain rule, we arrive at

$$\begin{aligned} \frac{d}{ds} \mathbf{f}^*(\mathbf{x}(s), s) &= 0 \\ \iff (\nabla_{\mathbf{x}} \mathbf{f}^*) (\mathbf{x}(s), s) \cdot \underbrace{\frac{d}{ds} \mathbf{x}(s)}_{\text{ODE velocity}} + \left( \frac{\partial}{\partial s} \mathbf{f}^* \right) (\mathbf{x}(s), s) &= 0, \end{aligned} \quad (11.3.1)$$

where the trajectory  $\mathbf{x}(s)$  follows the PF-ODE

$$\frac{d}{ds} \mathbf{x}(s) = \mathbf{v}^*(\mathbf{x}(s), s).$$

This relationship shows that the consistency function  $\mathbf{f}^*$  remains constant along any solution trajectory of the ODE. The velocity field  $\mathbf{v}^*$  can be estimated in practice either from a pre-trained diffusion model (when such a model is available) or from a direct one point approximation, such as  $\alpha'_s \mathbf{x}_0 + \sigma'_s \epsilon$ , as explained in Section 11.2.

Equation (11.3.1) suggests a natural way to design a training objective in continuous time. One approach is to enforce the condition by minimizing the residual in a manner similar to physics informed neural networks (PINNs) (Raissi, 2018; Boffi *et al.*, 2024):

$$\min_{\theta} \mathbb{E}_{s, \mathbf{x}_0, \epsilon} \left[ \left\| \frac{d}{ds} \mathbf{f}_{\theta}(\mathbf{x}_s, s) \right\|_2^2 \right].$$

In practice, however, Song *et al.* (2023) and Lu and Song (2024) observed that a different formulation works better in training. Instead of directly

enforcing the differential condition, they consider the continuous time limit of the discrete approximation as  $\Delta s \rightarrow 0$ :

$$\mathcal{L}_{\text{CM}}^{\Delta s}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) := \mathbb{E} \left[ \omega(s) \left\| \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s) - \mathbf{f}_{\boldsymbol{\theta}^-}(\Psi_{s \rightarrow s-\Delta s}(\mathbf{x}_s), s - \Delta s) \right\|_2^2 \right]. \quad (11.3.2)$$

Taking the limit  $\Delta s \rightarrow 0$  in Equation (11.3.2) is equivalent to letting the number of time steps  $N \rightarrow \infty$  in Equations (11.2.4) and (11.2.6).

We summarize this key idea in the following proposition.

### Proposition 11.3.1: Continuous-Time Consistency Training

The following convergence result holds:

$$\lim_{\Delta s \rightarrow 0} \frac{1}{\Delta s} \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{CM}}^{\Delta s}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{CM}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^-).$$

Here,

$$\mathcal{L}_{\text{CM}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) := \mathbb{E}_{s, \mathbf{x}_0, \epsilon} \left[ 2\omega(t) \mathbf{f}_{\boldsymbol{\theta}}^\top(\mathbf{x}_s, s) \cdot \frac{d}{ds} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) \right],$$

and the total differentiation identity,

$$\frac{d}{ds} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) = \partial_s \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) + (\partial_{\mathbf{x}} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)) \mathbf{v}^*(\mathbf{x}_s, s). \quad (11.3.3)$$

#### Proof for Proposition.

A first-order Taylor expansion of the stop-gradient target around  $(\mathbf{x}_s, s)$  shows that the loss  $\mathcal{L}_{\text{CM}}^{\Delta s}$  behaves, up to  $\mathcal{O}(\Delta s^2)$ , like an inner product between the student update  $\nabla_{\boldsymbol{\theta}} \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_s, s)$  and the tangent change  $\frac{d}{ds} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)$ . Consequently, the scaled gradient satisfies

$$\lim_{\Delta s \rightarrow 0} \frac{1}{\Delta s} \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{CM}}^{\Delta s} = \nabla_{\boldsymbol{\theta}} \mathbb{E} \left[ \tilde{\omega}(s) \mathbf{f}_{\boldsymbol{\theta}}^\top(\mathbf{x}_s, s) \cdot \frac{d}{ds} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) \right],$$

which is the claimed identity. We defer the proof to Section D.5. ■

The result above is written under the gradient operator  $\nabla_{\boldsymbol{\theta}}$  so that terms involving  $\boldsymbol{\theta}^-$  vanish, since  $\boldsymbol{\theta}^-$  is treated as constant under stop-gradient. Note that  $\frac{d}{ds} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)$  denotes the total derivative along the oracle trajectory, rather than a simple partial time derivative.

In summary, the continuous time consistency model can be trained by minimizing the following objective (ignoring the factor 2):

$$\mathcal{L}_{\text{CM}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) := \mathbb{E}_{s, \mathbf{x}_0, \epsilon} \left[ \omega(s) \mathbf{f}_{\boldsymbol{\theta}}^\top(\mathbf{x}_s, s) \cdot \frac{d}{ds} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) \right]. \quad (11.3.4)$$

### 11.3.2 Training Continuous-Time Consistency Model

Similar to the discrete time case discussed in Section 11.2.1, we now clarify the practical approximation of the tangent term in Equation (11.3.4), which involves the inaccessible oracle velocity  $\mathbf{v}^*$ :

$$\frac{d}{ds} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) = \partial_s \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) + (\partial_{\mathbf{x}} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)) \mathbf{v}^*(\mathbf{x}_s, s).$$

After training a continuous-time CM, sampling follows the same procedure as in the discrete time case (Section 11.2.2).

**Continuous-Time Consistency Distillation.** If a pre-trained diffusion model is available such that  $\mathbf{v}_{\phi^\times} \approx \mathbf{v}^*$ , then the tangent vector  $\frac{d}{ds} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)$  in Equation (11.3.3) can be approximated by the surrogate

$$\frac{d}{ds} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) \approx \partial_s \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) + (\partial_{\mathbf{x}} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)) \mathbf{v}_{\phi^\times}(\mathbf{x}_s, s). \quad (11.3.5)$$

We denote the resulting objective as  $\mathcal{L}_{\text{CM}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \phi^\times)$ . Accordingly, Proposition 11.3.1 can be restated as

$$\lim_{N \rightarrow \infty} N \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{CD}}^N(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \phi^\times) = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{CM}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \phi^\times).$$

**Continuous-Time Consistency Training (from Scratch).** On the other hand, if a pre-trained diffusion model is not available, the oracle velocity  $\mathbf{v}^*$  can be approximated using the one point conditional estimate  $\alpha'_s \mathbf{x}_0 + \sigma'_s \boldsymbol{\epsilon}$ . In this case, the tangent vector  $\frac{d}{ds} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)$  in Equation (11.3.3) is replaced by the surrogate

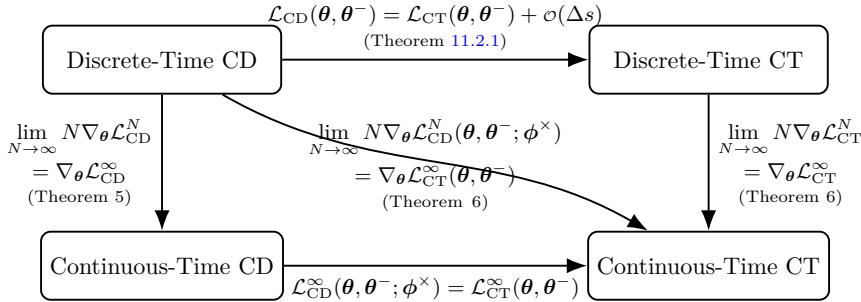
$$\frac{d}{ds} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) \approx \partial_s \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s) + (\partial_{\mathbf{x}} \mathbf{f}_{\boldsymbol{\theta}^-}(\mathbf{x}_s, s)) (\alpha'_s \mathbf{x}_0 + \sigma'_s \boldsymbol{\epsilon}). \quad (11.3.6)$$

We denote the resulting objective as  $\mathcal{L}_{\text{CT}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^-)$ , which corresponds to the training from scratch setting. Accordingly, Proposition 11.3.1 can be restated as

$$\lim_{N \rightarrow \infty} N \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{CT}}^N(\boldsymbol{\theta}, \boldsymbol{\theta}^-) = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{CT}}^{\infty}(\boldsymbol{\theta}, \boldsymbol{\theta}^-).$$

So far, we have introduced all the fundamental approaches listed in Table 11.1 to realize the learning of the consistency function  $\Psi_{s \rightarrow 0}$ . To provide a

clearer overview, Figure 11.3 summarizes the relationships among the different loss functions for training consistency functions. The figure also indicates whether each method relies on a pre-trained diffusion model and distinguishes between continuous time and discrete time objectives.



**Figure 11.3:** Diagram showing relationships between discrete/continuous-time CD and CT under the  $\ell_2$  distance metric:  $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2^2$ . The marked theorems follow the labeling in (Song *et al.*, 2023). Whenever the theorems involve CT, we assume a perfect score:  $\mathbf{s}_{\phi^x}(\mathbf{x}, t) \equiv \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ .  $\mathcal{L}_{CT}^\infty$  is defined in Equation (11.3.4), while  $\mathcal{L}_{CD}^\infty$  is defined in Equation (11.3.5).

However, the tangent vector  $\frac{d}{ds} \mathbf{f}_{\theta^-}$  often causes instability during training. In the following optional section, we present techniques from *Simplifying, Stabilizing and Scaling Continuous Time Consistency Models* (sCM) (Lu and Song, 2024) that mitigate these issues.

### 11.3.3 (Optional) Practical Considerations of Continuous-Time Consistency Training

Our interest lies in the training from scratch scenario, since it yields a stand-alone generative model that does not rely on external pre-trained diffusion models. Hence, we focus our discussion on the continuous time case.

In practice, however, training directly with Equation (11.3.4) is often unstable, as the term  $\frac{d}{ds} \mathbf{f}_{\theta^-}$  can exhibit large or unbounded time derivatives, leading to exploding gradients during optimization. To overcome this, suitable parameterizations and stabilization strategies are typically required (Geng *et al.*, 2025b; Lu and Song, 2024). As summarized in Section 6.2.2, the main factors that influence stable training include the *diffusion process*, *parameterization choices*, *time weighting function*, and *time sampling distribution*, all of which should be carefully designed and disentangled also in continuous-time CM.

**Diffusion Process.** Instead of using the standard diffusion parameterization  $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$  with  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , Lu and Song (2024) adopt a trigonometric schedule. This schedule, although mathematically equivalent to the original form (as shown in Equation (6.3.4)), provides a cleaner structure and a better separation in the training objective, which contributes to improved stability during training<sup>5</sup>. In addition, they incorporate the standard deviation  $\sigma_d$  of the data distribution  $p_{\text{data}}$ , in line with EDM’s design in Section D.6.1:

$$\mathbf{x}_s := \cos(s) \mathbf{x}_0 + \sin(s) \mathbf{z}, \quad \text{where } \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma_d^2 \mathbf{I}). \quad (11.3.7)$$

This formulation is fully general. For any diffusion process of the form  $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$  with  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , we can equivalently write:

$$\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \frac{\sigma_s}{\sigma_d} \cdot (\sigma_d \boldsymbol{\epsilon}),$$

by defining  $\mathbf{z} := \sigma_d \boldsymbol{\epsilon}$ ,  $\alpha'_s := \alpha_s$ , and  $\sigma'_s := \frac{\sigma_s}{\sigma_d}$ . The transformed pair  $(\alpha'_s, \sigma'_s)$  can then be mapped to the trigonometric form  $(\cos(s), \sin(s))$  using the normalization described in Equation (6.3.5).

**Parametrizations.** By considering the analogous principles of EDM in Section D.6.1, Lu and Song (2024) propose the following parametrization for the neural network similar to Equation (D.6.1):

$$\mathbf{f}_{\theta}(\mathbf{x}, s) := c_{\text{skip}}(s) \mathbf{x} + c_{\text{out}}(s) \mathbf{F}_{\theta}(c_{\text{in}}(s) \mathbf{x}, c_{\text{noise}}(s)).$$

Here,  $c_{\text{skip}}(s)$ ,  $c_{\text{out}}(s)$ , and  $c_{\text{in}}(s)$  can be derived using the same criteria presented in Section D.6.1 (see Appendix B of Lu and Song (2024) for detailed derivations), and are given by

$$c_{\text{skip}}(s) = \cos(s), \quad c_{\text{out}}(s) = -\sigma_d \sin(s), \quad c_{\text{in}}(s) \equiv \frac{1}{\sigma_d}.$$

This is considered along with the default choice  $c_{\text{noise}}(s) = s$ , where  $\partial_s c_{\text{noise}}(s)$  is bounded to ensure training stability, as will be discussed around Equation (11.3.10). This leads to the following parametrization under the trigonometric schedule:

$$\mathbf{f}_{\theta}(\mathbf{x}, s) = \cos(s) \mathbf{x} - \sin(s) \sigma_d \mathbf{F}_{\theta}\left(\frac{\mathbf{x}}{\sigma_d}, c_{\text{noise}}(s)\right). \quad (11.3.8)$$

We note that this parametrization also ensures that the neural network always satisfies the boundary condition  $\mathbf{f}_{\theta}(\mathbf{x}, 0) \equiv \mathbf{x}$  for all  $\mathbf{x}$ , which is an essential property of a consistency function.

---

<sup>5</sup>Intuitively, both the trigonometric functions and their derivatives are bounded, which helps prevent scale explosion in terms like  $\frac{d}{ds} \mathbf{f}_{\theta}$ . A detailed discussion is provided later.

**Techniques for Stabilizing Tangent Training.** Under the trigonometric schedule and the network parametrization described in Equation (11.3.8), the gradient of the loss in Equation (11.3.4) becomes

$$\nabla_{\theta} \mathcal{L}_{\text{CT}}^{\infty}(\theta, \theta^-) = \nabla_{\theta} \mathbb{E}_{s, \mathbf{x}_0, \epsilon} \left[ -\omega(s) \sigma_d \sin(s) \mathbf{F}_{\theta}^{\top} \left( \frac{\mathbf{x}_s}{\sigma_d}, s \right) \cdot \frac{d\mathbf{f}_{\theta^-}}{ds}(\mathbf{x}_s, s) \right]. \quad (11.3.9)$$

In theory, training with the gradient update in Equation (11.3.9) may be sufficient to learn a consistency function. However, Lu and Song (2024) empirically observed that the training process can become unstable in practice due to the behavior of the tangent function, given by

$$\begin{aligned} & \underbrace{\frac{d\mathbf{f}_{\theta^-}(\mathbf{x}_s, s)}{ds}}_{\mathbf{A.}} = \\ & -\cos(s) \left( \sigma_d \nabla_{\mathbf{x}_s} \mathbf{F}_{\theta^-} \left( \frac{\mathbf{x}_s}{\sigma_d}, s \right) - \frac{d\mathbf{x}_s}{ds} \right) - \sin(s) \left( \mathbf{x}_s + \sigma_d \frac{d\mathbf{F}_{\theta^-}}{ds} \left( \frac{\mathbf{x}_s}{\sigma_d}, c_{\text{noise}}(s) \right) \right). \end{aligned}$$

In particular, instability was observed in the term

$$\underbrace{\sin(s) \frac{d\mathbf{F}_{\theta^-}}{ds} \left( \frac{\mathbf{x}_s}{\sigma_d}, c_{\text{noise}}(s) \right)}_{\mathbf{B.}} = \sin(s) \nabla_{\mathbf{x}_s} \mathbf{F}_{\theta^-} \frac{d\mathbf{x}_s}{ds} + \sin(s) \partial_s \mathbf{F}_{\theta^-}.$$

More specifically, the instability arises from the component

$$\sin(s) \partial_s \mathbf{F}_{\theta^-} = \sin(s) \underbrace{\frac{\partial c_{\text{noise}}(s)}{\partial s} \cdot \frac{\partial \text{emb}(c_{\text{noise}})}{\partial c_{\text{noise}}}}_{\mathbf{C.}} \cdot \frac{\partial \mathbf{F}_{\theta^-}}{\partial \text{emb}(c_{\text{noise}})}. \quad (11.3.10)$$

Here, we follow a common practice in the DM and CM literature by applying a positional or Fourier embedding, denoted by  $\text{emb}(\cdot)$ , to the time variable  $c_{\text{noise}}(s)$ :

$$s \mapsto c_{\text{noise}}(s) \mapsto \text{emb}(c_{\text{noise}}(s)) \mapsto \mathbf{F}_{\theta^-} \left( \frac{\mathbf{x}_s}{\sigma_d}, \text{emb}(c_{\text{noise}}(s)) \right).$$

Therefore, some additional empirical techniques are introduced to mitigate the instability:

- **A. Tangent Normalization.** Explicitly normalize the tangent function by replacing  $\frac{d}{ds} \mathbf{f}_{\theta^-}$  with  $\frac{\frac{d}{ds} \mathbf{f}_{\theta^-}}{\|\frac{d}{ds} \mathbf{f}_{\theta^-}\|_2 + c}$ , where  $c > 0$  is a constant set empirically. Alternatively, clipping the tangent within  $[-1, 1]$  can also effectively cap its variance.

- **B. Tangent Warm-Up.** Since the term  $\sin(s)(\mathbf{x}_s + \sigma_d \frac{d}{ds} \mathbf{F}_{\theta^-})$  may induce instability, an optional technique can be applied by replacing the coefficient  $\sin(s)$  with  $r \cdot \sin(s)$ , where  $r$  linearly increases from 0 to 1 over the first few training iterations.
- **C. Time Embedding.** In light of the derivative chain in Equation (11.3.10), Lu and Song (2024) opted for a smaller magnitude parameter to control the derivative  $\frac{\partial_{\text{emb}}(c_{\text{noise}})}{\partial c_{\text{noise}}}$ . For a similar reason,  $c_{\text{noise}}(s) = s$  is chosen, where  $\partial_s c_{\text{noise}}(s) = 1 - a$  bounded constant.

On top of these, architectural changes for improved normalization (for stability) and efficient JVP-based computation of  $\frac{d}{ds} \mathbf{f}_{\theta^-}$  are often necessary, but beyond our scope.

**Time-Weighting Function.** Manual design of the time-weighting function  $\omega(s)$  may lead to suboptimal performance. To address this, following a similar approach to EDM-2 (Karras *et al.*, 2024), Lu and Song (2024) learn an adaptive weighting function  $\omega_\varphi(s)$  to balance the training loss variance across different times  $s$  (see Equation (11.3.11) for the desired outcome).

To elaborate further, we observe that the objective function in Equation (11.3.9) takes the form

$$\mathbb{E}_{s, \mathbf{x}_0, \epsilon} [\mathbf{F}_\theta^\top \mathbf{y}], \quad \text{with } \mathbf{y} = -\omega(s) \sigma_d \sin(s) \frac{d\mathbf{f}_{\theta^-}}{ds}.$$

Since  $\mathbf{y}$  is a vector independent of  $\theta$ , Equation (11.3.9) is equivalent to

$$\nabla_\theta \mathbb{E}_{s, \mathbf{x}_0, \epsilon} [\mathbf{F}_\theta^\top \mathbf{y}] = \frac{1}{2} \nabla_\theta \mathbb{E}_{s, \mathbf{x}_0, \epsilon} \|\mathbf{F}_\theta - \mathbf{F}_{\theta^-} + \mathbf{y}\|_2^2.$$

Based on this observation, Lu and Song (2024) propose additionally training an adaptive weighting network  $\omega_\varphi(s)$  to estimate the loss norm, formulated as the following minimization problem:

$$\min_\varphi \mathbb{E}_{s, \mathbf{x}_0, \epsilon} \left[ \frac{e^{\omega_\varphi(s)}}{D} \|\mathbf{F}_\theta - \mathbf{F}_{\theta^-} + \mathbf{y}\|_2^2 - \omega_\varphi(s) \right].$$

To understand the effect of the adaptive weighting, observe that the optimal solution  $\omega^*(s)$  (obtained by taking the partial derivative of the above objective with respect to  $\omega_\varphi$ ) satisfies

$$\mathbb{E}_{s, \mathbf{x}_0, \epsilon} \left[ \frac{e^{\omega^*(s)}}{D} \|\mathbf{F}_\theta - \mathbf{F}_{\theta^-} + \mathbf{y}\|_2^2 \right] = 1. \quad (11.3.11)$$

That is, after rescaling, the expected (weighted) loss across different  $s$  is kept uniform. As a result, the adaptive weighting effectively reduces the variance of the training loss across different time steps, leading to more balanced and stable training.

**Time Sampling Distribution.** Lu and Song (2024) opt to sample  $\tan(s)$  from a log-normal proposal distribution (Karras *et al.*, 2022), that is,

$$e^{\sigma_d \tan(s)} \sim \mathcal{N}(\cdot; P_{\text{mean}}, P_{\text{std}}^2). \quad (11.3.12)$$

Here,  $P_{\text{mean}}$  and  $P_{\text{std}}$  are two hyper-parameters.

**Summary of Training Objective.** In summary of the aforementioned discussion, the final training loss is expressed as:

$$\begin{aligned} \mathcal{L}_{\text{sCM}}(\theta, \varphi) := \\ \mathbb{E}_{s, \mathbf{x}_0, \epsilon} \left[ \frac{e^{\omega_\varphi(s)}}{D} \left\| \mathbf{F}_\theta \left( \frac{\mathbf{x}_s}{\sigma_d}, s \right) - \mathbf{F}_{\theta^-} \left( \frac{\mathbf{x}_s}{\sigma_d}, s \right) - \cos(s) \frac{d\mathbf{f}_{\theta^-}}{ds} (\mathbf{x}_s, s) \right\|_2^2 - \omega_\varphi(s) \right]. \end{aligned}$$

Here,  $s$  is sampled according to Equation (11.3.12), and  $\mathbf{x}_s$  is computed via Equation (11.3.7). The model trained with this loss is referred to as *sCM*, and its training procedure is summarized in Algorithm 10.

---

**Algorithm 10** Training of Continuous-time Consistency Models (sCM)
 

---

**Input:** dataset  $\mathcal{D}$  with std.  $\sigma_d$ , pre-trained DM  $\mathbf{F}_{\text{pretrain}}$  with parameter  $\theta_{\text{pretrain}}$ , model  $\mathbf{F}_\theta$ , weighting  $\omega_\varphi$ , learning rate  $\eta$ , proposal  $(P_{\text{mean}}, P_{\text{std}})$ , constant  $c$ , warmup iteration  $H$

- 1: **Init:**  $\theta \leftarrow \theta_{\text{pretrain}}$ , Iters  $\leftarrow 0$
- 2: **Repeat**
- 3:    $\mathbf{x}_0 \sim \mathcal{D}$ ,  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma_d^2 \mathbf{I})$ ,  $\tau \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$ ,  $s \leftarrow \arctan\left(\frac{e^\tau}{\sigma_d}\right)$
- 4:    $\mathbf{x}_s \leftarrow \cos(s)\mathbf{x}_0 + \sin(s)\mathbf{z}$
- 5:   **if** consistency training **then**
- 6:      $\frac{d\mathbf{x}_s}{ds} \leftarrow \cos(s)\mathbf{z} - \sin(s)\mathbf{x}_0$
- 7:   **else**
- 8:      $\frac{d\mathbf{x}_s}{ds} \leftarrow \sigma_d \mathbf{F}_{\text{pretrain}}\left(\frac{\mathbf{x}_s}{\sigma_d}, s\right)$
- 9:   **end if**
- 10:    $r \leftarrow \min\left(1, \frac{\text{Iters}}{H}\right)$  ▷ Tangent warmup
- 11:    $\mathbf{w} \leftarrow -\cos^2(s)(\sigma_d \mathbf{F}_\theta^- - \frac{d\mathbf{x}_s}{ds}) - r \cos(s) \sin(s) \left( \mathbf{x}_s + \sigma_d \frac{d\mathbf{F}_\theta^-}{ds} \right)$
- 12:    $\mathbf{w} \leftarrow \frac{\mathbf{w}}{\|\mathbf{w}\| + c}$  ▷ Tangent normalization
- 13:    $\mathcal{L}_{\text{sCM}}(\theta, \varphi) \leftarrow \frac{e^{\omega_\varphi(s)}}{D} \left\| \mathbf{F}_\theta\left(\frac{\mathbf{x}_s}{\sigma_d}, s\right) - \mathbf{F}_{\theta^-}\left(\frac{\mathbf{x}_s}{\sigma_d}, s\right) - \mathbf{w} \right\|_2^2 - \omega_\varphi(s)$  ▷ Adaptive weighting
- 14:    $(\theta, \varphi) \leftarrow (\theta, \varphi) - \eta \nabla_{\theta, \varphi} \mathcal{L}_{\text{sCM}}(\theta, \varphi)$
- 15:   Iters  $\leftarrow$  Iters + 1
- 16: **until** convergence

---

## 11.4 General Flow Map: Consistency Trajectory Model

Consistency Trajectory Model (CTM) (Kim *et al.*, 2024a) is among the first methods to learn a *general* flow map  $\Psi_{s \rightarrow t}$ .

**Setup of CTM in Practice.** Similar to the CM family, CTM originally follows the formulation of EDM (Karras *et al.*, 2022) (Section D.6), using the PF-ODE in  $\mathbf{x}$ -prediction form with the noise schedule  $\alpha_t = 1$  and  $\sigma_t = t$ . Under this setup, the PF-ODE becomes

$$\frac{d\mathbf{x}(\tau)}{d\tau} = \frac{\mathbf{x}(\tau) - \mathbb{E}[\mathbf{x}|\mathbf{x}(\tau)]}{\tau}.$$

Starting from  $\mathbf{x}_s$  at time  $s$  and evolving to a later time  $t \leq s$ , the corresponding flow map (solution) can be written equivalently as

$$\mathbf{x}_s + \int_s^t \frac{\mathbf{x}_\tau - \mathbb{E}[\mathbf{x}|\mathbf{x}_\tau]}{\tau} d\tau.$$

CTM adopts an Euler-inspired parameterization: applying a single-step Euler solver (equivalently, DDIM; see Equation (9.2.4)) to the PF-ODE yields

$$\mathbf{x}_{s \rightarrow t}^{\text{Euler}} = \mathbf{x}_s - (s-t) \frac{\mathbf{x}_s - \mathbb{E}[\mathbf{x}|\mathbf{x}_s]}{s} = \frac{t}{s} \mathbf{x}_s + \left(1 - \frac{t}{s}\right) \mathbb{E}[\mathbf{x}|\mathbf{x}_s],$$

where  $\mathbf{x}_{s \rightarrow t}^{\text{Euler}}$  approximates the solution at time  $t$  given the state  $\mathbf{x}_s$  at time  $s$ .

While the EDM setup provides a simple illustrative case, CTM allows broader noise schedules defined by an arbitrary linear Gaussian forward kernel  $(\alpha_t, \sigma_t)$  and expresses the PF-ODE in  $\mathbf{v}$ -prediction form:

$$\Psi_{s \rightarrow t}(\mathbf{x}_s) = \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du.$$

In the discussion that follows, we focus on this general formulation.

### 11.4.1 CTM Parametrization for Flexible Transition Learning

Following the single-step Euler solver of the PF-ODE above, CTM rewrites the oracle flow map  $\Psi_{s \rightarrow t}$  as a convex combination of the input  $\mathbf{x}_s$  and a residual function  $\mathbf{g}^*$ :

$$\Psi_{s \rightarrow t}(\mathbf{x}_s) := \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du = \frac{t}{s} \mathbf{x}_s + \underbrace{\frac{s-t}{s} \left[ \mathbf{x}_s + \frac{s}{s-t} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du \right]}_{=: \mathbf{g}^*}.$$

where the residual term  $\mathbf{g}^*$  is defined as

$$\mathbf{g}^*(\mathbf{x}_s, s, t) := \mathbf{x}_s + \frac{s}{s-t} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du. \quad (11.4.1)$$

This motivates the neural parameterization

$$\mathbf{G}_\theta(\mathbf{x}_s, s, t) := \frac{t}{s} \mathbf{x}_s + \frac{s-t}{s} \mathbf{g}_\theta(\mathbf{x}_s, s, t), \quad (11.4.2)$$

where  $\mathbf{g}_\theta$  is a neural network that aims at  $\mathbf{g}_\theta \approx \mathbf{g}^*$ , hence  $\mathbf{G}_\theta(\mathbf{x}_s, s, t)$  is trained to approximate the oracle flow map,

$$\mathbf{G}_\theta(\mathbf{x}_s, s, t) \approx \Psi_{s \rightarrow t}(\mathbf{x}_s).$$

Therefore, CTM naturally fits within the general consistency-mapping framework of Equation (10.1.4), which aligns the learned mapping with the oracle flow map.

Moreover, this formulation inherently satisfies the initial condition

$$\mathbf{G}_\theta(\mathbf{x}_s, s, s) = \mathbf{x}_s,$$

without requiring any explicit enforcement during training.

**Advantages of CTM's Parametrizations.** A crucial characteristic of  $\mathbf{g}^*$  becomes evident when taking the limit as  $t$  approaches  $s$  (i.e., the same ending time as the starting time):

### Proposition 11.4.1: Properties of $\mathbf{g}^*$

#### (i) Recovering Diffusion Model:

$$\mathbf{g}^*(\mathbf{x}_s, s, s) = \lim_{t \rightarrow s} \mathbf{g}^*(\mathbf{x}_s, s, t) = \mathbf{x}_s - s\mathbf{v}^*(\mathbf{x}_s, s).$$

#### (ii) Integration Representation:

$$\mathbf{g}^*(\mathbf{x}_s, s, t) = \mathbf{x}_s - s\mathbf{v}^*(\mathbf{x}_s, s) + \mathcal{O}(|t-s|).$$

### Proof for Proposition.

From the definition of  $\mathbf{g}^*$ , we obtain

$$\lim_{s \rightarrow t} \mathbf{g}^*(\mathbf{x}_s, s, t) = \mathbf{x}_t - s \lim_{s \rightarrow t} \frac{1}{t-s} \int_s^t \mathbf{v}^*(\mathbf{x}_\tau, \tau) d\tau = \mathbf{x}_s - s\mathbf{v}^*(\mathbf{x}_s, s).$$

This proved the first identity. For the second claim, from the Taylor expansion, we have

$$\begin{aligned}\mathbf{g}^*(\mathbf{x}_s, s, t) &= \mathbf{x}_s - \frac{s}{s-t} \int_t^s \mathbf{v}^*(\mathbf{x}_\tau, \tau) d\tau \\ &= \mathbf{x}_s - \frac{s}{s-t} \left[ (s-t)\mathbf{v}^*(\mathbf{x}_s, s) + \mathcal{O}((t-s)^2) \right] \\ &= \mathbf{x}_s - s\mathbf{v}^*(\mathbf{x}_s, s) + \mathcal{O}(|t-s|).\end{aligned}$$

From this proposition, we can conclude that

1. Estimating  $\mathbf{g}^*$  enables approximating not only the finite  $s$ -to- $t$  transition (for  $s \leq t$ ) but also the *infinitesimal*  $s$ -to- $s$  transition characterized by the instantaneous velocity  $\mathbf{v}^*$ .
2.  $\mathbf{g}^*(\mathbf{x}_s, s, t)$  is interpreted as the oracle velocity  $\mathbf{v}^*$  added with a residual term of the Taylor expansion.

Therefore, by leveraging CTM's parameterization in Equation (11.4.2), learning  $\mathbf{G}_\theta \approx \Psi_{s \rightarrow t}$  (or equivalently,  $\mathbf{g}_\theta \approx \mathbf{g}^*$ ) enables both long-jump capability via  $\mathbf{G}_\theta$ , and recovery of the diffusion model's velocity (or equivalently, the score function/denoiser) via  $\mathbf{g}_\theta$ . This parameterization is thus key: by learning  $\mathbf{g}^*$ , CTM unifies the strengths of diffusion models and consistency model (special flow map) under a single framework.

In the next two sections, we first present CTM's consistency loss (Section 11.4.2), which supports both distillation and training-from-scratch, and enforces the semigroup property to achieve  $\mathbf{G}_\theta(\cdot, s, t) \approx \Psi_{s \rightarrow t}(\cdot, s, t)$ . We then describe auxiliary losses (Section 11.4.3) that arise naturally from the parameterization in Equation (11.4.2), including diffusion model loss and GAN loss, which further improve CTM's performance significantly.

### 11.4.2 Consistency Loss in CTM

CTM aims to approximate the oracle solution map

$$\mathbf{G}_\theta(\cdot, s, t) \approx \Psi_{s \rightarrow t}(\cdot, s, t),$$

for any  $s \geq t$ . Since the oracle  $\Psi_{s \rightarrow t}$  is usually not available in closed form, CTM builds a feasible regression target by enforcing the *semigroup property* (Equation (11.2.1)): for any  $s \geq u \geq t$ ,

$$\Psi_{u \rightarrow t} \circ \Psi_{s \rightarrow u} = \Psi_{s \rightarrow t}.$$

Depending on whether a pre-trained diffusion model is available, the flow map  $\Psi_{s \rightarrow t}$  can be approximated in different ways. Throughout, we assume  $s \geq u \geq t \in [0, T]$ .

**Training via Distillation.** Assume access to a pre-trained diffusion model producing  $\mathbf{v}_{\phi^\times}(\mathbf{x}_s, s) \approx \mathbf{v}^*(\mathbf{x}_s, s)$ . Then the PF-ODE is approximated by the empirical dynamics

$$\frac{d\mathbf{x}(\tau)}{d\tau} = \mathbf{v}_{\phi^\times}(\mathbf{x}_\tau, \tau). \quad (11.4.3)$$

CTM trains  $\mathbf{G}_\theta$  to match a numerical solver  $\text{Solver}_{s \rightarrow t}(\mathbf{x}_s; \phi^\times)$  applied to this empirical ODE, which serves as a computable proxy for the oracle:

$$\mathbf{G}_\theta(\mathbf{x}_s, s, t) \approx \text{Solver}_{s \rightarrow t}(\mathbf{x}_s; \phi^\times) \approx \Psi_{s \rightarrow t}(\mathbf{x}_s, s, t).$$

With a strong teacher, the solver can recover  $\Psi_{s \rightarrow t}$  up to discretization error, so the optimal student closely matches the ground truth (see (Kim *et al.*, 2024a), Propositions 3 and 4).

However, solving across the full interval  $[t, s]$  during training loop can be costly when  $s$  and  $t$  are far apart. To improve efficiency and provide a smoother signal, CTM introduces *soft consistency matching*, which operationalizes the semigroup property. As illustrated in Figure 11.4, CTM compares two predictions at time  $t$ : the direct student output  $\mathbf{G}_\theta(\mathbf{x}_s, s, t)$ , and a mixed teacher–student path that first advances the teacher from  $s$  to a random  $u \sim \mathcal{U}[t, s]$ , then lets the student jump from  $u$  to  $t$ :

$$\mathbf{G}_{\theta^-}(\text{Solver}_{s \rightarrow u}(\mathbf{x}_s; \phi^\times), u, t).$$

The student is trained to match this composite prediction:

$$\underbrace{\mathbf{G}_\theta(\mathbf{x}_s, s, t)}_{\approx \Psi_{s \rightarrow t}(\mathbf{x}_s)} \approx \underbrace{\mathbf{G}_{\theta^-}(\text{Solver}_{s \rightarrow u}(\mathbf{x}_s; \phi^\times), u, t)}_{\approx \Psi_{u \rightarrow t}(\Psi_{s \rightarrow u}(\mathbf{x}_s))}, \quad (11.4.4)$$

where  $\theta^-$  is a stop gradient copy of  $\mathbf{G}_\theta$ .

By varying  $u$ , CTM interpolates between global and local supervision:

- **Global Consistency** ( $u = s$ ): the student mimics the teacher over the full interval  $(t, s)$ , receiving the most informative teacher signal.
- **Local Consistency** ( $u = s - \Delta s$ ): the student learns from a short teacher step near  $s$ ; when  $s = 0$ , this reduces to consistency distillation.

To reinforce sample quality while aligning trajectories, both predictions are mapped to time 0 by the stop gradient student and compared in a feature space metric  $d$ :

$$\begin{aligned}\mathbf{x}_{\text{est}}(\mathbf{x}_s, s, t) &:= \mathbf{G}_{\theta^-}(\mathbf{G}_{\theta}(\mathbf{x}_s, s, t), t, 0), \\ \mathbf{x}_{\text{target}}(\mathbf{x}_s, s, u, t) &:= \mathbf{G}_{\theta^-}(\mathbf{G}_{\theta^-}(\mathbf{Solver}_{s \rightarrow u}(\mathbf{x}_s; \phi^{\times}), u, t), t, 0).\end{aligned}$$

The CTM consistency loss is

$$\mathcal{L}_{\text{consist}}(\boldsymbol{\theta}; \phi^{\times}) := \mathbb{E}_{s \in [0, T]} \mathbb{E}_{t \in [0, s]} \mathbb{E}_{u \in [t, s]} \mathbb{E}_{\mathbf{x}_0} \mathbb{E}_{\mathbf{x}_s | \mathbf{x}_0} \left[ d(\mathbf{x}_{\text{est}}, \mathbf{x}_{\text{target}}) \right], \quad (11.4.5)$$

which encourages the student to match the empirical PF-ODE solution while preserving generation quality.

**Training from Scratch.** Leveraging CTM's special parameterization (Proposition 11.4.1(i)),

$$\mathbf{g}^*(\mathbf{x}_{\tau}, \tau, \tau) = \mathbf{x}_{\tau} - \tau \mathbf{v}^*(\mathbf{x}_{\tau}, \tau) \implies \mathbf{v}^*(\mathbf{x}_{\tau}, \tau) = \frac{\mathbf{x}_{\tau} - \mathbf{g}^*(\mathbf{x}_{\tau}, \tau, \tau)}{\tau}.$$

We can therefore replace the oracle residual function  $\mathbf{g}^*(\cdot, \tau, \tau)$  with CTM's own estimate  $\mathbf{g}_{\theta^-}(\cdot, \tau, \tau)$  for  $\tau \in [0, T]$ , which yields a self-induced empirical PF-ODE:

$$\frac{d\mathbf{x}(\tau)}{d\tau} = \frac{\mathbf{x}(\tau) - \mathbf{g}_{\theta^-}(\mathbf{x}(\tau), \tau, \tau)}{\tau}. \quad (11.4.6)$$

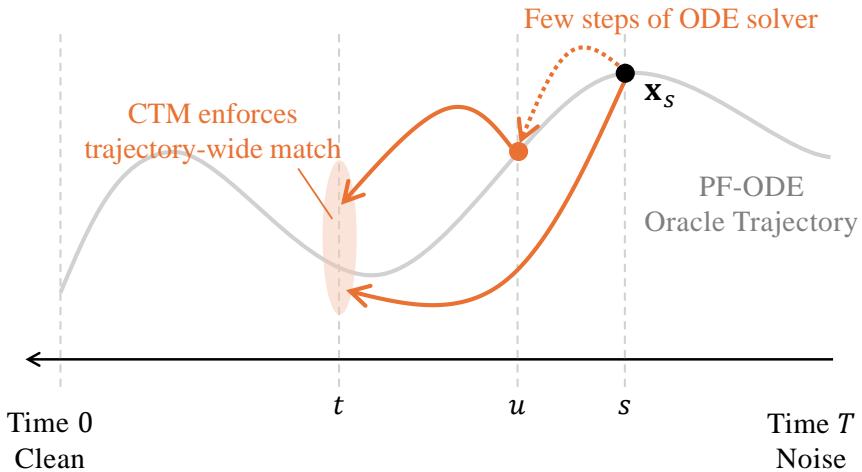
We then approximate the oracle solution map by solving this ODE and training the student to match the solver output:

$$\mathbf{G}_{\theta}(\mathbf{x}_s, s, t) \approx \mathbf{Solver}_{s \rightarrow t}(\mathbf{x}_s; \boldsymbol{\theta}^-) \approx \Psi_{s \rightarrow t}(\mathbf{x}_s, s, t).$$

As in the distillation case Equation (11.4.4), full integration over  $[t, s]$  can be costly when  $s$  and  $t$  are far apart. CTM therefore enforces the semigroup property to obtain a shorter supervision path:

$$\underbrace{\mathbf{G}_{\theta}(\mathbf{x}_s, s, t)}_{\approx \Psi_{s \rightarrow t}(\mathbf{x}_s)} \approx \underbrace{\mathbf{G}_{\theta^-}(\mathbf{Solver}_{s \rightarrow u}(\mathbf{x}_s; \boldsymbol{\theta}^-), u, t)}_{\approx \Psi_{u \rightarrow t}(\Psi_{s \rightarrow u}(\mathbf{x}_s))},$$

where  $u \sim \mathcal{U}[t, s]$  and  $\boldsymbol{\theta}^-$  is a stop gradient copy of the student. The only change from distillation is that the external teacher  $\mathbf{v}_{\phi^{\times}}$  is replaced by the self-induced teacher  $\mathbf{g}_{\theta^-}$ .



**Figure 11.4: Illustration of CTM’s semigroup property.** For any  $s \geq u \geq t$ , CTM enforces  $\mathbf{G}_\theta(\mathbf{x}_s, s, t) \approx \mathbf{G}_{\theta^-}(\text{Solver}_{s \rightarrow u}(\mathbf{x}_s), u, t)$ , i.e., a short solver segment  $s \rightarrow u$  followed by a CTM “jump” to  $t$  matches the direct CTM map  $s \rightarrow t$ . The solver may be a pre-trained diffusion or a CTM’s self-induced teacher.

To couple trajectory matching with sample quality, both predictions are mapped to time 0 using the stop gradient student and compared in feature space. The target without any pre-trained model is

$$\hat{\mathbf{x}}_{\text{target}} := \mathbf{G}_{\theta^-}(\mathbf{G}_{\theta^-}(\text{Solver}_{s \rightarrow u}(\mathbf{x}_s; \theta^-), u, t), t, 0),$$

which replaces  $\mathbf{x}_{\text{target}}$  in Equation (11.4.5), and leads to:

$$\mathcal{L}_{\text{consist}}(\theta; \theta^-) := \mathbb{E}_{s \in [0, T]} \mathbb{E}_{t \in [0, s]} \mathbb{E}_{u \in [t, s]} \mathbb{E}_{\mathbf{x}_0} \mathbb{E}_{\mathbf{x}_s | \mathbf{x}_0} [d(\mathbf{x}_{\text{est}}, \hat{\mathbf{x}}_{\text{target}})], \quad (11.4.7)$$

Conceptually, this is self-distillation within CTM: the model supplies its own short horizon teacher signals while the student learns the full transition.

### 11.4.3 Auxiliary Losses in CTM

(Self-)distillation can underperform the teacher because it optimizes only teacher generated targets, lacking direct supervision from real data. By contrast, CTM can naturally incorporate data driven regularizers, for example by augmenting its objective with denoising score matching and an adversarial (GAN) term (Goodfellow *et al.*, 2014), to better learn the flow map.

**Natural Integration of Diffusion Loss.** The diffusion–model loss (more precisely, the conditional flow matching loss; see Equation (5.2.9)) integrates naturally into CTM and provides a fixed regression target that facilitates the learning of the flow map model. To see this, note that we have

$$\mathbf{v}^*(\mathbf{x}_s, s) = \frac{\mathbf{x}_s - \mathbf{g}^*(\mathbf{x}_s, s, s)}{s}, \quad \mathbf{g}^*(\mathbf{x}_s, s, s) \approx \mathbf{g}_\theta(\mathbf{x}_s, s, s).$$

This naturally induces a velocity parametrization through  $\mathbf{g}_\theta$ :

$$\mathbf{v}_\theta(\mathbf{x}_s, s) := \frac{1}{s}(\mathbf{x}_s - \mathbf{g}_\theta(\mathbf{x}_s, s, s)).$$

Using the linear Gaussian path

$$\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}, \quad \mathbf{x}_0 \sim p_{\text{data}}, \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}),$$

the diffusion model loss can be written as

$$\mathcal{L}_{\text{DM}}(\theta) := \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}, s} [w(s) \|\mathbf{v}_\theta(\mathbf{x}_s, s) - (\alpha'_s \mathbf{x}_0 + \sigma'_s \boldsymbol{\epsilon})\|_2^2]. \quad (11.4.8)$$

$\mathcal{L}_{\text{DM}}$  improves accuracy when  $t$  is close to  $s$  by explicitly supervising small jumps along the trajectory. In this regime, the factor  $1 - \frac{t}{s}$  in Equation (11.4.2) approaches zero, which can weaken gradients and slow learning;  $\mathcal{L}_{\text{DM}}$  supplies a stronger local signal and stabilizes training.

Conceptually, Equations (11.4.5) and (11.4.7) enforce trajectory matching (zeroth order), while Equation (11.4.8) enforces slope matching (first order).

**(Optional) GAN Loss.** While consistency and diffusion model loss provide strong regression signals, they can yield overly smooth outputs. CTM therefore optionally adds an adversarial term to encourage sharper, more realistic samples by aligning the generator distribution with the data distribution. With a discriminator  $D_\zeta$  that distinguishes real  $\mathbf{x}_0 \sim p_{\text{data}}$  from generated  $\mathbf{x}_{\text{est}}(\mathbf{x}_s, s, t)$ , the objective is

$$\begin{aligned} \mathcal{L}_{\text{GAN}}(\theta, \zeta) := \\ \mathbb{E}_{\mathbf{x}_0} [\log D_\zeta(\mathbf{x}_0)] + \mathbb{E}_{s \in [0, T]} \mathbb{E}_{t \in [0, s]} \mathbb{E}_{\mathbf{x}_0} \mathbb{E}_{\mathbf{x}_s | \mathbf{x}_0} [\log(1 - D_\zeta(\mathbf{x}_{\text{est}}(\mathbf{x}_s, s, t)))], \end{aligned}$$

where  $D_\zeta$  is maximized and  $\mathbf{G}_\theta$  is minimized. Intuitively, the discriminator acts as an adaptive perceptual distance that encourages realistic detail. Theoretically, the GAN term drives distributional matching (Jensen–Shannon divergence) between  $p_{\text{data}}$  and the model distribution induced by  $\mathbf{G}_\theta$  (Goodfellow *et al.*, 2014), which can raise fidelity beyond the teacher.

**Overall CTM Objective.** In summary, CTM unifies (self-)distillation, diffusion, and GAN losses into a single training framework:

$$\mathcal{L}_{\text{CTM}}(\boldsymbol{\theta}, \boldsymbol{\zeta}) := \mathcal{L}_{\text{consist}}(\boldsymbol{\theta}; \boldsymbol{\phi}^{\times}/\boldsymbol{\theta}^-) + \lambda_{\text{DM}} \mathcal{L}_{\text{DM}}(\boldsymbol{\theta}) + \lambda_{\text{GAN}} \mathcal{L}_{\text{GAN}}(\boldsymbol{\theta}, \boldsymbol{\zeta}),$$

where the teacher is either an external pre-trained model  $\boldsymbol{\phi}^{\times}$  or the self-induced teacher  $\boldsymbol{\theta}^-$ . The regression style components  $\mathcal{L}_{\text{consist}}$  and  $\mathcal{L}_{\text{DM}}$  act as strong regularizers, while the optional GAN term improves fine scale detail without sacrificing stability (Kim *et al.*, 2024b).

#### 11.4.4 Flexible Sampling with CTM

CTM learns the general flow map  $\Psi_{s \rightarrow t}$  for any  $s > t$ , which means it supports anytime to anytime transitions. This property enables flexible sampling strategies. For example, CTM proposes  $\gamma$  *sampling*, where the hyperparameter  $\gamma$  controls the stochasticity during generation. In addition, CTM can reuse standard inference techniques developed for diffusion models, such as ODE based solvers and exact likelihood computation.

In what follows, we fix a discrete time grid for sampling  $T = \tau_0 > \tau_1 > \tau_2 > \dots > \tau_M = 0$ .

---

##### Algorithm 11 CTM’s $\gamma$ -sampling

---

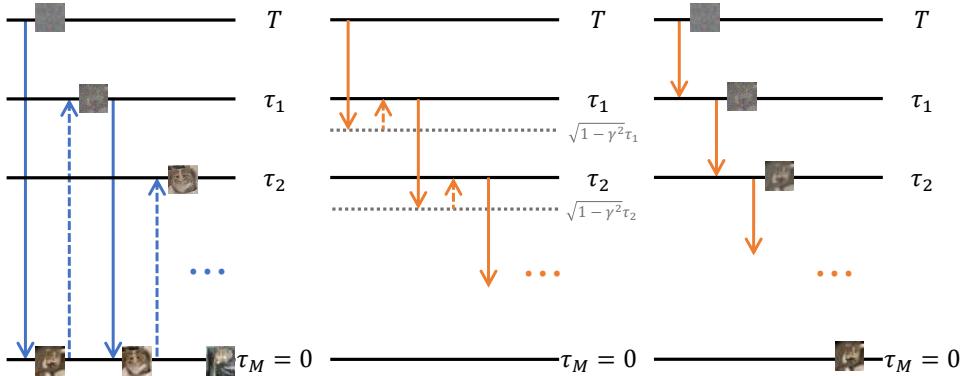
**Input:** Trained CTM  $\mathbf{G}_{\boldsymbol{\theta}^{\times}}$ ,  $\gamma \in [0, 1]$ ,  $T = \tau_0 > \tau_1 > \tau_2 > \dots > \tau_M = 0$ .

- 1: Start from  $\mathbf{x}_{\tau_0} \sim p_{\text{prior}} = \mathcal{N}(\mathbf{0}, T^2 \mathbf{I})$
- 2: **for**  $n = 0$  to  $M - 1$  **do**
- 3:      $\tilde{\tau}_{n+1} \leftarrow \sqrt{1 - \gamma^2} \tau_{n+1}$
- 4:     Denoise  $\mathbf{x}_{\tilde{\tau}_{n+1}} \leftarrow \mathbf{G}_{\boldsymbol{\theta}^{\times}}(\mathbf{x}_{\tau_n}, \tau_n, \tilde{\tau}_{n+1})$
- 5:     Diffuse  $\mathbf{x}_{\tau_{n+1}} \leftarrow \mathbf{x}_{\tilde{\tau}_{n+1}} + \gamma \tau_{n+1} \boldsymbol{\epsilon}$ , where  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- 6: **end for**

**Output:**  $\mathbf{x}_{\tau_M}$

---

**Methodology of  $\gamma$ -Sampling.** CTM’s  $\gamma$ -sampling introduces a unified family of samplers that arises naturally from learning a general flow map model. It encompasses prior approaches, such as CM’s multistep sampling (see Algorithm 9) and time-stepping-style sampling, which is conceptually similar to ODE solvers. The parameter  $\gamma$  directly controls the degree of semantic change during generation, making  $\gamma$  sampling a flexible and task aware strategy for diverse downstream applications.



**Figure 11.5: Illustration of  $\gamma$ -sampling with varying  $\gamma$  value.** The procedure alternates between denoising with a network evaluation and adding noise in reverse,  $(\tau_n \xrightarrow{\text{Denoise}} \sqrt{1 - \gamma^2} \tau_{n+1} \xrightarrow{\text{Noisify}} \tau_{n+1})_{n=0}^{M-1}$ . The leftmost panel illustrates  $\gamma = 1$ , corresponding to the fully stochastic case. The rightmost panel shows  $\gamma = 0$ , corresponding to the fully deterministic case. The middle panel depicts intermediate values  $\gamma \in (0, 1)$ , which interpolate between these two extremes.

- Figure 11.5-(Left): When  $\gamma = 1$ , it coincides to the multistep sampling introduced in CM (i.e., a special flow map  $\Psi_{s \rightarrow 0}$ ), which is fully stochastic and results in semantic variation when the number of steps changes.
- Figure 11.5-(Right): When  $\gamma = 0$ , it reduces to fully deterministic time-stepping, which estimates the solution trajectory of the PF-ODE. A key distinction between  $\gamma$  sampling with  $\gamma = 0$  and conventional time-stepping ODE-based sampling is that CTM avoids the discretization errors of numerical solvers.
- Figure 11.5-(Middle): When  $0 < \gamma < 1$ ,  $\gamma$ -sampling interpolates between the two extremes by allowing a controlled amount of stochasticity to be injected during sampling.

We highlight that the ability to realize samplers with  $\gamma \in (0, 1]$  is possible only when the model learns the general flow map  $\Psi_{s \rightarrow t}$ .

**Analysis of  $\gamma$ -Sampling.** CTM empirically observed that CM’s multistep sampling degrades in quality once the number of steps  $M \geq 4$ . To explain this phenomenon, CTM analyzed the underlying cause: when  $\gamma \neq 0$ , each neural “jump” introduces a small mismatch, and these mismatches accumulate as the model iteratively maps states toward time zero. This error accumulation

explains why long multi-step runs can perform poorly. We formalize this idea in the following proposition.

### Proposition 11.4.2: (Informal) 2-steps $\gamma$ -sampling

Let  $\tau \in (0, T)$  and  $\gamma \in [0, 1]$ . Let  $p_{\theta^*, 2}$  denote as the density obtained from the  $\gamma$ -sampler with the optimal CTM, following the transition sequence  $T \rightarrow \sqrt{1 - \gamma^2}\tau \rightarrow \tau \rightarrow 0$ , starting from  $p_{\text{prior}}$ . Then

$$\mathcal{D}_{\text{TV}}(p_{\text{data}}, p_{\theta^*, 2}) = \mathcal{O}\left(\sqrt{T - \sqrt{1 - \gamma^2}\tau + \tau}\right).$$

Here,  $\mathcal{D}_{\text{TV}}$  denotes the total variation between distributions (see Equation (1.1.4)).

#### Proof for Proposition.

We refer the reader to Theorem 8 of Kim *et al.* (2024a) for the general case when the number of sampling steps is  $M$ .

The insights from the above theorem can be summarized as follows:

- **When  $\gamma = 1$**  (corresponding to CM's multistep sampling): The method performs iterative long-range transitions from  $\tau_n$  to 0 at each step  $n$ . This leads to error accumulation on the order of

$$\mathcal{O}\left(\sqrt{T + \tau_1 + \dots + \tau_M}\right).$$

- **When  $\gamma = 0$**  (corresponding to CTM's deterministic multistep sampling): Such temporal overlap between transitions is eliminated. This avoids error accumulation and yields a tighter bound of  $\mathcal{O}(\sqrt{T})$ . Empirically, CTM with  $\gamma = 0$  provides a favorable trade-off between sampling speed and sample quality: increasing the number of sampling steps improves generation quality without introducing instability.

**CTM Supports Diffusion Inference.** Since CTM learns the score function (or denoiser) directly through  $\mathbf{g}_\theta$ , thanks to its parametrization in Equation (11.4.2), it is compatible with inference techniques originally developed for diffusion models. For instance, one can compute exact likelihoods (Section 4.2.2) or apply advanced samplers such as DDIM or DPM (Chapter 9) for generation, by using  $\mathbf{g}_\theta(\cdot, s, s)$ .

## 11.5 General Flow Map: Mean Flow

Just as diffusion models admit many equivalent parameterizations and training objectives, a general flow map  $\Psi_{s \rightarrow t}$  can also be learned in multiple plausible ways. In this section, we introduce *Mean Flow* (MF) (Geng *et al.*, 2025a), a later representative of the general flow map family  $\Psi_{s \rightarrow t}$  that illustrates an alternative yet principled perspective on how such maps can be effectively learned.

### 11.5.1 Modeling and Training of Mean Flow

In contrast to CM and CTM, which build on the EDM framework, MF is based on the flow matching formulation ( $\alpha_t = 1 - t$  and  $\sigma_t = t$  for  $t \in [0, 1]$ ). Rather than directly parameterizing the flow map, MF learns the *average drift* over an interval  $[t, s]$  (with  $t < s$ ):

$$\mathbf{h}_\theta(\mathbf{x}_s, s, t) \approx \mathbf{h}^*(\mathbf{x}_s, s, t) := \frac{1}{t-s} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du.$$

The corresponding oracle loss is

$$\mathbb{E}_{t < s} \mathbb{E}_{\mathbf{x}_s \sim p_s} [w(s) \|\mathbf{h}_\theta(\mathbf{x}_s, s, t) - \mathbf{h}^*(\mathbf{x}_s, s, t)\|_2^2]. \quad (11.5.1)$$

In particular, when  $s \rightarrow t$ , the loss function reduces to the flow matching loss:

$$\mathbb{E}_t \mathbb{E}_{\mathbf{x}_t \sim p_t} [w(t) \|\mathbf{h}_\theta(\mathbf{x}_t, t, t) - \mathbf{v}^*(\mathbf{x}_t, t)\|_2^2], \quad (11.5.2)$$

learning the instantaneous velocity. We will see later in Section 11.5.3 that MF remains consistent with the general objective in Equation (10.1.4), but approaches it from a different (while equivalent) perspective. Since the oracle regression target  $\mathbf{h}^*(\mathbf{x}_s, s, t)$  does not admit a closed form in general, MF constructs a surrogate by exploiting an identity obtained from differentiating

$$(t-s) \mathbf{h}^*(\mathbf{x}_s, s, t) = \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du$$

with respect to  $s$ . This yields

$$\begin{aligned} \mathbf{h}^*(\mathbf{x}_s, s, t) &= \mathbf{v}^*(\mathbf{x}_s, s) - (s-t) \frac{d}{ds} \mathbf{h}^*(\mathbf{x}_s, s, t) \\ &= \mathbf{v}^*(\mathbf{x}_s, s) - (s-t) [(\partial_{\mathbf{x}} \mathbf{h}^*)(\mathbf{x}_s, s, t) \mathbf{v}^*(\mathbf{x}_s, s) + \partial_s \mathbf{h}^*(\mathbf{x}_s, s, t)], \end{aligned}$$

where the second line applies the chain rule together with

$$\frac{d}{ds} \mathbf{x}_s = \mathbf{v}^*(\mathbf{x}_s, s).$$

Motivated by this identity, MF replaces the intractable oracle with a stop-gradient surrogate, leading to the practical training objective

$$\mathcal{L}_{\text{MF}}(\boldsymbol{\theta}) := \mathbb{E}_{t < s} \mathbb{E}_{\mathbf{x}_s \sim p_s} \left[ w(s) \|\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) - \mathbf{h}_{\boldsymbol{\theta}}^{\text{tgt}}(\mathbf{x}_s, s, t)\|_2^2 \right], \quad (11.5.3)$$

where the regression target is defined as

$$\begin{aligned} \mathbf{h}_{\boldsymbol{\theta}}^{\text{tgt}}(\mathbf{x}_s, s, t) := \\ \mathbf{v}^*(\mathbf{x}_s, s) - (s - t) \underbrace{\left[ (\partial_{\mathbf{x}} \mathbf{h}_{\boldsymbol{\theta}})(\mathbf{x}_s, s, t) \mathbf{v}^*(\mathbf{x}_s, s) + \partial_s \mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, t) \right]}_{\text{JVP}}. \end{aligned}$$

In practice, the oracle velocity  $\mathbf{v}^*$  cannot be computed in closed form and must instead be approximated. Two common strategies are available: relying on a pre-trained diffusion model (*distillation*) or constructing a direct estimator from data (*training from scratch*). Regardless of the choice, one ultimately needs to compute a *Jacobian–vector product* (JVP) of the target network  $\mathbf{h}_{\boldsymbol{\theta}}$ :

$$[\partial_{\mathbf{x}} \mathbf{h}_{\boldsymbol{\theta}}, \partial_s \mathbf{h}_{\boldsymbol{\theta}}, \partial_t \mathbf{h}_{\boldsymbol{\theta}}]^\top \cdot [\mathbf{v}^*, 1, 0]$$

**Distillation.** Use a pre-trained diffusion model with a flow matching backbone,  $\mathbf{v}_{\phi^x} \approx \mathbf{v}^*$ .

**Training from scratch.** Use the one point conditional velocity  $\alpha'_s \mathbf{x}_0 + \sigma'_s \boldsymbol{\epsilon}$ , obtained from the forward noise injection  $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$  with  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . This gives an unbiased single sample estimate of the instantaneous drift at level  $s$  when evaluated at paired  $(\mathbf{x}_0, \boldsymbol{\epsilon})$ .

### 11.5.2 Sampling of Mean Flow

Once a MF  $\mathbf{h}_{\boldsymbol{\theta}^x}$  is trained, it naturally recovers a proxy of the flow map. For any starting point  $\mathbf{x}_s$ , the map from  $s$  to  $t$  is (approximately) given by

$$\Psi_{s \rightarrow t}(\mathbf{x}_s) = \mathbf{x}_s + (t - s) \mathbf{h}^*(\mathbf{x}_s, s, t) \approx \mathbf{x}_s + (t - s) \mathbf{h}_{\boldsymbol{\theta}^x}(\mathbf{x}_s, s, t).$$

This enables both one-step and multi-step sampling. For example, drawing  $\mathbf{x}_T \sim p_{\text{prior}}$ , the one-step generation of a clean sample is

$$\mathbf{x}_0 \leftarrow \mathbf{x}_T + T \mathbf{h}_{\boldsymbol{\theta}^x}(\mathbf{x}_T, T, 0).$$

Alternatively, multi-step generation can be performed by preparing a time grid and applying the map sequentially, in the same time-stepping manner used in CTM. Since MF learns a general flow map, it also supports  $\gamma$ -sampling as in CTM, where a controllable hyperparameter  $\gamma$  injects stochasticity into the sampling process.

### 11.5.3 Equivalence of CTM and MF

At first sight CTM and MF may appear unrelated. In fact, both are simply different parameterizations of the same oracle flow map  $\Psi_{s \rightarrow t}$ , with their training losses (CTM's consistency loss versus Equation (11.5.1)) differing only in time weighting (Hu *et al.*, 2025).

**Relationship of Parameterizations.** Both methods operate under the same general framework but represent the learned function in distinct ways. The flow map can be written equivalently as

$$\begin{aligned}\Psi_{s \rightarrow t}(\mathbf{x}_s) &= \mathbf{x}_s + \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du \\ &= \frac{t}{s} \mathbf{x}_s + \frac{s-t}{s} \underbrace{\left[ \mathbf{x}_s + \frac{s}{s-t} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du \right]}_{\approx \mathbf{g}_\theta} \\ &= \mathbf{x}_s + (t-s) \underbrace{\left[ \frac{1}{t-s} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du \right]}_{\approx \mathbf{h}_\theta}.\end{aligned}$$

Here, the first is the definition of the flow map, the second form highlights the CTM parametrization through  $\mathbf{g}_\theta$  (see Equations (11.4.1) and (11.4.2)), while the last highlights the MF parametrization through  $\mathbf{h}_\theta$ .

**Relationship of Training Loss.** Given the above reinterpretation of the oracle flow map  $\Psi_{s \rightarrow t}$  in terms of the CTM parametrization

$$\mathbf{g}_\theta(\mathbf{x}_s, s, t) \approx \mathbf{g}^*(\mathbf{x}_s, s, t) := \mathbf{x}_s + \frac{s}{s-t} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du$$

and the MF parametrization

$$\mathbf{h}_\theta(\mathbf{x}_s, s, t) \approx \mathbf{h}^*(\mathbf{x}_s, s, t) := \frac{1}{t-s} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du,$$

we now show that the training losses of CTM and MF are in fact equivalent.

Consider the relation

$$\mathbf{g}_\theta(\mathbf{x}_s, s, t) := \mathbf{x}_s - s \mathbf{h}_\theta(\mathbf{x}_s, s, t),$$

and take  $d(\mathbf{x}, \mathbf{y}) := \|\mathbf{x} - \mathbf{y}\|^2$  as an example. Substituting into Equation (10.1.4) and viewing  $\mathbf{G}_\theta$  as CTM's flow-map parameterization (Equation (11.4.2))

gives

$$\begin{aligned}
& d(\mathbf{G}_\theta(\mathbf{x}_s, s, t), \Psi_{s \rightarrow t}(\mathbf{x}_s)) \\
&= \|\mathbf{G}_\theta(\mathbf{x}_s, s, t) - \Psi_{s \rightarrow t}(\mathbf{x}_s)\|^2 \\
&= \left\| \left( \frac{t}{s} \mathbf{x}_s + \frac{s-t}{s} \mathbf{g}_\theta(\mathbf{x}_s, s, t) \right) - \left( \frac{t}{s} \mathbf{x}_s + \frac{s-t}{s} \left[ \mathbf{x}_s + \frac{s}{s-t} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du \right] \right) \right\|^2 \\
&= \left( \frac{s-t}{s} \right)^2 \left\| \mathbf{g}_\theta(\mathbf{x}_s, s, t) - \left( \mathbf{x}_s + \frac{s}{s-t} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du \right) \right\|^2 \tag{11.5.4}
\end{aligned}$$

$$\begin{aligned}
&= \left( \frac{s-t}{s} \right)^2 \left\| (\mathbf{x}_s - s \mathbf{h}_\theta(\mathbf{x}_s, s, t)) - \left( \mathbf{x}_s + \frac{s}{s-t} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du \right) \right\|^2 \\
&= \left( \frac{s-t}{s} \right)^2 \left\| (\mathbf{x}_s - s \mathbf{h}_\theta(\mathbf{x}_s, s, t)) - \left( \mathbf{x}_s + \frac{s}{s-t} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du \right) \right\|^2 \\
&= (s-t)^2 \left\| \mathbf{h}_\theta(\mathbf{x}_s, s, t) - \left( \frac{1}{t-s} \int_s^t \mathbf{v}^*(\mathbf{x}_u, u) du \right) \right\|^2 \tag{11.5.5}
\end{aligned}$$

Hence,

$$\frac{1}{s^2} \left\| \mathbf{g}_\theta(\mathbf{x}_s, s, t) - \mathbf{g}^*(\mathbf{x}_s, s, t) \right\|^2 = \left\| \mathbf{h}_\theta(\mathbf{x}_s, s, t) - \mathbf{h}^*(\mathbf{x}_s, s, t) \right\|^2.$$

Thus CTM and MF losses are fundamentally equivalent up to a weighting function. Moreover, setting  $t = 0$  in either case recovers the CM setting ( $\Psi_{s \rightarrow 0}$ ), where each state maps directly to the clean data.

**Auxiliary Loss in Practice.** In CTM, training is performed with the consistency loss in Equation (11.4.7) jointly with its self-defined diffusion model loss in Equation (11.4.8). A similar strategy is adopted in MF. As shown in Equation (11.5.2), when  $s \rightarrow t$ , the MF loss reduces to the standard flow matching objective. In practice, MF controls the ratio between pairs with  $s \neq t$  and those with  $s = t$ ; consequently, the overall optimization becomes a mixture of the MF objective in Equation (11.5.3) and the flow matching objective in Equation (11.5.2).

Both parametrizations are able to provide a smooth transition from diffusion-model training, which learns instantaneous velocity with a fixed regression target, to flow-map learning, which employs a stop-gradient pseudo-regression target.

**Both CTM and MF Parameterizations Enable Flexible Inference.** Both CTM ( $\mathbf{G}_\theta(\mathbf{x}_s, s, t)$ ) and MF ( $\mathbf{h}_\theta(\mathbf{x}_s, s, t)$ ) aim to approximate the underlying

flow map  $\Psi_{s \rightarrow t}$ :

$$\mathbf{G}_\theta(\mathbf{x}_s, s, t) \approx \Psi_{s \rightarrow t}, \quad \text{and} \quad \mathbf{x}_s + (t - s)\mathbf{h}_\theta(\mathbf{x}_s, s, t) \approx \Psi_{s \rightarrow t}.$$

Since both models learn an explicit mapping between any two time steps, they naturally support CTM's  $\gamma$ -sampling and remain compatible with inference techniques originally developed for diffusion models, such as guidance (Chapter 8), exact likelihood computation (Equation (4.2.7)), and accelerated sampling with higher-order solvers (Chapter 9). This compatibility arises because their parameterizations recover the instantaneous diffusion drift in the infinitesimal limit  $t \rightarrow s$ :

$$\mathbf{g}^*(\mathbf{x}_s, s, s) = \mathbf{x}_s - \mathbf{v}^*(\mathbf{x}_s, s), \quad \text{and} \quad \mathbf{h}^*(\mathbf{x}_s, s, s) = \mathbf{v}^*(\mathbf{x}_s, s).$$

This property is not shared by specialized flow map formulations  $\Psi_{s \rightarrow 0}$ , such as those in the CM family. Thus, both CTM and MF can be regarded as flexible and general flow map formulations that generalize diffusion-based inference to direct time-to-time mappings.

**Conclusion.** This equivalence between CTM and MF is similar to the situation in diffusion models (Section 6.3), where different parameterizations ultimately describe the same underlying oracle target. In principle, these formulations are mathematically identical. In practice, however, their behavior can differ because of factors such as loss weighting, network design, or optimization dynamics, which may cause one approach to perform better than another under specific conditions.

This perspective suggests that CTM and MF are not the only possibility: other parametrizations of the flow map may also enable efficient and stable training, opening the door to new standalone generative models. Exploring these alternatives could further enrich the landscape of diffusion models and their flow map extensions, ultimately pushing the boundaries of what few-step generation can achieve.

## 11.6 Closing Remarks

This final chapter has brought our exploration full circle, culminating in a new paradigm for generative modeling: learning fast, few-step generators from scratch. Moving beyond the approaches of improving numerical solvers or distilling pre-trained models, we have focused on designing standalone training principles that are both principled and highly efficient by design.

The core innovation presented here is the direct learning of the flow map ( $\Psi_{s \rightarrow t}$ ) of the underlying probability flow ODE. The key to making this tractable without a teacher was leveraging the fundamental semi-group property of the ODE flow. This property, which dictates that a long trajectory can be decomposed into shorter segments, provides a powerful self-supervisory signal for training.

We began with Consistency Models (CMs), which pioneered this approach by learning the special flow map that transports any noisy state back to its clean origin ( $\Psi_{s \rightarrow 0}$ ). We then saw this idea generalized by Consistency Trajectory Models (CTM) and Mean Flow (MF), which learn the complete, anytime-to-anytime flow map  $\Psi_{s \rightarrow t}$  for all  $s, t$  satisfying  $s \geq t$ . While appearing different in their parameterization, we showed that these methods are fundamentally equivalent ways of approximating the same path integral that defines the flow map.

These flow map models represent a powerful synthesis of the principles developed throughout this monograph. They inherit the continuous-time foundation of the Score SDE framework and the deterministic transport view of Flow Matching, but reformulate the training objective to be self-contained and efficient.

By learning the solution map directly, these standalone models successfully unite the high sample quality of iterative diffusion processes with the inference speed of one-step generators. They resolve the fundamental trade-off between fidelity and speed, marking a significant milestone in generative modeling. This achievement represents not an end, but the beginning of a new chapter in the design of powerful, efficient, and controllable generative AI.

---

*The important thing is not to stop questioning. Curiosity has its own reason for existence.*

---

Albert Einstein

## **Appendices**

# A

---

## Crash Course on Differential Equations

---

Differential equations (DEs) are fundamental tools for modeling dynamic systems and can be broadly categorized into *ordinary differential equations* (ODEs), *stochastic differential equations* (SDEs), and *partial differential equations* (PDEs).

ODEs describe how a system's state changes over time according to a precise rule, so that knowing the starting point determines the future path exactly. SDEs add randomness to this evolution, modeling how noise or uncertainty influences the system's behavior, making the outcome probabilistic rather than fixed. PDEs explain how functions depending on several variables, such as time and space, evolve together, capturing phenomena like heat spreading, waves moving, or *the time evolution of probability densities in stochastic systems* (Spoiler: Fokker-Planck equation). These types of differential equations form a fundamental language for understanding how systems evolve over time and space under both deterministic and random influences.

In this chapter, we provide essential prerequisites on differential equations.

## A.1 Foundation of Ordinary Differential Equations

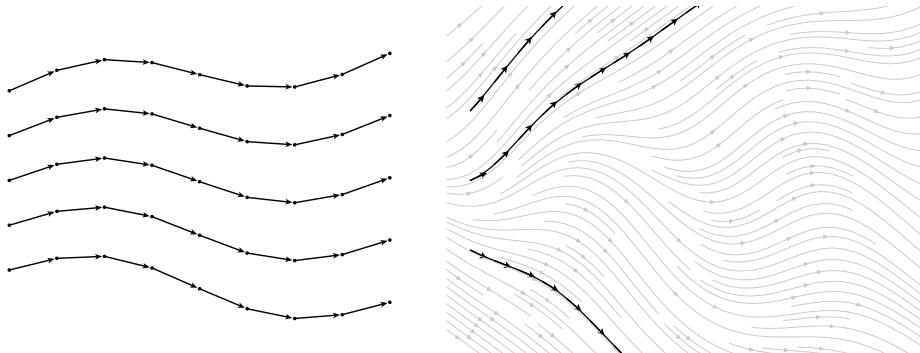
This section introduces the fundamental theory of ODEs, emphasizing the uniqueness of solutions given an initial condition. It also covers practical methods for solving ODEs using numerical solvers.

### A.1.1 Intuition of Ordinary Differential Equation

The deterministic process is called an *ordinary differential equation* (ODE). In the multivariate case, we consider systems of the form:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t), t), \quad (\text{A.1.1})$$

where  $\mathbf{x}(t) \in \mathbb{R}^D$  is a vector-valued function representing the state of the system at time  $t$ , and  $\mathbf{v} : \mathbb{R}^D \times \mathbb{R} \rightarrow \mathbb{R}^D$  is a vector field specifying the direction and magnitude of change at each point in space and time.



**Figure A.1: ODE illustration.** A velocity field  $\mathbf{v}(\mathbf{x}, t)$  assigns a drift vector at every point. A solution trajectory  $\mathbf{x}(t)$  is a path whose tangent always matches the local drift. The left panel shows step-by-step solver updates (dots and arrows) approximating the path, while the right panel shows the exact trajectories (black) flowing consistently with the velocity field. Without specifying an initial state  $\mathbf{x}(0)$ , there are infinitely many trajectories whose instantaneous changes match the same velocity field. Once  $\mathbf{x}(0)$  is fixed, however, the ODE determines a unique path  $\mathbf{x}(t)$  that flows according to the drift.

**High-Level Intuition for Solving ODEs.** To build intuition, imagine the vector field  $\mathbf{v}(\mathbf{x}, t)$  as a dynamic landscape of arrows that tells you how a point  $\mathbf{x}$  should move at any given time  $t$ . Solving the differential equation means tracing out a curve  $\mathbf{x}(t)$  through this field such that the tangent (i.e., the instantaneous velocity) of the curve at any point aligns with the vector given by  $\mathbf{v}(\mathbf{x}(t), t)$ .

- **Vector Field Perspective:** The function  $\mathbf{v}(\mathbf{x}, t)$  defines how things should move: it gives the local “instructions” for motion or change.
- **Trajectory Perspective:** The solution  $\mathbf{x}(t)$  is a path that a particle would follow if it obeys the rule set by the vector field  $\mathbf{v}$  at every instant.

Thus, solving an ODE is like placing a particle in a flow field and observing where it goes over time.

### A.1.2 Existence and Uniqueness of Ordinary Differential Equations

So far, we have seen that solving an ODE means finding a path that follows the directions given by the vector field at every point. Intuitively, this is like tracing the trajectory of a particle as it moves along the flow defined by the velocities.

But this picture leads to an important question:

#### Question A.1.1

If we pick a starting point, can we be sure there really is a path that follows these directions? And if there is, is that path unique, or could the particle suddenly jump onto a different trajectory?

Answering these questions is essential because it tells us whether the system’s behavior can be reliably predicted from its starting position. The *Existence and Uniqueness Theorem* provides conditions on the vector field that guarantee exactly one path starting from any given initial point. This ensures the solution behaves consistently and forms a cornerstone of the theory of ODEs.

**Local (in Time) Existence and Uniqueness Theorem.** Below, we state a *local* version of the theorem, which asserts existence and uniqueness of a solution in a neighborhood of the initial time for a given initial condition.

**Theorem A.1.1: Local Existence and Uniqueness**

Let  $\mathbf{v}(\mathbf{x}, t)$  be a continuous function with respect to  $\mathbf{x}$  and  $t$  in a domain  $D \subseteq \mathbb{R}^D \times \mathbb{R}$ . If  $\mathbf{v}$  satisfies the Lipschitz condition with respect to  $\mathbf{x}$ :

$$\|\mathbf{v}(\mathbf{x}_1, t) - \mathbf{v}(\mathbf{x}_2, t)\| \leq L \|\mathbf{x}_1 - \mathbf{x}_2\| \quad \forall (\mathbf{x}_1, t), (\mathbf{x}_2, t) \in D,$$

where  $L > 0$  is a constant, then for every initial condition  $\mathbf{x}(t_0) = \mathbf{x}_0$ , there exists a unique solution  $\mathbf{x}(t)$  to Equation (A.1.1) defined on some interval  $[t_0 - \delta, t_0 + \delta]$ .

**Proof for Theorem.**

(Proof Outline) The Existence and Uniqueness Theorem can be demonstrated constructively using the Picard-Lindelöf iteration method. The method generates a sequence of functions  $\{\mathbf{x}_n(t)\}$  that converges to the solution  $\mathbf{x}(t)$ . The iteration is defined as:

$$\mathbf{x}_{n+1}(t) = \mathbf{x}_0 + \int_{t_0}^t \mathbf{v}(\mathbf{x}_n(s), s) ds.$$

- Start with an initial guess  $\mathbf{x}_0(t) = \mathbf{x}_0$ .
- Iteratively refine  $\mathbf{x}_n(t)$  using the integral form.
- Convergence is guaranteed under the Lipschitz condition by applying Contraction Mapping Theorem.

The essence of the proof is rooted in the Picard–Lindelöf iteration method, whose core idea is also leveraged in Section 9.8 to accelerate the sampling process of diffusion models.

**Global (in Time) Existence and Uniqueness Theorem.** While the Local Existence and Uniqueness Theorem guarantees the existence of solutions on a small time interval, the “global (in time) existence and uniqueness theorem” extends this result to the entire interval  $[t_0, T]$  under additional regularity conditions. A well-known result in this category is the *Carathéodory theorem*, which ensures the global existence and uniqueness of solutions to ODEs under two key assumptions: local Lipschitz continuity in the state variable and a linear growth bound.

- (i) **Local Lipschitz condition in  $\mathbf{x}$ :** There exists a function  $\text{Lip}(t)$ , integrable on  $[0, T]$ , such that for all  $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^D$ ,

$$\|\mathbf{v}(\mathbf{x}_1, t) - \mathbf{v}(\mathbf{x}_2, t)\| \leq \text{Lip}(t) \|\mathbf{x}_1 - \mathbf{x}_2\|.$$

- (ii) **Linear growth condition:** There exists a function  $M(t)$ , integrable on  $[0, T]$ , such that for all  $\mathbf{x} \in \mathbb{R}^D$ ,

$$\|\mathbf{v}(\mathbf{x}, t)\| \leq M(t)(1 + \|\mathbf{x}\|).$$

We refer the reader to (Reid, 1971) for a comprehensive discussion of the assumptions, formal statement, and detailed proof of the theorem.

### Remark.

To apply these theorems to the probability flow ODE in diffusion models (see Equation (4.1.7)), it may be necessary to impose additional assumptions, such as conditions (i) and (ii), on the score function  $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ . These assumptions can be reasonably accepted without further justification by readers not focused on technical details.

In summary, when an initial condition is given to an ODE defined by a time-dependent velocity field, the trajectory of the particle flow is uniquely determined.

**Uniqueness Implies Non-Intersection of Solutions** The uniqueness of solutions in ODEs, as guaranteed by the Local Existence and Uniqueness Theorem, implies a fundamental property: two different solution trajectories, starting from different initial conditions, cannot cross each other. This reflects the deterministic nature of ODEs, ensuring that each state evolves along a unique path. The following corollary formalizes this result.

#### Corollary A.1.1: Non-Intersection of Solutions

Consider two solutions  $\mathbf{x}_1(t)$  and  $\mathbf{x}_2(t)$  to the ODE

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t), t), \quad t \in [0, T].$$

Suppose they have distinct initial values  $\mathbf{x}_1(0) \neq \mathbf{x}_2(0)$ . Then, these solutions do not intersect on  $[0, T]$ , i.e.,

$$\mathbf{x}_1(t) \neq \mathbf{x}_2(t) \quad \text{for all } t \in [0, T].$$

### Proof for Corollary.

Assume, for the sake of contradiction, that there exists some  $t^* \in (0, T]$  such that

$$\mathbf{x}_1(t^*) = \mathbf{x}_2(t^*).$$

Define the first time at which the two solutions meet as

$$t_0 := \inf\{t \in [0, T] | \mathbf{x}_1(t) = \mathbf{x}_2(t)\}.$$

Since  $\mathbf{x}_1(0) \neq \mathbf{x}_2(0)$  and  $t^*$  is contained in this set, it follows that  $t_0 > 0$ . By continuity of  $\mathbf{x}_1$  and  $\mathbf{x}_2$ , we have

$$\mathbf{x}_1(t_0) = \mathbf{x}_2(t_0).$$

Consider the initial value problem

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_1(t_0).$$

By the uniqueness theorem for ODEs, both  $\mathbf{x}_1$  and  $\mathbf{x}_2$  must coincide on the interval  $[t_0, T]$ . Applying uniqueness backward in time similarly implies that the two solutions coincide on  $[0, t_0]$ . Therefore, the solutions satisfy

$$\mathbf{x}_1(t) = \mathbf{x}_2(t) \quad \text{for all } t \in [0, T],$$

which contradicts the assumption that  $\mathbf{x}_1(0) \neq \mathbf{x}_2(0)$ . Hence, we conclude that

$$\mathbf{x}_1(t) \neq \mathbf{x}_2(t) \quad \text{for all } t \in [0, T].$$

By guaranteeing non-intersecting solution paths, this theorem offers hidden yet crucial support for the flow map model (see Chapters 10 and 11).

### A.1.3 Exponential Integration Factor

Even ODE determined by a general time-varying velocity  $\mathbf{v}$  does not admit closed-form solution, in some special case, we can solve them analytically or reducing its formulation to a better structural one.

**An Illustrative Example.** Consider the following linear scalar ODE:

$$\frac{d\mathbf{x}(t)}{dt} = L(t)\mathbf{x}(t),$$

where  $L(t) \in \mathbb{R}$  is a continuous function. This equation is solvable in closed form, and its solution is well known (for any  $s$  and  $t$ ):

$$\mathbf{x}(t) = \mathbf{x}(s) \cdot \exp\left(\int_s^t L(\tau) d\tau\right).$$

This formula demonstrates how the solution evolves according to an exponential factor that accumulates the effect of the time-dependent coefficient  $L(t)$ . This motivates the use of *exponential integration factors*:

$$\mathcal{E}(s \rightarrow t) := \exp\left(\int_s^t L(\tau) d\tau\right), \quad (\text{A.1.2})$$

especially in more general settings where the dynamics include both linear and nonlinear components.

**Semilinear ODEs and Exponential Integration Factors.** We now consider a broader class of ODEs known as *semilinear* ODEs. These equations separate the dynamics into a linear part (in the state variable) and a nonlinear remainder:

$$\frac{d\mathbf{x}(t)}{dt} = L(t)\mathbf{x}(t) + \mathbf{N}(\mathbf{x}(t), t), \quad (\text{A.1.3})$$

where  $\mathbf{x}(t) \in \mathbb{R}^D$  is the state vector,  $L(t)$  is a scalar-valued continuous function, and  $\mathbf{N} : \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}^D$  is a nonlinear vector field. This semilinear structure arises naturally in many physical and engineering systems. In particular, it also appears in the probability flow ODE formulation of diffusion models (see Equation (4.1.7)). Recognizing this structure enables the use of exponential integration factors, which not only simplify analysis but also improve numerical stability. Specifically, this technique plays a central role in the design of fast diffusion ODE solvers (see Chapter 9).

**Step 1: Isolate the Non-Linear Term via an Integration Factor.** Observing that we can isolate the nonlinear part by subtracting the linear drift from the semilinear ODE in Equation (A.1.3):

$$\frac{d\mathbf{x}(t)}{dt} - L(t)\mathbf{x}(t) = \mathbf{N}(\mathbf{x}(t), t).$$

To absorb the linear term, we multiply both sides by the inverse integration factor:

$$\mathcal{E}^{-1}(s \rightarrow t) = \exp\left(-\int_s^t L(\tau) d\tau\right).$$

Now apply the product rule to the left-hand side:

$$\mathcal{E}^{-1}(s \rightarrow t) \left( \frac{d\mathbf{x}(t)}{dt} - L(t)\mathbf{x}(t) \right) = \frac{d}{dt} [\mathcal{E}^{-1}(s \rightarrow t)\mathbf{x}(t)].$$

Hence, the equation becomes:

$$\frac{d}{dt} [\mathcal{E}^{-1}(s \rightarrow t)\mathbf{x}(t)] = \mathcal{E}^{-1}(s \rightarrow t)\mathbf{N}(\mathbf{x}(t), t).$$

This transformation simplifies the original equation by isolating the nonlinear component, allowing us to focus entirely on the nonlinear dynamics in a transformed coordinate system.

**Step 2: Integrate Over Time.** We now integrate both sides from  $s$  to  $t$ :

$$\int_s^t \frac{d}{d\tau} [\mathcal{E}^{-1}(s \rightarrow \tau)\mathbf{x}(\tau)] d\tau = \int_s^t \mathcal{E}^{-1}(s \rightarrow \tau)\mathbf{N}(\mathbf{x}(\tau), \tau) d\tau.$$

The left-hand side is simply the difference of the transformed variable evaluated at  $t$  and  $s$ :

$$\mathcal{E}^{-1}(s \rightarrow t)\mathbf{x}(t) - \mathbf{x}(s).$$

Hence, we obtain:

$$\mathcal{E}^{-1}(s \rightarrow t)\mathbf{x}(t) = \mathbf{x}(s) + \int_s^t \mathcal{E}^{-1}(s \rightarrow \tau)\mathbf{N}(\mathbf{x}(\tau), \tau) d\tau.$$

**Step 3: Solve for  $\mathbf{x}(t)$ .** Multiplying both sides by the exponential flow  $\mathcal{E}(s \rightarrow t)$  gives the solution:

$$\mathbf{x}(t) = \underbrace{\mathcal{E}(s \rightarrow t)\mathbf{x}(s)}_{\text{linear part}} + \underbrace{\int_s^t \mathcal{E}(\tau \rightarrow t)\mathbf{N}(\mathbf{x}(\tau), \tau) d\tau}_{\text{nonlinear part}}. \quad (\text{A.1.4})$$

The solution naturally separates into a linear and a nonlinear component. Exponential integrators exploit this structure by solving the linear part in exactly closed form and discretizing only the nonlinear residual. This ensures that the step size is dictated by the nonlinear dynamics rather than by the potentially large linear coefficient, yielding updates that are both stable and accurate even with fewer steps (see the comparison between the exponential Euler update Equation (9.1.7) and the vanilla Euler update Equation (9.1.8)).

### A.1.4 Numerical Solvers of Ordinary Differential Equations

We consider the ODE in Equation (A.1.1) with an initial condition  $\mathbf{x}(0)$ . Solving this ODE involves finding a continuous trajectory  $\mathbf{x}(t)$  that satisfies the equation for all  $t \in [0, T]$ . Ideally, a closed-form solution is desirable, though it is rarely attainable in practice.

A useful perspective is to rewrite the ODE in its integral form:

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^t \mathbf{v}(\mathbf{x}(\tau), \tau) d\tau, \quad (\text{A.1.5})$$

which expresses the solution as the initial state plus the accumulated effect of the velocity over time. However, the integral is often intractable due to the nonlinear and time-dependent nature of  $\mathbf{v}$ , making closed-form solutions unavailable.

In such cases, we turn to *numerical methods*, which discretize time and iteratively approximate  $\mathbf{x}(t)$ . Common approaches include Euler's method, Runge–Kutta methods, and specialized integrators for stiff systems. These methods simulate the system step by step, providing practical approximations of the true trajectory.

#### Remark.

When  $\mathbf{v}$  takes the semilinear form in Equation (A.1.3), the solution admits an integral representation involving an exponential integration factor (Equation (A.1.4)), which separates the linear and nonlinear components. This structure enables efficient numerical solvers that focus solely on approximating the nonlinear term, reducing computational complexity and motivating tailored algorithms (see Chapter 9).

**Key Concepts.** Numerical solvers approximate the continuous dynamics of ODEs by discretizing time and estimating the state using the slope  $\mathbf{v}$  of the ODE. This involves:

- **Discretization:** Partition the time domain into discrete steps  $t_0, t_1, \dots, t_n$ .
- **Step Size:** The interval  $\Delta t_i = t_{i+1} - t_i$  is called the step size.
- **Approximation:** The solution at each step is estimated numerically; the accuracy depends on the step size and the method used.
- **Error Control:** Errors from discretization and approximation are monitored and controlled.

**High-Level Categorization of Numerical Solvers.** ODE solvers can be broadly categorized as:

- **Time-Stepping Methods:** These methods advance the solution step by step, e.g., explicit/implicit Euler, Runge-Kutta.
- **Time-Parallel Methods:** These methods leverage parallelism to compute solutions over different time intervals simultaneously, useful for large-scale problems.

**Common Numerical Solvers.** Among these, Euler, Heun, and Runge–Kutta are *single-step methods*, since each update uses only the current state  $(t_n, \mathbf{x}_n)$ . In contrast, *multi-step methods* (such as Adams–Bashforth or Adams–Moulton) compute  $\mathbf{x}_{n+1}$  using not only the current state  $\mathbf{x}_n$  but also several previous values  $\mathbf{x}_{n-1}, \mathbf{x}_{n-2}, \dots$ . They save work by reusing past information (history anchors) instead of re-evaluating everything within the current step. Such methods are not covered here, though related schemes (e.g., Adams–Bashforth, discussed in Sections 9.3 and 9.5) also exploit multiple past states.

Picard iteration, on the other hand, is of a different nature: it serves as a theoretical fixed-point construction, whose idea will be revisited in Section 9.8.

**Euler's Method.** Euler's method is the simplest time-stepping scheme:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}(\mathbf{x}_n, t_n),$$

where  $h$  is the step size. It has first-order accuracy: local error  $\mathcal{O}(h^2)$ , global error  $\mathcal{O}(h)$ . While easy to implement, it requires small  $h$  for stability and accuracy.

**Heun's Method (Improved Euler).** Heun's method is a second-order predictor-corrector scheme:

$$\begin{aligned} \text{Predict: } \mathbf{x}_{\text{pred}} &= \mathbf{x}_n + h\mathbf{v}(\mathbf{x}_n, t_n), \\ \text{Correct: } \mathbf{x}_{n+1} &= \mathbf{x}_n + \frac{h}{2} (\mathbf{v}(\mathbf{x}_n, t_n) + \mathbf{v}(\mathbf{x}_{\text{pred}}, t_n + h)). \end{aligned}$$

It achieves local error  $\mathcal{O}(h^3)$  and global error  $\mathcal{O}(h^2)$ . Karras *et al.* (2022) advocate Heun's method for solving ODEs in diffusion models, though higher-order methods such as DPM-Solvers (see Sections 9.4 and 9.5) typically yield better performance.

**Runge-Kutta Methods.** Runge-Kutta (RK) methods generalize Euler by using weighted averages of intermediate slopes. The fourth-order method (RK4) is a standard choice:

$$\begin{aligned}\mathbf{k}_1 &= \mathbf{v}(\mathbf{x}_n, t_n), \\ \mathbf{k}_2 &= \mathbf{v}(\mathbf{x}_n + \frac{h}{2}\mathbf{k}_1, t_n + \frac{h}{2}), \\ \mathbf{k}_3 &= \mathbf{v}(\mathbf{x}_n + \frac{h}{2}\mathbf{k}_2, t_n + \frac{h}{2}), \\ \mathbf{k}_4 &= \mathbf{v}(\mathbf{x}_n + h\mathbf{k}_3, t_n + h), \\ \mathbf{x}_{n+1} &= \mathbf{x}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4).\end{aligned}$$

RK4 balances accuracy and cost, making it widely used. DPM-Solver builds on similar ideas to achieve higher-order accurate integration tailored to diffusion models, leveraging their semilinear structure (see (Lu *et al.*, 2022b)'s Appendix B.6 for a comparison).

**Picard Iteration.** Picard iteration refines successive approximations to the solution via:

$$\mathbf{x}^{(k+1)}(t) = \mathbf{x}(0) + \int_0^t \mathbf{v}(\mathbf{x}^{(k)}(s), s) \, ds,$$

starting from an initial guess function  $\mathbf{x}^{(0)}(t)$  with  $\mathbf{x}^{(0)}(0) = \mathbf{x}(0)$ . While theoretically foundational, Picard iteration often converges slowly due to its strong dependence on the initial guess. Moreover, each iteration involves computing an integral over time, which can be computationally expensive.

**Solving ODEs in Forward and Reverse Time.** So far, we have considered solving the ODE in Equation (A.1.1) *forward in time*, evolving the solution from an initial condition  $\mathbf{x}(0)$  to later times  $t > 0$ .

In contrast, *reverse-time* integration computes the solution by stepping backward from a terminal condition  $\mathbf{x}(T)$  toward earlier times  $t < T$ . Reparameterizing time as  $T - t$  transforms the ODE into:

$$\frac{d\mathbf{x}(t)}{dt} = -\mathbf{v}(\mathbf{x}(t), T - t), \quad \mathbf{x}(0) = \mathbf{x}(T).$$

Reverse-time integration applies the same methods as forward-time integration, but on a decreasing time grid. With Euler and step size  $h > 0$ , starting from  $t_0 = T$  with  $\mathbf{x}_0 = \mathbf{x}(T)$ , the updates are

$$t_{n+1} = t_n - h, \quad \mathbf{x}_{n+1} = \mathbf{x}_n - h \mathbf{v}(\mathbf{x}_n, t_n).$$

Care must be taken to ensure numerical stability, especially for stiff problems (i.e., when some components of the state vector evolve much faster than others, requiring very small time steps for stable integration), as commonly encountered in PF-ODE sampling for diffusion models.

While time reversal for ODEs is theoretically straightforward, as it only requires a reparameterization of time due to the bijective mapping between  $\mathbf{x}(0)$  and  $\mathbf{x}(T)$ , this does not hold for SDEs. Their intrinsic randomness precludes direct time reversal, a point we elaborate on in the next section.

## A.2 Foundation of Stochastic Differential Equations

Stochastic Differential Equations (SDEs) are an extension of ordinary differential equations (ODEs) that incorporate randomness, providing a mathematical framework for modeling systems affected by uncertainty. This chapter introduces SDEs, beginning with the discretization of ODEs, extending to the discretization of SDEs, and culminating in a discussion of general SDEs, including Ito's calculus and Ito's formula.

### A.2.1 From ODEs to SDEs: An Intuitive Introduction

Let us begin with a ODE describing the deterministic evolution of a state variable  $\mathbf{x}(t) \in \mathbb{R}^D$ :

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), t), \quad \mathbf{x}(0) = \mathbf{x}_0. \quad (\text{A.2.1})$$

Here,  $\mathbf{f} : \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}^D$  is a time-dependent velocity field that governs the dynamics of  $\mathbf{x}(t)$ . The solution to this ODE is a smooth trajectory  $t \mapsto \mathbf{x}(t)$ , fully determined by the initial condition  $\mathbf{x}_0$ .

**Discretization Perspective.** To build intuition, consider an Euler discretization of Equation Equation (A.2.1) over small time steps  $\Delta t$ :

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\Delta t.$$

This approximation becomes more accurate as  $\Delta t \rightarrow 0$ , converging (under standard regularity conditions on  $\mathbf{f}$ ) to the exact solution of the ODE.

**Introducing Randomness: From ODE to SDE.** In many real-world systems, perfect knowledge of the dynamics is unrealistic. Noise, uncertainty, or unmodeled interactions may affect the evolution. To incorporate such randomness, we augment the ODE with a stochastic term:

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\Delta t + g(t)\sqrt{\Delta t} \cdot \boldsymbol{\epsilon}_t, \quad (\text{A.2.2})$$

where

- $g : [0, T] \rightarrow \mathbb{R}$  is a diffusion coefficient (possibly dependent on both state and time, though here assumed time-dependent only),
- $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_D)$  are i.i.d. standard Gaussian vectors.

This modified update rule reflects not just deterministic drift, but also random perturbations scaled by  $\sqrt{\Delta t}$ . The scaling ensures that the stochastic perturbation remains finite in the limit  $\Delta t \rightarrow 0$ . Importantly, this formulation gives rise to a *continuous-time stochastic process* as  $\Delta t \rightarrow 0$ , which leads us to the framework of SDE.

**Stochastic Differential Equations.** Formally, the limit of the discrete update Equation (A.2.2) as  $\Delta t \rightarrow 0$  defines the SDE:

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t) dt + g(t) d\mathbf{w}(t). \quad (\text{A.2.3})$$

Here,  $\mathbf{w}(t) \in \mathbb{R}^D$  is a *Wiener process* (standard Brownian motion), a continuous-time stochastic process characterized by:

- **Initial State:**  $\mathbf{w}(0) = 0$  almost surely;
- **Independent Increments:** for  $0 \leq s < t$ , the increment  $\mathbf{w}(t) - \mathbf{w}(s)$  is independent of the past;
- **Gaussian Increments:**

$$\mathbf{w}(t) - \mathbf{w}(s) \sim \mathcal{N}(\mathbf{0}, (t-s)\mathbf{I}_D) \quad (\text{A.2.4})$$

- **Continuity:** Sample paths  $t \mapsto \mathbf{w}(t)$  are almost surely continuous but nowhere differentiable.

In addition, the notation

$$d\mathbf{w}(t) := \mathbf{w}(t + dt) - \mathbf{w}(t)$$

is often used to denote the infinitesimal increment of the Wiener process.

While suggestive, this notation is heuristic and should not be interpreted as a classical differential (e.g., in the Riemann or Lebesgue sense), since Brownian paths are almost surely nowhere differentiable. Instead, it serves as a formal shorthand to express the Gaussian increments property:

$$d\mathbf{w}(t) \sim \mathcal{N}(0, dt \mathbf{I}_D),$$

meaning that over an infinitesimal time interval of length  $dt$ , the increment of the Wiener process behaves like a Gaussian random variable with zero mean and covariance  $dt \mathbf{I}_D$ .

### A.2.2 Further Explanation of Equation (A.2.3)

The SDE in Equation (A.2.3) should be understood in its *integral form*:

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^t \mathbf{f}(\mathbf{x}(s), s) ds + \int_0^t g(s) d\mathbf{w}(s), \quad (\text{A.2.5})$$

interpreted in the *Itô sense*. Here, the first term is a classical (Riemann or Lebesgue) integral representing the accumulated deterministic drift, while the second term is an *Itô stochastic integral*, which integrates with respect to the Wiener process  $\mathbf{w}(t)$ . We do not provide a full rigorous construction of the Itô integral, but offer the following intuition.

**Intuition for Itô Integration.** The Itô integral can be viewed as the limit (in probability) of discrete sums:

$$\sum_i g(t_i)(\mathbf{w}(t_{i+1}) - \mathbf{w}(t_i)),$$

where the integrand  $g(t)$  is evaluated at the *left endpoint*  $t_i$  of each subinterval. This left-point evaluation is crucial and distinguishes Itô integration from classical integrals, which often use midpoints or other evaluation rules.

Because Brownian paths are continuous yet almost surely nowhere differentiable, classical integration fails to apply. The Itô integral handles this irregularity, capturing the cumulative effect of stochastic fluctuations over time.

**Use of Differential Notation.** Expressions such as  $d\mathbf{x}(t)$ ,  $dt$ , and  $d\mathbf{w}(t)$  are not classical differentials. Instead, they are formal notations representing infinitesimal increments of the respective processes. While heuristic, they are widely used for their convenience in expressing SDEs analogously to ODEs and facilitate formal manipulations within Itô calculus.

How Itô calculus is applied in diffusion models will be explained in Chapter C.

**Comparison with ODEs.** In ODEs, e.g.,

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), t),$$

the integral form

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^t \mathbf{f}(\mathbf{x}(\tau), \tau) d\tau$$

is justified by the *Fundamental Theorem of Calculus*, which ensures that differentiable functions can be recovered from their derivatives.

By contrast, in SDEs such as Equation (A.2.3), there is no direct analog of this theorem because Brownian motion lacks differentiability, and stochastic integrals do not follow the classical chain rule. Instead, Itô calculus introduces alternative tools (e.g., Itô's lemma) to analyze and manipulate stochastic dynamics.

Thus, while the differential notation for SDEs is compact and intuitive, a rigorous understanding depends on interpreting them via their integral formulation using Itô integrals.

### A.2.3 A Numerical Solver for SDE.

Like ODEs, the SDE in Equation (A.2.3) admits a unique solution<sup>1</sup> if  $\mathbf{f}(\cdot, t)$  and  $g(\cdot)$  satisfy some smoothness conditions:  $\mathbf{f}(\cdot, t)$  is Lipschitz and of linear growth in  $\mathbf{x}$ , and  $g(\cdot)$  is square integrable.

For general SDEs as in Equation (A.2.3), closed-form solutions are generally unavailable, so numerical methods are necessary. A common approach is the *Euler–Maruyama method*, which generalizes Euler's method for ODEs and, indeed, we have already seen it in Equation (A.2.2). It approximates the drift term  $\mathbf{f}(\mathbf{x}(t), t)$  over a time step  $\Delta t$  and simulates the stochastic noise  $g(t) \, d\mathbf{w}(t)$  using Gaussian increments  $\sqrt{\Delta t} \boldsymbol{\epsilon}_t$  with  $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ .

Later, in Section C.1.5, we will see that a linear SDE admits a closed-form solution.

---

<sup>1</sup>The solution is in the *strong* sense, meaning that  $\mathbf{x}(t)$  satisfies the SDE in its integral form (see Equation (A.2.5)) with respect to the given Brownian motion  $\mathbf{w}(t)$  on a fixed probability space. We omit the detailed technical definitions here.

# B

---

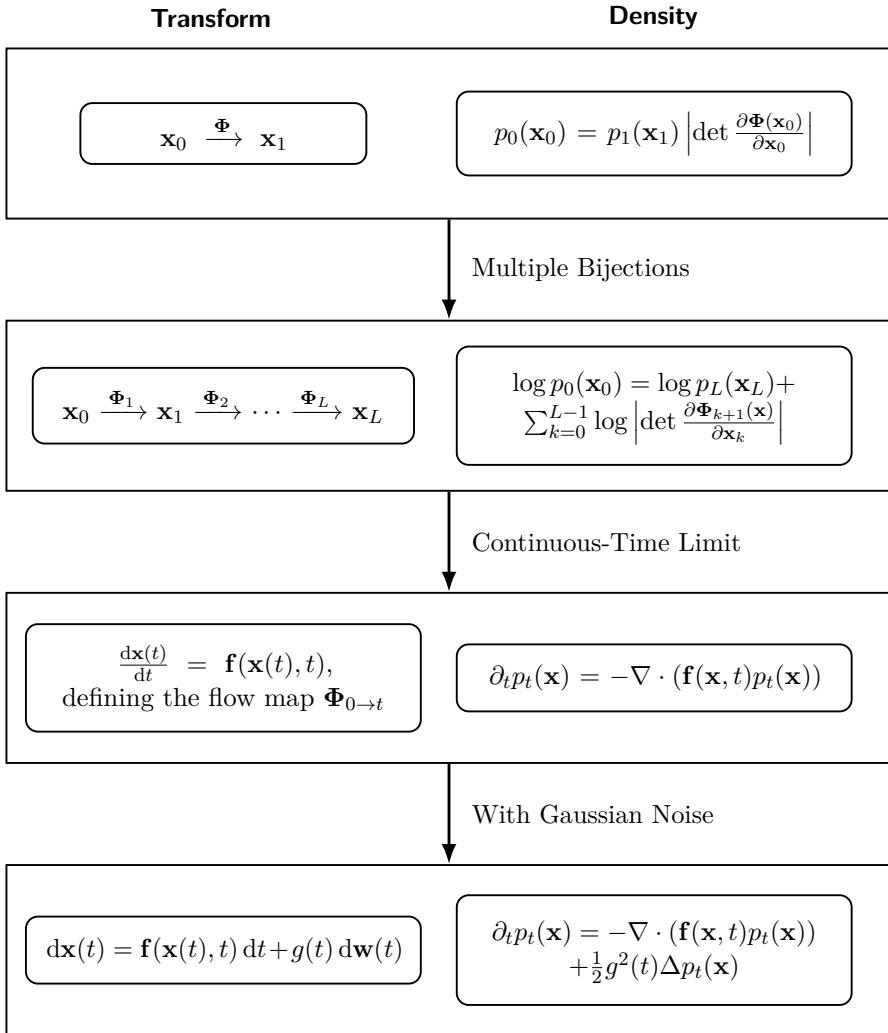
## Density Evolution: From Change of Variable to Fokker–Planck

---

Understanding how probability densities evolve under transformations is fundamental in both probability theory and generative modeling. In particular, diffusion models aim to construct generative processes whose induced density paths reverse a pre-defined forward process. This evolution is governed by the continuity equation or, in the stochastic case, the Fokker–Planck equation.

Although these names may sound unfamiliar or intimidating, they are in fact continuous-time analogues of the change-of-variable formula from basic calculus. In Section B.1, it builds up to them by presenting a progression of change-of-variable formulas, starting from deterministic bijections, and culminating in stochastic differential equations. This progression naturally bridges discrete mappings and continuous-time flow dynamics. See Figure B.1 for an overview of this unified framework.

In Section B.2, we provide a physical and intuitive interpretation of the continuity equation, emphasizing its connection to the conservation of density in dynamical systems.



**Figure B.1:** A unified *change-of-variables formula*. From top to bottom: (1) a single bijection and the ; (2) composition of multiple bijections; (3) continuous-time deterministic flow governed by an ODE and the associated continuity equation; (4) stochastic flow modeled by an SDE and the corresponding Fokker–Planck equation.

## B.1 Change-of-Variable Formula: From Deterministic Maps to Stochastic Flows

In this section, we aim to demystify the continuity equation and the Fokker–Planck equation by drawing analogies to the classic change-of-variable formula from calculus. We begin with the familiar single-variable case, extend it to the multivariate setting and to probability densities (Section B.1.1), then

generalize to compositions of bijective maps whose continuous-time limit leads to the continuity equation (Section B.1.2). Finally, we incorporate stochasticity by introducing random noise, which naturally extends the continuity equation to the Fokker–Planck equation (Section B.1.3).

### B.1.1 Change-of-Variable Formula for Deterministic Maps

We move particles according to a deterministic map and study how their *law* (density) evolves. The key principle is conservation of probability mass, grounded in a fundamental result from calculus and probability: the change-of-variable formula. This formula describes how integrals, and therefore probability densities, transform under smooth bijective mappings. To build intuition, we first consider a single update step, and then extend the discussion to sequential transformations.

**Single Update.** Think of a single update rule induced by applying a vector field (analogous to a force)  $\Psi : \mathbb{R}^D \rightarrow \mathbb{R}^D$  for one unit of time. Starting from an initial particle state  $\mathbf{x}_0$ , its next state is given by

$$\mathbf{x}_1 = \Psi(\mathbf{x}_0).$$

**Underlying Pattern (a Density) and How it Moves.** If the initial states follow an underlying “law/pattern” described by a density  $p_0$  (i.e.,  $\mathbf{x}_0 \sim p_0$ ), then applying  $\Psi$  produces a new density  $p_1$  for  $\mathbf{x}_1$  (i.e.,  $\mathbf{x}_1 \sim p_1$ ). Assuming  $\Psi$  is a smooth bijection,  $p_1$  is obtained from  $p_0$  via the standard *change-of-variables formula*:

$$p_1(\mathbf{x}_1) = p_0(\Psi^{-1}(\mathbf{x}_1)) \cdot \left| \det \left( \frac{\partial \Psi^{-1}}{\partial \mathbf{x}_1} \right) \right|. \quad (\text{B.1.1})$$

Here  $\frac{\partial \Psi}{\partial \mathbf{x}}$  is the Jacobian matrix of  $\Psi$ , denoted  $\partial_{\mathbf{x}} \Psi$ . Equivalently, in the original coordinates,

$$p_0(\mathbf{x}_0) = p_1(\Psi(\mathbf{x}_0)) |\det \partial_{\mathbf{x}} \Psi(\mathbf{x}_0)|.$$

In words,  $\Psi$  reshapes the density  $p_0$  into  $p_1$ . The factor  $|\det \partial_{\mathbf{x}} \Psi|$  represents the local change in volume; since probability mass is conserved, the density compensates by its inverse.

As a simple case, if  $\Psi$  is linear with an invertible matrix  $\mathbf{A}$  (i.e.,  $\mathbf{x}_1 = \mathbf{A}\mathbf{x}_0$ ), then

$$p_1(\mathbf{x}_1) = p_0(\mathbf{A}^{-1}\mathbf{x}_1) |\det \mathbf{A}^{-1}|.$$

Schematically, we can read it as:

$$\begin{array}{lll} \textbf{Sample:} & \mathbf{x}_0 & \xrightarrow{\Psi} \mathbf{x}_1 \\ \textbf{Density:} & p_{\mathbf{x}_0}(\mathbf{x}_0) & \xrightarrow{\Psi} p_{\mathbf{x}_1}(\mathbf{x}_1) \end{array}$$

**Why is Equation (B.1.1) the Change-of-Variables Formula?** This comes directly from the familiar rule in calculus.

**Single-Variable Case.** Let  $y = \Psi(x)$  be smooth and invertible. Rewriting an integral over  $y$  in terms of  $x$  gives

$$\int g(y) dy = \int g(\Psi(x)) \cdot |\Psi'(x)| dx,$$

where  $|\Psi'(x)|$  compensates for interval stretching or compression, ensuring area preservation.

**Multivariate Case.** For  $\Psi : \mathbb{R}^D \rightarrow \mathbb{R}^D$  with  $\mathbf{y} = \Psi(\mathbf{x})$ ,

$$\int g(\mathbf{y}) d\mathbf{y} = \int g(\Psi(\mathbf{x})) |\det(\partial_{\mathbf{x}} \Psi)| d\mathbf{x},$$

so infinitesimal volumes transform as

$$d\mathbf{y} = |\det(\partial_{\mathbf{x}} \Psi)| d\mathbf{x}.$$

From this, the density formula in Equation (B.1.1) follows:

$$\begin{aligned} p_{\mathbf{y}}(\mathbf{y}) &= \int_{\mathbb{R}^D} \delta(\mathbf{y} - \Psi(\mathbf{x})) p_{\mathbf{x}}(\mathbf{x}) d\mathbf{x} \\ &= p_{\mathbf{x}}(\Psi^{-1}(\mathbf{y})) \left| \det \left( \frac{\partial \Psi^{-1}}{\partial \mathbf{y}} \right) \right|. \end{aligned}$$

**Composing Multiple bijections.** We now apply several updates in sequence. Let  $\mathbf{x}_k = \Psi_k(\mathbf{x}_{k-1})$  for  $k = 1, \dots, L$ ; that is,

$$\mathbf{x}_0 \xrightarrow{\Psi_1} \mathbf{x}_1 \xrightarrow{\Psi_2} \dots \xrightarrow{\Psi_L} \mathbf{x}_L,$$

where each  $\Psi_k : \mathbb{R}^D \rightarrow \mathbb{R}^D$  is a smooth bijection. If the initial state follows density  $p_0$  (i.e.,  $\mathbf{x}_0 \sim p_0$ ), then the sequence of updates induces densities  $p_1, \dots, p_L$  for  $\mathbf{x}_1, \dots, \mathbf{x}_L$ .

Because probability mass is conserved at each step, the densities evolve according to

$$p_k(\mathbf{x}_k) = p_{k-1}(\mathbf{x}_{k-1}) \left| \det \partial_{\mathbf{x}_{k-1}} \Psi_k(\mathbf{x}_{k-1}) \right|^{-1}, \quad k = 1, \dots, L.$$

By recursion, the final density at  $\mathbf{x}_L$  is

$$p_{\mathbf{x}_L}(\mathbf{x}_L) = p_{\mathbf{x}_0}(\mathbf{x}_0) \cdot \prod_{k=1}^L \left| \det \left( \frac{\partial \Psi_k}{\partial \mathbf{x}_{k-1}} \right) \right|^{-1}. \quad (\text{B.1.2})$$

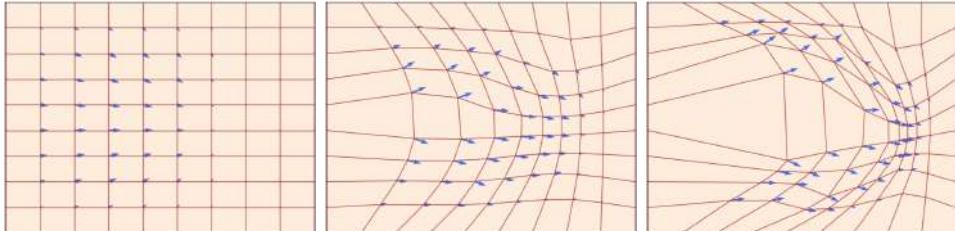
Equivalently, in log-density form:

$$\log p_{\mathbf{x}_L}(\mathbf{x}_L) = \log p_{\mathbf{x}_0}(\mathbf{x}_0) - \sum_{k=1}^L \log \left| \det \left( \frac{\partial \Psi_k}{\partial \mathbf{x}_{k-1}} \right) \right|.$$

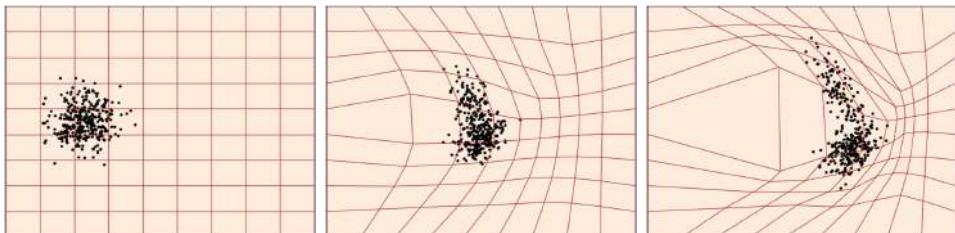
This expression reflects how each transformation  $\Psi_k$  stretches or contracts volume, as captured by the Jacobian determinant. The accumulation of these local volume changes along the transformation path determines the final probability density under the composed map.

Equation (B.1.2) serves as the core principle underlying Normalizing Flows (see Section 5.1.2).

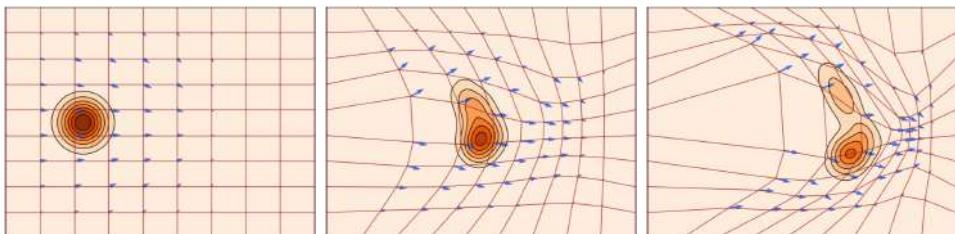
### B.1.2 Continuous-Time Limit: Continuity Equation



(a) Vector field illustrations. The arrows represent forces that would drag particles through space, deforming the underlying grid accordingly.



(b) Particle-cloud dynamics. A predefined vector field (interpreted as a force) generates a flow that transports particles from their initial state.



(c) Density evolution. As particles are advected by the vector field, the density contours deform accordingly, reflecting how the flow reshapes the underlying distribution.

**Figure B.2:** Illustrations of particle and density dynamics under a vector field. Each column shows successive time snapshots (left to right). These illustrations are adapted from Lipman *et al.* (2024) with the author's permission.

We now pass from discrete updates to a continuous description. Suppose the particle motion is driven by a time-varying velocity field  $\mathbf{f} : \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}^D$ . Imagine evolving a particle  $\mathbf{x}_0 \sim p_0$  through infinitely many small bijective updates. At each step  $t$  of length  $\Delta t > 0$ , the update is

$$\mathbf{x}_{t+\Delta t} = \Psi(\mathbf{x}_t) := \mathbf{x}_t + \Delta t \mathbf{f}(\mathbf{x}_t, t).$$

As  $\Delta t \rightarrow 0$ , the composition of these updates converges to a continuous flow governed by a velocity field  $\mathbf{f} : \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}^D$ :

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), t), \quad \mathbf{x}(0) = \mathbf{x}_0 \sim p_0. \quad (\text{B.1.3})$$

Under suitable smoothness assumptions (see Chapter A), this ODE admits a unique solution for each initial condition, which defines a deterministic flow map  $\Psi_{0 \rightarrow t} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ . In other words,  $\Psi_{0 \rightarrow t}$  brings the initial state  $\mathbf{x}_0$  to the solution of Equation (B.1.3) at time  $t$ :

$$\Psi_{0 \rightarrow t}(\mathbf{x}_0) = \mathbf{x}_0 + \int_0^t \mathbf{f}(\mathbf{x}(\tau), \tau) d\tau.$$

As a result, the whole distribution also moves: the initial density  $p_0$  is transported into the new density  $p_t$ , the law of  $\mathbf{x}(t)$ . Formally, this is written as a *pushforward*:

$$p_t = (\Psi_{0 \rightarrow t})_\# p_0.$$

When  $\Psi_{0 \rightarrow t}$  is smooth and invertible, this reduces to the familiar change-of-variables rule:

$$p_t(\mathbf{x}) = p_0(\Psi_{t \rightarrow 0}(\mathbf{x})) |\det \partial_{\mathbf{x}} \Psi_{t \rightarrow 0}(\mathbf{x})| = \int \delta(\mathbf{x} - \Psi_{t \rightarrow 0}(\mathbf{x}_0)) p_0(\mathbf{x}_0) d\mathbf{x}_0.$$

**Continuity Equation: How the Density Moves in Time.** Rather than writing a separate formula for the density at each time, we can describe how it moves continuously using a differential equation in space  $\mathbf{x}$  and time  $t$ . The idea is simple: probability mass is conserved, and the velocity field  $\mathbf{f}$  only redistributes it in space. This gives the *continuity equation*:

$$\frac{\partial}{\partial t} p_t(\mathbf{x}) + \nabla \cdot (p_t(\mathbf{x}) \mathbf{f}(\mathbf{x}, t)) = 0. \quad (\text{B.1.4})$$

Here the divergence term  $\nabla \cdot (p_t \mathbf{f})$  measures how the flow locally expands or compresses the density, ensuring total probability remains 1.

This partial differential equation (PDE) ensures that probability mass is conserved as the flow moves particles. In fact, it can be viewed as the continuous-time analogue of the change-of-variables formula.

**Derivation of Continuity Equation via Change-of-Variables Formula.** Conceptually, the continuity equation can also be obtained by taking the continuous-time limit of Equation (B.1.2). Here, however, we adopt a more direct derivation based on Equation (B.1.1).

**Discretization and Change-of-Variable Formula.** Consider

$$\mathbf{x}_{t+\Delta t} := \Psi(\mathbf{x}_t) = \mathbf{x}_t + \Delta t \mathbf{f}(\mathbf{x}_t, t),$$

which is actually the forward Euler discretization of the ODE in Equation (B.1.3) over a small time interval  $\Delta t > 0$ . The Jacobian of the map  $\Psi$  with respect to  $\mathbf{x}_t$  expands as

$$\frac{\partial \Psi}{\partial \mathbf{x}_t} = \mathbf{I} + \Delta t \nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x}_t, t) + \mathcal{O}(\Delta t^2),$$

so its determinant satisfies

$$\det\left(\frac{\partial \Psi}{\partial \mathbf{x}_t}\right) = 1 + \Delta t \nabla \cdot \mathbf{f}(\mathbf{x}_t, t) + \mathcal{O}(\Delta t^2).$$

This uses the standard expansion  $\det(\mathbf{I} + \Delta t \mathbf{A}) = 1 + \Delta t \operatorname{Tr}(\mathbf{A}) + \mathcal{O}(\Delta t^2)$  as  $\Delta t \rightarrow 0$ , along with  $\nabla \cdot \mathbf{f} = \operatorname{Tr}(\nabla_{\mathbf{x}} \mathbf{f})$ .

Applying the change-of-variables formula, the log-density evolves as

$$\log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) = \log p_t(\mathbf{x}_t) - \Delta t \nabla \cdot \mathbf{f}(\mathbf{x}_t, t) + \mathcal{O}(\Delta t^2).$$

Applying the change-of-variables formula, the log-density evolves as

$$\log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) = \log p_t(\mathbf{x}_t) - \Delta t \nabla \cdot \mathbf{f}(\mathbf{x}_t, t) + \mathcal{O}(\Delta t^2).$$

That is,

$$\log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) - \log p_t(\mathbf{x}_t) = -\Delta t \nabla \cdot \mathbf{f}(\mathbf{x}_t, t) + \mathcal{O}(\Delta t^2). \quad (\text{B.1.5})$$

**Using Taylor Expansion.** Now, we expand the left-hand side via multivariate Taylor expansion:

$$\begin{aligned} & \log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) - \log p_t(\mathbf{x}_t) \\ &= \Delta t \partial_t \log p_t(\mathbf{x}_t) + (\mathbf{x}_{t+\Delta t} - \mathbf{x}_t)^{\top} \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + \mathcal{O}(\Delta t^2). \end{aligned}$$

Substituting  $\mathbf{x}_{t+\Delta t} - \mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)\Delta t$  yields:

$$\begin{aligned} & \log p_{t+\Delta t}(\mathbf{x}_{t+\Delta t}) - \log p_t(\mathbf{x}_t) \\ &= \Delta t \partial_t \log p_t(\mathbf{x}_t) + \Delta t \mathbf{f}(\mathbf{x}_t, t)^{\top} \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + \mathcal{O}(\Delta t^2). \end{aligned}$$

Matching terms with Equation (B.1.5) and letting  $\Delta t \rightarrow 0$ , we conclude that

$$\partial_t \log p_t(\mathbf{x}_t) = -\nabla_{\mathbf{x}_t} \cdot \mathbf{f}(\mathbf{x}_t, t) - \mathbf{f}(\mathbf{x}_t, t)^{\top} \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t).$$

Exponentiating and using the product rule yields the continuity equation.

**Velocity First (Lagrangian) vs. Density First (Eulerian).** It is important to note a key asymmetry between particle dynamics and density dynamics. Starting from a velocity field gives a unique flow of particles and hence a unique density evolution. In contrast, prescribing only the density path does not pin down a single velocity field: many different flows can lead to the same sequence of densities.

**Velocity-First (Eulerian: Flow  $\Rightarrow$  Density).** So far, we have assumed that the velocity field  $\mathbf{f}$  is given. The particle ODE

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), t)$$

describes how each particle moves, while the density PDE

$$\partial_t p_t + \nabla \cdot (p_t \mathbf{f}_t) = 0$$

describes how the entire distribution of particles evolves. These two views are connected: moving particles according to the ODE automatically produces a density that satisfies the PDE. In this case, the particle flow  $\Psi_{0 \rightarrow t}$  is uniquely determined: starting from  $\mathbf{x}(0) \sim p_0$ , each trajectory  $\mathbf{x}(t)$  is fixed, and the resulting density  $p_t$  follows the continuity equation. Here, particle dynamics and density dynamics are fully consistent.

**Density-First (Eulerian: Density  $\not\Rightarrow$  Unique Flow).** If instead we begin only with the density path  $t \mapsto p_t$  (e.g., Section 5.3.2 in flow matching), the velocity field is no longer uniquely determined. For example, if a vector field  $\mathbf{w}_t$  satisfies

$$\nabla_{\mathbf{x}} \cdot (p_t(\mathbf{x}) \mathbf{w}_t(\mathbf{x})) = 0 \quad (\text{no net flux w.r.t. } p_t),$$

then both  $\mathbf{f}_t$  and  $\mathbf{f}_t + \mathbf{w}_t$  give rise to the same density evolution. Thus a single density path may correspond to many different flows, and choosing one particular particle flow  $\Psi_{0 \rightarrow t}$  amounts to picking a specific velocity field among these possibilities.

Not every given path  $p_t$  can actually arise from particles moving under some velocity field. The continuity equation (Equation (B.1.4)) provides the consistency check for whether a density path can be “generated by a flow”. We say that  $p_t$  is *realizable* (or *generated by  $\mathbf{f}$* ) if there exists a velocity field  $\mathbf{f}$  such that particles following

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), t)$$

produce exactly the densities  $p_t$  through the flow map  $\Psi_{0 \rightarrow t}$ . That is, realizability holds when  $p_t$  and  $\mathbf{f}$  together satisfy Equation (B.1.4).

Intuitively, realizability means that the snapshots of  $p_t$  over time can be explained by particles moving under some velocity field, rather than being an arbitrary sequence of distributions.

When this condition holds, the density  $p_t$  is nothing more than the push-forward of the initial density  $p_0$  along the flow map  $\Psi_{0 \rightarrow t}$ . In this case, the familiar change-of-variables formula applies:

$$\begin{aligned} p_t &= (\Psi_{0 \rightarrow t})_\# p_0 \\ &= p_0(\Psi_{t \rightarrow 0}(\mathbf{x})) |\det \partial_{\mathbf{x}} \Psi_{t \rightarrow 0}(\mathbf{x})| \\ &= \int \delta(\mathbf{x} - \Psi_{t \rightarrow 0}(\mathbf{x}_0)) p_0(\mathbf{x}_0) d\mathbf{x}_0. \end{aligned}$$

**(Optional) Conditioning.** If an additional conditioning variable  $\mathbf{z} \sim \pi(\mathbf{z})$  is introduced, the same reasoning applies for each fixed  $\mathbf{z}$ :

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}_t(\mathbf{x}(t)|\mathbf{z})$$

with pushforward  $p_t(\cdot|\mathbf{z}) = (\Psi_{0 \rightarrow t}(\cdot; \mathbf{z}))_\# p_0$ , and continuity equation

$$\partial_t p_t(\mathbf{x}|\mathbf{z}) + \nabla \cdot (p_t(\mathbf{x}|\mathbf{z}) \mathbf{v}_t(\mathbf{x}|\mathbf{z})) = 0$$

The marginal density is then

$$p_t(\mathbf{x}) = \int p_t(\mathbf{x}|\mathbf{z}) \pi(\mathbf{z}) d\mathbf{z}.$$

### B.1.3 Stochastic Processes: Fokker–Planck Equation

When noise is added, the dynamics follow the SDE as in Equation (A.2.3):

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t) dt + g(t) d\mathbf{w}(t).$$

Then, the density  $p_t(\mathbf{x})$  satisfies the Fokker–Planck equation:

$$\begin{aligned} \frac{\partial p_t(\mathbf{x})}{\partial t} &= -\nabla \cdot (\mathbf{f}(\mathbf{x}, t) p_t(\mathbf{x})) + \frac{1}{2} g^2(t) \Delta p_t(\mathbf{x}) \\ &= -\nabla \cdot \left( \left( \mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \right) p_t(\mathbf{x}) \right). \end{aligned}$$

Here,  $\Delta p_t = \nabla \cdot \nabla_{\mathbf{x}} p_t$  is the Laplacian operator. Here, the first term describes transport of probability mass by the deterministic drift  $\mathbf{f}$ , while the second term models the spreading (diffusion) of the density due to stochastic noise with variance proportional to  $\frac{1}{2}g^2(t)$ .

The derivation of the Fokker–Planck equation is more involved; we refer it to Section C.1.4.

## B.2 Intuition of the Continuity Equation

In this section, we give a physical interpretation of the continuity equation, highlighting its role as a conservation law for probability density in a dynamical system.

### B.2.1 Physical Interpretation of the Continuity Equation

Consider a small fixed control volume (a rectangular box) in 3D space centered at  $\mathbf{x} = (x, y, z)$  with side lengths  $\Delta x$ ,  $\Delta y$ , and  $\Delta z$ . Let  $p(\mathbf{x}, t)$  denote the density of a conserved quantity (e.g., mass or probability) at position  $\mathbf{x}$  and time  $t$ . The total amount of the quantity inside the box is:

$$\text{Total quantity in box} = p(\mathbf{x}, t)\Delta x\Delta y\Delta z.$$

**How Does the Total Change?** Changes in the total quantity can only arise from flux across the box's boundary. Let  $\mathbf{j}(\mathbf{x}, t)$  denote the flux vector, representing the amount of quantity flowing per unit area per unit time.

**Flux in the  $x$ -Direction.** The inflow through the left face (at  $x$ ) is approximately:

$$j_x(x, y, z, t)\Delta y\Delta z,$$

and the outflow through the right face (at  $x + \Delta x$ ) is:

$$j_x(x + \Delta x, y, z, t)\Delta y\Delta z.$$

Thus, the net flux in the  $x$ -direction is:

$$[j_x(x, y, z, t) - j_x(x + \Delta x, y, z, t)]\Delta y\Delta z.$$

**Net Flux in All Directions.** Analogous terms arise in the  $y$ - and  $z$ -directions:

$$\begin{aligned} & [j_y(x, y, z, t) - j_y(x, y + \Delta y, z, t)]\Delta x\Delta z, \\ & [j_z(x, y, z, t) - j_z(x, y, z + \Delta z, t)]\Delta x\Delta y. \end{aligned}$$

Summing all contributions, the total net outflux from the box is:

$$-\nabla \cdot \mathbf{j}(\mathbf{x}, t)\Delta x\Delta y\Delta z.$$

**Rate of Change Inside the Box.** The rate of change of the total quantity within the box is:

$$\frac{\partial p}{\partial t}(\mathbf{x}, t)\Delta x\Delta y\Delta z.$$

**Conservation Principle.** Assuming the quantity is conserved (e.g., total mass or probability is constant in time), the rate of change equals the negative of the net outflux:

$$\frac{\partial p}{\partial t}(\mathbf{x}, t)\Delta x \Delta y \Delta z = -\nabla \cdot \mathbf{j}(\mathbf{x}, t)\Delta x \Delta y \Delta z.$$

**Local Form.** Canceling the common volume factor (valid for any small box), we obtain the local form of the continuity equation:

$$\frac{\partial p}{\partial t} + \nabla \cdot \mathbf{j} = 0.$$

### B.2.2 Derivation of the Continuity Equation from Conservation Laws

The continuity equation formalizes the conservation of a physical quantity, such as mass or charge, in a dynamical system. Let  $p(\mathbf{x}, t)$  denote the density of the conserved quantity at position  $\mathbf{x} \in \mathbb{R}^D$  and time  $t \in [0, T]$ , and let  $\mathbf{v}(\mathbf{x}, t)$  denote the velocity field.

**Step 1: Rate of Change within a Control Volume.** Consider an arbitrary control volume  $V \subset \mathbb{R}^D$  with boundary  $\partial V$ . The total amount of the conserved quantity in  $V$  is

$$\int_V p(\mathbf{x}, t) dV,$$

whose time derivative gives the rate of accumulation:

$$\frac{\partial}{\partial t} \int_V p(\mathbf{x}, t) dV.$$

**Step 2: Net Flux Through the Boundary.** The quantity exits  $V$  through  $\partial V$  with outward normal vector  $\mathbf{n}$ . The net outward flux is

$$\int_{\partial V} p(\mathbf{x}, t) \mathbf{v}(\mathbf{x}, t) \cdot \mathbf{n} dS.$$

**Step 3: Conservation Principle.** Conservation implies that the rate of accumulation within  $V$  equals the negative of the net outward flux:

$$\frac{\partial}{\partial t} \int_V p dV + \int_{\partial V} p \mathbf{v} \cdot \mathbf{n} dS = 0.$$

**Step 4: Divergence Theorem.** Applying the divergence theorem to convert the surface integral to a volume integral:

$$\int_{\partial V} p \mathbf{v} \cdot \mathbf{n} dS = \int_V \nabla \cdot (p \mathbf{v}) dV.$$

Hence,

$$\frac{\partial}{\partial t} \int_V p dV + \int_V \nabla \cdot (p \mathbf{v}) dV = 0.$$

**Step 5: Local Form.** Since the control volume  $V$  is arbitrary, the integrand must vanish pointwise. This yields the continuity equation.

# C

---

## Behind the Scenes of Diffusion Models: Itô's Calculus and Girsanov's Theorem

---

(Score-based) Diffusion models are built on SDEs: a drift that pushes states and a Brownian term that jitters them. Unlike ODE paths, Brownian paths are nowhere differentiable, so the ordinary chain rule fails. In this section, we introduce two fundamental tools that make the math precise:

- **Itô's Formula** is the correct chain rule for stochastic trajectories. It tells us how a function  $\mathbf{h}(\mathbf{x}_t, t)$  evolves when  $\mathbf{x}_t$  follows an SDE. It enables derivations of the Fokker–Planck equation, moment dynamics, the Itô product rule, and the identities used in score-based training.
- **Girsanov's Theorem** is a change-of-measure result on *path probabilities*. It quantifies how likelihoods change when the noise is fixed but the drift is altered. This links score matching to path-space KL divergence and explains why learning the score in the reverse SDE corresponds to maximizing the data likelihood.

With these tools, the standard diffusion model derivations (Fokker–Planck, reverse time SDE, training objectives, and likelihood relations) follow cleanly and without hand waving.

## C.1 Itô's Formula: The Chain Rule for Random Processes

Standard calculus does not directly apply to stochastic processes because Wiener processes are not differentiable in the classical sense. Instead, we use Itô's calculus, which provides rules for working with stochastic integrals.

### C.1.1 Motivation: Why Do We Need a Special Chain Rule?

Consider a deterministic time-varying function  $\mathbf{y}_t$  that evolves smoothly with time  $t$  (e.g., an ODE). If we have a function  $\mathbf{h}(\mathbf{y}_t, t)$ , the usual chain rule tells us:

$$\frac{d\mathbf{h}}{dt} = \frac{\partial \mathbf{h}}{\partial t} + \nabla_{\mathbf{y}} \mathbf{h} \frac{d\mathbf{y}_t}{dt}.$$

Here,  $\nabla_{\mathbf{y}} \mathbf{h}$  is the Jacobian of  $\mathbf{h}$ . This works perfectly for deterministic paths  $\mathbf{y}_t$ .

#### Question C.1.1

But what happens if  $\mathbf{x}_t$  is a stochastic process, say, driven by an SDE

$$d\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t) dt + g(t) d\mathbf{w}_t$$

as in Equation (A.2.3)? What SDE does the process  $\mathbf{h}(\mathbf{x}_t, t)$  satisfy?

**Why the Ordinary Chain Rule Fails?** Naïvely applying the classical chain rule yields

$$d\mathbf{h} = \frac{\partial \mathbf{h}}{\partial t} dt + \nabla_{\mathbf{y}} \mathbf{h} \cdot d\mathbf{x}_t.$$

However, this neglects that Brownian increments satisfy  $d\mathbf{w}_t = \mathcal{O}(\sqrt{dt})$  and

$$(d\mathbf{w}_t)^2 = dt.$$

Thus, second-order terms in  $d\mathbf{w}_t$  do not vanish in stochastic calculus, unlike classical calculus where  $(dt)^2$  terms are negligible.

#### Example: Simple Example— $h(x_t) = x_t^2$

To see the intuition, let us consider the simple real-valued function  $h(x_t) = x_t^2 \in \mathbb{R}$  where the random variable  $x_t \in \mathbb{R}$  satisfies

$$dx_t = \sigma dw_t,$$

with a constant  $\sigma > 0$ . If we try the classical chain rule,

$$dh = 2x_t dx_t = 2x_t \sigma dw_t.$$

If this were true, the expectation of  $h(x_t)$  would be constant in time because

$$\mathbb{E}[dh] = 2\sigma\mathbb{E}[x_t \, dw_t] = 2\sigma\mathbb{E}[x_t] \underbrace{\mathbb{E}[dw_t]}_{=0} = 0.$$

But we know from classical Brownian motion properties (see Equation (A.2.4)) that

$$\mathbb{E}[x_t^2] = \sigma^2 t,$$

which grows linearly in time. So the ordinary chain rule misses an important term. ■

### C.1.2 Deriving 1D Itô's Formula from Taylor Expansion

**Deterministic Chain Rule via Taylor Expansion.** To understand why the classical chain rule fails for stochastic processes defined by SDEs, we first revisit it in the deterministic setting using Taylor expansion. We consider the scalar case:  $y_t \in \mathbb{R}$  and  $h(\cdot, \cdot) \in \mathbb{R}$ . Formally treating  $dy_t = y_{t+dt} - y_t$ , with  $dt \approx 0$ , we expand:

$$\begin{aligned} & h(y_{t+dt}, t + dt) - h(y_t, t) \\ &= \frac{\partial h}{\partial t} dt + \frac{\partial h}{\partial y} dy_t + \frac{1}{2} \left( \frac{\partial^2 h}{\partial y^2} (dy_t)^2 + 2 \frac{\partial^2 h}{\partial t \partial y} dt dy_t + \frac{\partial^2 h}{\partial t^2} (dt)^2 \right) + \mathcal{O}(dt^3), \end{aligned}$$

Here,  $dt dy_t = \left(\frac{dy_t}{dt}\right) (dt)^2 = \mathcal{O}(dt^2)$ , and similarly  $(dt)^2 = \mathcal{O}(dt^2)$ . Therefore, all the gray parts are ignorable, and the full differential is:

$$dh = \frac{\partial h}{\partial t} dt + \frac{\partial h}{\partial y} dy_t + \mathcal{O}(dt^2).$$

**Itô's Formula via Stochastic Taylor Expansion.** Now consider a stochastic process  $x_t \in \mathbb{R}$  governed by the SDE:

$$dx_t = f(x_t, t) dt + g(t) dw_t,$$

where  $w_t$  is standard Brownian motion. We aim to compute the differential of a scalar-valued function  $h(x_t, t)$ .

Using the stochastic Taylor expansion (Kloeden *et al.*, 1992), which retains second-order terms in  $dx_t$ , we have:

$$\begin{aligned} & h(x_{t+dt}, t + dt) - h(x_t, t) \\ &= \frac{\partial h}{\partial t} dt + \frac{\partial h}{\partial x} dx_t + \frac{1}{2} \left( \frac{\partial^2 h}{\partial x^2} (dx_t)^2 + 2 \frac{\partial^2 h}{\partial t \partial x} dt dx_t + \frac{\partial^2 h}{\partial t^2} (dt)^2 \right) + \dots \end{aligned}$$

**Negligible Cross Terms.** By the scaling property of Brownian motion (Equation (A.2.4)),

$$dw_t = \mathcal{O}(\sqrt{dt}) \quad \Rightarrow \quad dt \cdot dw_t = \mathcal{O}((dt)^{3/2}).$$

Therefore,

$$dt \cdot dx_t = dt(f dt + g dw_t) = f(dt)^2 + g \cdot dt \cdot dw_t = \mathcal{O}((dt)^{3/2}).$$

So, the gray terms are negligible:  $\mathcal{O}((dt)^{3/2})$  or smaller.

**Second-Order Term ( $dx_t$ )<sup>2</sup>.** Expanding using the SDE:

$$\begin{aligned} (dx_t)^2 &= (f dt + g dw_t)^2 \\ &= f^2(dt)^2 + 2fg dt dw_t + g^2(dw_t)^2 \\ &= \mathcal{O}((dt)^2) + \mathcal{O}((dt)^{3/2}) + g^2 \mathcal{O}(dt) \\ &= g^2(t) dt + \mathcal{O}((dt)^{3/2}). \end{aligned}$$

Combining terms, we obtain the differential:

$$dh(x_t, t) = \frac{\partial h}{\partial t} dt + \frac{\partial h}{\partial x} dx_t + \frac{1}{2} \frac{\partial^2 h}{\partial x^2} g^2(t) dt.$$

Substituting  $dx_t = f(x_t, t) dt + g(t) dw_t$  yields:

$$dh(x_t, t) = \left( \frac{\partial h}{\partial t} + f \frac{\partial h}{\partial x} + \frac{1}{2} g^2 \frac{\partial^2 h}{\partial x^2} \right) dt + g \frac{\partial h}{\partial x} dw_t.$$

This is the 1D version of *Itô's formula*.

### Example: Simple Example— $h(x_t) = x_t^2$

We revisit the simple example:  $h(x_t) = x_t^2$ , where the stochastic process  $x_t \in \mathbb{R}$  satisfies

$$dx_t = \sigma dw_t,$$

with a constant  $\sigma > 0$ . Applying Itô's formula correctly to  $h(x_t) = x_t^2$ , we obtain:

$$dh(x_t) = d(x_t^2) = 2x_t dx_t + \sigma^2 dt.$$

Substituting  $dx_t = \sigma dw_t$ , this becomes:

$$d(x_t^2) = 2x_t \sigma dw_t + \sigma^2 dt.$$

### C.1.3 Itô's Formula: The Chain Rule for SDEs

We summarize the one-dimensional Itô's formula derived above. Using similar arguments, the result extends naturally to the multi-dimensional setting. While we omit the detailed derivation, we state the general formula for completeness.

Finally, we illustrate an application of Itô's formula by deriving the *Itô product rule*, which enables computation of  $d(\mathbf{x}_t^\top \mathbf{y}_t)$  for stochastic processes  $\mathbf{x}_t$  and  $\mathbf{y}_t$ .

**1D Itô's Formula.** Let  $x_t \in \mathbb{R}$  be a stochastic process satisfying the SDE:

$$dx_t = f(x_t, t) dt + g(t) dw_t.$$

For a scalar function  $h: \mathbb{R} \times [0, T] \rightarrow \mathbb{R}$ , the process  $h(x_t, t)$  satisfies:

$$dh(x_t, t) = \left( \frac{\partial h}{\partial t} + f \frac{\partial h}{\partial x} + \frac{1}{2} g^2 \frac{\partial^2 h}{\partial x^2} \right) dt + g \frac{\partial h}{\partial x} dw_t.$$

**Multidimensional Itô's Formula with Scalar Output.** Let  $\mathbf{x}_t \in \mathbb{R}^D$  satisfy the SDE:

$$d\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t) dt + g(t) d\mathbf{w}_t,$$

where  $\mathbf{f}: \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}^D$ ,  $g: [0, T] \rightarrow \mathbb{R}$ , and  $\mathbf{w}_t \in \mathbb{R}^D$  is a  $D$ -dimensional Brownian motion. Let  $h: \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}$  be a scalar-valued function. Then  $h(\mathbf{x}_t, t)$  satisfies:

$$dh(\mathbf{x}_t, t) = \left( \frac{\partial h}{\partial t} + \nabla_{\mathbf{x}} h^\top \mathbf{f} + \frac{1}{2} g^2(t) \text{Tr}(\nabla_{\mathbf{x}}^2 h) \right) dt + g(t) \nabla_{\mathbf{x}} h^\top d\mathbf{w}_t, \quad (\text{C.1.1})$$

where  $\nabla_{\mathbf{x}} h \in \mathbb{R}^D$  is the gradient and  $\nabla_{\mathbf{x}}^2 h \in \mathbb{R}^{D \times D}$  is the Hessian matrix of  $h$  with respect to  $\mathbf{x}$ .

#### Example: Itô's Product Rule

Let  $\mathbf{x}_t, \mathbf{y}_t \in \mathbb{R}^D$  be vector-valued stochastic processes governed by the SDEs:

$$\begin{aligned} d\mathbf{x}_t &= \mathbf{a}(\mathbf{x}_t, t) dt + b(t) d\mathbf{w}_t, \\ d\mathbf{y}_t &= \mathbf{c}(\mathbf{y}_t, t) dt + d(t) d\mathbf{w}_t, \end{aligned}$$

where  $\mathbf{a}, \mathbf{c} : \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}^D$  are vector fields, and  $b(t), d(t) \in \mathbb{R}$  are

scalar-valued functions. Here,  $\mathbf{w}_t \in \mathbb{R}^D$  denotes a standard  $D$ -dimensional Brownian motion.

We aim to derive the SDE for the scalar-valued process

$$z(t) := \mathbf{x}_t^\top \mathbf{y}_t.$$

Applying the multivariate Itô formula to the bilinear function  $h(\mathbf{x}, \mathbf{y}) := \mathbf{x}^\top \mathbf{y}$ , we obtain:

$$d(\mathbf{x}^\top \mathbf{y}) = (d\mathbf{x})^\top \mathbf{y} + \mathbf{x}^\top d\mathbf{y} + \text{Tr}[d\mathbf{x} \cdot (d\mathbf{y})^\top].$$

The Itô correction term is computed as:

$$\begin{aligned} d\mathbf{x} \cdot (d\mathbf{y})^\top &= b(t) d\mathbf{w}_t \cdot [d(t) d\mathbf{w}_t]^\top \\ &= b(t)d(t) d\mathbf{w}_t \cdot d\mathbf{w}_t^\top \\ &= b(t)d(t) dt \cdot \mathbf{I}_D. \end{aligned}$$

Thus,

$$\text{Tr}[d\mathbf{x} \cdot (d\mathbf{y})^\top] = b(t)d(t) \text{Tr}(\mathbf{I}_D) dt = Db(t)d(t) dt.$$

Putting everything together, the resulting SDE is:

$$d(\mathbf{x}^\top \mathbf{y}) = (d\mathbf{x})^\top \mathbf{y} + \mathbf{x}^\top d\mathbf{y} + Db(t)d(t) dt \quad (\text{C.1.2})$$

#### C.1.4 Itô's Formula's Application: Derivation of Fokker-Planck Equation

In this section, we apply Itô's formula from Equation (C.1.1) to derive the Fokker-Planck equation, a PDE that characterizes the time evolution of the probability density  $p_t(\mathbf{x})$  associated with the  $D$ -dimensional diffusion process defined by the SDE in Equation (A.2.3).

**Step 1: Apply Itô's Formula.** Let  $\phi(\mathbf{x}, t)$  be a smooth test function  $\phi: \mathbb{R}^D \times [0, T] \rightarrow \mathbb{R}$ . By Itô's Formula:

$$d\phi(\mathbf{x}_t, t) = \left( \frac{\partial \phi}{\partial t} + \nabla_{\mathbf{x}} \phi^\top \mathbf{f}(\mathbf{x}_t, t) + \frac{1}{2} g^2(t) \text{Tr}[\nabla_{\mathbf{x}}^2 \phi] \right) dt + g(t) \nabla_{\mathbf{x}} \phi^\top d\mathbf{w}_t.$$

**Step 2: Take Expectation.** Taking expectation over  $p_t(\mathbf{x})$  and noting  $\mathbb{E}[d\mathbf{w}_t] = 0$ :

$$\mathbb{E}[d\phi(\mathbf{x}_t, t)] = \mathbb{E} \left[ \left( \frac{\partial \phi}{\partial t} + \nabla_{\mathbf{x}} \phi^\top \mathbf{f}(\mathbf{x}_t, t) + \frac{1}{2} g^2(t) \text{Tr}[\nabla_{\mathbf{x}}^2 \phi] \right) dt \right].$$

**Step 3: Express Expectation via Density.** This expectation can be written as:

$$\mathbb{E}[d\phi(\mathbf{x}_t, t)] = \int \left( \frac{\partial \phi}{\partial t} + \nabla_{\mathbf{x}} \phi^\top \mathbf{f}(\mathbf{x}, t) + \frac{1}{2} g^2(t) \text{Tr}[\nabla_{\mathbf{x}}^2 \phi] \right) p_t(\mathbf{x}) d\mathbf{x} dt.$$

**Step 4: Integrate by Parts.** Use integration by parts (divergence theorem) in  $\mathbb{R}^D$ :

$$\begin{aligned} \int \nabla_{\mathbf{x}} \phi^\top \mathbf{f} p_t d\mathbf{x} &= - \int \phi \nabla_{\mathbf{x}} \cdot (\mathbf{f} p_t) d\mathbf{x}, \\ \int \text{Tr}[\nabla_{\mathbf{x}}^2 \phi] p_t d\mathbf{x} &= - \int \phi \Delta p_t d\mathbf{x}. \end{aligned}$$

**Step 5: Substitute and Rearrange.** Substituting back:

$$\mathbb{E}[d\phi(\mathbf{x}_t, t)] = \int \phi(\mathbf{x}, t) \left[ \frac{\partial p_t}{\partial t} + \nabla_{\mathbf{x}} \cdot (\mathbf{f} p_t) - \frac{1}{2} g^2(t) \Delta p_t \right] d\mathbf{x} dt.$$

**Step 6: Conclude Fokker-Planck Equation.** Since  $\phi$  is arbitrary, the integrand must vanish:

$$\frac{\partial p_t}{\partial t} = -\nabla_{\mathbf{x}} \cdot (\mathbf{f}(\mathbf{x}, t) p_t(\mathbf{x})) + \frac{1}{2} g^2(t) \Delta p_t(\mathbf{x}),$$

which completes the derivation of the Fokker-Planck equation.

### C.1.5 Itô's Formula Application: Closed-Form Solution of a Linear SDE

This subsection demonstrates how to obtain a closed-form solution for a linear SDE by using an integration factor (similar to the ODE case) and Itô's formula. The approach mirrors classical techniques for solving linear ODEs, but adapted to the stochastic setting.

We consider a linear SDE of the form

$$d\mathbf{x}_t = f(t)\mathbf{x}_t dt + g(t)d\mathbf{w}_t, \quad (\text{C.1.3})$$

where  $f(t)$  and  $g(t)$  are deterministic functions and  $\mathbf{w}_t$  is a standard Wiener process.

**Closed-Form Solution of a Linear SDE.** We derive the explicit solution to the linear SDE in Equation (C.1.3) using the method of integrating factors. This type of forward SDE commonly arises in diffusion models (see Section 4.1).

**Step 1: Define an Integration Factor.** Let

$$\Psi(t) := \exp\left(-\int_0^t f(s)ds\right), \quad \text{and define } \mathbf{y}_t := \Psi(t)\mathbf{x}_t.$$

**Step 2: Apply Itô's Formula.** We apply Itô's formula to the function  $\mathbf{h}(\mathbf{x}, t) := \Psi(t)\mathbf{x}$ . This is actually a special case of the Itô product rule in Equation (C.1.2). Since  $\Psi(t)$  is deterministic, there is no cross-variation term, and the formula simplifies to:

$$\begin{aligned}\mathbf{d}\mathbf{y}_t &= d[\Psi(t)\mathbf{x}_t] \\ &= \Psi'(t)\mathbf{x}_t dt + \Psi(t)d\mathbf{x}_t \\ &= -f(t)\Psi(t)\mathbf{x}_t dt + \Psi(t)[f(t)\mathbf{x}_t dt + g(t)d\mathbf{w}_t] \\ &= \Psi(t)g(t)d\mathbf{w}_t.\end{aligned}$$

Hence,

$$\mathbf{y}_t = \mathbf{y}_0 + \int_0^t \Psi(s)g(s)d\mathbf{w}(s) = \mathbf{x}_0 + \int_0^t \Psi(s)g(s)d\mathbf{w}(s),$$

since  $\Psi(0) = 1$ .

**Step 3: Solve for  $\mathbf{x}_t$ .** Using  $\mathbf{x}_t = \Psi(t)^{-1}\mathbf{y}_t$ , we obtain

$$\mathbf{x}_t = e^{\int_0^t f(s)ds} \left[ \mathbf{x}_0 + \int_0^t e^{-\int_0^s f(r)dr} g(s)d\mathbf{w}(s) \right]. \quad (\text{C.1.4})$$

This provides an explicit solution to the vector-valued SDE.

Below, we demonstrate two alternative approaches to compute the analytical form of  $p_t(\mathbf{x}_t|\mathbf{x}_0)$ .

**Analysis of the Closed-Form Solution.** Equation (C.1.4) reconfirms that  $p_t(\mathbf{x}_t|\mathbf{x}_0)$  is Gaussian. To see this, define

$$\phi(s) := e^{-\int_0^s f(u)du} g(s),$$

which is a deterministic matrix-valued function of time (assuming  $f(u)$  and  $g(s)$  are deterministic). The Itô integral  $\int_0^t \phi(s) d\mathbf{w}_s$  is then a zero-mean Gaussian random variable, as it is the stochastic integral of a deterministic function with respect to Brownian motion. Therefore,  $\mathbf{x}_t|\mathbf{x}_0$  is an affine transformation of a Gaussian random variable and hence itself Gaussian. Its distribution is fully characterized by its conditional mean and covariance.

We define the conditional mean and covariance (given initial condition  $\mathbf{x}_0$ ) as

$$\mathbf{m}(t) := \mathbb{E}[\mathbf{x}_t|\mathbf{x}_0], \quad \mathbf{P}(t) := \mathbb{E}[(\mathbf{x}_t - \mathbf{m}(t))(\mathbf{x}_t - \mathbf{m}(t))^\top|\mathbf{x}_0].$$

**Mean.** Using linearity of expectation and the fact that the Itô integral of a deterministic function has zero mean:

$$\mathbf{m}(t) = \mathbb{E} \left[ e^{\int_0^t f(s) ds} \left( \mathbf{x}_0 + \int_0^t \phi(s) d\mathbf{w}_s \right) \middle| \mathbf{x}_0 \right] = e^{\int_0^t f(s) ds} \mathbf{x}_0.$$

**Covariance.** Let  $\mathbf{z}_t := \int_0^t \phi(s) d\mathbf{w}_s$ . Then  $\mathbf{x}_t - \mathbf{m}(t) = A(t)\mathbf{z}_t$ , so

$$\mathbf{P}(t) = e^{2 \int_0^t f(s) ds} \mathbb{E}[\mathbf{z}_t \mathbf{z}_t^\top].$$

By Itô isometry<sup>1</sup>,

$$\mathbb{E}[\mathbf{z}_t \mathbf{z}_t^\top] = \left( \int_0^t \phi^2(s) ds \right) \mathbf{I}_D,$$

hence,

$$\mathbf{P}(t) = e^{2 \int_0^t f(s) ds} \left( \int_0^t \left( e^{-\int_0^s f(u) du} g(s) \right)^2 ds \right) \mathbf{I}_D.$$

This shows the conditional covariance is isotropic.

**Derivation of Mean and Variance ODEs in Equation (4.3.3).** Alternatively, we can derive the moment evolution equations directly from the linear SDE Equation (C.1.3).

**Mean Evolution.** Taking the conditional expectation of both sides of the SDE and using linearity:

$$\frac{d\mathbf{m}(t)}{dt} = \mathbb{E}[f(t)\mathbf{x}_t | \mathbf{x}_0] = f(t)\mathbb{E}[\mathbf{x}_t | \mathbf{x}_0] = f(t)\mathbf{m}(t).$$

<sup>1</sup>Itô's isometry links stochastic integrals to standard integrals in expectation; we omit the proof as it requires the full machinery of Itô calculus. For a process  $\psi : [0, T] \rightarrow \mathbb{R}^{D \times D}$ , Itô isometry states

$$\mathbb{E} \left[ \left\| \int_0^T \psi(t) d\mathbf{w}_t \right\|^2 \right] = \mathbb{E} \left[ \int_0^T \|\psi(t)\|_F^2 dt \right],$$

where  $\|\psi(t)\|_F^2 = \sum_{i,j=1}^D |\psi_{ij}(t)|^2$  is the Frobenius norm. For  $\psi(t) \in \mathbb{R}^D$  (a vector), the integral is scalar and the isometry simplifies to

$$\mathbb{E} \left[ \left( \int_0^T \psi(t) d\mathbf{w}_t \right)^2 \right] = \mathbb{E} \left[ \int_0^T \|\psi(t)\|^2 dt \right].$$

**Covariance Evolution.** Define the centered process  $\tilde{\mathbf{x}}_t := \mathbf{x}_t - \mathbf{m}(t)$ . Applying Itô's product rule (see Equation (C.1.2)):

$$d(\tilde{\mathbf{x}}_t \tilde{\mathbf{x}}_t^\top) = d\tilde{\mathbf{x}}_t \cdot \tilde{\mathbf{x}}_t^\top + \tilde{\mathbf{x}}_t \cdot d\tilde{\mathbf{x}}_t^\top + d\tilde{\mathbf{x}}_t \cdot d\tilde{\mathbf{x}}_t^\top.$$

From the SDE, we compute:

$$d\tilde{\mathbf{x}}_t = d\mathbf{x}_t - d\mathbf{m}(t) = f(t)\tilde{\mathbf{x}}_t dt + g(t) d\mathbf{w}_t.$$

Substituting into the product rule and taking expectation:

$$\begin{aligned} \frac{d\mathbf{P}(t)}{dt} &= \mathbb{E}[f(t)\tilde{\mathbf{x}}_t \tilde{\mathbf{x}}_t^\top + \tilde{\mathbf{x}}_t \tilde{\mathbf{x}}_t^\top f(t) + g^2(t)\mathbf{I}_D] \\ &= 2f(t)\mathbf{P}(t) + g^2(t)\mathbf{I}_D. \end{aligned}$$

Thus, we recover the moment evolution equations in Equation (4.3.3).

## C.2 Change-of-Variable For Measures: Girsanov's Theorem in Diffusion Models

Diffusion models harness SDEs to transform simple noise into rich data distributions. At the heart of this transformation lies a profound idea: we can reinterpret randomness by modifying only the deterministic part of an SDE (the drift) while preserving its underlying stochasticity. This is precisely where Girsanov's theorem enters the picture.

**The Core Idea.** Consider an observed continuous trajectory that describes the data's evolution from time  $t = 0$  to  $t = T$ , denoted as  $\mathbf{x}_{0:T} := \{\mathbf{x}_t | t \in [0, T]\}$ . Girsanov's theorem addresses a fundamental question:

### Question C.2.1

*Given this single observed path, what is its likelihood if we assume it was generated by one SDE, versus if we assume it was generated by a different SDE?*

We compare two hypothetical models for generating the same trajectory. Both of these assumed SDEs share the same underlying pure randomness, represented by a standard Wiener process (Brownian motion)  $\mathbf{w}_t$ , but differ only in their deterministic “push” or “drift” function. We assume  $\mathbf{x}_0$  has the same initial distribution for both assumed generating processes.

To build intuition, imagine  $\mathbf{x}_{0:T}$  as a wiggly line drawn on paper. One hypothesis is that it was produced by a “robot painter” guided by a drift  $\mathbf{f}$  and perturbed by random noise scaled by  $g(t)$ , yielding likelihood  $p_{\mathbf{f}}(\mathbf{x}_{0:T})$ . Alternatively, we imagine a second robot, with a different drift  $\tilde{\mathbf{f}}$  but using the same noise process, generating the same line with likelihood  $p_{\tilde{\mathbf{f}}}(\mathbf{x}_{0:T})$ . Girsanov's theorem gives us a precise way to compare these two likelihoods for the exact same observed path. It quantifies how a change in drift affects the probability of generating a particular trajectory, while holding the randomness fixed.

**The Setup.** Let  $\mathbf{x}_t \in \mathbb{R}^D$  be our single, fixed, continuous path. We consider its likelihood under two SDE models, which differ only in their drift functions  $\mathbf{f}$  and  $\tilde{\mathbf{f}}$ . They share the same diffusion coefficient  $g(t) \in \mathbb{R}$  and the same underlying Wiener process  $\mathbf{w}_t$ :

$$\begin{aligned} d\mathbf{x}_t &= \mathbf{f}(\mathbf{x}_t, t) dt + g(t) d\mathbf{w}_t && (\text{Model with drift } \mathbf{f}) \\ d\mathbf{x}_t &= \tilde{\mathbf{f}}(\mathbf{x}_t, t) dt + g(t) d\mathbf{w}_t && (\text{Model with drift } \tilde{\mathbf{f}}) \end{aligned}$$

Let  $\boldsymbol{\delta}_t := \mathbf{f}(\mathbf{x}_t, t) - \tilde{\mathbf{f}}(\mathbf{x}_t, t)$  represent the difference in drifts for the given path  $\mathbf{x}_t$ .

**Girsanov's Likelihood Ratio.** Girsanov's theorem provides a fundamental likelihood ratio between these two ways of interpreting *the same observed path*. It states:

$$\frac{p_{\mathbf{f}}(\mathbf{x}_{0:T})}{p_{\tilde{\mathbf{f}}}(\mathbf{x}_{0:T})} = \exp \left( \int_0^T \boldsymbol{\delta}_t^\top g(t)^{-1} d\mathbf{w}_t - \frac{1}{2} \int_0^T \|g(t)^{-1} \boldsymbol{\delta}_t\|^2 dt \right).$$

This compact formula is an exponential of two integrals. The first is an Itô integral, while the second is a standard Riemann integral. This ratio is crucial in diffusion models, allowing us to bridge between different data generation processes and to evaluate model likelihoods.

Girsanov's theorem is best understood as a change-of-variable formula for *measures*. Just as a change of variables in calculus transforms an integral between coordinate systems via the Jacobian determinant, Girsanov's theorem provides the corresponding factor (the Radon–Nikodym derivative) to transform probabilities or expectations between two stochastic processes, when the drift changes but the diffusion remains the same.

### C.2.1 Girsanov's Theorem as a Bridge Between Likelihood Training and Score Matching

After understanding how Girsanov's theorem relates the likelihoods of a single path under different drift assumptions, we now delve into its implications for diffusion models.

Recall the forward SDE in diffusion models:

$$d\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t) dt + g(t) d\mathbf{w}_t,$$

which induces a *path distribution*  $P$  over full trajectories  $\mathbf{x}_{0:T} := \{\mathbf{x}_t\}_{t=0}^T$  (that is, the joint law of the process over the entire time interval). The reverse-time SDE, parameterized by a learnable score function  $\mathbf{s}_\phi(\mathbf{x}_t, t)$ , is given by

$$d\mathbf{x}_t = [\mathbf{f}(\mathbf{x}_t, t) - g^2(t) \mathbf{s}_\phi(\mathbf{x}_t, t)] dt + g(t) d\bar{\mathbf{w}}_t,$$

which in turn defines another path distribution  $P_\phi$  over trajectories.

**Two Notions in Diffusion Models.** In diffusion models, we navigate between two core perspectives for describing the stochastic process  $\mathbf{x}_{0:T}$ : the forward process and its reverse-time counterpart. These perspectives give rise to two distinct but related objectives:

- **Concept 1. Marginal Distribution Matching:** This goal constructs a reverse-time process whose marginals  $p_t(\mathbf{x}_t)$  match those of the forward SDE, starting from noise at time  $T$  and recovering the data distribution at  $t = 0$ . As emphasized, the Fokker–Planck equation ensures this marginal consistency for the reverse-time SDE.
- **Concept 2. Joint Path Distribution Matching:** This stronger objective seeks to match the full joint distribution over the entire trajectory  $P = p(\mathbf{x}_{0:T})$ . Rather than just matching snapshots at individual time steps, this condition ensures that the entire sequence of states and their temporal dependencies are faithfully reproduced.

Matching the full path distribution  $P$  ensures all marginals match. Formally, let  $\mathbf{x}_{0:T} := \{\mathbf{x}_t | t \in [0, T]\}$  be a stochastic process with joint distribution  $p(\mathbf{x}_{0:T})$ . Suppose another process with joint  $q(\mathbf{x}_{0:T})$  satisfies

$$p(\mathbf{x}_{0:T}) = q(\mathbf{x}_{0:T}).$$

Then for any  $t \in [0, T]$ , the marginal distributions are

$$p_t(\mathbf{x}_t) = \int p(\mathbf{x}_{0:T}) d\mathbf{x}_{[0,T] \setminus \{t\}}, \quad q_t(\mathbf{x}_t) = \int q(\mathbf{x}_{0:T}) d\mathbf{x}_{[0,T] \setminus \{t\}},$$

which implies

$$p_t(\mathbf{x}_t) = q_t(\mathbf{x}_t), \quad \forall t \in [0, T].$$

Thus, joint path matching implies marginal matching.

However, the reverse is not true: two processes may share identical marginals at every time step yet differ significantly in their temporal correlations. Marginal matching lacks the ability to capture these inter-time dependencies, which are encoded only in the joint distribution.

**Girsanov Bridges the Two Goals.** While reverse-time SDEs are primarily designed for marginal matching (Concept 1), Girsanov's theorem reveals a deeper connection: score matching across time also encourages joint path matching (Concept 2).

More precisely, Girsanov's theorem relates the forward path distribution  $P$  and the learned reverse path distribution  $P_\phi$ . The objective function in

score-based diffusion models is the KL divergence between these path measures:

$$\mathcal{D}_{\text{KL}}(P\|P_{\phi}) = \frac{1}{2} \mathbb{E}_P \left[ \int_0^T g^2(t) \|\mathbf{s}_{\phi}(\mathbf{x}_t, t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\|^2 dt \right] + \text{Const.}, \quad (\text{C.2.1})$$

Here, the constant does not depend on  $\phi$ , and we use the fact that the Itô integral has zero expectation under  $P$ . This expression shows that minimizing KL divergence between joint paths is equivalent to learning a score function  $\mathbf{s}_{\phi}$  that approximates the true score  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ . Thus, score matching, although framed as a marginal objective, effectively promotes alignment of the entire joint path distribution.

**Implicit Likelihood Training.** Beyond just matching path distributions, score matching implicitly allows diffusion models to achieve a fundamental goal of generative modeling: approximating the data likelihood (Song *et al.*, 2021).

The connection becomes clear through a powerful concept called the change-of-measure formula. This formula, illuminated by Girsanov’s theorem, allows us to express the logarithm of the marginal likelihood of the data at  $t = 0$  ( $p_{\phi}(\mathbf{x}_0)$ ) under our learned model.

$$\log p_{\phi}(\mathbf{x}_0) = \log \int p_T(\mathbf{x}_T) \cdot \frac{p_{\phi}(\mathbf{x}_{0:T})}{p(\mathbf{x}_{0:T})} p(\mathbf{x}_{0:T}) d\mathbf{x}_{0:T}. \quad (\text{C.2.2})$$

Here,  $p_T(\mathbf{x}_T)$  is the known distribution of noise at time  $T$  given by the forward SDE. The term  $\frac{p_{\phi}(\mathbf{x}_{0:T})}{p(\mathbf{x}_{0:T})}$  is the density ratio between the learned reverse process and the forward process for a given path  $\mathbf{x}_{0:T}$ —an object precisely quantified by Girsanov’s theorem. Essentially, this formula calculates the likelihood of generated data by re-weighting the known likelihood of noise based on how well our learned reverse dynamics explain the observed path’s trajectory.

We further draw connection back to the KL minimization in Equation (C.2.1) which concerns the discrepancy between the full forward path distribution and the learned reverse path distribution. The two, Equation (C.2.1) and Equation (C.2.2) are deeply intertwined: optimizing this score matching objective (the training loss) directly translates to learning the Girsanov density ratio, thereby implicitly maximizing the data likelihood ( $p_{\phi}(\mathbf{x}_0)$ ). This elegant connection beautifully ties together Girsanov’s theorem, score-based learning, and the ultimate generative modeling goal of assigning high probability to real data.

# D

---

## Supplementary Materials and Proofs

---

### D.1 Variational Perspective

#### D.1.1 Theorem 2.2.1: Equivalence Between Marginal and Conditional KL Minimization

*Proof.* Derivation of Equation (2.2.3).

We start by expanding the right-hand side expectation:

$$\begin{aligned} & \mathbb{E}_{p(\mathbf{x}_0, \mathbf{x}_i)} [\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \| p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))] \\ &= \int \int p(\mathbf{x}_0, \mathbf{x}_i) \mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \| p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)) d\mathbf{x}_0 d\mathbf{x}_i. \end{aligned}$$

By the definition of KL divergence,

$$\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \| p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)) = \int p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} d\mathbf{x}_{i-1}.$$

Substituting this into the expectation, we have

$$\int \int \int p(\mathbf{x}_0, \mathbf{x}_i) p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} d\mathbf{x}_{i-1} d\mathbf{x}_0 d\mathbf{x}_i.$$

Using the chain rule of probability,

$$p(\mathbf{x}_0, \mathbf{x}_i) = p(\mathbf{x}_i)p(\mathbf{x}_0|\mathbf{x}_i),$$

we rewrite the integral as

$$\int p(\mathbf{x}_i) \int p(\mathbf{x}_0|\mathbf{x}_i) \int p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} d\mathbf{x}_{i-1} d\mathbf{x}_0 d\mathbf{x}_i.$$

This allows us to express the expectation in nested form:

$$\mathbb{E}_{p(\mathbf{x}_i)} \left[ \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)} \left[ \mathbb{E}_{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)} \left[ \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} \right] \right] \right].$$

Next, we apply the decomposition of the logarithm:

$$\log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} = \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)}{p(\mathbf{x}_{i-1}|\mathbf{x}_i)} + \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)}.$$

Substituting this back into the expectation gives two terms:

$$\begin{aligned} & \mathbb{E}_{p(\mathbf{x}_i)} \left[ \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)} \left[ \mathbb{E}_{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)} \left[ \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)}{p(\mathbf{x}_{i-1}|\mathbf{x}_i)} \right] \right] \right] \\ & + \mathbb{E}_{p(\mathbf{x}_i)} \left[ \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)} \left[ \mathbb{E}_{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)} \left[ \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} \right] \right] \right]. \end{aligned}$$

Since the second logarithmic term does not depend on  $\mathbf{x}_0$ , by the law of total probability

$$\mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)} \left[ \mathbb{E}_{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0)} \left[ \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} \right] \right] = \mathbb{E}_{p(\mathbf{x}_{i-1}|\mathbf{x}_i)} \left[ \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i)}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} \right].$$

Similarly, the first term is the KL divergence

$$\mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)} [\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \| p(\mathbf{x}_{i-1}|\mathbf{x}_i))].$$

Putting it all together, we obtain the decomposition:

$$\begin{aligned} & \mathbb{E}_{p(\mathbf{x}_0, \mathbf{x}_i)} [\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \| p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))] \\ & = \mathbb{E}_{p(\mathbf{x}_i)} \left[ \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_i)} [\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}_0) \| p(\mathbf{x}_{i-1}|\mathbf{x}_i))] \right] \\ & + \mathbb{E}_{p(\mathbf{x}_i)} [\mathcal{D}_{\text{KL}}(p(\mathbf{x}_{i-1}|\mathbf{x}_i) \| p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i))]. \end{aligned}$$

**Proof of Optimality.** To prove:

$$p^*(\mathbf{x}_{i-1}|\mathbf{x}_i) = p(\mathbf{x}_{i-1}|\mathbf{x}_i) = \mathbb{E}_{p(\mathbf{x}|\mathbf{x}_i)} [p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})], \quad \mathbf{x}_i \sim p_i.$$

The first identity follows from the fact that the KL divergence  $\mathcal{D}_{\text{KL}}(p \| p_\phi)$  is minimized when  $p^* = p$ , assuming the parameterization is sufficiently expressive. The second identity follows directly from the law of total probability.

■

### D.1.2 Theorem 2.2.3: ELBO of Diffusion Model

**Proof.** For notational simplicity, we denote  $\mathbf{x}_{0:L} := (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_L)$ .

**Step 1: Apply Jensen's Inequality.** The marginal log-likelihood is given by:

$$\log p_\phi(\mathbf{x}) = \log \int p_\phi(\mathbf{x}, \mathbf{x}_{0:L}) d\mathbf{x}_0 \cdots d\mathbf{x}_L,$$

with the joint distribution:

$$p_\phi(\mathbf{x}, \mathbf{x}_{0:L}) = p_{\text{prior}}(\mathbf{x}_L) \prod_{i=1}^L p_\phi(\mathbf{x}_{i-1} | \mathbf{x}_i) \cdot p_\phi(\mathbf{x} | \mathbf{x}_0).$$

We introduce the variational distribution  $p(\mathbf{x}_{0:L} | \mathbf{x})$  and rewrite:

$$\log p_\phi(\mathbf{x}) = \log \int p(\mathbf{x}_{0:L} | \mathbf{x}) \frac{p_\phi(\mathbf{x}, \mathbf{x}_{0:L})}{p(\mathbf{x}_{0:L} | \mathbf{x})} d\mathbf{x}_0 \cdots d\mathbf{x}_L.$$

Applying Jensen's inequality ( $\log \mathbb{E}[Z] \geq \mathbb{E}[\log Z]$ ), we obtain the ELBO:

$$\log p_\phi(\mathbf{x}) \geq \mathbb{E}_{p(\mathbf{x}_{0:L} | \mathbf{x})} \left[ \log \frac{p_\phi(\mathbf{x}, \mathbf{x}_{0:L})}{p(\mathbf{x}_{0:L} | \mathbf{x})} \right] =: \mathcal{L}_{\text{ELBO}},$$

and thus,

$$-\log p_\phi(\mathbf{x}) \leq -\mathcal{L}_{\text{ELBO}}.$$

**Step 2: Expand the ELBO.** Assume the variational distribution factorizes as:

$$p(\mathbf{x}_{0:L} | \mathbf{x}) = p(\mathbf{x}_L | \mathbf{x}) \prod_{i=1}^L p(\mathbf{x}_{i-1} | \mathbf{x}_i, \mathbf{x}).$$

Substituting the joint and variational distributions into the ELBO:

$$\begin{aligned} \mathcal{L}_{\text{ELBO}} &= \mathbb{E}_{p(\mathbf{x}_{0:L} | \mathbf{x})} \left[ \log p_{\text{prior}}(\mathbf{x}_L) + \sum_{i=1}^L \log p_\phi(\mathbf{x}_{i-1} | \mathbf{x}_i) + \log p_\phi(\mathbf{x} | \mathbf{x}_0) \right. \\ &\quad \left. - \log p(\mathbf{x}_L | \mathbf{x}) - \sum_{i=1}^L \log p(\mathbf{x}_{i-1} | \mathbf{x}_i, \mathbf{x}) \right]. \end{aligned}$$

We now compute the negative ELBO by grouping terms according to their dependencies and applying marginalization:

$$\begin{aligned} -\mathcal{L}_{\text{ELBO}} &= \mathbb{E}_{p(\mathbf{x}_0 | \mathbf{x})} [-\log p_\phi(\mathbf{x} | \mathbf{x}_0)] + \mathbb{E}_{p(\mathbf{x}_L | \mathbf{x})} \left[ \log \frac{p(\mathbf{x}_L | \mathbf{x})}{p_{\text{prior}}(\mathbf{x}_L)} \right] \\ &\quad + \sum_{i=1}^L \mathbb{E}_{p(\mathbf{x}_i | \mathbf{x})} \left[ \mathbb{E}_{p(\mathbf{x}_{i-1} | \mathbf{x}_i, \mathbf{x})} \left[ \log \frac{p(\mathbf{x}_{i-1} | \mathbf{x}_i, \mathbf{x})}{p_\phi(\mathbf{x}_{i-1} | \mathbf{x}_i)} \right] \right]. \end{aligned}$$

To justify the last term, we use the factorization:

$$p(\mathbf{x}_i, \mathbf{x}_{i-1} | \mathbf{x}) = p(\mathbf{x}_i | \mathbf{x}) \cdot p(\mathbf{x}_{i-1} | \mathbf{x}_i, \mathbf{x}),$$

which leads to:

$$\begin{aligned}
& \mathbb{E}_{p(\mathbf{x}_{0:L}|\mathbf{x})} \left[ \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} \right] \\
&= \int p(\mathbf{x}_i|\mathbf{x}) \left[ \int p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x}) \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} d\mathbf{x}_{i-1} \right] d\mathbf{x}_i \\
&= \mathbb{E}_{p(\mathbf{x}_i|\mathbf{x})} \left[ \mathbb{E}_{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})} \left[ \log \frac{p(\mathbf{x}_{i-1}|\mathbf{x}_i, \mathbf{x})}{p_\phi(\mathbf{x}_{i-1}|\mathbf{x}_i)} \right] \right].
\end{aligned}$$

We may refer to the three terms in  $-\mathcal{L}_{\text{ELBO}}$  as:

$$\mathcal{L}_{\text{recon.}}, \quad \mathcal{L}_{\text{prior}}, \quad \mathcal{L}_{\text{diffusion}},$$

corresponding respectively to the reconstruction loss, the prior KL, and the stepwise diffusion KL. This completes the derivation. ■

## D.2 Score-Based Perspective

### D.2.1 Proposition 3.2.1: Tractable Score Matching via Integration by Parts

**Proof.** **Expanding  $\mathcal{L}_{\text{SM}}(\phi)$ .** Let us expand the squared difference inside the expectation:

$$\begin{aligned}\mathcal{L}_{\text{SM}}(\phi) &= \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \left[ \|\mathbf{s}_\phi(\mathbf{x})\|_2^2 - 2\langle \mathbf{s}_\phi(\mathbf{x}), \mathbf{s}(\mathbf{x}) \rangle + \|\mathbf{s}(\mathbf{x})\|_2^2 \right] \\ &= \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \left[ \|\mathbf{s}_\phi(\mathbf{x})\|_2^2 \right] - \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\langle \mathbf{s}_\phi(\mathbf{x}), \mathbf{s}(\mathbf{x}) \rangle] \\ &\quad + \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \left[ \|\mathbf{s}(\mathbf{x})\|_2^2 \right].\end{aligned}$$

We now focus on the cross-product term:

$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\langle \mathbf{s}_\phi(\mathbf{x}), \mathbf{s}(\mathbf{x}) \rangle].$$

Using the fact that

$$\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) = \frac{\nabla_{\mathbf{x}} p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x})},$$

and assuming  $p_{\text{data}}(\mathbf{x})$  is not zero (e.g., on its support), the cross-product term becomes:

$$\begin{aligned}\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\langle \mathbf{s}_\phi(\mathbf{x}), \mathbf{s}(\mathbf{x}) \rangle] &= \int \mathbf{s}_\phi(\mathbf{x})^\top \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) p_{\text{data}}(\mathbf{x}) d\mathbf{x} \\ &= \int \mathbf{s}_\phi(\mathbf{x})^\top \nabla_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) d\mathbf{x} \\ &= \sum_{i=1}^D \int s_\phi^{(i)}(\mathbf{x}) \partial_{x_i} p_{\text{data}}(\mathbf{x}) d\mathbf{x},\end{aligned}$$

where  $s_\phi^{(i)}(\mathbf{x})$  is the  $i$ -th component of the score function

$$\mathbf{s}_\phi = (s_\phi^{(1)}, s_\phi^{(2)}, \dots, s_\phi^{(D)}).$$

**Integration by Parts.** We use the following integration-by-parts formula (Evans, 2010), which is derived from standard calculus:

**Lemma.** Let  $u, v$  be differentiable real-valued functions on a ball  $\mathbb{B}(\mathbf{0}, R) \subset \mathbb{R}^D$  of radius  $R > 0$ . Then for  $i = 1, \dots, D$ , the formula holds:

$$\int_{\mathbb{B}(\mathbf{0}, R)} u \partial_{x_i} v d\mathbf{x} = - \int_{\mathbb{B}(\mathbf{0}, R)} v \partial_{x_i} u d\mathbf{x} + \int_{\partial \mathbb{B}(\mathbf{0}, R)} u v \nu_i dS,$$

where  $\nu = (\nu_1, \dots, \nu_D)$  is the outward unit normal to the boundary  $\partial \mathbb{B}(\mathbf{0}, R)$ —a sphere with radius  $R > 0$ , and  $dS$  is the surface measure on  $\partial \mathbb{B}(\mathbf{0}, R)$ .

We apply this formula to  $u(\mathbf{x}) := s_\phi^{(i)}(\mathbf{x})$  and  $v(\mathbf{x}) = p_{\text{data}}(\mathbf{x})$  for all  $i = 1, \dots, D$ , assuming that

$$|u(\mathbf{x})v(\mathbf{x})| \rightarrow 0 \quad \text{as } R \rightarrow \infty.$$

Summing the results over all  $i = 1, \dots, D$ , we get:

$$\begin{aligned} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\langle \mathbf{s}_\phi(\mathbf{x}), \mathbf{s}(\mathbf{x}) \rangle] &= - \sum_{i=1}^D \int \partial_{x_i} s_\phi^{(i)}(\mathbf{x}) p_{\text{data}}(\mathbf{x}) d\mathbf{x} \\ &= -\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\text{Tr}(\nabla_{\mathbf{x}} \mathbf{s}_\phi(\mathbf{x}))]. \end{aligned}$$

Combining all results, we have:

$$\begin{aligned} \mathcal{L}_{\text{SM}}(\phi) &= \underbrace{\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \left[ \text{Tr}(\nabla_{\mathbf{x}} \mathbf{s}_\phi(\mathbf{x})) + \frac{1}{2} \|\mathbf{s}_\phi(\mathbf{x})\|_2^2 \right]}_{\tilde{\mathcal{L}}_{\text{SM}}(\phi)} \\ &\quad + \underbrace{\frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\|\mathbf{s}(\mathbf{x})\|_2^2]}_{=:C}, \end{aligned}$$

where  $C$  depends only on the distribution  $p_{\text{data}}$ , which concludes the proof. ■

## D.2.2 Theorem 3.3.1: Equivalence Between SM and DSM Minimization

**Proof.** Expanding both  $\mathcal{L}_{\text{SM}}(\phi; \sigma)$  and  $\mathcal{L}_{\text{DSM}}(\phi; \sigma)$ , we have:

$$\begin{aligned} \mathcal{L}_{\text{SM}}(\phi; \sigma) &= \frac{1}{2} \mathbb{E}_{\tilde{\mathbf{x}} \sim p_\sigma(\tilde{\mathbf{x}})} \left[ \|\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)\|_2^2 - 2\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)^\top \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}) \right. \\ &\quad \left. + \|\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})\|_2^2 \right], \end{aligned}$$

$$\begin{aligned} \mathcal{L}_{\text{DSM}}(\phi; \sigma) &= \frac{1}{2} \mathbb{E}_{p_{\text{data}}(\mathbf{x}) p_\sigma(\tilde{\mathbf{x}} | \mathbf{x})} \left[ \|\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)\|_2^2 - 2\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)^\top \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} | \mathbf{x}) \right. \\ &\quad \left. + \|\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} | \mathbf{x})\|_2^2 \right]. \end{aligned}$$

Subtracting the two equations yields:

$$\begin{aligned}
& \mathcal{L}_{\text{SM}}(\phi; \sigma) - \mathcal{L}_{\text{DSM}}(\phi; \sigma) \\
&= \frac{1}{2} \left( \mathbb{E}_{\tilde{\mathbf{x}} \sim p_\sigma(\tilde{\mathbf{x}})} \|\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)\|_2^2 - \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})} \|\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)\|_2^2 \right) \\
&\quad - \left( \mathbb{E}_{\tilde{\mathbf{x}} \sim p_\sigma(\tilde{\mathbf{x}})} \left[ \mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)^\top \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}) \right] \right. \\
&\quad \left. - \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})} \left[ \mathbf{s}_\phi(\tilde{\mathbf{x}}; \mathbf{x})^\top \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) \right] \right) \\
&+ \frac{1}{2} \left( \mathbb{E}_{\tilde{\mathbf{x}} \sim p_\sigma(\tilde{\mathbf{x}})} \|\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})\|_2^2 - \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})} \|\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\|_2^2 \right).
\end{aligned}$$

Next, we address the three terms one at a time. For the first term, since  $p_\sigma(\tilde{\mathbf{x}}) = \int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p_{\text{data}}(\mathbf{x}) d\mathbf{x}$ , we can rewrite it as:

$$\begin{aligned}
\mathbb{E}_{\tilde{\mathbf{x}} \sim p_\sigma(\tilde{\mathbf{x}})} \|\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)\|_2^2 &= \int \left( \int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p_{\text{data}}(\mathbf{x}) d\mathbf{x} \right) \|\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)\|_2^2 d\tilde{\mathbf{x}} \\
&= \int p_{\text{data}}(\mathbf{x}) \int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) \|\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)\|_2^2 d\tilde{\mathbf{x}} d\mathbf{x} \\
&= \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})} \|\mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)\|_2^2.
\end{aligned}$$

Thus, the first term is zero. For the second term:

$$\begin{aligned}
& \mathbb{E}_{\tilde{\mathbf{x}} \sim p_\sigma(\tilde{\mathbf{x}})} \left[ \mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)^\top \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}) \right] \\
&= \int p_\sigma(\tilde{\mathbf{x}}) \mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)^\top \frac{\nabla_{\tilde{\mathbf{x}}} p_\sigma(\tilde{\mathbf{x}})}{p_\sigma(\tilde{\mathbf{x}})} d\tilde{\mathbf{x}} \\
&= \int \mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)^\top \nabla_{\tilde{\mathbf{x}}} \int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p_{\text{data}}(\mathbf{x}) d\mathbf{x} d\tilde{\mathbf{x}} \\
&= \int \int \mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)^\top \nabla_{\tilde{\mathbf{x}}} p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p_{\text{data}}(\mathbf{x}) d\tilde{\mathbf{x}} d\mathbf{x} \\
&= \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})} \left[ \mathbf{s}_\phi(\tilde{\mathbf{x}}; \sigma)^\top \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) \right].
\end{aligned} \tag{D.2.1}$$

Thus, it is also zero. For the third term, note that:

$$C := \frac{1}{2} \left( \mathbb{E}_{\tilde{\mathbf{x}} \sim p_\sigma(\tilde{\mathbf{x}})} \|\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})\|_2^2 - \mathbb{E}_{p_{\text{data}}(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})} \|\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\|_2^2 \right)$$

depends only on  $p_{\text{data}}(\mathbf{x})$  and  $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ , and hence it is constant with respect to  $\phi$ . ■

### D.2.3 Lemma 3.3.2: Tweedie's Formula

We first state a more general form of Tweedie's formula, which considers time dependent Gaussian perturbations, and we provide its proof below.

**Tweedie's Identity with Time-Dependent Parameters.** Let  $\mathbf{x}_t \sim \mathcal{N}(\cdot; \alpha_t \mathbf{x}_0, \sigma_t^2 \mathbf{I})$  be a Gaussian random vector. Then Tweedie's formula says

$$\alpha_t \mathbb{E}_{\mathbf{x}_0 \sim p(\mathbf{x}_0 | \mathbf{x}_t)} [\mathbf{x}_0 | \mathbf{x}_t] = \mathbf{x}_t + \sigma_t^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t),$$

where the expectation is taken over the posterior distribution  $p(\mathbf{x}_0 | \mathbf{x}_t)$  of  $\mathbf{x}_0$  given the observed  $\mathbf{x}_t$ , and  $p_t(\mathbf{x}_t)$  is the marginal density of  $\mathbf{x}_t$ .

**Proof.**

**Marginal Density and Its Score.** We recall that the marginal density of  $\mathbf{x}_t$  is given by

$$p_t(\mathbf{x}_t) = \int p_t(\mathbf{x}_t | \mathbf{x}_0) p_0(\mathbf{x}_0) d\mathbf{x}_0.$$

We now compute the score function:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \frac{\nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t)}{p_t(\mathbf{x}_t)} = \frac{1}{p_t(\mathbf{x}_t)} \int \nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t | \mathbf{x}_0) p_0(\mathbf{x}_0) d\mathbf{x}_0.$$

We therefore need to compute the gradient of the conditional density.

**Gradient of the Conditional and Rearrangement.** The gradient of the conditional Gaussian density is:

$$\nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t | \mathbf{x}_0) = -p_t(\mathbf{x}_t | \mathbf{x}_0) \cdot \sigma_t^{-2} (\mathbf{x}_t - \alpha_t \mathbf{x}_0).$$

Substituting this into the previous expression, we have:

$$\begin{aligned} \nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t) &= \int \nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t | \mathbf{x}_0) p_0(\mathbf{x}_0) d\mathbf{x}_0 \\ &= -\sigma_t^{-2} \int (\mathbf{x}_t - \alpha_t \mathbf{x}_0) p_t(\mathbf{x}_t | \mathbf{x}_0) p_0(\mathbf{x}_0) d\mathbf{x}_0 \\ &= -\sigma_t^{-2} \int (\mathbf{x}_t - \alpha_t \mathbf{x}_0) p(\mathbf{x}_0 | \mathbf{x}_t) p_t(\mathbf{x}_t) d\mathbf{x}_0 \\ &= -p_t(\mathbf{x}_t) \sigma_t^{-2} \left( \mathbf{x}_t - \alpha_t \mathbb{E}_{p(\mathbf{x}_0 | \mathbf{x}_t)} [\mathbf{x}_0 | \mathbf{x}_t] \right). \end{aligned}$$

Dividing both sides by  $p_t(\mathbf{x}_t)$ , we obtain:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = -\sigma_t^{-2} \left( \mathbf{x}_t - \alpha_t \mathbb{E}_{p(\mathbf{x}_0 | \mathbf{x}_t)} [\mathbf{x}_0 | \mathbf{x}_t] \right).$$

Rearranging yields:

$$\mathbf{x}_t + \sigma_t^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \alpha_t \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)}[\mathbf{x}_0|\mathbf{x}_t].$$

This completes the derivation.

#### D.2.4 Stein's Identity and Surrogate SURE Objective

**Stein's Identity.** Stein's identity is the integration-by-parts technique that turns expectations under an unknown density into expectations of observable functions and their derivatives, which cancels the partition function and enabling unbiased, tractable objectives and tests without ever evaluating the unknown density or the partition function. We begin with the simplest one-dimensional case and then extend it to the form needed to prove the surrogate loss for SURE.

**1D, Standard Normal Case.** If  $z \sim \mathcal{N}(0, 1)$  and  $f$  has suitable decay, then Stein's identity states:

$$\mathbb{E}[f'(z)] = \mathbb{E}[Zf(z)].$$

Denote  $\phi(z) := \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$ , the one-dimensional standard normal density. The proof follows by integration by parts, using  $\phi'(z) = -z\phi(z)$ , together with the vanishing boundary term. To see this precisely, we compute

$$\mathbb{E}[f'(Z)] = \int_{-\infty}^{\infty} f'(z)\phi(z) dz.$$

By integration by parts, with  $u = f(z)$  and  $dv = \phi'(z) dz$ , we obtain

$$\int f'(z)\phi(z) dz = \left[ f(z)\phi(z) \right]_{-\infty}^{\infty} - \int f(z)\phi'(z) dz.$$

Since  $\phi'(z) = -z\phi(z)$  and  $f(z)\phi(z) \rightarrow 0$  as  $|z| \rightarrow \infty$  (decay condition), the boundary term vanishes and we have

$$\mathbb{E}[f'(z)] = \int f(z)z\phi(z) dz = \mathbb{E}[zf(z)].$$

This completes the derivation and proves the one-dimensional case of Stein's identity.

**Multivariate, Standard Normal Case.** If  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_D)$  and  $g : \mathbb{R}^D \rightarrow \mathbb{R}$ , then Stein's identity is

$$\mathbb{E}[\nabla g(\mathbf{z})] = \mathbb{E}[\mathbf{z}g(\mathbf{z})].$$

Equivalently, for  $\mathbf{u} : \mathbb{R}^D \rightarrow \mathbb{R}^D$ ,

$$\mathbb{E}[\nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{u}(\mathbf{z})] = \mathbb{E}[\mathbf{z}^\top \mathbf{u}(\mathbf{z})]. \quad (\text{D.2.2})$$

**Identity Needed for SURE.** With  $\tilde{\mathbf{x}} = \mathbf{x} + \sigma \mathbf{z}$ , where  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_D)$ , and any vector function  $\mathbf{a}$  of suitable regularity,

$$\mathbb{E}[(\tilde{\mathbf{x}} - \mathbf{x})^\top \mathbf{a}(\tilde{\mathbf{x}}) | \mathbf{x}] = \sigma^2 \mathbb{E}[\nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{a}(\tilde{\mathbf{x}}) | \mathbf{x}]. \quad (\text{D.2.3})$$

This is obtained by applying Equation (D.2.2) and using the chain rule.

**Deriving SURE from the Conditional MSE.** Let  $\mathbf{D} : \mathbb{R}^D \rightarrow \mathbb{R}^D$  be a denoiser and define

$$R(\mathbf{D}; \mathbf{x}) := \mathbb{E} [\|\mathbf{D}(\tilde{\mathbf{x}}) - \mathbf{x}\|_2^2 | \mathbf{x}] .$$

Expand around  $\tilde{\mathbf{x}}$ :

$$\begin{aligned} R(\mathbf{D}; \mathbf{x}) &= \mathbb{E} [\|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|^2 | \mathbf{x}] + 2\mathbb{E} [(\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}})^\top (\tilde{\mathbf{x}} - \mathbf{x}) | \mathbf{x}] + \mathbb{E} [\|\tilde{\mathbf{x}} - \mathbf{x}\|^2 | \mathbf{x}] \\ &= \mathbb{E} [\|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|^2 | \mathbf{x}] + 2 \left( \underbrace{\mathbb{E}[(\tilde{\mathbf{x}} - \mathbf{x})^\top \mathbf{D}(\tilde{\mathbf{x}}) | \mathbf{x}]}_{\sigma^2 \mathbb{E}[\nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{D}(\tilde{\mathbf{x}}) | \mathbf{x}]} - \underbrace{\mathbb{E}[(\tilde{\mathbf{x}} - \mathbf{x})^\top \tilde{\mathbf{x}} | \mathbf{x}]}_{\sigma^2 D} \right) \\ &\quad + \underbrace{\mathbb{E}[\|\tilde{\mathbf{x}} - \mathbf{x}\|^2 | \mathbf{x}]}_{\sigma^2 D} \\ &= \mathbb{E} [\|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|^2 + 2\sigma^2 \nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{D}(\tilde{\mathbf{x}}) - D\sigma^2 | \mathbf{x}] . \end{aligned}$$

Therefore the *observable* surrogate

$$\text{SURE}(\mathbf{D}; \tilde{\mathbf{x}}) := \|\mathbf{D}(\tilde{\mathbf{x}}) - \tilde{\mathbf{x}}\|_2^2 + 2\sigma^2 \nabla_{\tilde{\mathbf{x}}} \cdot \mathbf{D}(\tilde{\mathbf{x}}) - D\sigma^2$$

satisfies  $\mathbb{E} [\text{SURE}(\mathbf{D}; \tilde{\mathbf{x}}) | \mathbf{x}] = R(\mathbf{D}; \mathbf{x})$ . Minimizing SURE (in expectation or empirically) is thus equivalent to minimizing the true conditional MSE while using only noisy observations.

### D.2.5 Theorem 4.1.1: Marginal Alignment via Fokker–Planck

**Proof.**

**Part 1: Fokker–Planck Equation for the Forward SDE.** Consider the forward SDE:

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t) dt + g(t) d\mathbf{w}(t).$$

The diffusion matrix is  $\sigma(t) = g(t)\mathbf{I}_D$ , so  $\sigma(t)\sigma(t)^T = g^2(t)\mathbf{I}_D$ . The Fokker–Planck equation for the marginal density  $p_t(\mathbf{x})$  of  $\mathbf{x}(t)$  is:

$$\partial_t p_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot [\mathbf{f}(\mathbf{x}, t)p_t(\mathbf{x})] + \frac{1}{2} \sum_{i,j=1}^D \frac{\partial^2}{\partial x_i \partial x_j} [(g^2(t)\delta_{ij})p_t(\mathbf{x})].$$

Compute the diffusion term:

$$\sum_{i,j=1}^D \frac{\partial^2}{\partial x_i \partial x_j} [g^2(t) \delta_{ij} p_t(\mathbf{x})] = \sum_{i=1}^D \frac{\partial^2}{\partial x_i^2} [g^2(t) p_t(\mathbf{x})] = g^2(t) \Delta_{\mathbf{x}} p_t(\mathbf{x}).$$

Thus:

$$\partial_t p_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot [\mathbf{f}(\mathbf{x}, t) p_t(\mathbf{x})] + \frac{1}{2} g^2(t) \Delta_{\mathbf{x}} p_t(\mathbf{x}).$$

Now, rewrite using:

$$\tilde{\mathbf{f}}(\mathbf{x}, t) = \mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}).$$

Since  $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) = \frac{\nabla_{\mathbf{x}} p_t(\mathbf{x})}{p_t(\mathbf{x})}$ , compute:

$$\nabla_{\mathbf{x}} \cdot [\tilde{\mathbf{f}}(\mathbf{x}, t) p_t(\mathbf{x})] = \nabla_{\mathbf{x}} \cdot \left[ \mathbf{f}(\mathbf{x}, t) p_t(\mathbf{x}) - \frac{1}{2} g^2(t) \frac{\nabla_{\mathbf{x}} p_t(\mathbf{x})}{p_t(\mathbf{x})} p_t(\mathbf{x}) \right].$$

The second term is:

$$\nabla_{\mathbf{x}} \cdot \left[ -\frac{1}{2} g^2(t) \nabla_{\mathbf{x}} p_t(\mathbf{x}) \right] = -\frac{1}{2} g^2(t) \Delta_{\mathbf{x}} p_t(\mathbf{x}).$$

Thus:

$$\nabla_{\mathbf{x}} \cdot [\tilde{\mathbf{f}}(\mathbf{x}, t) p_t(\mathbf{x})] = \nabla_{\mathbf{x}} \cdot [\mathbf{f}(\mathbf{x}, t) p_t(\mathbf{x})] - \frac{1}{2} g^2(t) \Delta_{\mathbf{x}} p_t(\mathbf{x}).$$

Therefore:

$$\partial_t p_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot [\tilde{\mathbf{f}}(\mathbf{x}, t) p_t(\mathbf{x})],$$

verifying the Fokker-Planck equation in both forms.

**Part 2: PF-ODE Marginal Densities.** Consider the PF-ODE:

$$\frac{d\tilde{\mathbf{x}}(t)}{dt} = \tilde{\mathbf{f}}(\tilde{\mathbf{x}}(t), t), \quad \tilde{\mathbf{f}}(\mathbf{x}, t) = \mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}).$$

**Forward Direction:**  $\tilde{\mathbf{x}}(0) \sim p_0$ . Let  $\tilde{\mathbf{x}}(t)$  follow the PF-ODE with  $\tilde{\mathbf{x}}(0) \sim p_0$ . The flow map  $\Psi_t : \mathbb{R}^D \rightarrow \mathbb{R}^D$  is defined by:

$$\frac{d}{dt} \Psi_t(\mathbf{x}_0) = \tilde{\mathbf{f}}(\Psi_t(\mathbf{x}_0), t), \quad \Psi_0(\mathbf{x}_0) = \mathbf{x}_0.$$

Since  $\tilde{\mathbf{x}}(t) = \Psi_t(\tilde{\mathbf{x}}(0))$ , the density  $\tilde{p}_t(\mathbf{x})$  of  $\tilde{\mathbf{x}}(t)$  satisfies the continuity equation:

$$\partial_t \tilde{p}_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot [\tilde{\mathbf{f}}(\mathbf{x}, t) \tilde{p}_t(\mathbf{x})].$$

Since  $\tilde{\mathbf{x}}(0) \sim p_0$ , we have  $\tilde{p}_0(\mathbf{x}) = p_0(\mathbf{x})$ . From Part 1,  $p_t(\mathbf{x})$  satisfies:

$$\partial_t p_t(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot [\tilde{\mathbf{f}}(\mathbf{x}, t) p_t(\mathbf{x})].$$

Both  $\tilde{p}_t$  and  $p_t$  satisfy the same continuity equation with the same initial condition  $p_0$ . Assuming sufficient smoothness (e.g.,  $\tilde{\mathbf{f}} \in C^1$ ), the solution is unique in some appropriate function space, so  $\tilde{p}_t = p_t$ . Thus,  $\tilde{\mathbf{x}}(t) \sim p_t$  for all  $t \in [0, T]$ .

**Backward Direction:**  $\tilde{\mathbf{x}}(T) \sim p_T$ . Now, let  $\tilde{\mathbf{x}}(t)$  follow the PF-ODE backward from  $t = T$  to  $t = 0$ , with  $\tilde{\mathbf{x}}(T) \sim p_T$ . The ODE is:

$$\frac{d}{dt} \tilde{\mathbf{x}}(t) = \tilde{\mathbf{f}}(\tilde{\mathbf{x}}(t), t).$$

Let  $s = T - t$ , so the backward ODE becomes:

$$\frac{d}{ds} \tilde{\mathbf{x}}(T-s) = -\tilde{\mathbf{f}}(\tilde{\mathbf{x}}(T-s), T-s).$$

The density  $\tilde{p}_{T-s}(\mathbf{x})$  of  $\tilde{\mathbf{x}}(T-s)$  satisfies:

$$\partial_s \tilde{p}_{T-s}(\mathbf{x}) = \nabla_{\mathbf{x}} \cdot [\tilde{\mathbf{f}}(\mathbf{x}, T-s) \tilde{p}_{T-s}(\mathbf{x})].$$

Since  $\tilde{\mathbf{x}}(T) \sim p_T$ , we have  $\tilde{p}_T = p_T$ . The Fokker-Planck equation for  $p_t$  at  $t = T-s$  is:

$$\partial_t p_{T-s}(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot [\tilde{\mathbf{f}}(\mathbf{x}, T-s) p_{T-s}(\mathbf{x})].$$

Since  $\partial_t = -\partial_s$ , we get:

$$\partial_s p_{T-s}(\mathbf{x}) = \nabla_{\mathbf{x}} \cdot [\tilde{\mathbf{f}}(\mathbf{x}, T-s) p_{T-s}(\mathbf{x})].$$

Both  $\tilde{p}_{T-s}$  and  $p_{T-s}$  satisfy the same PDE with the same initial condition at  $s = 0$  ( $\tilde{p}_T = p_T$ ). Uniqueness implies  $\tilde{p}_{T-s} = p_{T-s}$ , so  $\tilde{\mathbf{x}}(t) = \tilde{\mathbf{x}}(T-s) \sim p_{T-s} = p_t$ , for all  $t \in [0, T]$ .

**Part 3: Reverse-Time SDE Marginal Densities.** Consider the reverse-time SDE:

$$d\bar{\mathbf{x}}(t) = [\mathbf{f}(\bar{\mathbf{x}}(t), t) - g^2(t) \nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t))] dt + g(t) d\bar{\mathbf{w}}(t),$$

with  $\bar{\mathbf{x}}(0) \sim p_T$ , where  $\bar{\mathbf{w}}(t)$  is a standard Wiener process in reverse time, defined as  $\bar{\mathbf{w}}(t) = \mathbf{w}(T-t) - \mathbf{w}(T)$ . We need to show  $\bar{\mathbf{x}}(t) \sim p_{T-t}$ .

Rewrite the drift:

$$\mathbf{f}(\mathbf{x}, t) = \tilde{\mathbf{f}}(\mathbf{x}, t) + \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}),$$

so:

$$\mathbf{f}(\mathbf{x}, t) - g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) = \tilde{\mathbf{f}}(\mathbf{x}, t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}).$$

The reverse-time SDE is:

$$d\bar{\mathbf{x}}(t) = \left[ \tilde{\mathbf{f}}(\bar{\mathbf{x}}(t), t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\bar{\mathbf{x}}(t)) \right] dt + g(t) d\bar{\mathbf{w}}(t).$$

Let  $s = T - t$ , so  $\bar{\mathbf{x}}(t) = \bar{\mathbf{x}}(T - s)$ , and  $dt = -ds$ . The SDE becomes:

$$\begin{aligned} d\bar{\mathbf{x}}(T-s) &= \left[ -\tilde{\mathbf{f}}(\bar{\mathbf{x}}(T-s), T-s) + \frac{1}{2}g^2(T-s)\nabla_{\mathbf{x}} \log p_{T-s}(\bar{\mathbf{x}}(T-s)) \right] ds \\ &\quad + g(T-s) d\bar{\mathbf{w}}(T-s). \end{aligned}$$

Since  $\bar{\mathbf{w}}(t) = \mathbf{w}(T-t) - \mathbf{w}(T)$ , we have  $d\bar{\mathbf{w}}(T-s) = -d\mathbf{w}(s)$ , where  $\mathbf{w}(s)$  is a standard Wiener process. Thus, let  $\bar{\mathbf{w}}'(s) = -\mathbf{w}(s)$ , a standard Wiener process, so:

$$d\bar{\mathbf{x}}(s) = \left[ \tilde{\mathbf{f}}(\bar{\mathbf{x}}(s), T-s) - \frac{1}{2}g^2(T-s)\nabla_{\mathbf{x}} \log p_{T-s}(\bar{\mathbf{x}}(s)) \right] ds + g(T-s) d\bar{\mathbf{w}}'(s).$$

The Fokker-Planck equation for the density  $\bar{p}_s(\mathbf{x})$  of  $\bar{\mathbf{x}}(s)$  is:

$$\begin{aligned} \partial_s \bar{p}_s(\mathbf{x}) &= -\nabla_{\mathbf{x}} \cdot \left[ \left( \tilde{\mathbf{f}}(\mathbf{x}, T-s) - \frac{1}{2}g^2(T-s)\nabla_{\mathbf{x}} \log p_{T-s}(\mathbf{x}) \right) \bar{p}_s(\mathbf{x}) \right] \\ &\quad + \frac{1}{2}g^2(T-s)\Delta_{\mathbf{x}} \bar{p}_s(\mathbf{x}). \end{aligned}$$

Assume  $\bar{p}_s = p_{T-s}$ . The Fokker-Planck equation for  $p_{T-s}$  is:

$$\partial_t p_{T-s}(\mathbf{x}) = -\nabla_{\mathbf{x}} \cdot [\tilde{\mathbf{f}}(\mathbf{x}, T-s)p_{T-s}(\mathbf{x})].$$

Since  $\partial_t = -\partial_s$ :

$$\partial_s p_{T-s}(\mathbf{x}) = \nabla_{\mathbf{x}} \cdot [\tilde{\mathbf{f}}(\mathbf{x}, T-s)p_{T-s}(\mathbf{x})].$$

Substitute  $\bar{p}_s = p_{T-s}$ :

$$\begin{aligned} \partial_s p_{T-s} &= -\nabla_{\mathbf{x}} \cdot \left[ \tilde{\mathbf{f}}(\mathbf{x}, T-s)p_{T-s}(\mathbf{x}) - \frac{1}{2}g^2(T-s)\frac{\nabla_{\mathbf{x}} p_{T-s}(\mathbf{x})}{p_{T-s}(\mathbf{x})}p_{T-s}(\mathbf{x}) \right] \\ &\quad + \frac{1}{2}g^2(T-s)\Delta_{\mathbf{x}} p_{T-s}(\mathbf{x}). \end{aligned}$$

The extra term is:

$$\begin{aligned} &-\nabla_{\mathbf{x}} \cdot \left[ -\frac{1}{2}g^2(T-s)\nabla_{\mathbf{x}} p_{T-s}(\mathbf{x}) \right] + \frac{1}{2}g^2(T-s)\Delta_{\mathbf{x}} p_{T-s}(\mathbf{x}) \\ &= \frac{1}{2}g^2(T-s)\Delta_{\mathbf{x}} p_{T-s}(\mathbf{x}) - \frac{1}{2}g^2(T-s)\Delta_{\mathbf{x}} p_{T-s}(\mathbf{x}) = 0. \end{aligned}$$

Thus,  $\bar{p}_s = p_{T-s}$  satisfies the Fokker-Planck equation. Since  $\bar{\mathbf{x}}(0) \sim p_T$ , we have  $\bar{p}_0 = p_T$ , matching the initial condition. Uniqueness (under sufficient smoothness) ensures  $\bar{p}_s = p_{T-s}$ , so  $\bar{\mathbf{x}}(t) = \bar{\mathbf{x}}(T-s) \sim p_{T-t}$ . ■

### D.2.6 Proposition 4.2.1: Minimizer of SM and DSM

**Proof.** To find the minimizer  $\mathbf{s}^*$ , we first consider a fixed time  $t$  and analyze the inner expectation in the objective function:

$$\mathcal{J}(t, \phi) := \mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}} \mathbb{E}_{\mathbf{x}_t \sim p_t(\cdot | \mathbf{x}_0)} \left[ \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t | \mathbf{x}_0)\|_2^2 \right].$$

For this expectation to be minimized, we need to find  $\mathbf{s}_\phi(\mathbf{x}_t, t)$  that minimizes the expected squared error for each  $\mathbf{x}_t$ . We can rewrite this expectation using the joint distribution of  $X_0$  and  $X_t$ :

$$\mathcal{J}(t, \phi) = \iint p_{\text{data}}(\mathbf{x}_0) p_t(\mathbf{x}_t | \mathbf{x}_0) \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t | \mathbf{x}_0)\|_2^2 d\mathbf{x}_0 d\mathbf{x}_t.$$

For each fixed  $\mathbf{x}_t$ , we need to minimize:

$$\int p(\mathbf{x}_0 | X_t = \mathbf{x}_t) p_t(\mathbf{x}_t) \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t | \mathbf{x}_0)\|_2^2 d\mathbf{x}_0.$$

Since  $p_t(\mathbf{x}_t)$  is constant with respect to  $\mathbf{s}_\phi(\mathbf{x}_t, t)$ , this is equivalent to minimizing:

$$\int p(\mathbf{x}_0 | X_t = \mathbf{x}_t) \|\mathbf{s}_\phi(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t | \mathbf{x}_0)\|_2^2 d\mathbf{x}_0$$

This is minimized when  $\mathbf{s}_\phi(\mathbf{x}_t, t)$  equals the conditional expectation:

$$\mathbf{s}^*(\mathbf{x}_t, t) = \mathbb{E}_{X_0 \sim p(X_0 | X_t = \mathbf{x}_t)} [\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t | X_0)].$$

Now we need to prove that this equals  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ . By Bayes' rule and the definition of marginal probability:

$$p_t(\mathbf{x}_t) = \int p_t(\mathbf{x}_t | \mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0.$$

Taking the logarithm and then the gradient with respect to  $\mathbf{x}_t$ :

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \frac{\nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t)}{p_t(\mathbf{x}_t)} = \frac{\nabla_{\mathbf{x}_t} \int p_t(\mathbf{x}_t | \mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0}{\int p_t(\mathbf{x}_t | \mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0}.$$

Under suitable regularity conditions, we can exchange the gradient and integral:

$$\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \frac{\int \nabla_{\mathbf{x}_t} p_t(\mathbf{x}_t | \mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0}{\int p_t(\mathbf{x}_t | \mathbf{x}_0) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0}.$$

■

### D.2.7 Closed-Form Score Function of a Gaussian

For future reference, we summarize the formula for the score of a general multivariate normal distribution as the following lemma:

#### Lemma D.2.1: Score of Gaussian

Let  $\mathbf{x} \in \mathbb{R}^D$  and consider the multivariate normal distribution

$$p(\tilde{\mathbf{x}}|\mathbf{x}) := \mathcal{N}(\tilde{\mathbf{x}}; \boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where  $\boldsymbol{\mu} \in \mathbb{R}^D$  is the mean and  $\boldsymbol{\Sigma} \in \mathbb{R}^{D \times D}$  is an invertible covariance matrix. Its score function is

$$\nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}|\mathbf{x}) = -\boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu}). \quad (\text{D.2.4})$$

#### Proof for Lemma.

The density function of  $p(\tilde{\mathbf{x}}|\mathbf{x})$  is given by:

$$p(\tilde{\mathbf{x}}|\mathbf{x}) = \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\tilde{\mathbf{x}} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu})\right).$$

To compute the score function, we first take the log of  $p(\tilde{\mathbf{x}}|\mathbf{x})$ :

$$\log p(\tilde{\mathbf{x}}|\mathbf{x}) = -\frac{D}{2} \log(2\pi) - \frac{1}{2} \log |\boldsymbol{\Sigma}| - \frac{1}{2}(\tilde{\mathbf{x}} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu}).$$

Now, we compute the gradient of  $\log p(\tilde{\mathbf{x}}|\mathbf{x})$  with respect to  $\tilde{\mathbf{x}}$ :

$$\nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}|\mathbf{x}) = -\frac{1}{2} \nabla_{\tilde{\mathbf{x}}} \left( (\tilde{\mathbf{x}} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu}) \right).$$

Using the chain rule, we get:

$$\nabla_{\tilde{\mathbf{x}}} \left( (\tilde{\mathbf{x}} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu}) \right) = 2\boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu}).$$

Thus, the score function is:

$$\nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}|\mathbf{x}) = -\boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \boldsymbol{\mu}). \quad (\text{D.2.5})$$

## D.3 Flow-Based Perspective

### D.3.1 Lemma 5.1.1: Instantaneous Change of Variables

**Proof.** **Approach 1: Change-of-Variables Formula.** We denote  $p(\mathbf{x}(t), t)$  by  $p_t(\mathbf{x}_t)$ . Starting from the ODE discretization

$$\mathbf{z}_{t+\Delta t} = \mathbf{z}_t + \Delta t \mathbf{F}(\mathbf{z}_t, t),$$

the change-of-variables formula for normalizing flows (Equation (5.1.1)) gives

$$\begin{aligned} \log p_{t+\Delta t}(\mathbf{z}_{t+\Delta t}) &= \log p_t(\mathbf{z}_t) - \log \left| \det (\mathbf{I} + \Delta t \nabla_{\mathbf{z}} \mathbf{F}(\mathbf{z}_t, t)) \right| \\ &= \log p_t(\mathbf{z}_t) - \text{Tr} \left( \log(\mathbf{I} + \Delta t \nabla_{\mathbf{z}} \mathbf{F}(\mathbf{z}_t, t)) \right) \\ &= \log p_t(\mathbf{z}_t) - \Delta t \text{Tr} (\nabla_{\mathbf{z}} \mathbf{F}(\mathbf{z}_t, t)) + \mathcal{O}(\Delta t^2), \end{aligned}$$

where we used  $\log \det = \text{Tr} \log$  and the expansion for small  $\Delta t$ . Taking the limit  $\Delta t \rightarrow 0$  yields the continuous-time differential equation for the log-density. Indeed, the same trick is applied in Equation (5.1.6).

**Approach 2: Continuity Equation.** We can also leverage the continuity equation, which essentially serves as the change-of-variables formula:

$$\partial_t p(\mathbf{z}, t) = -\nabla_{\mathbf{z}} \cdot (\mathbf{F}(\mathbf{z}, t)p(\mathbf{z}, t)).$$

Expanding the divergence,

$$\partial_t p = -((\nabla_{\mathbf{z}} \cdot \mathbf{F})p + \mathbf{F} \cdot \nabla_{\mathbf{z}} p).$$

Along trajectories  $\mathbf{z}(t)$  satisfying  $\frac{d\mathbf{z}}{dt} = \mathbf{F}(\mathbf{z}(t), t)$ , the total time derivative is

$$\begin{aligned} \frac{d}{dt} p(\mathbf{z}(t), t) &= \nabla_{\mathbf{z}} p \cdot \frac{d\mathbf{z}}{dt} + \partial_t p \\ &= \nabla_{\mathbf{z}} p \cdot \mathbf{F} - ((\nabla_{\mathbf{z}} \cdot \mathbf{F})p + \mathbf{F} \cdot \nabla_{\mathbf{z}} p) \\ &= -(\nabla_{\mathbf{z}} \cdot \mathbf{F})p. \end{aligned}$$

Dividing by  $p(\mathbf{z}(t), t)$ , we conclude

$$\frac{d}{dt} \log p(\mathbf{z}(t), t) = -\nabla_{\mathbf{z}} \cdot \mathbf{F}(\mathbf{z}(t), t).$$

■

### D.3.2 Theorem 5.2.2: Mass Conservation Criterion

**Some Prerequisites: Flow Map and Flow-Induced Density.** For any initial position  $\mathbf{x}_0 \in \mathbb{R}^D$ , the flow map  $\Psi_t : \mathbb{R}^D \rightarrow \mathbb{R}^D$  is the unique solution of the ODE

$$\frac{d}{dt}\Psi_t(\mathbf{x}_0) = \mathbf{v}_t(\Psi_t(\mathbf{x}_0)), \quad \Psi_0(\mathbf{x}_0) = \mathbf{x}_0.$$

Under our regularity assumptions,  $\Psi_t$  is continuously differentiable in both  $t$  and  $\mathbf{x}_0$ .

The flow-induced density  $p_t^{\text{fwd}}$  is the pushforward of the initial density  $p_0$  by  $\Psi_t$ :

$$p_t^{\text{fwd}}(\mathbf{x}) = p_0(\Psi_t^{-1}(\mathbf{x})) \left| \det(\nabla \Psi_t^{-1}(\mathbf{x})) \right|.$$

This gives the density at  $\mathbf{x}$  and time  $t$  of particles that started from  $p_0 = p_{\text{data}}$  and evolved under the velocity field  $\mathbf{v}_t$ .

**Informal Proof: Sufficient Condition:**  $p_t^{\text{fwd}} = p_t \Rightarrow$  **Continuity Equation.**

In Section B.1.2 we obtained a *strong solution* of the continuity equation by taking the continuous time limit of the discrete change of variable formula. In that approach, the density  $p_t$  is assumed smooth enough that all derivatives exist classically and the PDE holds pointwise. Here, we offer a complementary derivation in the *weak sense*: the continuity equation is imposed only after integrating against arbitrary smooth test functions, which relaxes the regularity requirements on both  $p_t$  and the velocity field  $\mathbf{v}_t$ . This weak formulation is not only more rigorous since it accommodates less regular solutions but is also the standard framework in PDE theory and numerical analysis.

For any smooth test function  $\varphi(\mathbf{x})$  with compact support, the pushforward property gives:

$$\begin{aligned} \int p_t^{\text{fwd}}(\mathbf{x}) \varphi(\mathbf{x}) d\mathbf{x} &= \int p_0(\Psi_t^{-1}(\mathbf{x})) \left| \det(\nabla \Psi_t^{-1}(\mathbf{x})) \right| \varphi(\mathbf{x}) d\mathbf{x} \\ &= \int p_0(\mathbf{y}) \varphi(\Psi_t(\mathbf{y})) d\mathbf{y}, \end{aligned}$$

where the second equality follows from the change of variables  $\mathbf{x} = \Psi_t(\mathbf{y})$ , with  $d\mathbf{y} = \left| \det(\nabla \Psi_t^{-1}(\mathbf{x})) \right| d\mathbf{x}$ .

Differentiate both sides with respect to  $t$ :

$$\frac{d}{dt} \int p_t^{\text{fwd}}(\mathbf{x}) \varphi(\mathbf{x}) d\mathbf{x} = \frac{d}{dt} \int p_0(\mathbf{y}) \varphi(\Psi_t(\mathbf{y})) d\mathbf{y}.$$

The left-hand side is:

$$\int \frac{\partial p_t^{\text{fwd}}}{\partial t}(\mathbf{x}) \varphi(\mathbf{x}) d\mathbf{x}.$$

On the right-hand side:

$$\int p_0(\mathbf{y}) \nabla \varphi(\Psi_t(\mathbf{y})) \cdot \mathbf{v}_t(\Psi_t(\mathbf{y})) d\mathbf{y},$$

since  $\frac{\partial \Psi_t}{\partial t}(\mathbf{y}) = \mathbf{v}_t(\Psi_t(\mathbf{y}))$ . Change variables to  $\mathbf{x} = \Psi_t(\mathbf{y})$ , so

$$d\mathbf{y} = \left| \det \left( \nabla \Psi_t^{-1}(\mathbf{x}) \right) \right| d\mathbf{x},$$

and

$$p_0(\mathbf{y}) = p_t^{\text{fwd}}(\mathbf{x}) \left| \det \left( \nabla \Psi_t(\mathbf{y}) \right) \right| = \frac{p_t^{\text{fwd}}(\mathbf{x})}{\left| \det \left( \nabla \Psi_t^{-1}(\mathbf{x}) \right) \right|}.$$

Thus, the right-hand side becomes:

$$\int p_t^{\text{fwd}}(\mathbf{x}) \nabla \varphi(\mathbf{x}) \cdot \mathbf{v}_t(\mathbf{x}) d\mathbf{x}.$$

Apply integration by parts, using the compact support of  $\varphi$ :

$$\int p_t^{\text{fwd}}(\mathbf{x}) \nabla \varphi(\mathbf{x}) \cdot \mathbf{v}_t(\mathbf{x}) d\mathbf{x} = - \int \varphi(\mathbf{x}) \nabla \cdot (p_t^{\text{fwd}}(\mathbf{x}) \mathbf{v}_t(\mathbf{x})) d\mathbf{x}.$$

Equating both sides:

$$\int \left[ \frac{\partial p_t^{\text{fwd}}}{\partial t}(\mathbf{x}) + \nabla \cdot (p_t^{\text{fwd}}(\mathbf{x}) \mathbf{v}_t(\mathbf{x})) \right] \varphi(\mathbf{x}) d\mathbf{x} = 0.$$

Since  $\varphi$  is arbitrary, we conclude:

$$\frac{\partial p_t^{\text{fwd}}}{\partial t} + \nabla \cdot (p_t^{\text{fwd}} \mathbf{v}_t) = 0.$$

Given  $p_t^{\text{fwd}} = p_t$ , this implies:

$$\frac{\partial p_t}{\partial t} + \nabla \cdot (p_t \mathbf{v}_t) = 0.$$

**Necessary Condition: Continuity Equation**  $\Rightarrow p_t^{\text{fwd}} = p_t$ . Suppose  $p_t$  satisfies the continuity equation:

$$\frac{\partial p_t}{\partial t} + \nabla \cdot (p_t \mathbf{v}_t) = 0,$$

with the initial condition  $p_0(\mathbf{x}) = p_{\text{data}}(\mathbf{x})$ . We know  $p_t^{\text{fwd}}$  satisfies the same continuity equation, as shown above, and:

$$p_0^{\text{fwd}}(\mathbf{x}) = p_0(\Psi_0^{-1}(\mathbf{x})) \left| \det \left( \nabla \Psi_0^{-1}(\mathbf{x}) \right) \right| = p_0(\mathbf{x}),$$

since  $\Psi_0(\mathbf{x}) = \mathbf{x}$ . Thus, both densities share the same initial condition  $p_0 = p_{\text{data}}$ .

The continuity equation can be rewritten as:

$$\frac{\partial p}{\partial t} + \mathbf{v}_t \cdot \nabla p + p \nabla \cdot \mathbf{v}_t = 0.$$

This is a first-order linear PDE. Assuming  $\mathbf{v}_t$  is continuously differentiable and globally Lipschitz, and  $p_t$  is sufficiently smooth, the method of characteristics guarantees a unique solution in the space of smooth functions. Since  $p_t$  and  $p_t^{\text{fwd}}$  satisfy the same PDE and initial condition, we conclude:

$$p_t(\mathbf{x}) = p_t^{\text{fwd}}(\mathbf{x})$$

for all  $t \in [0, 1]$  and  $\mathbf{x} \in \mathbb{R}^D$ .

This completes the proof of the equivalence. ■

## D.4 Theoretical Supplement: A Unified and Systematic View on Diffusion Models

### D.4.1 Proposition 6.3.1: Equivalence of Parametrizations

**Proof:** As in Theorem 4.2.1 on the DSM loss, the global optimum of the matching loss

$$\mathbb{E}_t \left[ \omega(t) \mathbb{E}_{\mathbf{x}_0, \epsilon} \left[ \|\cdot\|_2^2 \right] \right]$$

is attained when the inner expectation

$$\mathbb{E}_{\mathbf{x}_0, \epsilon} \left[ \|\cdot\|_2^2 \right]$$

is minimized for each fixed  $t$ . Since this is a standard mean squared error problem, the minimizer is unique. From denoising score matching (Vincent, 2011), Theorem 4.2.1 shows the optimal score function satisfies

$$\mathbf{s}^*(\mathbf{x}_t, t) = \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)} [\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t|\mathbf{x}_0)] = \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t).$$

Using the identity  $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t|\mathbf{x}_0) = -\frac{1}{\sigma_t} \boldsymbol{\epsilon}$  for  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , we obtain

$$\mathbf{s}^*(\mathbf{x}_t, t) = -\frac{1}{\sigma_t} \boldsymbol{\epsilon}^*(\mathbf{x}_t, t),$$

where  $\boldsymbol{\epsilon}^*(\mathbf{x}_t, t) = \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)} [\boldsymbol{\epsilon}|\mathbf{x}_t]$  is the optimal  $\boldsymbol{\epsilon}$ -prediction for  $\mathcal{L}_{\text{noise}}(\boldsymbol{\phi})$ . For the  $\mathbf{x}$ -prediction loss  $\mathcal{L}_{\text{clean}}$ , the optimal estimator is

$$\mathbf{x}^*(\mathbf{x}_t, t) = \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)} [\mathbf{x}_0|\mathbf{x}_t],$$

which, by Tweedie's formula, relates to the score via

$$\alpha_t \mathbf{x}^*(\mathbf{x}_t, t) = \mathbf{x}_t + \sigma_t^2 \mathbf{s}^*(\mathbf{x}_t, t).$$

For the  $\mathbf{v}$ -prediction loss  $\mathcal{L}_{\text{velocity}}$ , the optimal estimator is

$$\begin{aligned} \mathbf{v}^*(\mathbf{x}_t, t) &= \mathbb{E}_{p(\mathbf{x}_0|\mathbf{x}_t)} [\alpha'_t \mathbf{x}_0 + \sigma'_t \boldsymbol{\epsilon}|\mathbf{x}_t] \\ &= \alpha'_t \mathbf{x}^* + \sigma'_t \boldsymbol{\epsilon}^*. \end{aligned}$$

Substituting the expressions for  $\mathbf{x}^*$  and  $\boldsymbol{\epsilon}^*$  in terms of  $\mathbf{s}^*$  yields

$$\begin{aligned} \mathbf{v}^*(\mathbf{x}_t, t) &= \frac{\alpha'_t}{\alpha_t} \mathbf{x}_t + \left( \frac{\alpha'_t}{\alpha_t} \sigma_t^2 - \sigma_t \sigma'_t \right) \mathbf{s}^*(\mathbf{x}_t, t) \\ &= f(t) \mathbf{x}_t - \frac{1}{2} g^2(t) \mathbf{s}^*(\mathbf{x}_t, t), \end{aligned}$$

which completes the derivation. ■

## D.5 Theoretical Supplement: Learning Fast Diffusion-Based Generators

### D.5.1 Knowledge Distillation Loss as an Instance of the General Framework Equation (10.1.4)

We begin with the oracle KD loss

$$\mathcal{L}_{\text{KD}}^{\text{oracle}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_T \sim p_T} \|\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{x}_T, T, 0) - \Psi_{T \rightarrow 0}(\mathbf{x}_T)\|_2^2,$$

with  $p_T = p_{\text{prior}}$ . For the deterministic ODE flow map  $\Psi$  (semigroup, bijective along the trajectory), the marginals satisfy the pushforward identity

$$p_t = \Psi_{0 \rightarrow t} \# p_{\text{data}} = \Psi_{T \rightarrow t} \# p_{\text{prior}};$$

hence,

$$\Psi_{s \rightarrow T} \# p_s = p_T \quad \text{and} \quad \Psi_{T \rightarrow 0} \circ \Psi_{s \rightarrow T} = \Psi_{s \rightarrow 0}.$$

Changing variables  $\mathbf{x}_T = \Psi_{s \rightarrow T}(\mathbf{x}_s)$  with  $\mathbf{x}_s \sim p_s$  gives

$$\mathcal{L}_{\text{KD}}^{\text{oracle}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_s \sim p_s} \left\| \mathbf{G}_{\boldsymbol{\theta}}(\Psi_{s \rightarrow T}(\mathbf{x}_s), T, 0) - \Psi_{s \rightarrow 0}(\mathbf{x}_s) \right\|_2^2.$$

Define the pulled-back student  $\tilde{\mathbf{G}}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, 0) := \mathbf{G}_{\boldsymbol{\theta}}(\Psi_{s \rightarrow T}(\mathbf{x}_s), T, 0)$  to express the same loss in the unified flow-map form (at  $t = 0$ ):

$$\mathcal{L}_{\text{KD}}^{\text{oracle}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}_s \sim p_s} \left\| \tilde{\mathbf{G}}_{\boldsymbol{\theta}}(\mathbf{x}_s, s, 0) - \Psi_{s \rightarrow 0}(\mathbf{x}_s) \right\|_2^2.$$

This derivation relies on change of variables through the oracle flow and the semigroup property. ■

### D.5.2 Parameter–Flow Interpretation to Proposition 10.2.1

From the derivation of Proposition 10.2.1, we can interpret the gradient of VSD as a parameter-induced transport flow, where adjusting the model parameters moves particles in data space to align their motion with the score mismatch between the student and teacher distributions.

Let  $t \sim p(t)$ ,  $\mathbf{z} \sim p(\mathbf{z})$ ,  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  and

$$\hat{\mathbf{x}}_t = \alpha_t \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z}) + \sigma_t \boldsymbol{\epsilon}.$$

Define the *sample (particle) velocity*

$$\hat{\mathbf{v}}_{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t) := \partial_{\boldsymbol{\theta}} \hat{\mathbf{x}}_t = \alpha_t \partial_{\boldsymbol{\theta}} \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{z}),$$

and the *velocity field* in  $\mathbf{x}$ -space as the conditional average

$$\mathbf{v}_{\boldsymbol{\theta}}(\mathbf{x}) := \mathbb{E}[\hat{\mathbf{v}}_{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t) | \hat{\mathbf{x}}_t = \mathbf{x}].$$

With this definition, at each fixed  $t$  the density obeys the parameter–flow continuity equation

$$\partial_{\theta} p_t^{\theta}(\mathbf{x}) + \nabla_{\mathbf{x}} \cdot (p_t^{\theta}(\mathbf{x}) \mathbf{v}_{\theta}(\mathbf{x})) = 0.$$

Here  $\mathbf{v}_{\theta}(\hat{\mathbf{x}}_t) = \partial_{\theta} \hat{\mathbf{x}}_t$  is the *parameter-induced particle velocity* in data space (with  $t$  fixed). Equivalently, at each fixed  $t$  the density satisfies the continuity equation in  $\theta$ :

$$\partial_{\theta} p_t^{\theta}(\mathbf{x}) + \nabla_{\mathbf{x}} \cdot (p_t^{\theta}(\mathbf{x}) \mathbf{v}_{\theta}(\mathbf{x})) = 0.$$

Thus the gradient of  $\mathcal{L}_{\text{VSD}}$  takes a transport form,

$$\nabla_{\theta} \mathcal{L}_{\text{VSD}} = \mathbb{E}[\omega(t) \langle \underbrace{\nabla_{\mathbf{x}} \log p_t^{\theta} - \nabla_{\mathbf{x}} \log p_t}_{\text{score mismatch at fixed } t} \underbrace{\mathbf{v}_{\theta}}_{\text{parameter–flow velocity}} \rangle],$$

which says: adjust the parameter–flow so that particle motion aligns with the local score mismatch, reducing the divergence along the trajectory.

### D.5.3 Theorem 11.2.1: CM Equals CT up to $\mathcal{O}(\Delta s)$

**Proof:** Step 1: DDIM Update (with Oracle Score) Is the Conditional Mean.

$$\begin{aligned}\hat{\mathbf{x}}_{s'} &:= \frac{\alpha_{s'}}{\alpha_s} \mathbf{x}_s + \sigma_s^2 \left( \frac{\alpha_{s'}}{\alpha_s} - \frac{\sigma_{s'}}{\sigma_s} \right) \nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s) \\ &= \frac{\alpha_{s'}}{\alpha_s} (\mathbf{x}_s + \sigma_s^2 \nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s)) - \sigma_{s'} \sigma_s \nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s) \\ &= \alpha_{s'} \mathbb{E}[\mathbf{x}_0 | \mathbf{x}_s] + \sigma_{s'} \left( \frac{\mathbf{x}_s - \alpha_s \mathbb{E}[\mathbf{x}_0 | \mathbf{x}_s]}{\sigma_s} \right) \\ &= \alpha_{s'} \mathbb{E}[\mathbf{x}_0 | \mathbf{x}_s] + \sigma_{s'} \mathbb{E}[\boldsymbol{\epsilon} | \mathbf{x}_s] \\ &= \mathbb{E}[\alpha_{s'} \mathbf{x}_0 + \sigma_{s'} \boldsymbol{\epsilon} | \mathbf{x}_s] \\ &= \mathbb{E}[\mathbf{x}_{s'} | \mathbf{x}_s].\end{aligned}$$

Here, we use the Tweedie's formula  $\mathbb{E}[\mathbf{x}_0 | \mathbf{x}_s] = \frac{\mathbf{x}_s + \sigma_s^2 \nabla_{\mathbf{x}_s} \log p_s(\mathbf{x}_s)}{\alpha_s}$  and  $\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sigma_s \boldsymbol{\epsilon}$  in the third and fourth equalities.

**Step 2: Expand CT Around the Conditional Mean  $\hat{\mathbf{x}}_{s'}$ .**

$$\begin{aligned}
\mathcal{L}_{\text{CT}} &= \mathbb{E}_{s, \mathbf{x}_s} \mathbb{E}_{\mathbf{x}_{s'} | \mathbf{x}_s} \left[ w(s) d(\mathbf{f}_{\theta}(\mathbf{x}_s, s), \mathbf{f}_{\theta^-}(\mathbf{x}_{s'}, s')) \right] \\
&\stackrel{(1)}{=} \mathbb{E}_{s, \mathbf{x}_s} \mathbb{E}_{\mathbf{x}_{s'} | \mathbf{x}_s} \left[ w(s) d(\mathbf{f}_{\theta}(\mathbf{x}_s, s), \mathbf{f}_{\theta^-}(\hat{\mathbf{x}}_{s'}, s')) \right. \\
&\quad + w(s) \partial_2 d(\mathbf{f}_{\theta}(\mathbf{x}_s, s), \mathbf{f}_{\theta^-}(\hat{\mathbf{x}}_{s'}, s')) [\partial_1 \mathbf{f}_{\theta^-}(\hat{\mathbf{x}}_{s'}, s') (\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'})] \\
&\quad \left. + w(s) \mathcal{O}(\|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2) \right] \\
&\stackrel{(2)}{=} \mathbb{E}_{s, \mathbf{x}_s} \left[ w(s) d(\mathbf{f}_{\theta}(\mathbf{x}_s, s), \mathbf{f}_{\theta^-}(\hat{\mathbf{x}}_{s'}, s')) \right] \\
&\quad + \mathbb{E}_{s, \mathbf{x}_s} \left[ w(s) \partial_2 d(\mathbf{f}_{\theta}(\mathbf{x}_s, s), \mathbf{f}_{\theta^-}(\hat{\mathbf{x}}_{s'}, s')) [\partial_1 \mathbf{f}_{\theta^-}(\hat{\mathbf{x}}_{s'}, s') \mathbb{E}_{\mathbf{x}_{s'} | \mathbf{x}_s} (\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'})] \right] \\
&\quad + \mathbb{E}_{s, \mathbf{x}_s} \mathbb{E}_{\mathbf{x}_{s'} | \mathbf{x}_s} \left[ w(s) \mathcal{O}(\|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2) \right] \\
&\stackrel{(3)}{=} \mathbb{E}_{s, \mathbf{x}_s} \left[ w(s) d(\mathbf{f}_{\theta}(\mathbf{x}_s, s), \mathbf{f}_{\theta^-}(\hat{\mathbf{x}}_{s'}, s')) \right] + \mathbb{E}_{s, \mathbf{x}_s} \mathbb{E}_{\mathbf{x}_{s'} | \mathbf{x}_s} \left[ w(s) \mathcal{O}(\|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2) \right] \\
&= \mathbb{E}_{s, \mathbf{x}_s} \left[ w(s) d(\mathbf{f}_{\theta}(\mathbf{x}_s, s), \mathbf{f}_{\theta^-}(\hat{\mathbf{x}}_{s'}, s')) \right] + \mathcal{O}(\mathbb{E}_{s, \mathbf{x}_s} \mathbb{E}_{\mathbf{x}_{s'} | \mathbf{x}_s} \|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2) \\
&= \mathcal{L}_{\text{CM}} + \mathcal{O}(\mathbb{E}_{s, \mathbf{x}_s} \mathbb{E}_{\mathbf{x}_{s'} | \mathbf{x}_s} \|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2) \\
&\stackrel{(4)}{=} \mathcal{L}_{\text{CM}} + \mathcal{O}(\Delta s)
\end{aligned}$$

Here, (1) applies a second-order Taylor expansion of

$$h(\mathbf{x}') := d(\mathbf{f}_{\theta}(\mathbf{x}_s, s), \mathbf{f}_{\theta^-}(\mathbf{x}', s'))$$

around  $\hat{\mathbf{x}}_{s'}$  in its second argument. (2) applies the tower property

$$\mathbb{E}_{s, \mathbf{x}_s} \mathbb{E}_{\mathbf{x}_{s'} | \mathbf{x}_s} [\cdot] = \mathbb{E}_{s, \mathbf{x}_s} [\cdot]$$

and notes that, inside  $\mathbb{E}_{\mathbf{x}_{s'} | \mathbf{x}_s}$ , the only  $\mathbf{x}_{s'}$ -dependence is through  $(\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'})$ , so all other factors are treated as constants and the inner expectations reduce to  $\mathbb{E}_{\mathbf{x}_{s'} | \mathbf{x}_s} (\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'})$  and  $\mathbb{E}_{\mathbf{x}_{s'} | \mathbf{x}_s} \|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2$ . (3) uses  $\mathbb{E}[\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'} | \mathbf{x}_s] = \mathbf{0}$  because  $\hat{\mathbf{x}}_{s'} = \mathbb{E}[\mathbf{x}_{s'} | \mathbf{x}_s]$ . (4) uses the linear-Gaussian scheduler, which gives

$$\mathbb{E}[\|\mathbf{x}_{s'} - \hat{\mathbf{x}}_{s'}\|^2 | \mathbf{x}_s] = \mathcal{O}(\Delta s^2),$$

thus leading to a total remainder of  $\mathcal{O}(\Delta s)$ . ■

#### D.5.4 Proposition 11.3.1: Continuous-Time Limit of the CT Gradient

**Proof:** We simplify the notation by proving for Equation (11.3.2):

$$\mathcal{L}_{\text{CM}}^{\Delta s}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) := \mathbb{E} \left[ \omega(s) \|\mathbf{f}_{\theta}(\mathbf{x}_s, s) - \mathbf{f}_{\theta^-}(\Psi_{s \rightarrow s - \Delta s}(\mathbf{x}_s), s - \Delta s)\|_2^2 \right].$$

For notational simplicity, we write  $\tilde{\mathbf{x}}_{s-\Delta s} := \Psi_{s \rightarrow s-\Delta s}(\mathbf{x}_s)$

Expanding the MSE loss, we will have

$$\begin{aligned} & \| \mathbf{f}_{\theta}(\mathbf{x}_s, s) - \mathbf{f}_{\theta^-}(\tilde{\mathbf{x}}_{s-\Delta s}, s - \Delta s) \|_2^2 \\ &= \| \underbrace{(\mathbf{f}_{\theta}(\mathbf{x}_s, s) - \mathbf{f}_{\theta^-}(\mathbf{x}_s, s)) + (\mathbf{f}_{\theta^-}(\mathbf{x}_s, s) - \mathbf{f}_{\theta^-}(\tilde{\mathbf{x}}_{s-\Delta s}, s - \Delta s))}_{=: \delta \mathbf{f}} \|_2^2 \\ &= \| \delta \mathbf{f} \|_2^2 + 2\delta \mathbf{f}^\top \cdot \frac{d}{ds} \mathbf{f}_{\theta^-}(\mathbf{x}_s, s) \Delta s + \left\| \frac{d}{ds} \mathbf{f}_{\theta^-}(\mathbf{x}_s, s) \right\|_2^2 |\Delta s|^2 + \mathcal{O}(|\Delta s|^3). \end{aligned}$$

Here, we apply a Taylor expansion at  $(\mathbf{x}_s, s)$ :

$$\mathbf{f}_{\theta^-}(\mathbf{x}_s, s) - \mathbf{f}_{\theta^-}(\tilde{\mathbf{x}}_{s-\Delta s}, s - \Delta s) = \frac{d}{ds} \mathbf{f}_{\theta^-}(\mathbf{x}_s, s) \Delta s + \mathcal{O}(|\Delta s|^2),$$

together with the first-order expansion of the oracle flow map,

$$\mathbf{x}_s - \Psi_{s \rightarrow s-\Delta s}(\mathbf{x}_s) = \mathbf{v}^*(\mathbf{x}_s, s) \Delta s + \mathcal{O}(|\Delta s|^2),$$

and the total differentiation identity,

$$\frac{d}{ds} \mathbf{f}_{\theta^-}(\mathbf{x}_s, s) = \partial_s \mathbf{f}_{\theta^-}(\mathbf{x}_s, s) + (\partial_{\mathbf{x}} \mathbf{f}_{\theta^-}(\mathbf{x}_s, s)) \mathbf{v}^*(\mathbf{x}_s, s),$$

to simplify the expression.

Since  $\theta^-$  is treated as a constant and just the same copy as  $\theta$  (i.e., no gradient through it), the gradient of  $\mathcal{L}_{\text{CM}}^{\Delta s}(\theta, \theta^-)$  with respect to  $\theta$  is:

$$\nabla_{\theta} \mathcal{L}_{\text{CM}}^{\Delta s}(\theta, \theta^-) = 2\mathbb{E} \left[ \omega(s) \nabla_{\theta} \mathbf{f}_{\theta}(\mathbf{x}_s, s)^\top \cdot \frac{d}{ds} \mathbf{f}_{\theta^-}(\mathbf{x}_s, s) \right] \Delta s + \mathcal{O}(|\Delta s|^2).$$

Dividing by  $\Delta s$  and taking the limit yields:

$$\lim_{\Delta s \rightarrow 0} \frac{1}{\Delta s} \nabla_{\theta} \mathcal{L}_{\text{CM}}^{\Delta s}(\theta, \theta^-) = \nabla_{\theta} \mathbb{E} \left[ 2\omega(t) \mathbf{f}_{\theta}^\top(\mathbf{x}_s, s) \cdot \frac{d}{ds} \mathbf{f}_{\theta^-}(\mathbf{x}_s, s) \right].$$

This proves the identity. ■

## D.6 (Optional) Elucidating Diffusion Model (EDM)

We introduce specific criteria for neural network parameterization design in the  $\mathbf{x}$ -prediction model, as proposed in Elucidating Diffusion Models (EDM) (Karras *et al.*, 2022). EDM provides a simple recipe that makes the training process easier to optimize and more reliable. The  $\mathbf{x}$ -prediction model is expressed as a time dependent skip connection combined with a scaled residual (Equation (D.6.1)). The central idea is to normalize both the inputs and the regression targets to unit variance at all times, and to adjust the skip path so that residual errors are not amplified as time evolves. This recipe has become a widely adopted default in diffusion model implementations and extends naturally to flow map learning, especially the family of Consistency Models.

### D.6.1 Criteria for Neural Network $\mathbf{x}_\phi$ Design

EDM considers a parametrization of the  $\mathbf{x}$ -prediction model, denoted with a slight abuse of notation as  $\mathbf{x}_\phi(\mathbf{x}, t)$ <sup>1</sup>, in the following form:

$$\mathbf{x}_\phi(\mathbf{x}, t) := c_{\text{skip}}(t)\mathbf{x} + c_{\text{out}}(t)\mathbf{F}_\phi(c_{\text{in}}(t)\mathbf{x}, c_{\text{noise}}(t)). \quad (\text{D.6.1})$$

Here,  $c_{\text{skip}}(t)$ ,  $c_{\text{out}}(t)$ ,  $c_{\text{in}}(t)$ , and  $c_{\text{noise}}(t)$  are time-dependent functions. They are chosen to enhance stability and performance during training, based on practical considerations that will be introduced shortly.

Plugging this in Equation (6.2.5), then the objective function becomes after straightforward algebraic manipulation<sup>2</sup>:

$$\mathbb{E}_{\mathbf{x}_0, \epsilon, t} \left[ \omega(t) c_{\text{out}}^2(t) \|\mathbf{F}_\phi(c_{\text{in}}(t)\mathbf{x}_t, c_{\text{noise}}(t)) - \mathbf{x}_{\text{tgt}}(t)\|_2^2 \right]. \quad (\text{D.6.2})$$

<sup>1</sup>As discussed, all four prediction types are equivalent and can be reduced to the  $\mathbf{x}$ -prediction case. EDM adopts this formulation, which is both well-studied and naturally aligned with the goal of generating clean samples of flow map models.

<sup>2</sup>We start from the  $\mathbf{x}$ -prediction diffusion loss by substituting the parameterization given in Equation (D.6.1):

$$\begin{aligned} & \mathbb{E}_{\mathbf{x}_0, \epsilon, t} [\omega(t) \|\mathbf{x}_\phi(\mathbf{x}_t, t) - \mathbf{x}_0\|_2^2] \\ &= \mathbb{E}_{\mathbf{x}_0, \epsilon, t} \left[ \omega(t) \left\| c_{\text{skip}}(t) \underbrace{(\alpha_t \mathbf{x}_0 + \sigma_t \epsilon)}_{\mathbf{x}_t} + c_{\text{out}}(t) \mathbf{F}_\phi(c_{\text{in}}(t)\mathbf{x}_t, c_{\text{noise}}(t)) - \mathbf{x}_0 \right\|_2^2 \right] \\ &= \mathbb{E}_{\mathbf{x}_0, \epsilon, t} \left[ \omega(t) c_{\text{out}}^2(t) \left\| \mathbf{F}_\phi(c_{\text{in}}(t)\mathbf{x}, c_{\text{noise}}(t)) - \underbrace{\left( \frac{(1 - c_{\text{skip}}(t)\alpha_t) \mathbf{x}_0 - c_{\text{skip}}(t)\sigma_t \epsilon}{c_{\text{out}}(t)} \right)}_{\mathbf{x}_{\text{tgt}}(t)} \right\|_2^2 \right] \\ &= \text{Equation (D.6.1)}. \end{aligned}$$

Here, the regression target  $\mathbf{x}_{\text{tgt}}(t)$  is obtained as:

$$\mathbf{x}_{\text{tgt}}(t) = \frac{(1 - c_{\text{skip}}(t)\alpha_t) \mathbf{x}_0 - c_{\text{skip}}(t)\sigma_t \boldsymbol{\epsilon}}{c_{\text{out}}(t)}.$$

By incorporating the standard deviation of  $p_{\text{data}}$ , denoted as  $\sigma_d$ , EDM proposes the following design criterion for network parameterization, which may be described as the *unit variance criterion*.

### Unit Variance of Input.

$$\begin{aligned} \text{Var}_{\mathbf{x}_0, \boldsymbol{\epsilon}} [c_{\text{in}}(t)\mathbf{x}_t] &= 1 \\ \iff c_{\text{in}}^2(t) \text{Var}_{\mathbf{x}_0, \boldsymbol{\epsilon}} [\alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}] &= 1 \\ \iff c_{\text{in}}(t) &= \frac{1}{\sqrt{\sigma_d^2 \alpha_t^2 + \sigma_t^2}}, \end{aligned}$$

taking the positive root.

### Unit Variance of Training Target.

$$\begin{aligned} \text{Var}_{\mathbf{x}_0, \boldsymbol{\epsilon}} [\mathbf{x}_{\text{tgt}}(t)] &= 1 \\ \iff c_{\text{out}}^2(t) &= (1 - c_{\text{skip}}(t)\alpha_t)^2 \sigma_d^2 + c_{\text{skip}}^2(t)\sigma_t^2, \end{aligned} \quad (\text{D.6.3})$$

with centered data ( $\mathbb{E}[\mathbf{x}_0] = \mathbf{0}$ ).

**Minimize the Error Amplification from  $\mathbf{F}_\phi$  to  $\mathbf{x}_\phi$ .** EDM aims to mitigate the amplification of the network's learning error from  $\mathbf{F}_\phi$  to  $\mathbf{x}_\phi$ . This is achieved by selecting  $c_{\text{skip}}$  to minimize  $c_{\text{out}}$ :

$$c_{\text{skip}}^* \in \arg \min_{c_{\text{skip}}} c_{\text{out}}^2.$$

Using the standard approach of solving  $\frac{\partial c_{\text{out}}}{\partial c_{\text{skip}}} = 0$  for the critical point  $c_{\text{skip}}^*$ , we obtain

$$c_{\text{skip}}^*(t) = \frac{\alpha_t \sigma_d^2}{\alpha_t^2 \sigma_d^2 + \sigma_t^2}.$$

Substituting this into Equation (D.6.3), the optimal value is given by

$$c_{\text{out}}^*(t) = \pm \frac{\sigma_t \sigma_d}{\sqrt{\alpha_t^2 \sigma_d^2 + \sigma_t^2}}.$$

By convention, we use the nonnegative branch for the output scale:

$$c_{\text{out}}^*(t) = \frac{\sigma_t \sigma_d}{\sqrt{\alpha_t^2 \sigma_d^2 + \sigma_t^2}} \quad (\geq 0),$$

which ensures  $c_{\text{out}}(0) = 0$  and  $c_{\text{out}}(t) \rightarrow \sigma_d$  as  $\sigma_t$  is sufficiently large, yielding the intuitive limits  $\mathbf{x}_\phi(\mathbf{x}_t, 0) \approx \mathbf{x}_t$  and  $\mathbf{x}_\phi(\mathbf{x}_t, t) \approx \sigma_d \mathbf{F}_\phi(\cdot)$ .

We summarize these coefficients as follows:

Denoting  $R_t := \alpha_t^2 \sigma_d^2 + \sigma_t^2$ , we have the following selections:

$$c_{\text{in}}(t) = \frac{1}{\sqrt{R_t}}, \quad c_{\text{skip}}(t) = \frac{\alpha_t \sigma_d^2}{R_t}, \quad c_{\text{out}}(t) = \frac{\sigma_t \sigma_d}{\sqrt{R_t}}. \quad (\text{D.6.4})$$

With Equation (D.6.4), the regression target  $\mathbf{x}_{\text{tgt}}(t)$  simplifies to

$$\mathbf{x}_{\text{tgt}}(t) = \frac{1}{\sigma_d} \frac{\sigma_t \mathbf{x}_0 - \alpha_t \sigma_d^2 \boldsymbol{\epsilon}}{\sqrt{R_t}}.$$

Additionally, substituting these expressions into Equation (D.6.1) and Equation (D.6.2) allows us to simplify both the parametrization and the loss function, yielding:

$$\mathbf{x}_\phi(\mathbf{x}, t) = \frac{\alpha_t \sigma_d^2}{R_t} \mathbf{x} + \frac{\sigma_t \sigma_d}{\sqrt{R_t}} \mathbf{F}_\phi \left( \frac{1}{\sqrt{R_t}} \mathbf{x}, c_{\text{noise}}(t) \right),$$

and

$$\mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}, t} \left[ \omega(t) \frac{\sigma_t^2}{R_t} \left\| \sigma_d \mathbf{F}_\phi \left( \frac{1}{\sqrt{R_t}} \mathbf{x}_t, c_{\text{noise}}(t) \right) - \left( \frac{\sigma_t \mathbf{x}_0 - \alpha_t \sigma_d^2 \boldsymbol{\epsilon}}{\sqrt{R_t}} \right) \right\|_2^2 \right].$$

With this parameterization and the conditions  $\alpha_0 \approx 1$  and  $\sigma_0 \approx 0$ , we observe that

$$c_{\text{skip}}(0) \approx 1 \quad \text{and} \quad c_{\text{out}}(0) \approx 0.$$

**Selection of  $c_{\text{noise}}(t)$ .** It provides a noise-level embedding to  $\mathbf{F}_\phi$ ; any one-to-one mapping of the noise level  $\sigma_t$  (e.g.,  $c_{\text{noise}}(t) = \log \sigma_t$ ) is suitable.

## D.6.2 A Common EDM Special Case: $\alpha_t = 1$ , $\sigma_t = t$

We consider the simplified noise schedule used in EDM, where  $\alpha_t = 1$  and  $\sigma_t = t$ , which also appears in the discussion of CTM in Section 11.4. Under this setting, the forward process becomes

$$\mathbf{x}_t = \mathbf{x}_0 + t \boldsymbol{\epsilon}, \quad \text{with } \mathbf{x}_0 \sim p_{\text{data}}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

corresponding to the perturbation kernel

$$p_t(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \mathbf{x}_0, t^2 \mathbf{I}).$$

Accordingly, the prior distribution at the terminal time is set as

$$p_{\text{prior}}(\mathbf{x}_T) := \mathcal{N}(\mathbf{x}_T; \mathbf{0}, T^2 \mathbf{I}).$$

The marginal density induced by the perturbation kernel is given by the convolution:

$$p_t(\mathbf{x}) = \int \mathcal{N}(\cdot; \mathbf{0}, t^2 \mathbf{I}) p_{\text{data}}(\mathbf{x}_0) d\mathbf{x}_0.$$

Under this setup, the PF-ODE based on  $\mathbf{x}$ -prediction  $\mathbf{x}_{\phi^\times}$  (see Equation (6.3.2)) simplifies to

$$\frac{d\mathbf{x}(t)}{dt} = \frac{\mathbf{x}(t) - \mathbf{x}_{\phi^\times}(\mathbf{x}(t), t)}{t}.$$

Substituting this formulation into Equation (D.6.4), the neural network parameterization coefficients become

$$c_{\text{in}}(t) = \frac{1}{\sqrt{\sigma_d^2 + t^2}}, \quad c_{\text{skip}}(t) = \frac{\sigma_d^2}{\sigma_d^2 + t^2}, \quad c_{\text{out}}(t) = \pm \frac{t\sigma_d}{\sqrt{\sigma_d^2 + t^2}}. \quad (\text{D.6.5})$$

From these expressions, we observe:

- When  $t \approx 0$ , the noise level is negligible, so  $c_{\text{skip}} \approx 1$  and  $c_{\text{out}} \approx 0$ . In this limit, the skip path dominates and the network essentially passes through its input,

$$\mathbf{x}_\phi(\mathbf{x}, t) \approx \mathbf{x}.$$

- When  $t \gg 0$ , the input is heavily corrupted by noise, so  $c_{\text{skip}} \approx 0$  and  $c_{\text{out}} \approx \sigma_d$ . In this regime, the skip path vanishes and the model output is determined entirely by the learned residual,

$$\mathbf{x}_\phi(\mathbf{x}, t) \approx \sigma_d \mathbf{F}_\phi(c_{\text{in}}(t)\mathbf{x}, c_{\text{noise}}(t)),$$

meaning that the network  $\mathbf{F}_\phi$  predicts a scaled proxy of the clean signal from a normalized noisy input; at high noise levels the model output is therefore determined entirely by the learned denoising function.

In short, the parameterization smoothly interpolates from an identity map at small  $t$  to a scaled residual predictor on standardized inputs at large  $t$ .

## References

---

- Ackley, D. H., G. E. Hinton, and T. J. Sejnowski. (1985). “A learning algorithm for Boltzmann machines”. *Cognitive science*. 9(1): 147–169.
- Albergo, M. S., N. M. Boffi, and E. Vanden-Eijnden. (2023). “Stochastic interpolants: A unifying framework for flows and diffusions”. *arXiv preprint arXiv:2303.08797*.
- Albergo, M. S. and E. Vanden-Eijnden. (2023). “Building Normalizing Flows with Stochastic Interpolants”. In: *The Eleventh International Conference on Learning Representations*.
- Altschuler, J., J. Niles-Weed, and P. Rigollet. (2017). “Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration”. *Advances in neural information processing systems*. 30.
- Anderson, B. D. (1982). “Reverse-time diffusion equation models”. *Stochastic Processes and their Applications*. 12(3): 313–326.
- Atkinson, K., W. Han, and D. E. Stewart. (2009). *Numerical solution of ordinary differential equations*. Vol. 81. John Wiley & Sons.
- Bansal, A., H.-M. Chu, A. Schwarzschild, S. Sengupta, M. Goldblum, J. Geiping, and T. Goldstein. (2023). “Universal guidance for diffusion models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 843–852.
- Behrmann, J., W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen. (2019). “Invertible residual networks”. In: *International conference on machine learning*. PMLR. 573–582.
- Benamou, J.-D. and Y. Brenier. (2000). “A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem”. *Numerische Mathematik*. 84(3): 375–393.

- Boffi, N. M., M. S. Albergo, and E. Vanden-Eijnden. (2024). “Flow Map Matching”. *arXiv preprint arXiv:2406.07507*.
- Bradley, R. A. and M. E. Terry. (1952). “Rank analysis of incomplete block designs: I. the method of paired comparisons”. *Biometrika*. 39(3/4): 324–345.
- Caluya, K. F. and A. Halder. (2021). “Wasserstein proximal algorithms for the Schrödinger bridge problem: Density control with nonlinear drift”. *IEEE Transactions on Automatic Control*. 67(3): 1163–1178.
- Chen, R. T., J. Behrmann, D. K. Duvenaud, and J.-H. Jacobsen. (2019). “Residual flows for invertible generative modeling”. *Advances in Neural Information Processing Systems*. 32.
- Chen, R. T., Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. (2018). “Neural ordinary differential equations”. *Advances in neural information processing systems*. 31.
- Chen, T., G.-H. Liu, and E. Theodorou. (2022). “Likelihood Training of Schrödinger Bridge using Forward-Backward SDEs Theory”. In: *International Conference on Learning Representations*.
- Chen, Y., T. T. Georgiou, and M. Pavon. (2016). “On the relation between optimal transport and Schrödinger bridges: A stochastic control viewpoint”. *Journal of Optimization Theory and Applications*. 169: 671–691.
- Chen, Y., T. T. Georgiou, and M. Pavon. (2021). “Stochastic control liaisons: Richard sinkhorn meets gaspard monge on a schrodinger bridge”. *Siam Review*. 63(2): 249–313.
- Choi, J., S. Kim, Y. Jeong, Y. Gwon, and S. Yoon. (2021). “ILVR: Conditioning Method for Denoising Diffusion Probabilistic Models”. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE. 14347–14356.
- Chung, H., J. Kim, M. T. Mccann, M. L. Klasky, and J. C. Ye. (2022). “Diffusion posterior sampling for general noisy inverse problems”. *arXiv preprint arXiv:2209.14687*.
- Chung, H., J. Kim, M. T. Mccann, M. L. Klasky, and J. C. Ye. (2023). “Diffusion Posterior Sampling for General Noisy Inverse Problems”. In: *The Eleventh International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=OnD9zGAGT0k>.
- Csiszár, I. (1963). “Eine informationstheoretische Ungleichung und ihre Anwendung auf den Beweis der Ergodizität von Markoffschen Ketten”. *A Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*. 8(1-2): 85–108.

- Cuturi, M. (2013). “Sinkhorn distances: Lightspeed computation of optimal transport”. *Advances in neural information processing systems*. 26.
- Dai Pra, P. (1991). “A stochastic control approach to reciprocal diffusion processes”. *Applied mathematics and Optimization*. 23(1): 313–329.
- Daras, G., H. Chung, C.-H. Lai, Y. Mitsufuji, J. C. Ye, P. Milanfar, A. G. Dimakis, and M. Delbracio. (2024). “A survey on diffusion models for inverse problems”. *arXiv preprint arXiv:2410.00083*.
- Daras, G., Y. Dagan, A. Dimakis, and C. Daskalakis. (2023). “Consistent diffusion models: Mitigating sampling drift by learning to be consistent”. *Advances in Neural Information Processing Systems*. 36: 42038–42063.
- De Bortoli, V. (2022). “Convergence of denoising diffusion models under the manifold hypothesis”. *arXiv preprint arXiv:2208.05314*.
- De Bortoli, V., J. Thornton, J. Heng, and A. Doucet. (2021). “Diffusion schrödinger bridge with applications to score-based generative modeling”. *Advances in Neural Information Processing Systems*. 34: 17695–17709.
- Dhariwal, P. and A. Nichol. (2021). “Diffusion models beat gans on image synthesis”. *Advances in Neural Information Processing Systems*. 34: 8780–8794.
- Efron, B. (2011). “Tweedie’s formula and selection bias”. *Journal of the American Statistical Association*. 106(496): 1602–1614.
- Esser, P., S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, et al. (2024). “Scaling rectified flow transformers for high-resolution image synthesis”. In: *Forty-first International Conference on Machine Learning*.
- Evans, L. C. (2010). *Partial differential equations*. Providence, R.I.: American Mathematical Society.
- Fournier, N. and A. Guillin. (2015). “On the rate of convergence in Wasserstein distance of the empirical measure”. *Probability theory and related fields*. 162(3): 707–738.
- Frey, B. J., G. E. Hinton, and P. Dayan. (1995). “Does the wake-sleep algorithm produce good density estimators?” *Advances in neural information processing systems*. 8.
- Genevay, A., G. Peyré, and M. Cuturi. (2018). “Learning generative models with sinkhorn divergences”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 1608–1617.
- Geng, Z., M. Deng, X. Bai, J. Z. Kolter, and K. He. (2025a). “Mean flows for one-step generative modeling”. *arXiv preprint arXiv:2505.13447*.

- Geng, Z., A. Pokle, W. Luo, J. Lin, and J. Z. Kolter. (2025b). “Consistency Models Made Easy”. In: *The Thirteenth International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=xQVxo9dSID>.
- Geng, Z., A. Pokle, W. Luo, J. Lin, and J. Z. Kolter. (2024). “Consistency models made easy”. *arXiv preprint arXiv:2406.14548*.
- Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. (2014). “Generative adversarial nets”. *Advances in neural information processing systems*. 27.
- He, Y., N. Murata, C.-H. Lai, Y. Takida, T. Uesaka, D. Kim, W.-H. Liao, Y. Mitsufuji, J. Z. Kolter, R. Salakhutdinov, et al. (2023). “Manifold preserving guided diffusion”. In: *International Conference on Learning Representations*.
- He, Y., N. Murata, C.-H. Lai, Y. Takida, T. Uesaka, D. Kim, W.-H. Liao, Y. Mitsufuji, J. Z. Kolter, R. Salakhutdinov, and S. Ermon. (2024). “Manifold Preserving Guided Diffusion”. In: *The Twelfth International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=o3BxOLoxm1>.
- Heitz, E., L. Belcour, and T. Chambon. (2023). “Iterative  $\alpha$ -(de) blending: A minimalist deterministic diffusion model”. In: *ACM SIGGRAPH 2023 Conference Proceedings*. 1–8.
- Hertrich, J., A. Chambolle, and J. Delon. (2025). “On the Relation between Rectified Flows and Optimal Transport”. *arXiv preprint arXiv:2505.19712*.
- Hinton, G. E. (2002). “Training products of experts by minimizing contrastive divergence”. *Neural computation*. 14(8): 1771–1800.
- Ho, J., A. Jain, and P. Abbeel. (2020). “Denoising diffusion probabilistic models”. *Advances in Neural Information Processing Systems*. 33: 6840–6851.
- Ho, J. and T. Salimans. (2021). “Classifier-Free Diffusion Guidance”. In: *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*.
- Hochbruck, M. and A. Ostermann. (2005). “Explicit exponential Runge–Kutta methods for semilinear parabolic problems”. *SIAM Journal on Numerical Analysis*. 43(3): 1069–1090.
- Hochbruck, M. and A. Ostermann. (2010). “Exponential integrators”. *Acta Numerica*. 19: 209–286.
- Hu, Z., C.-H. Lai, Y. Mitsufuji, and S. Ermon. (2025). “CMT: Mid-Training for Efficient Learning of Consistency, Mean Flow, and Flow Map Models”. *arXiv preprint arXiv:2509.24526*.

- Huang, C.-W., R. T. Chen, C. Tsirigotis, and A. Courville. (2021). “Convex Potential Flows: Universal Probability Distributions with Optimal Transport and Convex Optimization”. In: *International Conference on Learning Representations*.
- Hutchinson, M. F. (1989). “A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines”. *Communications in Statistics-Simulation and Computation*. 18(3): 1059–1076.
- Hyvärinen, A. and P. Dayan. (2005). “Estimation of non-normalized statistical models by score matching.” *Journal of Machine Learning Research*. 6(4).
- Iserles, A. (2009). *A first course in the numerical analysis of differential equations*. No. 44. Cambridge university press.
- Karras, T., M. Aittala, T. Aila, and S. Laine. (2022). “Elucidating the design space of diffusion-based generative models”. *Advances in Neural Information Processing Systems*. 35: 26565–26577.
- Karras, T., M. Aittala, J. Lehtinen, J. Hellsten, T. Aila, and S. Laine. (2023). “Analyzing and improving the training dynamics of diffusion models”. *arXiv preprint arXiv:2312.02696*.
- Karras, T., M. Aittala, J. Lehtinen, J. Hellsten, T. Aila, and S. Laine. (2024). “Analyzing and Improving the Training Dynamics of Diffusion Models”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 24174–24184.
- Kim, D., C.-H. Lai, W.-H. Liao, N. Murata, Y. Takida, T. Uesaka, Y. He, Y. Mitsufuji, and S. Ermon. (2024a). “Consistency trajectory models: Learning probability flow ode trajectory of diffusion”. In: *International Conference on Learning Representations*.
- Kim, D., C.-H. Lai, W.-H. Liao, Y. Takida, N. Murata, T. Uesaka, Y. Mitsufuji, and S. Ermon. (2024b). “PaGoDA: Progressive Growing of a One-Step Generator from a Low-Resolution Diffusion Teacher”. *arXiv preprint arXiv:2405.14822*.
- Kim, D., S. Shin, K. Song, W. Kang, and I.-C. Moon. (2022). “Soft truncation: A universal training technique of score-based diffusion model for high precision score estimation”. In: *International Conference on Machine Learning*. PMLR. 11201–11228.
- Kingma, D., T. Salimans, B. Poole, and J. Ho. (2021). “Variational diffusion models”. *Advances in neural information processing systems*. 34: 21696–21707.
- Kingma, D. P. and R. Gao. (2023). “Understanding Diffusion Objectives as the ELBO with Simple Data Augmentation”. In: *Thirty-seventh Conference on Neural Information Processing Systems*.

- Kingma, D. P. and M. Welling. (2013). “Auto-encoding variational bayes”. *arXiv preprint arXiv:1312.6114*.
- Kloeden, P. E., E. Platen, P. E. Kloeden, and E. Platen. (1992). *Stochastic differential equations*. Springer.
- Lai, C.-H., Y. Takida, N. Murata, T. Uesaka, Y. Mitsufuji, and S. Ermon. (2023a). “FP-Diffusion: Improving score-based diffusion models by enforcing the underlying score fokker-planck equation”. In: *International Conference on Machine Learning*. PMLR. 18365–18398.
- Lai, C.-H., Y. Takida, T. Uesaka, N. Murata, Y. Mitsufuji, and S. Ermon. (2023b). “On the Equivalence of Consistency-Type Models: Consistency Models, Consistent Diffusion Models, and Fokker-Planck Regularization”. *arXiv preprint arXiv:2306.00367*.
- Larochelle, H. and I. Murray. (2011). “The neural autoregressive distribution estimator”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 29–37.
- Lavenant, H. and F. Santambrogio. (2022). “The flow map of the fokker–planck equation does not provide optimal transport”. *Applied Mathematics Letters*. 133: 108225.
- LeCun, Y., S. Chopra, R. Hadsell, M. Ranzato, F. Huang, et al. (2006). “A tutorial on energy-based learning”. *Predicting structured data*. 1(0).
- Léonard, C. (2012). “From the Schrödinger problem to the Monge–Kantorovich problem”. *Journal of Functional Analysis*. 262(4): 1879–1920.
- Léonard, C. (2014). “A survey of the Schrödinger problem and some of its connections with optimal transport”. *Discrete and Continuous Dynamical Systems-Series A*. 34(4): 1533–1574.
- Lipman, Y., R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le. (2022). “Flow Matching for Generative Modeling”. In: *The Eleventh International Conference on Learning Representations*.
- Lipman, Y., M. Havasi, P. Holderrieth, N. Shaul, M. Le, B. Karrer, R. T. Chen, D. Lopez-Paz, H. Ben-Hamu, and I. Gat. (2024). “Flow matching guide and code”. *arXiv preprint arXiv:2412.06264*.
- Liu, G.-H., A. Vahdat, D.-A. Huang, E. Theodorou, W. Nie, and A. Anandkumar. (2023). “I $\hat{2}$ SB: Image-to-Image Schrödinger Bridge”. In: *International Conference on Machine Learning*. PMLR. 22042–22062.
- Liu, Q. (2022). “Rectified flow: A marginal preserving approach to optimal transport”. *arXiv preprint arXiv:2209.14577*.

- Liu, X., C. Gong, *et al.* (2022). “Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow”. In: *The Eleventh International Conference on Learning Representations*.
- Lou, A., C. Meng, and S. Ermon. (2024). “Discrete Diffusion Modeling by Estimating the Ratios of the Data Distribution”. In: *International Conference on Machine Learning*. PMLR. 32819–32848.
- Lu, C. and Y. Song. (2024). “Simplifying, stabilizing and scaling continuous-time consistency models”. *arXiv preprint arXiv:2410.11081*.
- Lu, C., K. Zheng, F. Bao, J. Chen, C. Li, and J. Zhu. (2022a). “Maximum Likelihood Training for Score-based Diffusion ODEs by High Order Denoising Score Matching”. In: *International Conference on Machine Learning*. PMLR. 14429–14460.
- Lu, C., Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. (2022b). “Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps”. *Advances in Neural Information Processing Systems*. 35: 5775–5787.
- Lu, C., Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. (2022c). “Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models”. *arXiv preprint arXiv:2211.01095*.
- Luan, V. T. (2021). “Efficient exponential Runge–Kutta methods of high order: construction and implementation”. *BIT Numerical Mathematics*. 61(2): 535–560.
- Luhman, E. and T. Luhman. (2021). “Knowledge distillation in iterative generative models for improved sampling speed”. *arXiv preprint arXiv:2101.02388*.
- Luo, W., T. Hu, S. Zhang, J. Sun, Z. Li, and Z. Zhang. (2023). “Diff-instruct: A universal approach for transferring knowledge from pre-trained diffusion models”. *Advances in Neural Information Processing Systems*. 36: 76525–76546.
- Ma, N., M. Goldstein, M. S. Albergo, N. M. Boffi, E. Vanden-Eijnden, and S. Xie. (2024). “Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers”. In: *European Conference on Computer Vision*. Springer. 23–40.
- Maoutsa, D., S. Reich, and M. Opper. (2020). “Interacting particle solutions of fokker–planck equations through gradient–log–density estimation”. *Entropy*. 22(8): 802.
- Meng, C., K. Choi, J. Song, and S. Ermon. (2022). “Concrete score matching: Generalized score matching for discrete data”. *Advances in Neural Information Processing Systems*. 35: 34532–34545.

- Meng, C., R. Rombach, R. Gao, D. Kingma, S. Ermon, J. Ho, and T. Salimans. (2023). “On distillation of guided diffusion models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14297–14306.
- Meng, C., Y. Song, W. Li, and S. Ermon. (2021a). “Estimating high order gradients of the data distribution by denoising”. *Advances in Neural Information Processing Systems*. 34: 25359–25369.
- Meng, C., Y. Song, J. Song, J. Wu, J.-Y. Zhu, and S. Ermon. (2021b). “Sdedit: Image synthesis and editing with stochastic differential equations”. *arXiv preprint arXiv:2108.01073*.
- Mikami, T. and M. Thieullen. (2006). “Duality theorem for the stochastic optimal control problem”. *Stochastic processes and their applications*. 116(12): 1815–1835.
- Mikami, T. and M. Thieullen. (2008). “Optimal transportation problem by stochastic optimal control”. *SIAM Journal on Control and Optimization*. 47(3): 1127–1139.
- Mokady, R., A. Hertz, K. Aberman, Y. Pritch, and D. Cohen-Or. (2023). “Null-text inversion for editing real images using guided diffusion models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6038–6047.
- Neklyudov, K., R. Brekelmans, D. Severo, and A. Makhzani. (2023). “Action matching: Learning stochastic dynamics from samples”. In: *International Conference on Machine Learning*. PMLR. 25858–25889.
- Nowozin, S., B. Cseke, and R. Tomioka. (2016). “f-gan: Training generative neural samplers using variational divergence minimization”. *Advances in neural information processing systems*. 29.
- Øksendal, B. (2003). “Stochastic differential equations”. In: *Stochastic differential equations*. Springer. 65–84.
- Onken, D., S. W. Fung, X. Li, and L. Ruthotto. (2021). “Ot-flow: Fast and accurate continuous normalizing flows via optimal transport”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. No. 10. 9223–9232.
- Pavon, M. and A. Wakolbinger. (1991). “On free energy, stochastic control, and Schrödinger processes”. In: *Modeling, Estimation and Control of Systems with Uncertainty: Proceedings of a Conference held in Sopron, Hungary, September 1990*. Springer. 334–348.
- Peters, J., K. Mulling, and Y. Altun. (2010). “Relative entropy policy search”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 24. No. 1. 1607–1612.

- Peyré, G., M. Cuturi, *et al.* (2019). “Computational optimal transport: With applications to data science”. *Foundations and Trends® in Machine Learning*. 11(5-6): 355–607.
- Pontryagin, L. S. (2018). *Mathematical theory of optimal processes*. Routledge.
- Poole, B., A. Jain, J. T. Barron, and B. Mildenhall. (2023). “DreamFusion: Text-to-3D using 2D Diffusion”. In: *The Eleventh International Conference on Learning Representations*.
- Pra, P. D. and M. Pavon. (1990). “On the Markov processes of Schrödinger, the Feynman-Kac formula and stochastic control”. In: *Realization and Modelling in System Theory: Proceedings of the International Symposium MTNS-89, Volume I*. Springer. 497–504.
- Radford, A., J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.* (2021). “Learning transferable visual models from natural language supervision”. In: *International conference on machine learning*. PMLR. 8748–8763.
- Rafailov, R., A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. (2023). “Direct preference optimization: Your language model is secretly a reward model”. *Advances in neural information processing systems*. 36: 53728–53741.
- Raissi, M. (2018). “Deep hidden physics models: Deep learning of nonlinear partial differential equations”. *Journal of Machine Learning Research*. 19(25): 1–24.
- Reid, W. (1971). *Ordinary Differential Equations. Applied mathematics series*. Wiley.
- Rezende, D. and S. Mohamed. (2015). “Variational inference with normalizing flows”. In: *International conference on machine learning*. PMLR. 1530–1538.
- Saharia, C., W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, K. Ghasemipour, R. Gontijo Lopes, B. Karagol Ayan, T. Salimans, *et al.* (2022). “Photorealistic text-to-image diffusion models with deep language understanding”. *Advances in Neural Information Processing Systems*. 35: 36479–36494.
- Salimans, T. and J. Ho. (2021). “Progressive Distillation for Fast Sampling of Diffusion Models”. In: *International Conference on Learning Representations*.
- Särkkä, S. and A. Solin. (2019). *Applied stochastic differential equations*. Vol. 10. Cambridge University Press.
- Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. (2017). “Proximal policy optimization algorithms”. *arXiv preprint arXiv:1707.06347*.

- Shih, A., S. Belkhale, S. Ermon, D. Sadigh, and N. Anari. (2023). “Parallel Sampling of Diffusion Models”. *arXiv preprint arXiv:2305.16317*.
- Sinkhorn, R. (1964). “A relationship between arbitrary positive matrices and doubly stochastic matrices”. *The annals of mathematical statistics*. 35(2): 876–879.
- Sohl-Dickstein, J., E. Weiss, N. Maheswaranathan, and S. Ganguli. (2015). “Deep unsupervised learning using nonequilibrium thermodynamics”. In: *International Conference on Machine Learning*. PMLR. 2256–2265.
- Song, J., C. Meng, and S. Ermon. (2020a). “Denoising Diffusion Implicit Models”. In: *International Conference on Learning Representations*.
- Song, Y. and P. Dhariwal. (2024). “Improved Techniques for Training Consistency Models”. In: *The Twelfth International Conference on Learning Representations*.
- Song, Y., P. Dhariwal, M. Chen, and I. Sutskever. (2023). “Consistency models”. *arXiv preprint arXiv:2303.01469*.
- Song, Y., C. Durkan, I. Murray, and S. Ermon. (2021). “Maximum likelihood training of score-based diffusion models”. *Advances in Neural Information Processing Systems*. 34: 1415–1428.
- Song, Y. and S. Ermon. (2019). “Generative modeling by estimating gradients of the data distribution”. *Advances in Neural Information Processing Systems*. 32.
- Song, Y., S. Garg, J. Shi, and S. Ermon. (2020b). “Sliced score matching: A scalable approach to density and score estimation”. In: *Uncertainty in Artificial Intelligence*. PMLR. 574–584.
- Song, Y., J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. (2020c). “Score-Based Generative Modeling through Stochastic Differential Equations”. In: *International Conference on Learning Representations*.
- Su, X., J. Song, C. Meng, and S. Ermon. (2022). “Dual diffusion implicit bridges for image-to-image translation”. *arXiv preprint arXiv:2203.08382*.
- Tabak, E. G. and E. Vanden-Eijnden. (2010). “Density estimation by dual ascent of the log-likelihood”. *Commun. Math. Sci.* 8(1): 217–233.
- Tong, A., K. FATRAS, N. Malkin, G. Huguet, Y. Zhang, J. Rector-Brooks, G. Wolf, and Y. Bengio. (2024). “Improving and generalizing flow-based generative models with minibatch optimal transport”. *Transactions on Machine Learning Research*.
- Turner, C. V. (2013). “A family of nonparametric density estimation algorithms”. *Communications on Pure and Applied Mathematics*. 66(2): 145–164.

- Uria, B., M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle. (2016). “Neural autoregressive distribution estimation”. *Journal of Machine Learning Research*. 17(205): 1–37.
- Vahdat, A. and J. Kautz. (2020). “NVAE: A deep hierarchical variational autoencoder”. *Advances in neural information processing systems*. 33: 19667–19679.
- Villani, C. *et al.* (2008). *Optimal transport: old and new*. Vol. 338. Springer.
- Vincent, P. (2011). “A connection between score matching and denoising autoencoders”. *Neural computation*. 23(7): 1661–1674.
- Wallace, B., M. Dang, R. Rafailov, L. Zhou, A. Lou, S. Purushwalkam, S. Ermon, C. Xiong, S. Joty, and N. Naik. (2024). “Diffusion model alignment using direct preference optimization”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8228–8238.
- Wang, Z., C. Lu, Y. Wang, F. Bao, C. Li, H. Su, and J. Zhu. (2023). “Prolific-dreamer: High-fidelity and diverse text-to-3d generation with variational score distillation”. *Advances in Neural Information Processing Systems*. 36: 8406–8441.
- Xu, Y., M. Deng, X. Cheng, Y. Tian, Z. Liu, and T. S. Jaakkola. (2023). “Restart Sampling for Improving Generative Processes”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. URL: <https://openreview.net/forum?id=wFuemocyHZ>.
- Ye, H., H. Lin, J. Han, M. Xu, S. Liu, Y. Liang, J. Ma, J. Y. Zou, and S. Ermon. (2024). “Tfg: Unified training-free guidance for diffusion models”. *Advances in Neural Information Processing Systems*. 37: 22370–22417.
- Yin, T., M. Gharbi, T. Park, R. Zhang, E. Shechtman, F. Durand, and B. Freeman. (2024a). “Improved distribution matching distillation for fast image synthesis”. *Advances in neural information processing systems*. 37: 47455–47487.
- Yin, T., M. Gharbi, R. Zhang, E. Shechtman, F. Durand, W. T. Freeman, and T. Park. (2024b). “One-Step Diffusion with Distribution Matching Distillation”. In: *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 6613–6623.
- Yu, J., Y. Wang, C. Zhao, B. Ghanem, and J. Zhang. (2023). “Freedom: Training-free energy-guided conditional diffusion model”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 23174–23184.
- Zhang, Q. and Y. Chen. (2022). “Fast Sampling of Diffusion Models with Exponential Integrator”. In: *The Eleventh International Conference on Learning Representations*.

- Zheng, K., C. Lu, J. Chen, and J. Zhu. (2023). “Dpm-solver-v3: Improved diffusion ode solver with empirical model statistics”. *Advances in Neural Information Processing Systems*. 36: 55502–55542.
- Zhou, M., H. Zheng, Z. Wang, M. Yin, and H. Huang. (2024). “Score identity distillation: Exponentially fast distillation of pretrained diffusion models for one-step generation”. In: *Forty-first International Conference on Machine Learning*.