An Introduction to Flow Matching and Diffusion Models

Peter Holderrieth and Ezra Erives

Website: https://diffusion.csail.mit.edu/

1 Introduction

> Creating noise from data is easy; creating data from noise is generative modeling.
>
> Song et al. [30]

1.1 Overview

In recent years, we all have witnessed a tremendous revolution in artificial intelligence (AI). Image generators like *Stable Diffusion 3* can generate photorealistic and artistic images across a diverse range of styles, video models like Metaa's *Movie Gen Video* can generate highly realistic movie clips, and large language models like *ChatGPT* can generate seemingly human-level responses to text prompts. At the heart of this revolution lies a new ability of AI systems: the ability to **generate** objects. While previous generations of AI systems were mainly used for **prediction**, these new AI system are creative: they dream or come up with new objects based on user-specified input. Such **generative AI** systems are at the core of this recent AI revolution.

The goal of this class is to teach you two of the most widely used generative AI algorithms: **denoising diffusion models** [32] and **flow matching** [14, 16, 1, 15]. These models are the backbone of the best image, audio, and video generation models (e.g., *Stable Diffusion 3* and *Movie Gen Video*), and have most recently became the state-of-the-art in scientific applications such as protein structures (e.g., *AlphaFold3* is a diffusion model). Without a doubt, understanding these models is truly an extremely useful skill to have.

All of these generative models generate objects by iteratively converting **noise** into **data**. This evolution from noise to data is facilitated by the simulation of **ordinary or stochastic differential equations (ODEs/SDEs)**. Flow matching and denoising diffusion models are a family of techniques that allow us to construct, train, and simulate, such ODEs/SDEs at large scale with deep neural networks. While these models are rather simple to implement, the technical nature of SDEs can make these models difficult to understand. In this course, our goal is to provide a self-contained introduction to the necessary mathematical toolbox regarding differential equations to enable you to systematically understand these models. Beyond being widely applicable, we believe that the theory behind flow and diffusion models is elegant in its own right. Therefore, most importantly, we hope that this course will be a lot of fun to you.

Remark 1 (Additional Resources)
While these lecture notes are self-contained, there are two additional resources that we encourage you to use:

1. **Lecture recordings:** These guide you through each section in a lecture format.
2. **Labs:** These guide you in implementing your own diffusion model from scratch. We highly recommend that you "get your hands dirty" and code.

You can find these on our course website: https://diffusion.csail.mit.edu/.

1.2 Course Structure

We give a brief overview over of this document.

- **Section 1, Generative Modeling as Sampling:** We formalize what it means to "generate" an image, video, protein, etc. We will translate the problem of e.g., "how to generate an image of a dog?" into the more precise problem of sampling from a probability distribution.
- **Section 2, Flow and Diffusion Models:** Next, we explain the machinery of generation. As you can guess by the name of this class, this machinery consists of simulating ordinary and stochastic differential equations. We provide an introduction to differential equations and explain how to construct them with neural networks.
- **Section 3, Constructing a Training Target:** To train our generative model, we must first pin down precisely what it is that our model is supposed to approximate. In other words, what's the ground truth? We will introduce the celebrated **Fokker-Planck equation**, which will allow us to formalize the notion of ground truth.
- **Section 4, Training:** This section formulates a **training objective**, allowing us to approximate the training target, or ground truth, of the previous section. With this, we are ready to provide a minimal implementation of flow matching and denoising diffusion models.
- **Section 5, Conditional Image Generation:** We learn how to build a conditional image generator. To do so, we formulate how to condition our samples on a prompt (e.g. "an image of a cat"). We then discuss common neural network architectures and survey state-of-the-art models for both image and video generation.

Required background. Due to the technical nature of this subject, we recommend some base level of mathematical maturity, and in particular some familiarity with probability theory. For this reason, we included a brief reminder section on probability theory in appendix A. Don't worry if some of the concepts there are unfamiliar to you.

1.3 Generative Modeling As Sampling

Let's begin by thinking about various data types, or **modalities**, that we might encounter, and how we will go about representing them numerically:

1. **Image:** Consider images with $H \times W$ pixels where H describes the height and W the width of the image, each with three color channels (RGB). For every pixel and every color channel, we are given an intensity value in $\mathbb{R}$. Therefore, an image can be represented by an element $z \in \mathbb{R}^{H \times W \times 3}$.
2. **Video:** A video is simply a series of images in time. If we have T time points or **frames**, a video would therefore be represented by an element $z \in \mathbb{R}^{T \times H \times W \times 3}$.
3. **Molecular structure:** A naive way would be to represent the structure of a molecule by a matrix $z = (z^1, \dots, z^N) \in \mathbb{R}^{3 \times N}$ where N is the number of atoms in the molecule and each $z^i \in \mathbb{R}^3$ describes the location of that atom. Of course, there are other, more sophisticated ways of representing such a molecule.

In all of the above examples, the object that we want to generate can be mathematically represented as a vector (potentially after flattening). Therefore, throughout this document, we will have:

Key Idea 1 (Objects as Vectors)
We identify the objects being generated as vectors $z \in \mathbb{R}^d$.

A notable exception to the above is text data, which is typically modeled as a discrete object via autoregressive language models (such as *ChatGPT*). While flow and diffusion models for discrete data have been developed, this course focuses exclusively on applications to continuous data.

Generation as Sampling. Let us define what it means to "generate" something. For example, let's say we want to generate an image of a dog. Naturally, there are *many* possible images of dogs that we would be happy with. In particular, there is no one single "best" image of a dog. Rather, there is a spectrum of images that fit better or worse. In machine learning, it is common to think of this diversity of possible images as a *probability distribution*. We call it the **data distribution** and denote it as p_{data}. In the example of dog images, this distribution would therefore give higher likelihood to images that look more like a dog. Therefore, how "good" an image/video/molecule fits - a rather subjective statement - is replaced by how "likely" it is under the data distribution p_{data}. With this, we can mathematically express the task of generation as sampling from the (unknown) distribution p_{data}:

Key Idea 2 (Generation as Sampling)
Generating an object z is modeled as sampling from the data distribution $z \sim p_{\text{data}}$.

A **generative model** is a machine learning model that allows us to generate samples from p_{data}. In machine learning, we require data to train models. In generative modeling, we usually assume access to a finite number of examples sampled independently from p_{data}, which together serve as a proxy for the true distribution.

Key Idea 3 (Dataset)
A dataset consists of a finite number of samples $z_1, \dots, z_N \sim p_{\text{data}}$.

For images, we might construct a dataset by compiling publicly available images from the internet. For videos, we might similarly use YouTube as a database. For protein structures, we can use experimental data bases from sources such as the Protein Data Bank (PDB) that collected scientific measurements over decades. As the size of our dataset grows very large, it becomes an increasingly better representation of the underlying distribution p_{data}.

Conditional Generation. In many cases, we want to generate an object **conditioned** on some data y. For example, we might want to generate an image conditioned on $y =$"a dog running down a hill covered with snow with mountains in the background". We can rephrase this as sampling from a **conditional distribution**:

Key Idea 4 (Conditional Generation)
Conditional generation involves sampling from $z \sim p_{\text{data}}(\cdot|y)$, where y is a conditioning variable.

We call $p_{\text{data}}(\cdot|y)$ the **conditional data distribution**. The conditional generative modeling task typically involves learning to condition on an arbitrary, rather than fixed, choice of y. Using our previous example, we might

alternatively want to condition on a different text prompt, such as $y =$"a photorealistic image of a cat blowing out birthday candles". We therefore seek a single model which may be conditioned on any such choice of y. It turns out that techniques for unconditional generation are readily generalized to the conditional case. Therefore, for the first 3 sections, we will focus almost exclusively on the unconditional case (keeping in mind that conditional generation is what we're building towards).

From Noise to Data. So far, we have discussed the *what* of generative modeling: generating samples from p_{data}. Here, we will briefly discuss the *how*. For this, we assume that we have access to some **initial distribution** p_{init} that we can easily sample from, such as the Gaussian $p_{\text{init}} = \mathcal{N}(0, I_d)$. The goal of a generative modeling is then to transform samples from $x \sim p_{\text{init}}$ into samples from p_{data}. We note that p_{init} does not have to be so simple as a Gaussian. As we shall see, there are interesting use cases for leveraging this flexibility. Despite this, in the majority of applications we take it to be a simple Gaussian and it is important to keep that in mind.

Summary We summarize our discussion so far as follows.

Summary 2 (Generation as Sampling)
We summarize the findings of this section:

1. In this class, we consider the task of generating objects that are represented as vectors $z \in \mathbb{R}^d$ such as images, videos, or molecular structures.
2. Generation is the task of generating samples from a probability distribution p_{data} having access to a dataset of samples $z_1, \dots, z_N \sim p_{\text{data}}$ during training.
3. Conditional generation assumes that we condition the distribution on a label y and we want to sample from $p_{\text{data}}(\cdot|y)$ having access to data set of pairs $(z_1, y) \dots, (z_N, y)$ during training.
4. Our goal is to train a generative model to transform samples from a simple distribution p_{init} (e.g. a Gaussian) into samples from p_{data}.

2 Flow and Diffusion Models

In the previous section, we formalized generative modeling as sampling from a data distribution p_{data}. Further, we saw that sampling could be achieved via the transformation of samples from a simple distribution p_{init}, such as the Gaussian $\mathcal{N}(0, I_d)$, to samples from the target distribution p_{data}. In this section, we describe how the desired transformation can be obtained as the simulation of a suitably constructed differential equation. For example, flow matching and diffusion models involve simulating **ordinary differential equations** (ODEs) and **stochastic differential equations** (SDEs), respectively. The goal of this section is therefore to define and construct these generative models as they will be used throughout the remainder of the notes. Specifically, we first define ODEs and SDEs, and discuss their simulation. Second, we describe how to parameterize an ODE/SDE using a deep neural network. This leads to the definition of a flow and diffusion model and the fundamental algorithms to sample from such models. In later sections, we then explore how to train these models.

2.1 Flow Models

We start by defining **ordinary differential equations (ODEs)**. A solution to an ODE is defined by a **trajectory**, i.e. a function of the form

$$X : [0, 1] \to \mathbb{R}^d, \quad t \mapsto X_t,$$

that maps from time t to some location in space $\mathbb{R}^d$. Every ODE is defined by a **vector field** u, i.e. a function of the form

$$u : \mathbb{R}^d \times [0, 1] \to \mathbb{R}^d, \quad (x, t) \mapsto u_t(x),$$

i.e. for every time t and location x we get a vector $u_t(x) \in \mathbb{R}^d$ specifying a velocity in space (see fig. 1). An ODE imposes a condition on a trajectory: we want a trajectory X that "follows along the lines" of the vector field u_t, starting at the point x_0. We may formalize such a trajectory as being the solution to the equation:

$$\frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t(X_t) \qquad \blacktriangleright \text{ ODE} \tag{1a}$$

$$X_0 = x_0 \qquad \blacktriangleright \text{ initial conditions} \tag{1b}$$

Equation (1a) requires that the derivative of X_t is specified by the direction given by u_t. Equation (1b) requires that we start at x_0 at time $t = 0$. We may now ask: if we start at $X_0 = x_0$ at $t = 0$, where are we at time t (what is X_t)? This question is answered by a function called the **flow**, which is a solution to the ODE

$$\psi : \mathbb{R}^d \times [0, 1] \mapsto \mathbb{R}^d, \quad (x_0, t) \mapsto \psi_t(x_0) \tag{2a}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} \psi_t(x_0) = u_t(\psi_t(x_0)) \qquad \blacktriangleright \text{ flow ODE} \tag{2b}$$

$$\psi_0(x_0) = x_0 \qquad \blacktriangleright \text{ flow initial conditions} \tag{2c}$$

For a given initial condition $X_0 = x_0$, a trajectory of the ODE is recovered via $X_t = \psi_t(X_0)$. Therefore, vector fields, ODEs, and flows are, intuitively, three descriptions of the same object: **vector fields define ODEs whose**

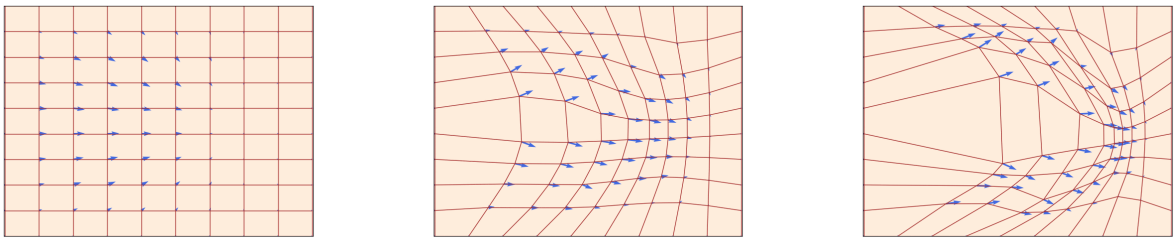

Figure 1: A flow $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (red square grid) is defined by a velocity field $u_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with blue arrows) that prescribes its instantaneous movements at all locations (here, $d = 2$). We show three different times t. As one can see, a flow is a diffeomorphism that "warps" space. Figure from [15].

solutions are flows. As with every equation, we should ask ourselves about an ODE: Does a solution exist and if so, is it unique? A fundamental result in mathematics is "yes!" to both, as long we impose weak assumptions on u_t:

Theorem 3 (Flow existence and uniqueness)
If $u : \mathbb{R}^d \times [0, 1] \to \mathbb{R}^d$ is continuously differentiable with a bounded derivative, then the ODE in (2) has a unique solution given by a flow ψ_t. In this case, ψ_t is a **diffeomorphism** for all t, i.e. ψ_t is continuously differentiable with a continuously differentiable inverse ψ_t^{-1}.

Note that the assumptions required for the existence and uniqueness of a flow are almost always fulfilled in machine learning, as we use neural networks to parameterize $u_t(x)$ and they always have bounded derivatives. Therefore, theorem 3 should not be a concern for you but rather good news: **flows exist and are unique solutions to ODEs in our cases of interest.** A proof can be found in [20, 4].

Example 4 (Linear Vector Fields)
Let us consider a simple example of a vector field $u_t(x)$ that is a simple linear function in x, i.e. $u_t(x) = -\theta x$ for $\theta > 0$. Then the function

$$\psi_t(x_0) = \exp(-\theta t)\, x_0 \tag{3}$$

defines a flow ψ solving the ODE in eq. (2). You can check this yourself by checking that $\psi_0(x_0) = x_0$ and computing

$$\frac{\mathrm{d}}{\mathrm{d}t}\psi_t(x_0) \overset{(3)}{=} \frac{\mathrm{d}}{\mathrm{d}t}\left(\exp(-\theta t)\, x_0\right) \overset{(i)}{=} -\theta \exp(-\theta t)\, x_0 \overset{(3)}{=} -\theta\psi_t(x_0) = u_t(\psi_t(x_0)),$$

where in (i) we used the chain rule. In fig. 3, we visualize a flow of this form converging to 0 exponentially.

Simulating an ODE. In general, it is not possible to compute the flow ψ_t explicitly if u_t is not as simple as a linear function. In these cases, one uses **numerical methods** to simulate ODEs. Fortunately, this is a classical and

well researched topic in numerical analysis, and a myriad of powerful methods exist [11]. One of the simplest and most intuitive methods is the **Euler method**. In the Euler method, we initialize with $X_0 = x_0$ and update via

$$X_{t+h} = X_t + hu_t(X_t) \quad (t = 0, h, 2h, 3h, \dots, 1-h) \tag{4}$$

where $h = n^{-1} > 0$ is a step size hyperparameter with $n \in \mathbb{N}$. For this class, the Euler method will be good enough. To give you a taste of a more complex method, let us consider **Heun's method** defined via the update rule

$$\begin{aligned} X'_{t+h} &= X_t + hu_t(X_t) && \blacktriangleright \text{ initial guess of new state} \\ X_{t+h} &= X_t + \frac{h}{2}(u_t(X_t) + u_{t+h}(X'_{t+h})) && \blacktriangleright \text{ update with average } u \text{ at current and guessed state} \end{aligned}$$

Intuitively, the Heun's method is as follows: it takes a first guess X'_{t+h} of what the next step could be but corrects the direction initially taken via an updated guess.

Flow models. We can now construct a generative model via an ODE. Remember that our goal was to convert a simple distribution p_{init} into a complex distribution p_{data}. The simulation of an ODE is thus a natural choice for this transformation. A **flow model** is described by the ODE

$$\begin{aligned} X_0 &\sim p_{\text{init}} && \blacktriangleright \text{ random initialization} \\ \frac{\mathrm{d}}{\mathrm{d}t} X_t &= u_t^\theta(X_t) && \blacktriangleright \text{ ODE} \end{aligned}$$

where the vector field u_t^θ is a neural network u_t^θ with parameters θ. For now, we will speak of u_t^θ as being a generic neural network; i.e. a continuous function $u_t^\theta : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ with parameters θ. Later, we will discuss particular choices of neural network architectures. Our goal is to make the endpoint X_1 of the trajectory have distribution p_{data}, i.e.

$$X_1 \sim p_{\text{data}} \quad \Leftrightarrow \quad \psi_1^\theta(X_0) \sim p_{\text{data}}$$

where ψ_t^θ describes the flow induced by u_t^θ. Note however: although it is called *flow model*, **the neural network parameterizes the vector field, not the flow**. In order to compute the flow, we need to simulate the ODE. In algorithm 1, we summarize the procedure how to sample from a flow model.

Algorithm 1 Sampling from a Flow Model with Euler method

Require: Neural network vector field u_t^θ, number of steps n
1: Set $t = 0$
2: Set step size $h = \frac{1}{n}$
3: Draw a sample $X_0 \sim p_{\text{init}}$
4: **for** $i = 1, \dots, n-1$ **do**
5: $\quad X_{t+h} = X_t + hu_t^\theta(X_t)$
6: $\quad$ Update $t \leftarrow t + h$
7: **end for**
8: **return** X_1

2.2 Diffusion Models

Stochastic differential equations (SDEs) extend the deterministic trajectories from ODEs with **stochastic** trajectories. A stochastic trajectory is commonly called a **stochastic process** $(X_t)_{0\leq t\leq 1}$ and is given by

$$\begin{aligned} X_t \text{ is a random variable for every } 0 \leq t \leq 1 \\ X : [0,1] \to \mathbb{R}^d, \quad t \mapsto X_t \text{ is a random trajectory for every draw of } X \end{aligned}$$

In particular, when we simulate the same stochastic process twice, we might get different outcomes because the dynamics are designed to be random.

Brownian Motion. SDEs are constructed via a **Brownian motion** - a fundamental stochastic process that came out of the study physical diffusion processes. You can think of a Brownian motion as a continuous random walk. Let us define it: A **Brownian motion** $W = (W_t)_{0\leq t\leq 1}$ is a stochastic process such that $W_0 = 0$, the trajectories $t \mapsto W_t$ are continuous, and the following two conditions hold:

1. **Normal increments:** $W_t - W_s \sim \mathcal{N}(0, (t-s)I_d)$ for all $0 \leq s < t$, i.e. increments have a Gaussian distribution with variance increasing linearly in time (I_d is the identity matrix).
2. **Independent increments:** For any $0 \leq t_0 < t_1 < \cdots < t_n = 1$, the increments $W_{t_1} - W_{t_0}, \ldots, W_{t_n} - W_{t_{n-1}}$ are independent random variables.

Brownian motion is also called a **Wiener process**, which is why we denote it with a "W".[1] We can easily simulate a Brownian motion approximately with step size $h > 0$ by setting $W_0 = 0$ and updating

$$W_{t+h} = W_t + \sqrt{h}\epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I_d) \quad (t = 0, h, 2h, \ldots, 1-h) \quad (5)$$

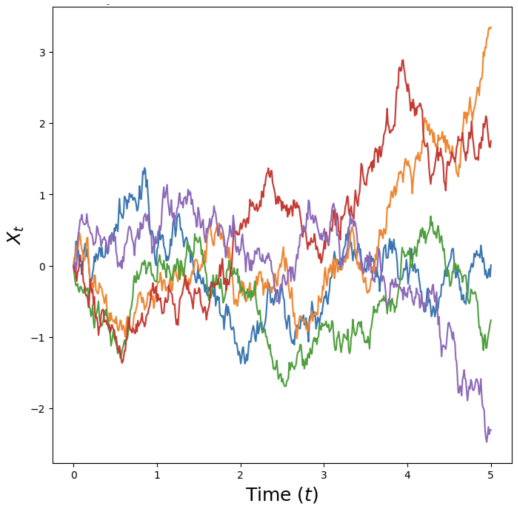

Figure 2: Sample trajectories of a Brownian motion W_t in dimension $d = 1$ simulated using eq. (5).

In fig. 2, we plot a few example trajectories of a Brownian motion.
Brownian motion is as central to the study of stochastic processes as the Gaussian distribution is to the study of probability distributions. From finance to statistical physics to epidemiology, the study of Brownian motion has far reaching applications beyond machine learning. In finance, for example, Brownian motion is used to model the price of complex financial instruments. Also just as a mathematical construction, Brownian motion is fascinating: For example, while the paths of a Brownian motion are continuous (so that you could draw it without ever lifting a pen), they are infinitely long (so that you would never stop drawing).

From ODEs to SDEs. The idea of an SDE is to extend the deterministic dynamics of an ODE by adding stochastic dynamics driven by a Brownian motion. Because everything is stochastic, we may no longer take the derivative as

[1] Nobert Wiener was a famous mathematician who taught at MIT. You can still see his portraits hanging at the MIT math department.

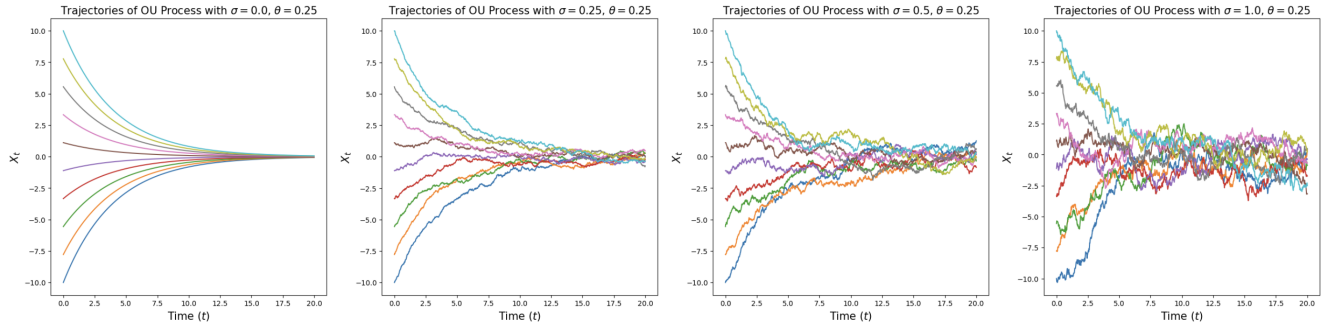

Figure 3: Illustration of Ornstein-Uhlenbeck processes (eq. (8)) in dimension $d = 1$ for $\theta = 0.25$ and various choices of σ (increasing from left to right). For $\sigma = 0$, we recover a flow (smooth, deterministic trajectories) that converges to the origin as $t \to \infty$. For $\sigma > 0$ we have random paths which converge towards the Gaussian $\mathcal{N}(0, \frac{\sigma^2}{2\theta})$ as $t \to \infty$.

in eq. (1a). Hence, we need to find an **equivalent formulation of ODEs that does not use derivatives**. For this, let us therefore rewrite trajectories $(X_t)_{0 \le t \le 1}$ of an ODE as follows:

$$\begin{aligned} & \frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t(X_t) && \blacktriangleright \text{ expression via derivatives} \\ \overset{(i)}{\Leftrightarrow} \quad & \frac{1}{h}\left(X_{t+h} - X_t\right) = u_t(X_t) + R_t(h) && \\ \Leftrightarrow \quad & X_{t+h} = X_t + h u_t(X_t) + h R_t(h) && \blacktriangleright \text{ expression via infinitesimal updates} \end{aligned}$$

where $R_t(h)$ describes a negligible function for small h, i.e. such that $\lim_{h \to 0} R_t(h) = 0$, and in (i) we simply use the definition of derivatives. The derivation above simply restates what we already know: A trajectory $(X_t)_{0 \le t \le 1}$ of an ODE takes, at every timestep, a small step in the direction $u_t(X_t)$. We may now amend the last equation to make it stochastic: A trajectory $(X_t)_{0 \le t \le 1}$ of an SDE takes, at every timestep, a small step in the direction $u_t(X_t)$ *plus* some contribution from a Brownian motion:

$$X_{t+h} = X_t + \underbrace{h u_t(X_t)}_{\text{deterministic}} + \sigma_t \underbrace{(W_{t+h} - W_t)}_{\text{stochastic}} + \underbrace{h R_t(h)}_{\text{error term}} \tag{6}$$

where $\sigma_t \ge 0$ describes the **diffusion coefficient** and $R_t(h)$ describes a stochastic error term such that the standard deviation $\mathbb{E}[\|R_t(h)\|^2]^{1/2} \to 0$ goes to zero for $h \to 0$. The above describes a **stochastic differential equation (SDE)**. It is common to denote it in the following symbolic notation:

$$\mathrm{d}X_t = u_t(X_t)\mathrm{d}t + \sigma_t \mathrm{d}W_t \qquad \blacktriangleright \text{ SDE} \tag{7a}$$

$$X_0 = x_0 \qquad \blacktriangleright \text{ initial condition} \tag{7b}$$

However, always keep in mind that the "$\mathrm{d}X_t$"-notation above is a purely informal notation of eq. (6). Unfortunately, SDEs do not have a flow map ϕ_t anymore. This is because the value X_t is not fully determined by $X_0 \sim p_{\text{init}}$ anymore as the evolution itself is stochastic. Still, in the same way as for ODEs, we have:

Theorem 5 (SDE Solution Existence and Uniqueness)
If $u : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ is continuously differentiable with a bounded derivative and σ_t is continuous, then the SDE in (7) has a solution given by the unique stochastic process $(X_t)_{0 \leq t \leq 1}$ satisfying eq. (6).

If this was a stochastic calculus class, we would spend several lectures proving this theorem and constructing SDEs with full mathematical rigor, i.e. constructing a Brownian motion from first principles and constructing the process X_t via **stochastic integration**. As we focus on machine learning in this class, we refer to [18] for a more technical treatment. Finally, note that every ODE is also an SDE - simply with a vanishing diffusion coefficient $\sigma_t = 0$. Therefore, for the remainder of this class, **when we speak about SDEs, we consider ODEs as a special case**.

Example 6 (Ornstein-Uhlenbeck Process)
Let us consider a constant diffusion coefficient $\sigma_t = \sigma \geq 0$ and a constant linear drift $u_t(x) = -\theta x$ for $\theta > 0$, yielding the SDE

$$\mathrm{d}X_t = -\theta X_t \mathrm{d}t + \sigma dW_t. \tag{8}$$

A solution $(X_t)_{0 \leq t \leq 1}$ to the above SDE is known as an **Ornstein-Uhlenbeck (OU) process**. We visualize it in fig. 3. The vector field $-\theta x$ pushes the process back to its center 0 (as I always go the inverse direction of where I am), while the diffusion coefficient σ always adds more noise. This process converges towards a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ if we simulate it for $t \to \infty$. Note that for $\sigma = 0$, we have a flow with linear vector field that we have studied in eq. (3).

Simulating an SDE. If you struggle with the abstract definition of an SDE so far, then don't worry about it. A more intuitive way of thinking about SDEs is given by answering the question: How might we simulate an SDE? The simplest such scheme is known as the **Euler-Maruyama method**, and is essentially to SDEs what the Euler method is to ODEs. Using the Euler-Maruyama method, we initialize $X_0 = x_0$ and update iteratively via

$$X_{t+h} = X_t + h u_t(X_t) + \sqrt{h} \sigma_t \epsilon_t, \qquad \epsilon_t \sim \mathcal{N}(0, I_d) \tag{9}$$

where $h = n^{-1} > 0$ is a step size hyperparameter for $n \in \mathbb{N}$. In other words, to simulate using the Euler-Maruyama method, we take a small step in the direction of $u_t(X_t)$ as well as add a little bit of Gaussian noise scaled by $\sqrt{h}\sigma_t$. When simulating SDEs in this class (such as in the accompanying labs), we will usually stick to the Euler-Maruyama method.

Diffusion Models. We can now construct a generative model via an SDE in the same way as we did for ODEs. Remember that our goal was to convert a simple distribution p_{init} into a complex distribution p_{data}. Like for ODEs, the simulation of an SDE randomly initialized with $X_0 \sim p_{\text{init}}$ is a natural choice for this transformation. To parameterize this SDE, we can simply parameterize its central ingredient - the vector field u_t - a neural network

Algorithm 2 Sampling from a Diffusion Model (Euler-Maruyama method)

Require: Neural network u_t^θ, number of steps n, diffusion coefficient σ_t
Set $t = 0$
Set step size $h = \frac{1}{n}$
Draw a sample $X_0 \sim p_{\text{init}}$
for $i = 1, \dots, n-1$ **do**
  Draw a sample $\epsilon \sim \mathcal{N}(0, I_d)$
  $X_{t+h} = X_t + h u_t^\theta(X_t) + \sigma_t \sqrt{h} \epsilon$
  Update $t \leftarrow t + h$
end for
return X_1

u_t^θ. A **diffusion model** is thus given by

$$\mathrm{d}X_t = u_t^\theta(X_t)\mathrm{d}t + \sigma_t \mathrm{d}W_t \quad \blacktriangleright \text{ SDE}$$
$$X_0 \sim p_{\text{init}} \quad \blacktriangleright \text{ random initialization}$$

In algorithm 2, we describe the procedure by which to sample from a diffusion model with the Euler-Maruyama method. We summarize the results of this section as follows.

Summary 7 (SDE generative model)
Throughout this document, a **diffusion model** consists of a neural network u_t^θ with parameters θ that parameterize a vector field and a fixed diffusion coefficient σ_t:

$$\textbf{Neural network: } u^\theta : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d,\ (x,t) \mapsto u_t^\theta(x) \text{ with parameters } \theta$$
$$\textbf{Fixed: } \sigma_t : [0,1] \to [0,\infty),\ t \mapsto \sigma_t$$

To obtain samples from our SDE model (i.e. generate objects), the procedure is as follows:

Initialization: $X_0 \sim p_{\text{init}}$ ▶ Initialize with simple distribution, e.g. a Gaussian
Simulation: $\mathrm{d}X_t = u_t^\theta(X_t)\mathrm{d}t + \sigma_t \mathrm{d}W_t$ ▶ Simulate SDE from 0 to 1
Goal: $X_1 \sim p_{\text{data}}$ ▶ Goal is to make X_1 have distribution p_{data}

A diffusion model with $\sigma_t = 0$ is a **flow model**.

3 Constructing the Training Target

In the previous section, we constructed flow and diffusion models where we obtain trajectories $(X_t)_{0 \leq t \leq 1}$ by simulating the ODE/SDE

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = u_t^\theta(X_t)\mathrm{d}t \qquad \text{(Flow model)} \tag{10}$$

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = u_t^\theta(X_t)\mathrm{d}t + \sigma_t \mathrm{d}W_t \qquad \text{(Diffusion model)} \tag{11}$$

where u_t^θ is a neural network and σ_t is a fixed diffusion coefficient. Naturally, if we just randomly initialize the parameters θ of our neural network u_t^θ, simulating the ODE/SDE will just produce nonsense. As always in machine learning, we need to train the neural network. We accomplish this by minimizing a loss function $\mathcal{L}(\theta)$, such as the **mean-squared error**

$$\mathcal{L}(\theta) = \| u_t^\theta(x) - \underbrace{u_t^{\text{target}}(x)}_{\text{training target}} \|^2,$$

where $u_t^{\text{target}}(x)$ is the **training target** that we would like to approximate. To derive a training algorithm, we proceed in two steps: In this chapter, our goal is to **find an equation for the training target** u_t^{target}. In the next chapter, we will describe a training algorithm that approximates the training target u_t^{target}. Naturally, like the neural network u_t^θ, the training target should itself be a vector field $u_t^{\text{target}} : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$. Further, u_t^{target} should do what we want u_t^θ to do: convert noise into data. **Therefore, the goal of this chapter is to derive a formula for the training target u_t^{ref} such that the corresponding ODE/SDE converts p_{init} into p_{data}.** Along the way we will encounter two fundamental results from physics and stochastic calculus: the **continuity equation** and the **Fokker-Planck equation**. As before, we will first describe the key ideas for ODEs before generalizing them to SDEs.

Remark 8

There are a number of different approaches to deriving a training target for flow and diffusion models. The approach we present here is both the most general and arguably most simple and is in line with recent state-of-the-art models. However, it might well differ from other, older presentations of diffusion models you have seen. Later, we will discuss alternative formulations.

Figure 4: Gradual interpolation from noise to data via a Gaussian conditional probability path for a collection of images.

3.1 Conditional and Marginal Probability Path

The first step of constructing the training target u_t^{target} is by specifying a **probability path**. Intuitively, a probability path specifies a gradual interpolation between noise p_{init} and data p_{data} (see fig. 4). We explain the construction in this section. In the following, for a data point $z \in \mathbb{R}^d$, we denote with δ_z the **Dirac delta** "distribution". This is the simplest distribution that one can imagine: sampling from δ_z always returns z (i.e. it is deterministic). A **conditional (interpolating) probability path** is a set of distribution $p_t(x|z)$ over $\mathbb{R}^d$ such that:

$$p_0(\cdot|z) = p_{\text{init}}, \quad p_1(\cdot|z) = \delta_z \quad \text{for all } z \in \mathbb{R}^d. \tag{12}$$

In other words, a conditional probability path gradually converts a *single* data point into the distribution p_{init} (see e.g. fig. 4). You can think of a probability path as a trajectory in the space of distributions. Every conditional probability path $p_t(x|z)$ induces a **marginal probability path** $p_t(x)$ defined as the distribution that we obtain by first sampling a data point $z \sim p_{\text{data}}$ from the data distribution and then sampling from $p_t(\cdot|z)$:

$$z \sim p_{\text{data}}, \quad x \sim p_t(\cdot|z) \quad \Rightarrow x \sim p_t \qquad \blacktriangleright \text{ sampling from marginal path} \tag{13}$$

$$p_t(x) = \int p_t(x|z) p_{\text{data}}(z) \mathrm{d}z \qquad \blacktriangleright \text{ density of marginal path} \tag{14}$$

Note that we know how to sample from p_t but we don't know the density values $p_t(x)$ as the integral is intractable. Check for yourself that because of the conditions on $p_t(\cdot|z)$ in eq. (12), the marginal probability path p_t interpolates between p_{init} and p_{data}:

$$p_0 = p_{\text{init}} \quad \text{and} \quad p_1 = p_{\text{data}}. \qquad \blacktriangleright \text{ noise-data interpolation} \tag{15}$$

Example 9 (Gaussian Conditional Probability Path)
One particularly popular probability path is the **Gaussian probability path**. This is the **probability path used by denoising diffusion models**. Let α_t, β_t be **noise schedulers**: two continuously differentiable, monotonic functions with $\alpha_0 = \beta_1 = 0$ and $\alpha_1 = \beta_0 = 1$. We then define the conditional probability path

$$p_t(\cdot|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d) \qquad \blacktriangleright \text{ Gaussian conditional path} \tag{16}$$

which, by the conditions we imposed on α_t and β_t, fulfills

$$p_0(\cdot|z) = \mathcal{N}(\alpha_0 z, \beta_0^2 I_d) = \mathcal{N}(0, I_d), \quad \text{and} \quad p_1(\cdot|z) = \mathcal{N}(\alpha_1 z, \beta_1^2 I_d) = \delta_z,$$

where we have used the fact that a normal distribution with zero variance and mean z is just δ_z. Therefore, this choice of $p_t(x|z)$ fulfills eq. (12) for $p_{\text{init}} = \mathcal{N}(0, I_d)$ and is therefore a valid conditional interpolating path. The Gaussian conditional probability path has several useful properties which makes it especially amenable to our goals, and because of this we will use it as our prototypical example of a conditional probability path for the rest of the section. In fig. 4, we illustrate its application to an image. We can express sampling from the

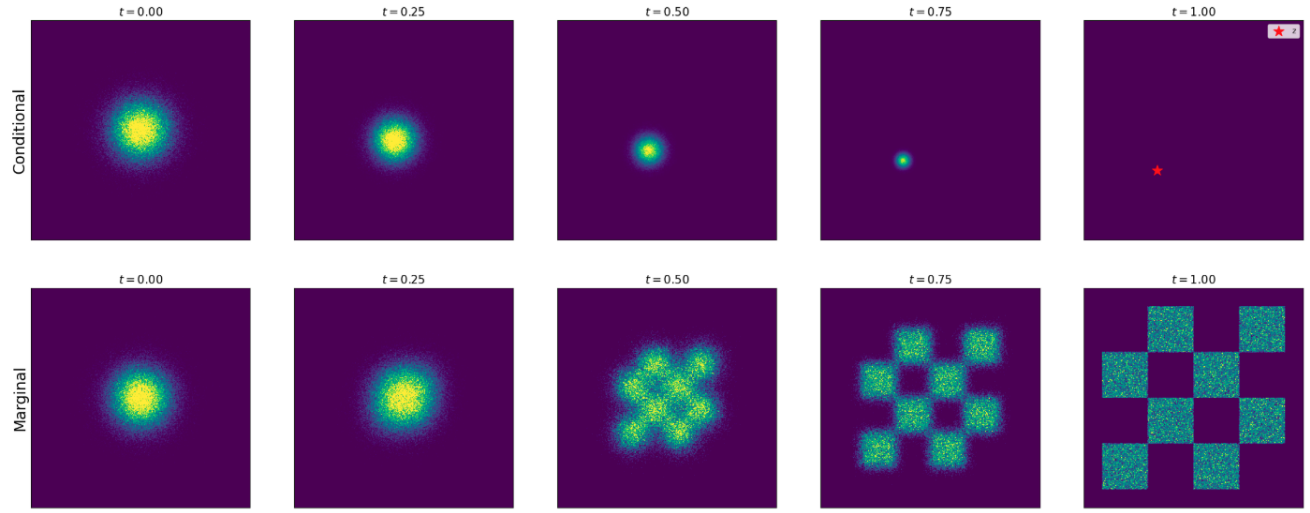

Figure 5: Illustration of a conditional (top) and marginal (bottom) probability path. Here, we plot a Gaussian probability path with $\alpha_t = t, \beta_t = 1 - t$. The conditional probability path interpolates a Gaussian $p_{\text{init}} = \mathcal{N}(0, I_d)$ and $p_{\text{data}} = \delta_z$ for single data point z. The marginal probability path interpolates a Gaussian and a data distribution p_{data} (Here, p_{data} is a toy distribution in dimension $d = 2$ represented by a chess board pattern.)

marginal path p_t as:

$$z \sim p_{\text{data}},\ \epsilon \sim p_{\text{init}} = \mathcal{N}(0, I_d) \ \Rightarrow\ x = \alpha_t z + \beta_t \epsilon \sim p_t \qquad \blacktriangleright \text{ sampling from marginal Gaussian path} \tag{17}$$

Intuitively, the above procedure adds more noise for lower t until time $t = 0$, at which point there is only noise. In fig. 5, we plot an example of such an interpolating path between Gaussian noise and a simple data distribution.

3.2 Conditional and Marginal Vector Fields

We proceed now by constructing a training target u_t^{target} for a flow model using the recently defined notion of a probability path p_t. The idea is to construct u_t^{target} from simple components that we can derive analytically by hand.

Theorem 10 (Marginalization trick)
For every data point $z \in \mathbb{R}^d$, let $u_t^{\text{target}}(\cdot|z)$ denote a **conditional vector field**, defined so that the corresponding ODE yields the conditional probability path $p_t(\cdot|z)$, viz.,

$$X_0 \sim p_{\text{init}}, \quad \frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t^{\text{target}}(X_t|z) \quad \Rightarrow \quad X_t \sim p_t(\cdot|z) \quad (0 \le t \le 1). \tag{18}$$

Then the **marginal vector field** $u_t^{\text{target}}(x)$, defined by

$$u_t^{\text{target}}(x) = \int u_t^{\text{target}}(x|z) \frac{p_t(x|z) p_{\text{data}}(z)}{p_t(x)} \mathrm{d}z, \tag{19}$$

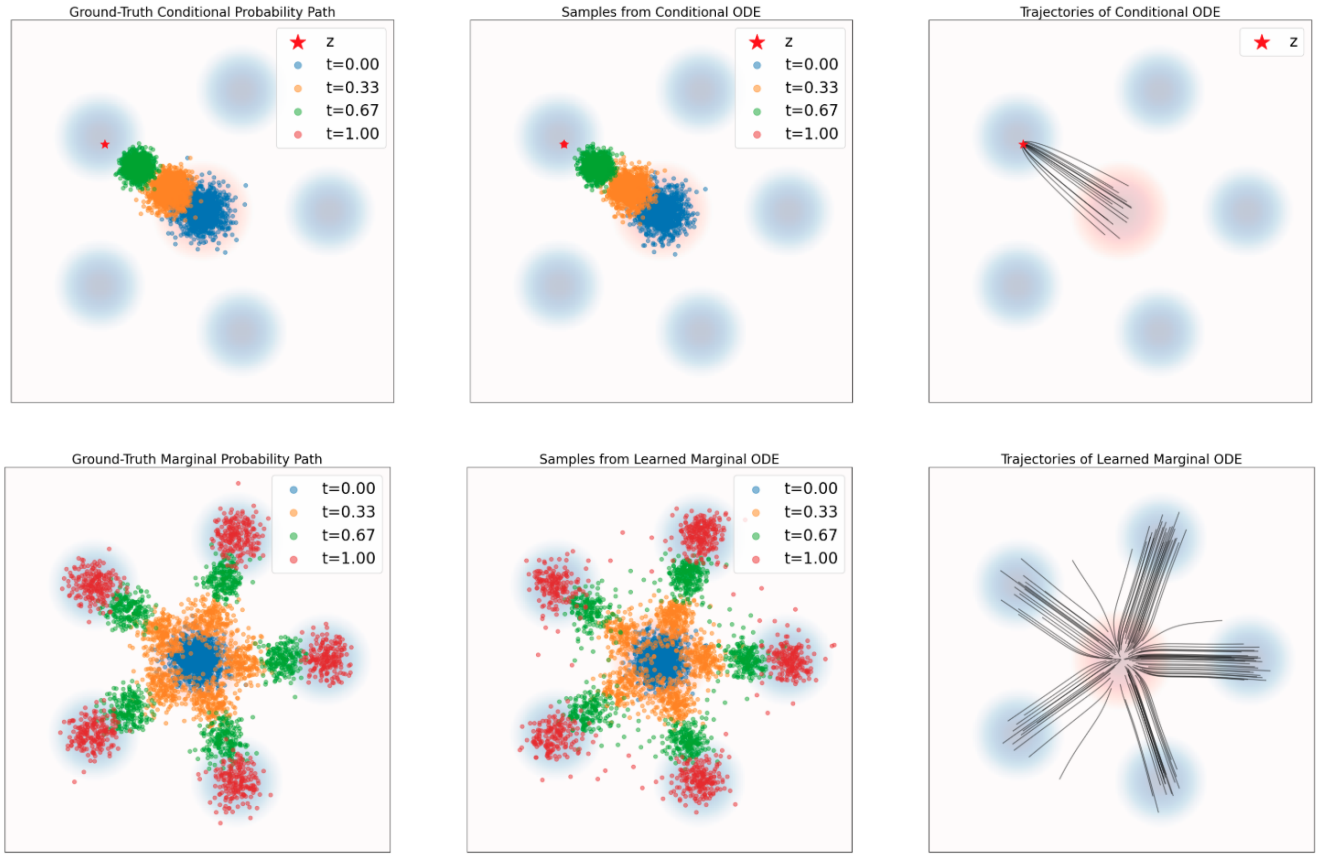

Figure 6: Illustration of theorem 10. Simulating a probability path with ODEs. Data distribution p_{data} in blue background. Gaussian p_{init} in red background. Top row: Conditional probability path. Left: Ground truth samples from conditional path $p_t(\cdot|z)$. Middle: ODE samples over time. Right: Trajectories by simulating ODE with $u_t^{\text{target}}(x|z)$ in eq. (21). Bottom row: Simulating a marginal probability path. Left: Ground truth samples from p_t. Middle: ODE samples over time. Right: Trajectories by simulating ODE with marginal vector field $u_t^{\text{flow}}(x)$. As one can see, the conditional vector field follows the conditional probability path and the marginal vector field follows the marginal probability path.

follows the marginal probability path, i.e.

$$X_0 \sim p_{\text{init}}, \quad \frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t^{\text{target}}(X_t) \quad \Rightarrow \quad X_t \sim p_t \quad (0 \leq t \leq 1). \tag{20}$$

In particular, $X_1 \sim p_{\text{data}}$ for this ODE, so that we might say "u_t^{target} converts noise p_{init} into data p_{data}".

See fig. 6 for an illustration. Before we prove the marginalization trick, let us first explain why it is useful: The marginalization trick from theorem 10 allows us to construct the marginal vector field from a conditional vector field. This simplifies the problem of finding a formula for a training target significantly as we can often find a conditional vector field $u_t^{\text{target}}(\cdot|z)$ satisfying eq. (18) analytically by hand (i.e. by just doing some algebra ourselves). Let us illustrate this by deriving a conditional vector field $u_t(x|z)$ for our running example of a Gaussian probability path.

Example 11 (Target ODE for Gaussian probability paths)
As before, let $p_t(\cdot|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$ for noise schedulers α_t, β_t (see eq. (16)). Let $\dot{\alpha}_t = \partial_t \alpha_t$ and $\dot{\beta}_t = \partial_t \beta_t$ denote respective time derivatives of α_t and β_t. Here, we want to show that the **conditional Gaussian vector field** given by

$$u_t^{\text{target}}(x|z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x \tag{21}$$

is a valid conditional vector field model in the sense of theorem 10: its ODE trajectories X_t satisfy $X_t \sim p_t(\cdot|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$ if $X_0 \sim \mathcal{N}(0, I_d)$. In fig. 6, we confirm this visually by comparing samples from the conditional probability path (ground truth) to samples from simulated ODE trajectories of this flow. As you can see, the distribution match. We will now prove this.

Proof. Let us construct a conditional flow model $\psi_t^{\text{target}}(x|z)$ first by defining

$$\psi_t^{\text{target}}(x|z) = \alpha_t z + \beta_t x. \tag{22}$$

If X_t is the ODE trajectory of $\psi_t^{\text{target}}(\cdot|z)$ with $X_0 \sim p_{\text{init}} = \mathcal{N}(0, I_d)$, then by definition

$$X_t = \psi_t^{\text{target}}(X_0|z) = \alpha_t z + \beta_t X_0 \sim \mathcal{N}(\alpha_t z, \beta^2 I_d) = p_t(\cdot|z).$$

We conclude that the trajectories are distributed like the conditional probability path (i.e, eq. (18) is fulfilled). It remains to extract the vector field $u_t^{\text{target}}(x|z)$ from $\psi_t^{\text{target}}(x|z)$. By the definition of a flow (eq. (2b)), it holds

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}\psi_t^{\text{target}}(x|z) = u_t^{\text{target}}(\psi_t^{\text{target}}(x|z)|z) \quad \text{for all } x, z \in \mathbb{R}^d \\
\overset{(i)}{\Leftrightarrow} \quad &\dot{\alpha}_t z + \dot{\beta}_t x = u_t^{\text{target}}(\alpha_t z + \beta_t x|z) \quad \text{for all } x, z \in \mathbb{R}^d \\
\overset{(ii)}{\Leftrightarrow} \quad &\dot{\alpha}_t z + \dot{\beta}_t \left(\frac{x - \alpha_t z}{\beta_t}\right) = u_t^{\text{target}}(x|z) \quad \text{for all } x, z \in \mathbb{R}^d \\
\overset{(iii)}{\Leftrightarrow} \quad &\left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x = u_t^{\text{target}}(x|z) \quad \text{for all } x, z \in \mathbb{R}^d
\end{aligned}$$

where in (i) we used the definition of $\psi_t^{\text{target}}(x|z)$ (eq. (22)), in (ii) we reparameterized $x \to (x - \alpha_t z)/\beta_t$, and in (iii) we just did some algebra. Note that the last equation is the conditional Gaussian vector field as we defined in eq. (21). This proves the statement.[a] □

[a] One can also double check this by plugging it into the continuity equation introduced later in this section.

The remainder of this section will now prove theorem 10 via the **continuity equation**, a fundamental result in mathematics and physics. To explain it, we will require the use of the **divergence** operator div, which we define as

$$\text{div}(v_t)(x) = \sum_{i=1}^{d} \frac{\partial}{\partial x_i} v_t(x) \tag{23}$$

Theorem 12 (Continuity Equation)
Let us consider an flow model with vector field u_t^{target} with $X_0 \sim p_{\text{init}}$. Then $X_t \sim p_t$ for all $0 \leq t \leq 1$ if and only if

$$\partial_t p_t(x) = -\text{div}(p_t u_t^{\text{target}})(x) \quad \text{for all } x \in \mathbb{R}^d, 0 \leq t \leq 1, \tag{24}$$

where $\partial_t p_t(x) = \frac{\mathrm{d}}{\mathrm{d}t} p_t(x)$ denotes the time-derivative of $p_t(x)$. Equation 24 is known as the **continuity equation**.

For the mathematically-inclined reader, we present a self-contained proof of the Continuity Equation in appendix B. Before we move on, let us try and understand intuitively the continuity equation. The left-hand side $\partial_t p_t(x)$ describes how much the probability $p_t(x)$ at x changes over time. Intuitively, the change should correspond to the net inflow of probability mass. For a flow model, a particle X_t follows along the vector field u_t^{target}. As you might recall from physics, the divergence measures a sort of net outflow from the vector field. Therefore, the negative divergence measures the net inflow. Scaling this by the total probability mass currently residing at x, we get that the net $-\text{div}(p_t u_t)$ measures the total inflow of probability mass. Since probability mass is conserved, the left-hand and right-hand side of the equation should be the same! We now proceed with a proof of the marginalization trick from theorem 10.

Proof. By theorem 12, we have to show that the marginal vector field u_t^{target}, as defined as in eq. (19), satisfies the continuity equation. We can do this by direct calculation:

$$\begin{aligned}
\partial_t p_t(x) &\overset{(i)}{=} \partial_t \int p_t(x|z) p_{\text{data}}(z) \mathrm{d}z \\
&= \int \partial_t p_t(x|z) p_{\text{data}}(z) \mathrm{d}z \\
&\overset{(ii)}{=} \int -\text{div}(p_t(\cdot|z) u_t^{\text{target}}(\cdot|z))(x) p_{\text{data}}(z) \mathrm{d}z \\
&\overset{(iii)}{=} -\text{div}\left(\int p_t(x|z) u_t^{\text{target}}(x|z) p_{\text{data}}(z) \mathrm{d}z \right) \\
&\overset{(iv)}{=} -\text{div}\left(p_t(x) \int u_t^{\text{target}}(x|z) \frac{p_t(x|z) p_{\text{data}}(z)}{p_t(x)} \mathrm{d}z \right)(x) \\
&\overset{(v)}{=} -\text{div}\left(p_t u_t^{\text{target}} \right)(x),
\end{aligned}$$

where in (i) we used the definition of $p_t(x)$ in eq. (13), in (ii) we used the continuity equation for the conditional probability path $p_t(\cdot|z)$, in (iii) we swapped the integral and divergence operator using eq. (23), in (iv) we multiplied and divided by $p_t(x)$, and in (v) we used eq. (19). The beginning and end of the above chain of equations show that the continuity equation is fulfilled for u_t^{target}. By theorem 12, this is enough to imply eq. (20), and we are done. □

3.3 Conditional and Marginal Score Functions

We just successfully constructed a training target for a flow model. We now extend this reasoning to SDEs. To do so, let us define the **marginal score function** of p_t as $\nabla \log p_t(x)$. We can use this to extend the ODE from the

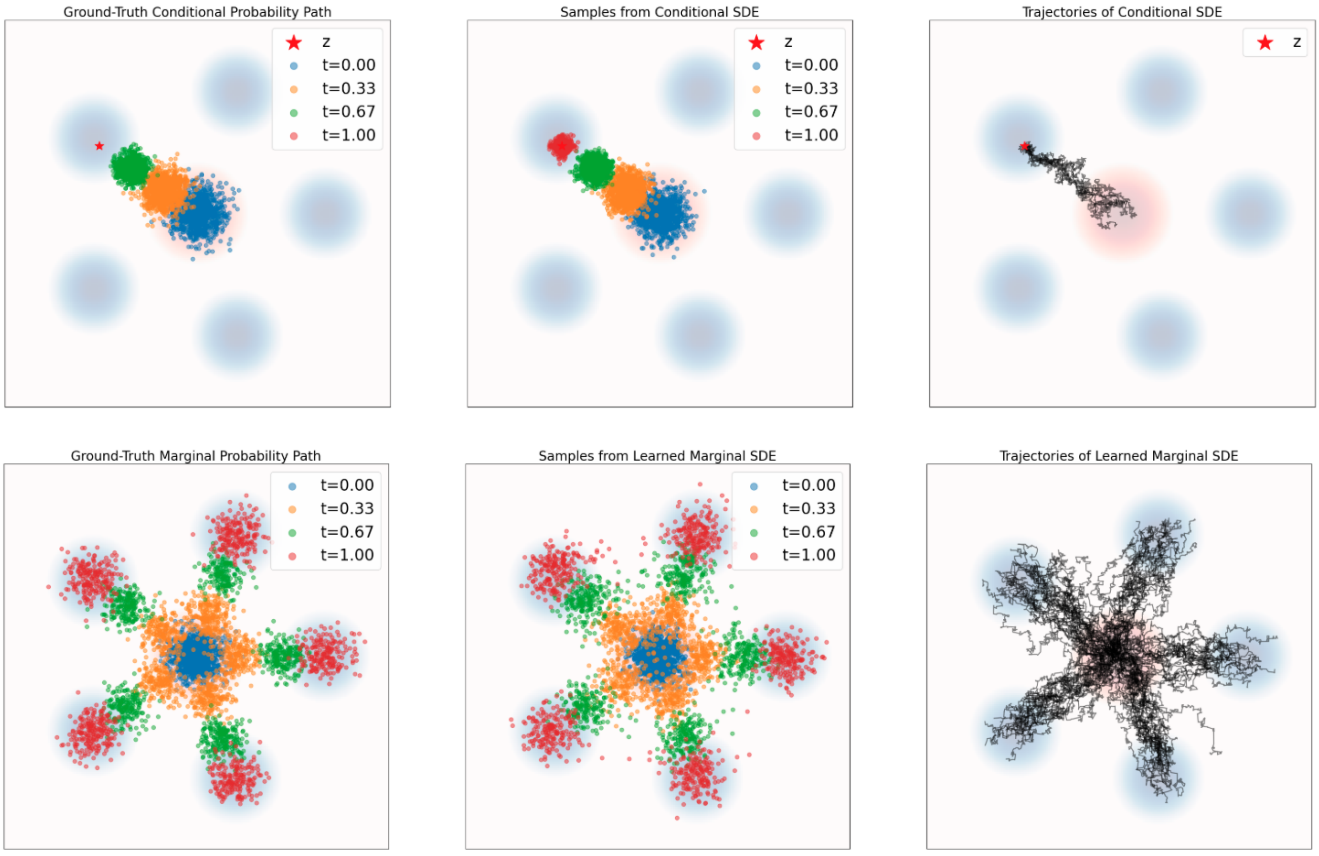

Figure 7: Illustration of theorem 13. Simulating a probability path with SDEs. This repeats the plots from fig. 6 with SDE sampling using eq. (25). Data distribution p_{data} in blue background. Gaussian p_{init} in red background. Top row: Conditional path. Bottom row: Marginal probability path. As one can see, the SDE transports samples from p_{init} into samples from δ_z (for the conditional path) and to p_{data} (for the marginal path).

previous section to an SDE, as the following result demonstrates.

Theorem 13 (SDE extension trick)

Define the conditional and marginal vector fields $u_t^{\text{target}}(x|z)$ and $u_t^{\text{target}}(x)$ as before. Then, for diffusion coefficient $\sigma_t \geq 0$, we may construct an SDE which follows the same probability path:

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = \left[u_t^{\text{target}}(X_t) + \frac{\sigma_t^2}{2}\nabla \log p_t(X_t)\right]\mathrm{d}t + \sigma_t \mathrm{d}W_t \tag{25}$$

$$\Rightarrow X_t \sim p_t \quad (0 \leq t \leq 1) \tag{26}$$

In particular, $X_1 \sim p_{\text{data}}$ for this SDE. The same identity holds if we replace the marginal probability $p_t(x)$ and vector field $u_t^{\text{target}}(x)$ with the conditional probability path $p_t(x|z)$ and vector field $u_t^{\text{target}}(x|z)$.

We illustrate the theorem in fig. 7. The formula in theorem 13 is useful because, similar to before, we can express the marginal score function via the **conditional score function** $\nabla \log p_t(x|z)$

$$\nabla \log p_t(x) = \frac{\nabla p_t(x)}{p_t(x)} = \frac{\nabla \int p_t(x|z) p_{\text{data}}(z) \mathrm{d}z}{p_t(x)} = \frac{\int \nabla p_t(x|z) p_{\text{data}}(z) \mathrm{d}z}{p_t(x)} = \int \nabla \log p_t(x|z) \frac{p_t(x|z) p_{\text{data}}(z)}{p_t(x)} \mathrm{d}z \quad (27)$$

and the conditional score function $\nabla \log p_t(x|z)$ is something we usually know analytically, as illustrated by the following example.

Example 14 (Score Function for Gaussian Probability Paths.)
For the Gaussian path $p_t(x|z) = \mathcal{N}(x; \alpha_t z, \beta_t^2 I_d)$, we can use the form of the Gaussian probability density (see eq. (81)) to get

$$\nabla \log p_t(x|z) = \nabla \log \mathcal{N}(x; \alpha_t z, \beta_t^2 I_d) = -\frac{x - \alpha_t z}{\beta_t^2}. \quad (28)$$

Note that the score is a linear function of x. This is a unique feature of Gaussian distributions.

In the remainder of this section, we will prove theorem 13 via the **Fokker-Planck equation**, which extends the continuity equation from ODEs to SDEs. To do so, let us first define the **Laplacian** operator Δ via

$$\Delta w_t(x) = \sum_{i=1}^{d} \frac{\partial^2}{\partial^2 x_i} w_t(x) = \operatorname{div}(\nabla w_t)(x). \quad (29)$$

Theorem 15 (Fokker-Planck Equation)
Let p_t be a probability path and let us consider the SDE

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = u_t(X_t)\mathrm{d}t + \sigma_t \mathrm{d}W_t.$$

Then X_t has distribution p_t for all $0 \leq t \leq 1$ if and only if the **Fokker-Planck equation** holds:

$$\partial_t p_t(x) = -\operatorname{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2} \Delta p_t(x) \quad \text{for all } x \in \mathbb{R}^d, 0 \leq t \leq 1. \quad (30)$$

A self-contained proof of the Fokker-Planck equation can be found in appendix B. Note that the continuity equation is recovered from the Fokker-Planck equation when $\sigma_t = 0$. The additional Laplacian term Δp_t might be hard to rationalize at first. Those familiar with physics will note that the same term also appears in the heat equation (which is in fact a special case of the Fokker-Planck equation). Heat diffuses through a medium. We also add a diffusion process (not a physical but a mathematical one) and hence we add this additional Laplacian term. Let us now use the Fokker-Planck equation to help us prove theorem 13.

Proof of Theorem 13. By theorem 15, we need to show that that the SDE defined in eq. (25) satisfies the Fokker-

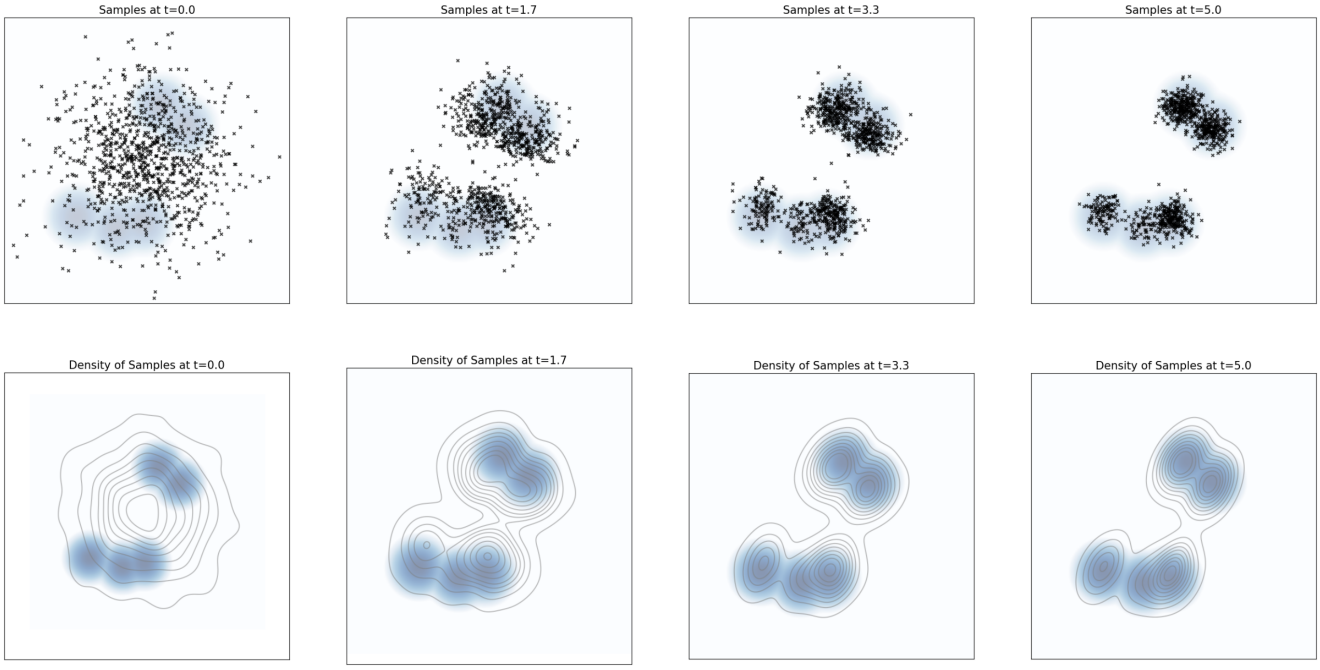

Figure 8: Top row: Particles evolving under the Langevin dynamics given by eq. (31), with $p(x)$ taken to be a Gaussian mixture with 5 modes. Bottom row: A kernel density estimate of the same samples shown in the top row. As one can see, the distribution of samples converges to the equilibrium distribution p (blue background colour).

Planck equation for p_t. We can do this by direction calculation:

$$\begin{aligned}
\partial_t p_t(x) &\overset{(i)}{=} -\operatorname{div}(p_t u_t^{\text{target}})(x) \\
&\overset{(ii)}{=} -\operatorname{div}(p_t u_t^{\text{target}})(x) - \frac{\sigma_t^2}{2}\Delta p_t(x) + \frac{\sigma_t^2}{2}\Delta p_t(x) \\
&\overset{(iii)}{=} -\operatorname{div}(p_t u_t^{\text{target}})(x) - \operatorname{div}(\frac{\sigma_t^2}{2}\nabla p_t)(x) + \frac{\sigma_t^2}{2}\Delta p_t(x) \\
&\overset{(iv)}{=} -\operatorname{div}(p_t u_t^{\text{target}})(x) - \operatorname{div}(p_t \left[\frac{\sigma_t^2}{2}\nabla \log p_t\right])(x) + \frac{\sigma_t^2}{2}\Delta p_t(x) \\
&\overset{(v)}{=} -\operatorname{div}\left(p_t \left[u_t^{\text{target}} + \frac{\sigma_t^2}{2}\nabla \log p_t\right]\right)(x) + \frac{\sigma_t^2}{2}\Delta p_t(x),
\end{aligned}$$

where in (i) we used the Contuity Equation, in (ii) we added and subtracted the same term, in (iii) we used the definition of the Laplacian (eq. (29)), in (iv) we used that $\nabla \log p_t = \frac{\nabla p_t}{p_t}$, and in (v) we used the linearity of the divergence operator. The above derivation shows that the SDE defined in eq. (25) satisfies the Fokker-Planck equation for p_t. By theorem 15, this implies $X_t \sim p_t$ for $0 \leq t \leq 1$, as desired. □

Remark 16 (Langevin dynamics.)
The above construction has a famous special case when the probability path is static, i.e. $p_t = p$ for a fixed

distribution p. In this case, we set $u_t^{\text{target}} = 0$ and obtain the SDE

$$\mathrm{d}X_t = \frac{\sigma_t^2}{2}\nabla \log p(X_t)\mathrm{d}t + \sigma_t dW_t, \tag{31}$$

which is commonly known as **Langevin dynamics**. The fact that p_t is static implies that $\partial_t p_t(x) = 0$. It follows immediately from theorem 13 that these dynamics satisfy the Fokker-Planck equation for the static path $p_t = p$ in theorem 13. Therefore, we may conclude that p is a stationary distribution of Langevin dynamics, so that

$$X_0 \sim p \quad \Rightarrow \quad X_t \sim p \quad (t \geq 0)$$

As with many Markov chains, these dynamics converge to the stationary distribution p under rather general conditions (see section 3.3). That is, if we instead we take $X_0 \sim p' \neq p$, so that $X_t \sim p'_t$, then under mild conditions $p_t \to p$. This fact makes Langevin dynamics extremely useful, and it accordingly serves as the basis for e.g., **molecular dynamics** simulations, and many other Markov chain Monte Carlo (MCMC) methods across Bayesian statistics and the natural sciences.

Let us summarize the results of this section.

Summary 17 (Derivation of the Training Target)
The flow training target is the marginal vector field u_t^{target}. To construct it, we choose a **conditional probability path** $p_t(x|z)$ that fulfils $p_0(\cdot|z) = p_{\text{init}}$, $p_1(\cdot|z) = \delta_z$. Next, we find a **conditional vector field** $u_t^{\text{flow}}(x|z)$ such that its corresponding flow $\psi_t^{\text{target}}(x|z)$ fulfills

$$X_0 \sim p_{\text{init}} \quad \Rightarrow \quad X_t = \psi_t^{\text{target}}(X_0|z) \sim p_t(\cdot|z),$$

or, equivalently, that u_t^{target} satisfies the continuity equation. Then the **marginal vector field** defined by

$$u_t^{\text{target}}(x) = \int u_t^{\text{target}}(x|z)\frac{p_t(x|z)p_{\text{data}}(z)}{p_t(x)}\mathrm{d}z, \tag{32}$$

follows the marginal probability path, i.e.,

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = u_t^{\text{target}}(X_t)\mathrm{d}t \Rightarrow X_t \sim p_t \quad (0 \leq t \leq 1). \tag{33}$$

In particular, $X_1 \sim p_{\text{data}}$ for this ODE, so that u_t^{target} "converts noise into data", as desired.

Extending to SDEs. For a time-dependent diffusion coefficient $\sigma_t \geq 0$, we can extend the above ODE to an SDE with the same marginal probability path:

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = \left[u_t^{\text{target}}(X_t) + \frac{\sigma_t^2}{2}\nabla \log p_t(X_t)\right]\mathrm{d}t + \sigma_t \mathrm{d}W_t \tag{34}$$

$$\Rightarrow X_t \sim p_t \quad (0 \leq t \leq 1), \tag{35}$$

where $\nabla \log p_t(x)$ is the **marginal score function**

$$\nabla \log p_t(x) = \int \nabla \log p_t(x|z)\frac{p_t(x|z)p_{\text{data}}(z)}{p_t(x)}\mathrm{d}z. \tag{36}$$

In particular, for the trajectories X_t of the above SDE, it holds that $X_1 \sim p_{\text{data}}$, so that the SDE "converts noise into data", as desired. An important example is the **Gaussian probability path**, yielding the formulae:

$$p_t(x|z) = \mathcal{N}(x; \alpha_t z, \beta_t^2 I_d) \tag{37}$$

$$u_t^{\text{flow}}(x|z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x \tag{38}$$

$$\nabla \log p_t(x|z) = -\frac{x - \alpha_t z}{\beta_t^2}, \tag{39}$$

for **noise schedulers** $\alpha_t, \beta_t \in \mathbb{R}$: continuously differentiable, monotonic functions such that $\alpha_0 = \beta_1 = 0$ $\alpha_1 = \beta_0 = 1$.

4 Training the Generative Model

In the last two sections, we showed how to construct a generative model with a vector field u_t^θ given by a neural network, and we derived a formula for the training target u_t^{target}. In this section, we will describe how to train the neural network u_t^θ to approximate the training target u_t^{target}. First, we restrict ourselves to ODEs again, in doing so recovering **flow matching**. Second, we explain how to extend the approach to SDEs via **score matching**. Finally, we consider the special case of Gaussian probability paths, in doing so recovering **denoising diffusion models**. With these tools, we will at last have an end-to-end procedure to train and sample from a generative model with ODEs and SDEs.

4.1 Flow Matching

As before, let us consider a flow model given by

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = u_t^\theta(X_t)\,\mathrm{d}t. \qquad \blacktriangleright \text{ flow model} \tag{40}$$

As we learned, we want the neural network u_t^θ to equal the marginal vector field u_t^{target}. In other words, we would like to find parameters θ so that $u_t^\theta \approx u_t^{\text{target}}$. In the following, we denote by $\text{Unif} = \text{Unif}_{[0,1]}$ the uniform distribution on the interval $[0,1]$, and by $\mathbb{E}$ the expected value of a random variable. An intuitive way of obtaining $u_t^\theta \approx u_t^{\text{target}}$ is to use a mean-squared error, i.e. to use the **flow matching loss** defined as

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t\sim\text{Unif}, x\sim p_t}[\|u_t^\theta(x) - u_t^{\text{target}}(x)\|^2] \tag{41}$$

$$\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, x\sim p_t(\cdot|z)}[\|u_t^\theta(x) - u_t^{\text{target}}(x)\|^2], \tag{42}$$

where $p_t(x) = \int p_t(x|z)p_{\text{data}}(z)\mathrm{d}z$ is the marginal probability path and in (i) we used the sampling procedure given by eq. (13). Intuitively, this loss says: First, draw a random time $t \in [0,1]$. Second, draw a random point z from our data set, sample from $p_t(\cdot|z)$ (e.g., by adding some noise), and compute $u_t^\theta(x)$. Finally, compute the mean-squared error between the output of our neural network and the marginal vector field $u_t^{\text{target}}(x)$. Unfortunately, we are *not* done here. While we do know the formula for u_t^{target} by theorem 10

$$u_t^{\text{target}}(x) = \int u_t^{\text{target}}(x|z)\frac{p_t(x|z)p_{\text{data}}(z)}{p_t(x)}\mathrm{d}z, \tag{43}$$

we cannot compute it efficiently because the above integral is intractable. Instead, we will exploit the fact that the **conditional** velocity field $u_t^{\text{target}}(x|z)$ is tractable. To do so, let us define the **conditional flow matching loss**

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{z\sim p_{\text{data}}, x\sim p_t(\cdot|z)}[\|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2]. \tag{44}$$

Note the difference to eq. (41): we use the conditional vector field $u_t^{\text{target}}(x|z)$ instead of the marginal vector $u_t^{\text{target}}(x)$. As we have an analytical formula for $u_t^{\text{target}}(x|z)$, we can minimize the above loss easily. But wait, what sense does it make to regress against the conditional vector field if it's the marginal vector field we care about? As it turns out, by *explicitly* regressing against the tractable, conditional vector field, we are *implicitly* regressing against the intractable, marginal vector field. The next result makes this intuition precise.

Theorem 18

The marginal flow matching loss equals the conditional flow matching loss up to a constant. That is,

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathcal{L}_{\mathrm{CFM}}(\theta) + C,$$

where C is independent of θ. Therefore, their gradients coincide:

$$\nabla_\theta \mathcal{L}_{\mathrm{FM}}(\theta) = \nabla_\theta \mathcal{L}_{\mathrm{CFM}}(\theta).$$

Hence, minimizing $\mathcal{L}_{\mathrm{CFM}}(\theta)$ with e.g., stochastic gradient descent (SGD) is equivalent to minimizing $\mathcal{L}_{\mathrm{FM}}(\theta)$ with in the same fashion. In particular, for the minimizer θ^* of $\mathcal{L}_{\mathrm{CFM}}(\theta)$, it will hold that $u_t^{\theta^*} = u_t^{\mathrm{target}}$ (assuming an infintely expressive parameterization).

Proof. The proof works by expanding the mean-squared error into three components and removing constants:

$$\begin{aligned}
\mathcal{L}_{\mathrm{FM}}(\theta) &\overset{(i)}{=} \mathbb{E}_{t\sim\mathrm{Unif},x\sim p_t}[\|u_t^\theta(x) - u_t^{\mathrm{target}}(x)\|^2] \\
&\overset{(ii)}{=} \mathbb{E}_{t\sim\mathrm{Unif},x\sim p_t}[\|u_t^\theta(x)\|^2 - 2u_t^\theta(x)^T u_t^{\mathrm{target}}(x) + \|u_t^{\mathrm{target}}(x)\|^2] \\
&\overset{(iii)}{=} \mathbb{E}_{t\sim\mathrm{Unif},x\sim p_t}\left[\|u_t^\theta(x)\|^2\right] - 2\mathbb{E}_{t\sim\mathrm{Unif},x\sim p_t}[u_t^\theta(x)^T u_t^{\mathrm{target}}(x)] + \underbrace{\mathbb{E}_{t\sim\mathrm{Unif}_{[0,1]},x\sim p_t}[\|u_t^{\mathrm{target}}(x)\|^2]}_{=:C_1} \\
&\overset{(iv)}{=} \mathbb{E}_{t\sim\mathrm{Unif},z\sim p_{\mathrm{data}},x\sim p_t(\cdot|z)}[\|u_t^\theta(x)\|^2] - 2\mathbb{E}_{t\sim\mathrm{Unif},x\sim p_t}[u_t^\theta(x)^T u_t^{\mathrm{target}}(x)] + C_1
\end{aligned}$$

where (i) holds by definition, in (ii) we used the formula $\|a-b\|^2 = \|a\|^2 - 2a^Tb + \|b\|^2$, in (iii) we define a constant C_1 and in (iv) we used the sampling procedure of p_t given by eq. (13). Let us reexpress the second summand:

$$\begin{aligned}
\mathbb{E}_{t\sim\mathrm{Unif},x\sim p_t}[u_t^\theta(x)^T u_t^{\mathrm{target}}(x)] &\overset{(i)}{=} \int_0^1 \int p_t(x) u_t^\theta(x)^T u_t^{\mathrm{target}}(x)\,\mathrm{d}x\,\mathrm{d}t \\
&\overset{(ii)}{=} \int_0^1 \int p_t(x) u_t^\theta(x)^T \left[\int u_t^{\mathrm{target}}(x|z)\frac{p_t(x|z)p_{\mathrm{data}}(z)}{p_t(x)}\mathrm{d}z\right]\mathrm{d}x\,\mathrm{d}t \\
&\overset{(iii)}{=} \int_0^1 \int\int\int u_t^\theta(x)^T u_t^{\mathrm{target}}(x|z) p_t(x|z) p_{\mathrm{data}}(z)\,\mathrm{d}z\,\mathrm{d}x\,\mathrm{d}t \\
&\overset{(iv)}{=} \mathbb{E}_{t\sim\mathrm{Unif},z\sim p_{\mathrm{data}},x\sim p_t(\cdot|z)}[u_t^\theta(x)^T u_t^{\mathrm{target}}(x|z)]
\end{aligned}$$

where in (i) we expressed the expected value as an integral, in (ii) we use eq. (43), in (iii) we use the fact that integrals are linear, in (iv) we express the integral as an expected value. Note that this was really the crucial step of the proof: The beginning of the equality used the marginal vector field $u_t^{\mathrm{target}}(x)$, while the end uses the

conditional vector field $u_t^{\text{target}}(x|z)$. We plug is into the equation for $\mathcal{L}_{\text{FM}}$ to get:

$$\begin{aligned}
\mathcal{L}_{\text{FM}}(\theta) &\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[\|u_t^\theta(x)\|^2] - 2\mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[u_t^\theta(x)^T u_t^{\text{target}}(x|z)] + C_1 \\
&\overset{(ii)}{=} \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[\|u_t^\theta(x)\|^2 - 2u_t^\theta(x)^T u_t^{\text{target}}(x|z) + \|u_t^{\text{target}}(x|z)\|^2 - \|u_t^{\text{target}}(x|z)\|^2] + C_1 \\
&\overset{(iii)}{=} \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[\|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2] + \underbrace{\mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[-\|u_t^{\text{target}}(x|z)\|^2]}_{C_2} + C_1 \\
&\overset{(iv)}{=} \mathcal{L}_{\text{CFM}}(\theta) + \underbrace{C_2 + C_1}_{=:C}
\end{aligned}$$

where in (i) we plugged in the derived equation, in (ii) we added and subtracted the same value, in (iii) we used the formula $\|a-b\|^2 = \|a\|^2 - 2a^Tb + \|b\|^2$ again, and in (iv) we defined a constant in θ. This finishes the proof. □

Once u_t^θ has been trained, we may simulate the flow model

$$\mathrm{d}X_t = u_t^\theta(X_t)\,\mathrm{d}t, \qquad X_0 \sim p_{\text{init}} \tag{45}$$

via e.g., algorithm 1 to obtain samples $X_1 \sim p_{\text{data}}$. This whole pipeline is called **flow matching** in the literature [14, 16, 1, 15]. The training procedure is summarized in algorithm 5 and visualized in fig. 9. Let us now instantiate the conditional flow matching loss for the choice of Gaussian probability paths:

Example 19 (Flow Matching for Gaussian Conditional Probability Paths)
Let us return to the example of Gaussian probability paths $p_t(\cdot|z) = \mathcal{N}(\alpha_t z; \beta_t^2 I_d)$, where we may sample from the conditional path via

$$\epsilon \sim \mathcal{N}(0, I_d) \quad \Rightarrow \quad x_t = \alpha_t z + \beta_t \epsilon \sim \mathcal{N}(\alpha_t z, \beta_t^2 I_d) = p_t(\cdot|z). \tag{46}$$

As we derived in eq. (21), the conditional vector field $u_t^{\text{target}}(x|z)$ is given by

$$u_t^{\text{target}}(x|z) = \left(\dot\alpha_t - \frac{\dot\beta_t}{\beta_t}\alpha_t\right) z + \frac{\dot\beta_t}{\beta_t} x, \tag{47}$$

where $\dot\alpha_t = \partial_t \alpha_t$ and $\dot\beta_t = \partial_t \beta_t$ are the respective time derivatives. Plugging in this formula, the conditional flow matching loss reads

$$\begin{aligned}
\mathcal{L}_{\text{CFM}}(\theta) &= \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim\mathcal{N}(\alpha_t z,\beta_t^2 I_d)}[\|u_t^\theta(x) - \left(\dot\alpha_t - \frac{\dot\beta_t}{\beta_t}\alpha_t\right) z - \frac{\dot\beta_t}{\beta_t} x\|^2] \\
&\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},\epsilon\sim\mathcal{N}(0,I_d)}[\|u_t^\theta(\alpha_t z + \beta_t\epsilon) - (\dot\alpha_t z + \dot\beta_t \epsilon)\|^2]
\end{aligned}$$

where in (i) we plugged in eq. (46) and replaced x by $\alpha_t z + \beta_t \epsilon$. Note the simplicity of $\mathcal{L}_{\text{CFM}}$: We sample a data point z, sample some noise ϵ and then we take a mean squared error. Let us make this even more concrete for the special case of $\alpha_t = t$, and $\beta_t = 1-t$. The corresponding probability $p_t(x|z) = \mathcal{N}(tz, (1-t)^2)$ is sometimes

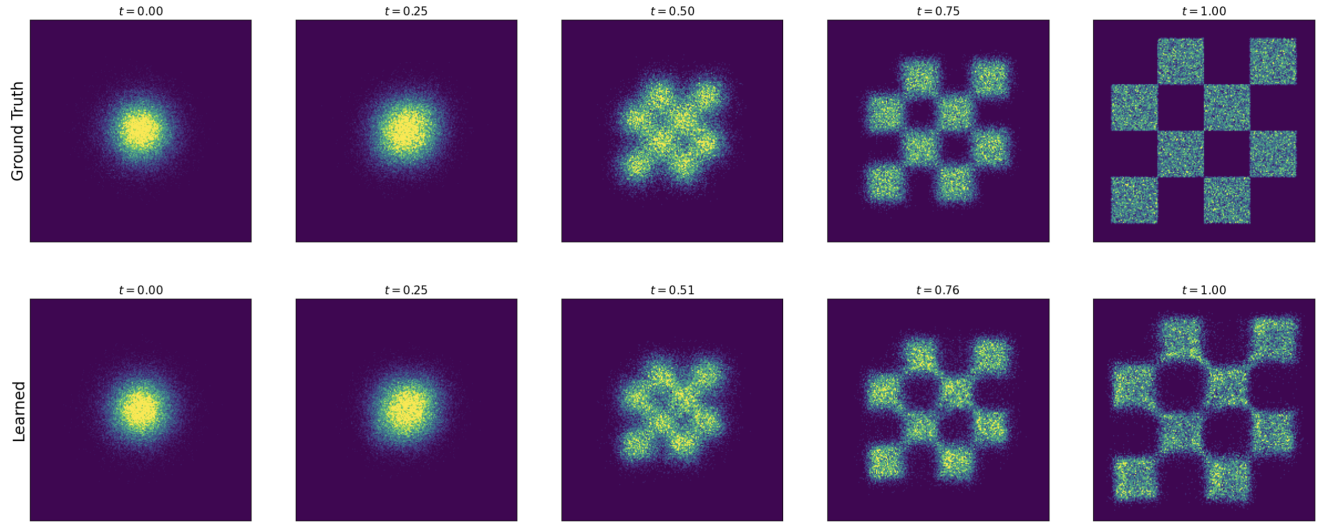

Figure 9: Illustration of theorem 18 with a Gaussian CondOT probability path: simulating an ODE from a trained flow matching model. The data distribution is the chess board pattern (top right). Top row: Histogram from ground truth marginal probability path $p_t(x)$. Bottom row: Histogram of samples from flow matching model. As one can see, the top row and bottom row match after training (up to training error). The model was trained using algorithm 5.

referred to as the (Gaussian) **CondOT probability path**. Then we have $\dot{\alpha}_t = 1, \dot{\beta}_t = -1$, so that

$$\mathcal{L}_{\text{cfm}}(\theta) = \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, \epsilon\sim\mathcal{N}(0,I_d)}[\|u_t^\theta(tz + (1-t)\epsilon) - (z-\epsilon)\|^2]$$

Many famous state-of-the-art models have been trained using this simple yet effective procedure, e.g. *Stable Diffusion 3*, Meta's *Movie Gen Video*, and probably many more proprietary models. In fig. 9, we visualize it in a simple example and in algorithm 5 we summarize the training procedure.

Algorithm 3 Flow Matching Training Procedure (here for Gaussian CondOT path $p_t(x|z) = \mathcal{N}(tz, (1-t)^2)$)

Require: A dataset of samples $z \sim p_{\text{data}}$, neural network u_t^θ

1: **for** each mini-batch of data **do**
2: Sample a data example z from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$.
4: Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$
5: Set $x = tz + (1-t)\epsilon$ (General case: $x \sim p_t(\cdot|z)$)
6: Compute loss

$$\mathcal{L}(\theta) = \|u_t^\theta(x) - (z-\epsilon)\|^2 \qquad \text{(General case: } = \|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2\text{)}$$

7: Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$.
8: **end for**

4.2 Score Matching

Let us extend the algorithm we just found from ODEs to SDEs. Remember we can extend the target ODE to an SDE with the same marginal distribution given by

$$\mathrm{d}X_t = \left[u_t^{\text{target}}(X_t) + \frac{\sigma_t^2}{2}\nabla \log p_t(X_t)\right]\mathrm{d}t + \sigma_t \mathrm{d}W_t \tag{48}$$

$$X_0 \sim p_{\text{init}}, \tag{49}$$

$$\Rightarrow X_t \sim p_t \quad (0 \le t \le 1) \tag{50}$$

where u_t^{target} is the marginal vector field and $\nabla \log p_t(x)$ is the **marginal score** function represented via the formula

$$\nabla \log p_t(x) = \int \nabla \log p_t(x|z) \frac{p_t(x|z)p_{\text{data}}(z)}{p_t(x)}\,\mathrm{d}z. \tag{51}$$

To approximate the marginal score $\nabla \log p_t$, we can use a neural network that we call **score network** $s_t^\theta : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$. In the same way as before, we can design a **score matching** loss and a **conditional score matching** loss:

$$\mathcal{L}_{\text{SM}}(\theta) = \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, x\sim p_t(\cdot|z)}[\|s_t^\theta(x) - \nabla \log p_t(x)\|^2] \quad \blacktriangleright \text{ score matching loss}$$

$$\mathcal{L}_{\text{CSM}}(\theta) = \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, x\sim p_t(\cdot|z)}[\|s_t^\theta(x) - \nabla \log p_t(x|z)\|^2] \quad \blacktriangleright \text{ conditional score matching loss}$$

where again the difference is using the marginal score $\nabla \log p_t(x)$ vs. using the conditional score $\nabla \log p_t(x|z)$. As before, we ideally would want to minimize the score matching loss but can't because we don't know $\nabla \log p_t(x)$. But similarly as before, the conditional score matching loss is a tractable alternative:

Theorem 20

The score matching loss equals the conditional score matching loss up to a constant:

$$\mathcal{L}_{\text{SM}}(\theta) = \mathcal{L}_{\text{CSM}}(\theta) + C,$$

where C is independent of parameters θ. Therefore, their gradients coincide:

$$\nabla_\theta \mathcal{L}_{\text{SM}}(\theta) = \nabla_\theta \mathcal{L}_{\text{CSM}}(\theta).$$

In particular, for the minimizer θ^*, it will hold that $s_t^{\theta^*} = \nabla \log p_t$.

Proof. Note that the formula for $\nabla \log p_t$ (eq. (51)) looks the same as the formula for u_t^{target} (eq. (43)). Therefore, the proof is identical to the proof of theorem 18 replacing u_t^{target} with $\nabla \log p_t$. □

The above procedure describes the vanilla procedure of training a diffusion model. After training, we can choose an arbitrary diffusion coefficient $\sigma_t \ge 0$ and then simulate the SDE

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = \left[u_t^\theta(X_t) + \frac{\sigma_t^2}{2}s_t^\theta(X_t)\right]\mathrm{d}t + \sigma_t \mathrm{d}W_t, \tag{52}$$

to generate samples $X_1 \sim p_{\text{data}}$. In theory, every σ_t should give samples $X_1 \sim p_{\text{data}}$ at perfect training. In practice, we encounter two types of errors: (1) numerical errors by simulating the SDE imperfectly and (2) training errors

(i.e., the model u_t^θ is not exactly equal to u_t^{target}). Therefore, there is an optimal unknown noise level σ_t - this can be determined empirically by just testing our different values of empirically (see e.g. [1, 12, 17]). At first sight, it might seem to be a disadvantage that we have to learn both s_t^θ and u_t^θ if we wanted to use diffusion model now as opposed to a flow model. However, note we can often directly s_t^θ and u_t^θ in a single network with two outputs, so that the additional computational effort is usually minimal. Further, as we will see now for the special case of the Gaussian probability path, s_t^θ and u_t^θ may be converted into one another so that we don't have to train them separately.

Remark 21 (Denoising Diffusion Models)
If you are familiar with diffusion models, you have probably encountered the term ***denoising* diffusion model**. This model has become so popular that most people nowadays drop the word "denoising" and simply use the term "diffusion model" to describe it. In the language of this document, these are simply diffusion models with Gaussian probability paths $p_t(\cdot|z) = \mathcal{N}(\alpha_t z; \beta_t^2 I_d)$. However, it is important to note that this might not be immediately obvious if you read some of the first diffusion model papers: they use a different time convention (time is inverted) - so you need apply an appropriate time re-scaling - and they construct their probability path via so-called **forward processes** (we will discuss this in section 4.3).

Example 22 (Denoising Diffusion Models: Score Matching for Gaussian Probability Paths)
First, let us instantiate the denoising score matching loss for the case of $p_t(x|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$. As we derived in eq. (28), the conditional score $\nabla \log p_t(x|z)$ has the formula

$$\nabla \log p_t(x|z) = -\frac{x - \alpha_t z}{\beta_t^2}. \tag{53}$$

Plugging in this formula, the conditional score matching loss becomes:

$$\begin{aligned}
\mathcal{L}_{\text{CSM}}(\theta) &= \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, x\sim p_t(\cdot|z)}[\|s_t^\theta(x) + \frac{x - \alpha_t z}{\beta_t^2}\|^2] \\
&\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, \epsilon\sim\mathcal{N}(0,I_d)}[\|s_t^\theta(\alpha_t z + \beta_t \epsilon) + \frac{\epsilon}{\beta_t}\|^2] \\
&= \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, \epsilon\sim\mathcal{N}(0,I_d)}[\frac{1}{\beta_t^2}\|\beta_t s_t^\theta(\alpha_t z + \beta_t \epsilon) + \epsilon\|^2]
\end{aligned}$$

where in (i) we plugged in eq. (46) and replaced x by $\alpha_t z + \beta_t \epsilon$. Note that the network s_t^θ essentially learns to predict the noise that was used to corrupt a data sample z. Therefore, the above training loss is also called **denoising score matching** and it was the one of the first procedures used to learn diffusion models. It was soon realized that the above loss is numerically unstable for $\beta_t \approx 0$ close to zero (i.e. denoising score matching only works if you add a sufficient amount of noise). In some of the first works on denoising diffusion models (see **Denoising Diffusion Probabilitic Models**, [9]) it was therefore proprosed to drop the constant $\frac{1}{\beta_t^2}$ in the loss and reparameterize s_t^θ into a **noise predictor** network $\epsilon_t^\theta : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ via:

$$-\beta_t s_t^\theta(x) = \epsilon_t^\theta(x) \quad \Rightarrow \quad \mathcal{L}_{\text{DDPM}}(\theta) = \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, \epsilon\sim\mathcal{N}(0,I_d)}\left[\|\epsilon_t^\theta(\alpha_t z + \beta_t \epsilon) - \epsilon\|^2\right]$$

As before, the network ϵ_t^θ essentially learns to predict the noise that was used to corrupt a data sample z. In algorithm 4, we summarize the training procedure.

Algorithm 4 Score Matching Training Procedure for Gaussian probability path

Require: A dataset of samples $z \sim p_{\text{data}}$, score network s_t^θ or noise predictor ϵ_t^θ

1: **for** each mini-batch of data **do**
2: Sample a data example z from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$.
4: Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$
5: Set $x_t = \alpha_t z + \beta_t \epsilon$ (General case: $x_t \sim p_t(\cdot|z)$)
6: Compute loss

$$\mathcal{L}(\theta) = \|s_t^\theta(x_t) + \frac{\epsilon}{\beta_t}\|^2 \qquad \text{(General case: } = \|s_t^\theta(x_t) - \nabla \log p_t(x_t|z)\|^2\text{)}$$

$$\text{Alternatively: } \mathcal{L}(\theta) = \|\epsilon_t^\theta(x_t) - \epsilon\|^2$$

7: Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$.
8: **end for**

Beyond its simplicity, there is another useful property of the Gaussian probability path: By learning s_t^θ or ϵ_t^θ, we also learn u_t^θ automatically and the other way around:

Proposition 1 (Conversion formula for Gaussian probability path)
For the Gaussian probability path $p_t(x|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$, it holds that that the conditional (resp. marginal) vector field can be converted into the conditional (resp. marginal) score:

$$u_t^{\text{target}}(x|z) = \left(\beta_t^2 \frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t \beta_t\right) \nabla \log p_t(x|z) + \frac{\dot{\alpha}_t}{\alpha_t} x$$

$$u_t^{\text{target}}(x) = \left(\beta_t^2 \frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t \beta_t\right) \nabla \log p_t(x) + \frac{\dot{\alpha}_t}{\alpha_t} x$$

where the formula for the above marginal vector field u_t^{target} is called **probability flow ODE** in the literature (more correctly, the corresponding ODE).

Proof. For the conditional vector field and conditional score, we can derive:

$$u_t^{\text{target}}(x|z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x \overset{(i)}{=} \left(\beta_t^2 \frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t \beta_t\right) \left(\frac{\alpha_t z - x}{\beta_t^2}\right) + \frac{\dot{\alpha}_t}{\alpha_t} x = \left(\beta_t^2 \frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t \beta_t\right) \nabla \log p_t(x|z) + \frac{\dot{\alpha}_t}{\alpha_t} x$$

where in (i) we just did some algebra. By taking integrals, the same identity holds for the marginal flow vector field and the marginal score function:

$$u^{\text{target}}(x) = \int u_t^{\text{target}}(x|z) \frac{p_t(x|z) p_{\text{data}}(z)}{p_t(x)} \mathrm{d}z = \int \left[\left(\beta_t^2 \frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t \beta_t\right) \nabla \log p_t(x|z) + \frac{\dot{\alpha}_t}{\alpha_t} x\right] \frac{p_t(x|z) p_{\text{data}}(z)}{p_t(x)} \mathrm{d}z$$

$$\overset{(i)}{=} \left(\beta_t^2 \frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t \beta_t\right) \nabla \log p_t(x) + \frac{\dot{\alpha}_t}{\alpha_t} x$$

where in (i) we used eq. (51). □

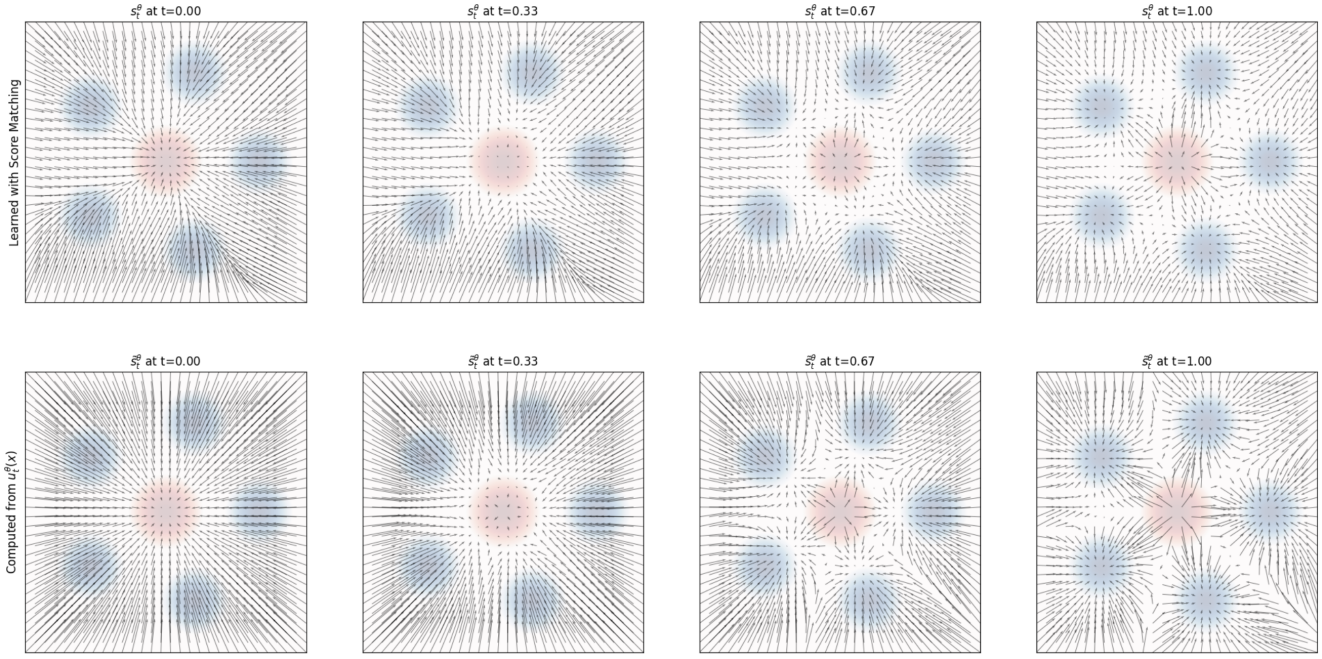

Figure 10: A comparison of the score, as obtained in two different ways. Top: A visualization of the score field $s_t^\theta(x)$ learned independently with score matching (see algorithm 4). Bottom: A visualization of the score field $\tilde{s}_t^\theta(x)$ parameterized using $u_t^\theta(x)$ as in eq. (55).

We can use the conversion formula to parameterize the score network s_t^θ and the vector field network u_t^θ into one another via

$$u_t^\theta = \left(\beta_t^2 \frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t \beta_t\right) s_t^\theta(x) + \frac{\dot{\alpha}_t}{\alpha_t} x. \tag{54}$$

Similarly, so long as $\beta_t^2 \dot{\alpha}_t - \alpha_t \dot{\beta}_t \beta_t \neq 0$ (always true for $t \in [0, 1)$), it follows that

$$s_t^\theta(x) = \frac{\alpha_t u_t^\theta(x) - \dot{\alpha}_t x}{\beta_t^2 \dot{\alpha}_t - \alpha_t \dot{\beta}_t \beta_t}. \tag{55}$$

Using this parameterization, it can be shown that the denoising score matching and the conditional flow matching losses are the same up to a constant. We conclude that **for Gaussian probability paths there is no need to separately train both the marginal score and the marginal vector field, as knowledge of one is sufficient to compute the other. In particular, we can choose whether we want to use flow matching or score matching to train it**. In fig. 10, we compare visually the score as approximated using score matching and the parameterized score using eq. (55). If we have trained a score network s_t^θ, we know by eq. (52) that we can use arbitrary $\sigma_t \geq 0$ to sample from the SDE

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = \left[\left(\beta_t^2 \frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t \beta_t + \frac{\sigma_t^2}{2}\right) s_t^\theta(x) + \frac{\dot{\alpha}_t}{\alpha_t} x\right] \mathrm{d}t + \sigma_t \mathrm{d}W_t \tag{56}$$

to obtain samples $X_1 \sim p_{\text{data}}$ (up to training and simulation error). This corresponds to **stochastic sampling from a denoising diffusion model.**

4.3 A Guide to the Diffusion Model Literature

There is a whole family of models around diffusion models and flow matching in the literature. When you read these papers, you will likely find a different (but equivalent) way of presenting the material from this class. This makes it sometimes a little confusing to read these papers. For this reason, we want to give a brief overview over various frameworks and their differences and put them also in their historical context. **This is not necessary to understand the remainder of this document** but rather intended to be a support for you in case you read the literature.

Discrete time vs. continuous time. The first denoising diffusion model papers [28, 29, 9] did not use SDEs but constructed Markov chains in **discrete time**, i.e. with time steps $t = 0, 1, 2, 3, \dots$. To this date, you will find a lot of works in the literature working with this discrete-time formulation. While this construction is appealing due to its simplicity, the disadvantage of the time-discrete approach is that it forces you to choose a time discretization before training. Further, the loss function needs to be approximated via an **evidence lower bound (ELBO)** - which is, as the name suggests, only a lower bound to the loss we actually want to minimize. Later, Song et al. [32] showed that these constructions were essentially an approximation of a time-continuous SDEs. Further, the ELBO loss becomes tight (i.e. it is not a lower bound anymore) in the continuous time case (e.g. note that theorem 18 and theorem 20 are equalities and not lower bounds - this would be different in the discrete time case). This made the SDE construction popular because it was considered mathematically "cleaner" and that one could control the simulation error via ODE/SDE samplers post training. It is important to note however that both models employ the same loss and are *not* fundamentally different.

"Forward process" vs probability paths. The first wave of denoising diffusion models [28, 29, 9, 32] did not use the term **probability path** but constructed a noising procedure of a data point $z \in \mathbb{R}^d$ via a so-called **forward process**. This is an SDE of the form

$$\bar{X}_0 = z, \quad \mathrm{d}\bar{X}_t = u_t^{\text{forw}}(\bar{X}_t)\mathrm{d}t + \sigma_t^{\text{forw}}\mathrm{d}\bar{W}_t \tag{57}$$

The idea is that after drawing a data point $z \sim p_{\text{data}}$ one simulates the forward process and thereby corrupts or "noises" the data. The forward process is designed such that for $t \to \infty$ its distribution converges to a Gaussian $\mathcal{N}(0, I_d)$. In other words, for $T \gg 0$ it holds that $\bar{X}_T \sim \mathcal{N}(0, I_d)$ approximately. Note that this essentially corresponds to a probability path: the conditional distribution of $\bar{X}_t$ given $\bar{X}_0 = z$ is a conditional probability path $\bar{p}_t(\cdot|z)$ and the distribution of $\bar{X}_t$ marginalized over $z \sim p_{\text{data}}$ corresponds to a marginal probability path $\bar{p}_t$.[2] However, note that with this construction, we need to know the distribution of $X_t|X_0 = z$ in closed form in order to train our models to avoid simulating the SDE. This essentially restrict the vector field u_t^{forw} to ones such that we know the distribution $\bar{X}_t|\bar{X}_0 = z$ in closed form. Therefore, throughout the diffusion model literature, vector fields in forward processes are always of the affine form, i.e. $u_t^{\text{forw}}(x) = a_t x$ for some continuous function a_t. For

[2] Note however that they use an **inverted time convention**: $\bar{p}_0(\cdot|z) = p_{\text{data}}$ here.

this choice, we can use known formulas of the conditional distribution [27, 31, 12]:

$$\bar{X}_t|\bar{X}_0 = z \sim \mathcal{N}\left(\alpha_t z, \beta_t^2 I\right), \quad \alpha_t = \exp\left(\int_0^t a_r \mathrm{d}r\right), \quad \beta_t^2 = \alpha_t^2 \int_0^t \frac{(\sigma_r^{\text{forw}})^2}{\alpha_r^2} dr$$

Note that these are simply Gaussian probability paths. Therefore, one can say that **a forward process is a specific way of constructing a (Gaussian) probability path.** The term probability path was introduced by flow matching [14] to both simplify the construction and make it more general at the same time: First, the "forward process" of diffusion models is never actually simulated (only samples from $\bar{p}_t(\cdot|z)$ are drawn during training). Second, a forward process only converges for $t \to \infty$ (i.e. we will never arrive at p_{init} in finite time). Therefore, we choose to use probability paths in this document.

Time-Reversals vs Solving the Fokker-Planck equation. The original description of diffusion models did not construct the training target u_t^{target} or $\nabla \log p_t$ via the Fokker-Planck equation (or Continuity equation) but via a **time-reversal** of the forward process [2]. A time-reversal $(X_t)_{0\leq t\leq T}$ is an SDE with the same distribution over trajectories inverted in time, i.e.

$$\mathbb{P}[\bar{X}_{t_1} \in A_1, \dots, \bar{X}_{t_n} \in A_n] = \mathbb{P}[X_{T-t_1} \in A_1, \dots, X_{T-t_n} \in A_n] \tag{58}$$

$$\text{for all } 0 \leq t_1, \dots, t_n \leq T, \text{ and } A_1, \dots, A_n \subset S \tag{59}$$

As shown in Anderson [2], one can obtain a time-reversal satisfying the above condition by the SDE:

$$\mathrm{d}X_t = \left[-u_t(X_t) + \sigma_t^2 \nabla \log p_t(X_t)\right] \mathrm{d}t + \sigma_t \mathrm{d}W_t, \quad u_t(x) = u_{T-t}^{\text{forw}}(x), \sigma_t = \bar{\sigma}_{T-t}$$

As $u_t(X_t) = a_t X_t$, the above corresponds to a specific instance of training target we derived in proposition 1 (this is not immediately trivial as different time conventions are used. See e.g. [15] for a derivation). However, for the purposes of generative modeling, we often only use the final point X_1 of the Markov process (e.g., as a generated image) and discard earlier time points. Therefore, whether a Markov process is a "true" time-reversal or follows along a probability path does not matter for many applications. Therefore, using a time-reversal is not necessary and often leads to suboptimal results, e.g. the probability flow ODE is often better [12, 17]. All ways of sampling from a diffusion models that are different from the time-reversal rely again on using the Fokker-Planck equation. We hope that this illustrates why nowadays many people construct the training targets directly via the Fokker-Planck equation - as pioneered by [14, 16, 1] and done in this class.

Flow Matching [14] and Stochastic Interpolants [1]. The framework that we present is most closely related to the frameworks of flow matching and **stochastic interpolants (SIs)**. As we learnt, flow matching restricts itself to flows. In fact, one of the key innovations of flow matching was to show that one does not need a construction via a forward process and SDEs but flow models alone can be trained in a scalable manner. Due to this restriction, you should keep in mind that sampling from a flow matching model will be deterministic (only the initial $X_0 \sim p_{\text{init}}$ will be random). Stochastic interpolants included both the pure flow and the SDE extension via "Langevin dynamics" that we use here (see theorem 13). Stochastic interpolants get their name from a **interpolant function** $I(t, x, z)$ intended to interpolate between two distributions. In the terminology we use here, this corresponds to a different

yet (mainly) equivalent way of constructing a conditional and marginal probability path. The advantage of flow matching and stochastic interpolants over diffusion models is both their simplicity and their generality: their training framework is very simple but at the same time they allow you to go from an arbitrary distribution p_{init} to an arbitrary distribution p_{data} - while denoising diffusion models only work for Gaussian initial distributions and Gaussian probability path. This opens up new possibilities for generative modeling that we will touch upon briefly later in this class.

Let us summarize the results of this section:

Summary 23 (Training the Generative Model)
Flow matching consists of training a neural network u_t^θ via minimizing the **conditional flow matching loss**

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{z\sim p_{\text{data}}, t\sim \text{Unif}, x\sim p_t(\cdot|z)}[\|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2] \quad \text{(conditional flow matching loss)} \tag{60}$$

where $u_t^{\text{target}}(x|z)$ is the conditional vector field (see algorithm 5). After training, one generates samples by simulating the corresponding ODE (see algorithm 1). To extend this to a diffusion model, we can use a **score network** s_t^θ and train it via **conditional score matching**

$$\mathcal{L}_{\text{CSM}}(\theta) = \mathbb{E}_{z\sim p_{\text{data}}, t\sim \text{Unif}, x\sim p_t(\cdot|z)}[\|s_t^\theta(x) - \nabla \log p_t(x|z)\|^2] \quad \text{(denoising score matching loss)} \tag{61}$$

For every diffusion coefficient $\sigma_t \geq 0$, simulating the SDE (e.g. via algorithm 2)

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = \left[u_t^\theta(X_t) + \frac{\sigma_t^2}{2} s_t^\theta(X_t)\right]\mathrm{d}t + \sigma_t \mathrm{d}W_t \tag{62}$$

will result in generating approximate samples from p_{data}. One can empirically find the optimal $\sigma_t \geq 0$.

Gaussian probability paths. For the special case of a Gaussian probability path $p_t(x|z) = \mathcal{N}(x; \alpha_t z, \beta_t^2 I_d)$, the conditional score matching is also called **denoising score matching**. This loss and conditional flow matching loss are then given by:

$$\begin{aligned}\mathcal{L}_{\text{CFM}}(\theta) =& \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, \epsilon\sim\mathcal{N}(0,I_d)}[\|u_t^\theta(\alpha_t z + \beta_t \epsilon) - (\dot\alpha_t z + \dot\beta_t \epsilon)\|^2] \\ \mathcal{L}_{\text{CSM}}(\theta) =& \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, \epsilon\sim\mathcal{N}(0,I_d)}[\|s_t^\theta(\alpha_t z + \beta_t \epsilon) + \frac{\epsilon}{\beta_t}\|^2]\end{aligned}$$

In this case, there is no need to train s_t^θ and u_t^θ separately as we can convert them post training via the formula:

$$u_t^\theta(x) = \left(\beta_t^2 \frac{\dot\alpha_t}{\alpha_t} - \dot\beta_t \beta_t\right) s_t^\theta(x) + \frac{\dot\alpha_t}{\alpha_t} x$$

Also here, after training we can simulate the SDE in eq. (62) via algorithm 2 to obtain samples X_1.

Denoising diffusion models. Denoising diffusion models are diffusion models with Gaussian probability paths. For this reason, it is sufficient for them to learn either u_t^θ or s_t^θ as they can be converted into one another.

While flow matching only allows for a simulation procedure that is deterministic via ODE, they allow for a simulation that is deterministic (probability flow ODE) or stochastic (SDE sampling). However, unlike flow matching or stochastic interpolants that allow to convert arbitrary distributions p_{init} into arbitrary distributions p_{data} via arbitrary probability paths p_t, denoising diffusion models only works for Gaussian initial distributions $p_{\text{init}} = \mathcal{N}(0, I_d)$ and a Gaussian probability path.

Literature Alternative formulations for diffusion models that are popular in the literature are:

Discrete-time: Approximations of SDEs via discrete-time Markov chains are often used.

2. **Inverted time convention:** It is popular to use an inverted time convention where $t = 0$ corresponds to p_{data} (as opposed to here where $t = 0$ corresponds to p_{init}).
3. **Forward process:** Forward processes (or noising processes) are ways of constructing (Gaussian) probability paths.
4. **Training target via time-reversal:** A training target can also be constructed via the time-reversal of SDEs. This is a specific instance of the construction presented here (with an inverted time convention).

5 Building an Image Generator

In the previous sections, we learned how to train a flow matching or diffusion model to sample from a distribution $p_{\text{data}}(x)$. This recipe is general and can be applied to a variety of different data types and applications. In this section, we learn how to apply this framework to build an image or video generator, such as e.g., *Stable Diffusion 3* and *Meta Movie Gen Video*. To build such a model, there are two main ingredients that we are missing: First, we will need to formulate **conditional generation** (**guidance**), e.g. how do we generate an image that fits a specific text prompt, and how our existing objectives may be suitably adapted to this end. We will also learn about classifier-free guidance, a popular technique used to enhance the quality of conditional generation. Second, we will discuss common neural network architectures, again focusing on those designed for images and videos. Finally, we will examine in depth the two state-of-the-art image and video models mentioned above - *Stable Diffusion* and *Meta MovieGen* - to give you a taste of how things are done at scale.

5.1 Guidance

So far, the generative models we considered were **unconditional**, e.g. an image model would simply generate *some* image. However, the task is not merely to generate an arbitrary object, but to generate an object **conditioned on some additional information**. For example, one might imagine a generative model for images which takes in a text prompt y, and then generates an image x **conditioned** on y. For fixed prompt y, we would thus like to sample from $p_{\text{data}}(x|y)$, that is, the data distribution **conditioned on** y. Formally, we think of y to live in a space $\mathcal{Y}$. When y corresponds to a text-prompt, for example, $\mathcal{Y}$ would likely be some continuous space like $\mathbb{R}^{d_y}$. When y corresponds to some discrete class label, $\mathcal{Y}$ would be discrete. In the lab, we will work with the MNIST dataset, in which case we will take $\mathcal{Y} = \{0, 1, \dots, 9\}$ to correspond to the identities of handwritten digits.

To avoid a notation and terminology clash with the use of the word "conditional" to refer to conditioning on $z \sim p_{\text{data}}$ (conditional probability path/vector field), we will make use of the term **guided** to refer specifically to conditioning on y.

Remark 24 (Guided vs. Conditional Terminology)
In these notes, we opt to use the term **guided** in place of **conditional** to refer to the act of conditioning on y. Here, we will refer to e.g., a **guided** vector field $u_t^{\text{target}}(x|y)$ and a **conditional** vector field $u_t^{\text{target}}(x|z)$. This terminology is consistent with other works such as [15].

The goal of **guided generative modeling** is thus to be able to sample from $p_{\text{data}}(x|y)$ *for any such* y. In the language of flow and score matching, and in which our generative models correspond to the simulation of ordinary and stochastic differential equations, this can be phrased as follows.

Key Idea 5 (Guided Generative Model)
We define a **guided diffusion model** to consist of a **guided vector field** $u_t^\theta(\cdot|y)$, parameterized by some neural

network, and a time-dependent diffusion coefficient σ_t, together given by

$$\textbf{Neural network: } u^\theta : \mathbb{R}^d \times \mathcal{Y} \times [0,1] \to \mathbb{R}^d, \ (x, y, t) \mapsto u_t^\theta(x|y)$$
$$\textbf{Fixed: } \sigma_t : [0,1] \to [0, \infty), \ t \mapsto \sigma_t$$

Notice the difference from summary 7: we are additionally guiding u_t^θ with the input $y \in \mathcal{Y}$. For any such $y \in \mathbb{R}^{d_y}$, samples may then be generated from such a model as follows:

Initialization:	$X_0 \sim p_{\text{init}}$	▶ Initialize with simple distribution (such as a Gaussian)
Simulation:	$\mathrm{d}X_t = u_t^\theta(X_t\|y)\mathrm{d}t + \sigma_t \mathrm{d}W_t$	▶ Simulate SDE from $t = 0$ to $t = 1$.
Goal:	$X_1 \sim p_{\text{data}}(\cdot\|y)$	▶ Goal is for X_1 to be distributed like $p_{\text{data}}(\cdot\|y)$.

When $\sigma_t = 0$, we say that such a model is a **guided flow model**.

5.1.1 Guidance for Flow Models

If we imagine fixing our choice of y, and take our data distribution as $p_{\text{data}}(x|y)$, then we have recovered the unguided generative problem, and can accordingly construct a generative model using the conditional flow matching objective, viz.,

$$\mathbb{E}_{z \sim p_{\text{data}}(\cdot|y), x \sim p_t(\cdot|z)} \|u_t^\theta(x|y) - u_t^{\text{target}}(x|z)\|^2. \tag{63}$$

Note that the label y does not affect the conditional probability path $p_t(\cdot|z)$ or the conditional vector field $u_t^{\text{target}}(x|z)$ (although in principle, we could make it dependent). Expanding the expectation over all such choices of y, and over all times $t \in \text{Unif}[0,1)$, we thus obtain a **guided conditional flow matching objective**

$$\mathcal{L}_{\text{CFM}}^{\text{guided}}(\theta) = \mathbb{E}_{(z,y) \sim p_{\text{data}}(z,y),\, t \sim \text{Unif}[0,1),\, x \sim p_t(\cdot|z)} \|u_t^\theta(x|y) - u_t^{\text{target}}(x|z)\|^2. \tag{64}$$

One of the main differences between the guided objective in eq. (64) and the unguided objective from eq. (44) is that here we are sampling $(z, y) \sim p_{\text{data}}$ rather than just $z \sim p_{\text{data}}$. The reason is that our data distribution is now, in principle, a joint distribution over e.g., both images z and text prompts y. In practice, this means that a PyTorch implementation of eq. (64) would involve a dataloader which returned batches of **both** z **and** y. The above procedure leads to a faithful generation procedure of $p_{\text{data}}(\cdot|y)$.

Classifier-Free Guidance. While the above conditional training procedure is theoretically valid, it was soon empirically realized that images samples with this procedure did not fit well enough to the desired label y. It was discovered that perceptual quality is increased when the effect of the guidance variable y is artificially reinforced. This insight was distilled into a technique known as **classifier-free guidance** that is widely used in the context of state-of-the-art diffusion models, and which we discuss next. For simplicity, we will focus here on the case of Gaussian probability paths. Recall from eq. (16) that a **Gaussian conditional probability path** is given by

$$p_t(\cdot|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$$

where the noise schedulers α_t and β_t are continuously differentiable, monotonic, and satisfy $\alpha_0 = \beta_1 = 0$ and $\alpha_1 = \beta_0 = 1$. To gain intuition for classifier-free guidance, we can use proposition 1 to rewrite the guided vector field $u_t^{\text{target}}(x|y)$ in the following form using the guided score function $\nabla \log p_t(x|y)$

$$u_t^{\text{target}}(x|y) = a_t x + b_t \nabla \log p_t(x|y), \tag{65}$$

where

$$(a_t, b_t) = \left(\frac{\dot{\alpha}_t}{\alpha_t}, \frac{\dot{\alpha}_t \beta_t^2 - \dot{\beta}_t \beta_t \alpha_t}{\alpha_t} \right). \tag{66}$$

However, notice that by Bayes' rule, we can rewrite the guided score as

$$\nabla \log p_t(x|y) = \nabla \log \left(\frac{p_t(x) p_t(y|x)}{p_t(y)} \right) = \nabla \log p_t(x) + \nabla \log p_t(y|x), \tag{67}$$

where we used that the gradient ∇ is taken with respect to the variable x, so that $\nabla \log p_t(y) = 0$. We may thus rewrite

$$u_t^{\text{target}}(x|y) = a_t x + b_t(\nabla \log p_t(x) + \nabla \log p_t(y|x)) = u_t^{\text{target}}(x) + b_t \nabla \log p_t(x|y).$$

Notice the shape of the above equation: The guided vector field $u_t^{\text{target}}(x|y)$ is a sum of the unguided vector field *plus* a guided score $\nabla \log p_t(x|y)$. As people observed that their image x did not fit their prompt y well enough, it was a natural idea to scale up the contribution of the $\nabla \log p_t(y|x)$ term, yielding

$$\tilde{u}_t(x|y) = u_t^{\text{target}}(x) + w b_t \nabla \log p_t(y|x),$$

where $w > 1$ is known as the **guidance scale**. Note that this is a heuristic: for $w \neq 1$, it holds that $\tilde{u}_t(x|y) \neq u_t^{\text{target}}(x|y)$, i.e. therefore not the true, guided vector field. However, empirical results have shown to yield preferable results (when $w > 1$).

Remark 25 (Where is the classifier?)
The term $\log p_t(y|x)$ can be considered as a sort of classifier of noised data (i.e. it gives the likelihoods of y given x). In fact, early works in diffusion trained actual classifiers and used them to the guide via the above procedure. This leads to **classifier guidance** [5, 30]. As it has been largely superseded by classifier-free guidance, we do not consider it here.

We may again apply the equality

$$\nabla \log p_t(x|y) = \nabla \log p_t(x) + \nabla \log p_t(y|x)$$

to obtain

$$\begin{aligned}\tilde{u}_t(x|y) &= u_t^{\text{target}}(x) + wb_t\nabla \log p_t(y|x) \\ &= u_t^{\text{target}}(x) + wb_t(\nabla \log p_t(x|y) - \nabla \log p_t(x)) \\ &= u_t^{\text{target}}(x) - (wa_t x + wb_t\nabla \log p_t(x)) + (wa_t x + wb_t\nabla \log p_t(x|y)) \\ &= (1-w)u_t^{\text{target}}(x) + wu_t^{\text{target}}(x|y).\end{aligned}$$

We may therefore express the scaled guided vector field $\tilde{u}_t(x|y)$ as the linear combination of the unguided vector field $u_t^{\text{target}}(x)$ with the guided vector field $u_t^{\text{target}}(x|y)$. The idea might then to to train both an unguided $u_t^{\text{target}}(x)$ (using e.g., eq. (44)) as well as a guided $u_t^{\text{target}}(x|y)$ (using e.g., eq. (64)), and then combine them at inference time to obtain $\tilde{u}_t(x|y)$. "But wait!", you might ask, "wouldn't we need to train two models then !?". It turns out we do can train both in model: we may thus augment our label set with a new, additional $\varnothing$ label that denotes **the absence of conditioning**. We can then treat $u_t^{\text{target}}(x) = u_t^{\text{target}}(x|\varnothing)$. With that, we do not need to train a separate model to reinforce the effect of a hypothetical classifier. This approach of training a conditional and unconditional model in one (and subsequently reinforcing the conditioning) is known as **classifier-free guidance** (CFG) [10].

Remark 26 (Derivation for general probability paths)
Note that the construction

$$\tilde{u}_t(x|y) = (1-w)u_t^{\text{target}}(x) + wu_t^{\text{target}}(x|y),$$

is equally valid for any choice probability path, not just a Gaussian one. When $w = 1$, it is straightforward to verify that $\tilde{u}_t(x|y) = u_t^{\text{target}}(x|y)$. Our derivation using Gaussian paths was simply to illustrate the intuition behind the construction, and in particular of amplifying the contribution of a "classifier" $\nabla \log p_t(y|x)$.

Training and Context-Free Guidance. We must now amend the guided conditional flow matching objective from eq. (64) to account for the possibility of $y = \varnothing$. The challenge is that when sampling $(z, y) \sim p_{\text{data}}$, we will never obtain $y = \varnothing$. It follows that we must introduce the possibility of $y = \varnothing$ artificially. To do so, we will define some hyperparameter η to be the probability that we discard the original label y, and replace it with $\varnothing$. We thus arrive at our **CFG conditional flow matching training objective**

$$\mathcal{L}_{\text{CFM}}^{\text{CFG}}(\theta) = \mathbb{E}_{\square}\|u_t^\theta(x|y) - u_t^{\text{target}}(x|z)\|^2 \tag{68}$$

$$\square = (z, y) \sim p_{\text{data}}(z, y),\ t \sim \text{Unif}[0, 1),\ x \sim p_t(\cdot|z), \text{replace } y = \varnothing \text{ with prob. } \eta \tag{69}$$

We summarize our findings below.

Summary 27 (Classifier-Free Guidance for Flow Models)
Given the unguided marginal vector field $u_t^{\text{target}}(x|\varnothing)$, the guided marginal vector field $u_t^{\text{target}}(x|y)$, and a **guidance scale** $w > 1$, we define the **classifier-free guided vector field** $\tilde{u}_t(x|y)$ by

$$\tilde{u}_t(x|y) = (1-w)u_t^{\text{target}}(x|\varnothing) + wu_t^{\text{target}}(x|y). \tag{70}$$

Figure 11: The effect of classifier guidance. The prompt here is the "class" chosen to be "Corgi" (a specific type of dog). Left: samples generated with no guidance (i.e., $w = 1$). Right: samples generated with classifier guidance and $w = 4$. As shown, classifier-free guidance improves the similarity to the prompt. Figure taken from [10].

By approximating $u_t^{\text{target}}(x|\varnothing)$ and $u_t^{\text{target}}(x|y)$ using the same neural network, we may leverage the following **classifier-free guidance CFM** (CFG-CFM) objective, given by

$$\mathcal{L}_{\text{CFM}}^{\text{CFG}}(\theta) = \mathbb{E}_{\square}\|u_t^\theta(x|y) - u_t^{\text{target}}(x|z)\|^2 \tag{71}$$

$$\square = (z, y) \sim p_{\text{data}}(z, y),\ t \sim \text{Unif}[0, 1),\ x \sim p_t(\cdot|z), \text{replace } y = \varnothing \text{ with prob. } \eta \tag{72}$$

In plain English, $\mathcal{L}_{\text{CFM}}^{\text{CFG}}$ might be approximated by

$(z, y) \sim p_{\text{data}}(z, y)$	▶ Sample (z, y) from data distribution.
$t \sim \text{Unif}[0, 1)$	▶ Sample t uniformly on $[0, 1)$.
$x \sim p_t(x\|z)$	▶ Sample x from the conditional probability path $p_t(x\|z)$.
with prob. η, $y \leftarrow \varnothing$	▶ Replace y with $\varnothing$ with probability η.
$\widehat{\mathcal{L}_{\text{CFM}}^{\text{CFG}}}(\theta) = \|u_t^\theta(x\|y) - u_t^{\text{target}}(x\|z)\|^2$	▶ Regress model against conditional vector field.

Above, we made use multiple times of the fact that $u_t^{\text{target}}(x|z) = u_t^{\text{target}}(x|z, y)$. At inference time, for a fixed choice of y, we may sample via

Initialization:	$X_0 \sim p_{\text{init}}(x)$	▶ Initialize with simple distribution (such as a Gaussian)
Simulation:	$\mathrm{d}X_t = \tilde{u}_t^\theta(X_t\|y)\mathrm{d}t$	▶ Simulate ODE from $t = 0$ to $t = 1$.
Samples:	X_1	▶ Goal is for X_1 to adhere to the guiding variable y.

Note that the distribution of X_1 is not necessarily aligned with $X_1 \sim p_{\text{data}}(\cdot|y)$ anymore if we use a weight $w > 1$. However, empirically, this shows better alignment with conditioning. In fig. 11, we illustrate class-based classifier-free guidance on 128x128 ImageNet, as in [10]. Similarly, in fig. 12, we visualize the affect of various guidance scales w when applying classifier-free guidance to sampling from the MNIST dataset of handwritten digits.

Figure 12: The effect of classifier-free guidance applied at various guidance scales for the MNIST dataset of handwritten digits. Left: Guidance scale set to $w = 1.0$. Middle: Guidance scale set to $w = 2.0$. Right: Guidance scale set to $w = 4.0$. You will generate a similar image yourself in the lab three!

Algorithm 5 Classifier-free guidance training for Gaussian probability path $p_t(x|z) = \mathcal{N}(x; \alpha_t z, \beta_t^2 I_d)$

Require: Paired dataset $(z, y) \sim p_{\text{data}}$, neural network u_t^θ

1: **for** each mini-batch of data **do**
2: Sample a data example (z, y) from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$.
4: Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$
5: Set $x = \alpha_t z + \beta_t \epsilon$
6: With probability p drop label: $y \leftarrow \varnothing$
7: Compute loss

$$\mathcal{L}(\theta) = \|u_t^\theta(x|y) - (\dot{\alpha}_t \epsilon + \dot{\beta}_t z)\|^2$$

8: Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$.
9: **end for**

5.1.2 Guidance for Diffusion Models

In this section we extend the reasoning of the previous section to diffusion models. First, in the same way that we obtained eq. (64), we may generalize the conditional score matching loss eq. (61) to obtain the **guided conditional score matching objective**

$$\mathcal{L}_{\text{CSM}}^{\text{guided}}(\theta) = \mathbb{E}_{\square}[\|s_t^\theta(x|y) - \nabla \log p_t(x|z)\|^2] \tag{73}$$

$$\square = (z, y) \sim p_{\text{data}}(z, y),\ t \sim \text{Unif},\ x \sim p_t(\cdot|z). \tag{74}$$

A guided score network $s_t^\theta(x|y)$ trained with eq. (73) might then be combined with the guided vector field $u_t^\theta(x|y)$ to simulate the SDE

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = \left[u_t^\theta(X_t|y) + \frac{\sigma_t^2}{2} s_t^\theta(X_t|y)\right] \mathrm{d}t + \sigma_t \mathrm{d}W_t.$$

Classifier-Free Guidance. We now extend the classifier-free guidance construction to the diffusion setting. By Bayes' rule (see eq. (67)),

$$\nabla \log p_t(x|y) = \nabla \log p_t(x) + \nabla \log p_t(y|x),$$

so that for **guidance scale** $w > 1$ we may define

$$\begin{aligned}\tilde{s}_t(x|y) &= \nabla \log p_t(x) + w\nabla \log p_t(y|x) \\ &= \nabla \log p_t(x) + w(\nabla \log p_t(x|y) - \nabla \log p_t(x)) \\ &= (1-w)\nabla \log p_t(x) + w\nabla \log p_t(x|y) \\ &= (1-w)\nabla \log p_t(x|\varnothing) + w\nabla \log p_t(x|y)\end{aligned}$$

We thus arrive at the CFG-compatible (that is, accounting for the possibility of $\varnothing$) objective

$$\mathcal{L}_{\text{DSM}}^{\text{CFG}}(\theta) = \mathbb{E}_{\square}\|s_t^\theta(x|y) - \nabla \log p_t(x|z)\|^2 \tag{75}$$

$$\square = (z,y) \sim p_{\text{data}}(z,y),\ t \sim \text{Unif}[0,1),\ x \sim p_t(\cdot|z),\ \text{replace } y = \varnothing \text{ with prob. } \eta, \tag{76}$$

where η is a hyperparameter (the probability of replacing y with $\varnothing$). We will refer $\mathcal{L}_{\text{CSM}}^{\text{CFG}}(\theta)$ as the **guided conditional score matching objective**. We recap as follows

Summary 28 (Classifier-Free Guidance for Diffusions)
Given the unguided marginal score $\nabla \log p_t(x|\varnothing)$, the guided marginal score field $\nabla \log p_t(x|y)$, and a **guidance scale** $w > 1$, we define the **classifier-free guided score** $\tilde{s}_t(x|y)$ by

$$\tilde{s}_t(x|y) = (1-w)\nabla \log p_t(x|\varnothing) + w\nabla \log p_t(x|y). \tag{77}$$

By approximating $\nabla \log p_t(x|\varnothing)$ and $\nabla \log p_t(x|y)$ using the same neural network $s_t^\theta(x|y)$, we may leverage the following **classifier-free guidance CSM** (CFG-CSM) objective, given by

$$\mathcal{L}_{\text{CSM}}^{\text{CFG}}(\theta) = \mathbb{E}_{\square}\|s_t^\theta(x|(1-\xi)y + \xi\varnothing) - \nabla \log p_t(x|z)\|^2 \tag{78}$$

$$\square = (z,y) \sim p_{\text{data}}(z,y),\ t \sim \text{Unif}[0,1),\ x \sim p_t(\cdot|z),\ \text{replace } y = \varnothing \text{ with prob. } \eta \tag{79}$$

In plain English, $\mathcal{L}_{\text{DSM}}^{\text{CFG}}$ might be approximated by

$(z,y) \sim p_{\text{data}}(z,y)$	▶ Sample (z,y) from data distribution.
$t \sim \text{Unif}[0,1)$	▶ Sample t uniformly on $[0,1)$.
$x \sim p_t(x\|z,y)$	▶ Sample x from cond. path $p_t(x\|z)$.
with prob. η, $y \leftarrow \varnothing$	▶ Replace y with $\varnothing$ with probability η.
$\widehat{\mathcal{L}_{\text{DSM}}^{\text{CFG}}}(\theta) = \|s_t^\theta(x\|y) - \nabla \log p_t(x\|z)\|^2$	▶ Regress model against conditional score.

At inference time, for a fixed choice of $w > 1$, we may combine $s_t^\theta(x|y)$ with a guided vector field $u_t^\theta(x|y)$ and

define

$$\tilde{s}_t^\theta(x|y) = (1-w)s_t^\theta(x|\varnothing) + ws_t^\theta(x|y),$$
$$\tilde{u}_t^\theta(x|y) = (1-w)u_t^\theta(x|\varnothing) + wu_t^\theta(x|y).$$

Then we may sample via

Initialization: $X_0 \sim p_{\text{init}}(x)$ ▶ Initialize with simple distribution (such as a Gaussian)

Simulation: $\mathrm{d}X_t = \left[\tilde{u}_t^\theta(X_t|y) + \frac{\sigma_t^2}{2}\tilde{s}_t^\theta(X_t|y)\right]\mathrm{d}t + \sigma_t \mathrm{d}W_t$ ▶ Simulate SDE from $t=0$ to $t=1$.

Samples: X_1 ▶ Goal is for X_1 to adhere to the guiding variable y.

5.2 Neural network architectures

We next discuss the design of neural networks for flow and diffusion models. Specifically, we answer the question of how to construct a neural network architecture that represents the (guided) vector field $u_t^\theta(x|y)$ with parameters θ. Note that the neural network must have 3 inputs - a vector $x \in \mathbb{R}^d$, a conditioning variable $y \in \mathcal{Y}$, and a time value $t \in [0,1]$ - and one output - a vector $u_t^\theta(x|y) \in \mathbb{R}^d$. For low-dimensional distributions (e.g. the toy distributions we have seen in previous sections), it is sufficient to parameterize $u_t^\theta(x|y)$ as a multi-layer perceptron (MLP), otherwise known as a fully connected neural network. That is, in this simple setting, a forward pass through $u_t^\theta(x|y)$ would involve concatenating our input x, y, and t, and passing them through an MLP. However, for complex, high-dimensional distributions, such as those over images, videos, and proteins, an MLP is rarely sufficient, and it is common to use special, application-specific architectures. For the remainder of this section, we will consider the case of **images** (and by extension, videos), and discuss two common architectures: the **U-Net** [25], and the **diffusion transformer** (DiT).

5.2.1 U-Nets and Diffusion Transformers

Before we dive into the specifics of these architectures, let us recall from the introduction that an image is simply a vector $x \in \mathbb{R}^{C_{\text{image}} \times H \times W}$. Here C_{image} denotes the number of **channels** (an RGB image typically would have $C_{\text{input}} = 3$ color channels), H denotes the **height** of the image in pixels, and W denotes the **width** of the image in pictures.

U-Nets. The **U-Net** architecture [25] is a specific type of convolutional neural network. Originally designed for image segmentation, its crucial feature is that both its input and its output have the shape of images (possibly with a different number of channels). This makes it ideal to parameterize a vector field $x \mapsto u_t^\theta(x|y)$ as for fixed y, t its input has the shape of an image and its output does, too. Therefore, U-Net were widely used in the development of diffusion models. A U-Net consists of a series of **encoders** $\mathcal{E}_i$, and a corresponding sequence of **decoders** $\mathcal{D}_i$, along with a latent processing block in between, which we shall refer to as a **midcoder** (midcoder is a term is not used in the literature usually). For sake of example, let us walk through the path taken by an image $x_t \in \mathbb{R}^{3\times 256\times 256}$ (we

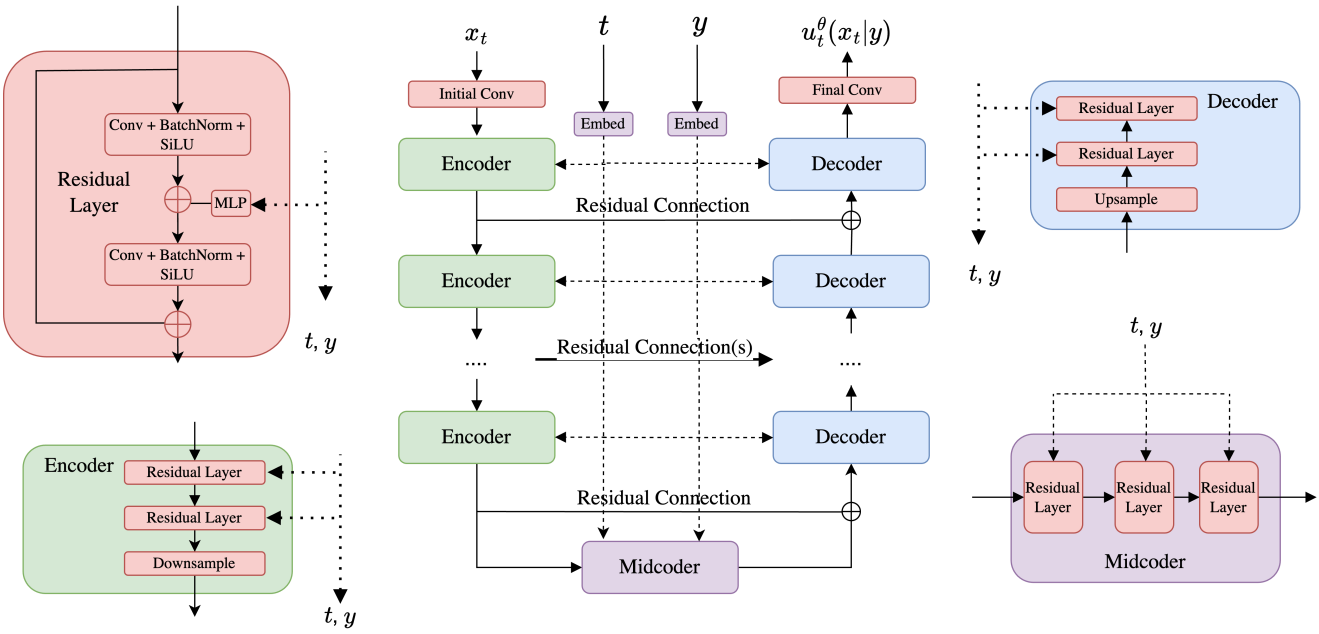

Figure 13: The simplified U-Net architecture used in lab three.

have taken $(C_{\text{input}}, H, W) = (3, 256, 256)$) as it is processed by the U-Net:

$$
\begin{aligned}
x_t^{\text{input}} &\in \mathbb{R}^{3\times256\times256} && \blacktriangleright \text{ Input to the U-Net.} \\
x_t^{\text{latent}} &= \mathcal{E}(x_t^{\text{input}}) \in \mathbb{R}^{512\times32\times32} && \blacktriangleright \text{ Pass through encoders to obtain latent.} \\
x_t^{\text{latent}} &= \mathcal{M}(x_t^{\text{latent}}) \in \mathbb{R}^{512\times32\times32} && \blacktriangleright \text{ Pass latent through midcoder.} \\
x_t^{\text{output}} &= \mathcal{D}(x_t^{\text{latent}}) \in \mathbb{R}^{3\times256\times256} && \blacktriangleright \text{ Pass through decoders to obtain output.}
\end{aligned}
$$

Notice that as the input passes through the encoders, the number of channels in its representation increases, while the height and width of the images are decreased. Both the encoder and the decoder usually consist of a series of convolutional layers (with activation functions, pooling operations, etc. in between). Not shown above are two points: First, the input $x_t^{\text{input}} \in \mathbb{R}^{3\times256\times256}$ is often fed into an initial pre-encoding block to increase the number of channels before being fed into the first encoder block. Second, the encoders and decoders are often connected by **residual connections**. The complete picture is shown in fig. 13. At a high level, most U-Nets involve some variant of what is described above. However, certain of the design choices described above may well differ from various implementations in practice. In particular, we opt above for a purely-convolutional architecture whereas it is common to include attention layers as well throughout the encoders and decoders. The U-Net derives its name from the "U"-like shape formed by its encoders and decoders (see fig. 13).

Diffusion Transformers. One alternative to U-Nets are **diffusion transformers** (DiTs), which dispense with convolutions and purely use **attention** [35, 19]. Diffusion transformers are based on **vision transformers** (ViTs), in which the big idea is essentially to divide up an image into patches, embed each of these patches, and then attend between the patches [6]. *Stable Diffusion 3*, trained with conditional flow matching, parameterizes the velocity field

$u_t^\theta(x)$ as a modified DiT, as we discuss later in section 5.3 [7].

Remark 29 (Working in Latent Space)
A common problem for large-scale applications is that the data is so high-dimensional that it consumes too much memory. For example, we might want to generate a high resolution image of 1000×10000 pixels leading to 1 million (!) dimensions. To reduce memory usage, a common design pattern is to work in a **latent space** that can be considered a compressed version of our data at lower resolution. Specifically, the usual approach is to combine a flow or diffusion model with a (variational) **autoencoder** [24]. In this case, one first encodes the training dataset in the **latent space** via an autoencoder, and then training the flow or diffusion model in the latent space. Sampling is performed by first sampling in the latent space using the trained flow or diffusion model, and then decoding of the output via the decoder. Intuitively, a well-trained autoencoder can be thought of as filtering out semantically meaningless details, allowing the generative model to "focus" on important, perceptually relevant features [24]. By now, nearly all state-of-the-art approaches to image and video generation involve training a flow or diffusion model in the latent space of an autoencoder - so called **latent diffusion models** [24, 34]. However, it is important to note: one also needs to train the autoencoder before training the diffusion models. Crucially, performance now depends also on how good the autoencoder compresses images into latent space and recovers aesthetically pleasing images.

5.2.2 Encoding the Guiding Variable.

Up until this point, we have glossed over how exactly the guiding (conditioning) variable y is fed into the neural network $u_t^\theta(x|y)$. Broadly, this process can be decomposed into two steps: embedding the raw input y_{raw} (e.g., the text prompt "a cat playing a trumpet, photorealistic") into some vector-valued input y, and feeding the resulting y into the actual model. We now proceed to describe each step in greater detail.

Embedding Raw Input. Here, we'll consider two cases: (1) where y_{raw} is a discrete class-label, and (2) where y_{raw} is a text-prompt. When $y_{\text{raw}} \in \mathcal{Y} \triangleq \{0, \dots, N\}$ is just a class label, then it is often easiest to simply learn a separate embedding vector for each of the $N+1$ possible values of y_{raw}, and set y to this embedding vector. One would consider the parameters of these embeddings to be included in the parameters of $u_t^\theta(x|y)$, and would therefore learn these during training. When y_{raw} is a text-prompt, the situation is more complex, and approaches largely rely on frozen, pre-trained models. Such models are trained to embed a discrete text input into a continuous vector that captures the relevant information. One such model is known as **CLIP** (Contrastive Language-Image Pre-training). CLIP is trained to learn a shared embedding space for both images and text-prompts, using a training loss designed to encourage image embeddings to be close to their corresponding prompts, while being farther from the embeddings of other images and prompts [22]. We might therefore take $y = \text{CLIP}(y_{\text{raw}}) \in \mathbb{R}^{d_{\text{CLIP}}}$ to be the embedding produced by a frozen, pre-trained CLIP model. In certain cases, it may be undesirable to compress the entire sequence into a single representation. In this case, one might additionally consider embedding the prompt using a pre-trained transformer so as to obtain a sequence of embeddings. It is also common to combine multiple such pretrained embeddings when conditioning so as to simultaneously reap the benefits of each model [7, 21].

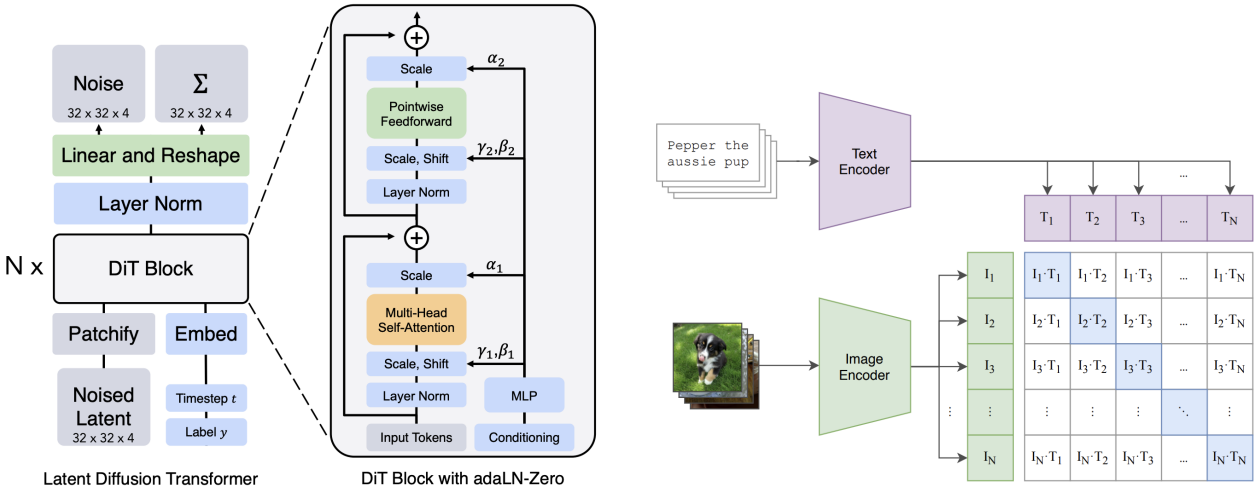

Figure 14: Left: An overview of the diffusion transformer architecture, taken from [19]. Right: A schematic of the contrastive CLIP loss, in which a shared image-text embedding space is learned, taken from [22].

Feeding in the Embedding. Suppose now that we have obtained our embedding vector $y \in \mathbb{R}^{d_y}$. Now what? The answer varies, but usually it is some variant of the following: feed it individually into every sub-component of the architecture for images. Let us briefly describe how this is accomplished in the U-Net implementation used in lab three, as depicted in fig. 13. At some intermediate point within the network, we would like to inject information from $y \in \mathbb{R}^{d_y}$ into the current activation $x_t^{\text{intermediate}} \in \mathbb{R}^{C \times H \times W}$. We might do so using the procedure below, given in PyTorch-esque pseudocode.

$$y = \text{MLP}(y) \in \mathbb{R}^C \quad \blacktriangleright \text{ Map } y \text{ from } \mathbb{R}^{d_y} \text{ to } \mathbb{R}^C.$$

$$y = \text{reshape}(y) \in \mathbb{R}^{C \times 1 \times 1} \quad \blacktriangleright \text{ Reshape } y \text{ to "look" like an image.}$$

$$x_t^{\text{intermediate}} = \text{broadcast\_add}(x_t^{\text{intermediate}}, y) \in \mathbb{R}^{C \times H \times W} \quad \blacktriangleright \text{ Add } y \text{ to } x_t^{\text{intermediate}} \text{ pointwise.}$$

One exception to this simple-pointwise conditioning scheme is when we have a sequence of embeddings as produced by some pretrained language model. In this case, we might consider using some sort of cross-attention scheme between our image (suitably patchified) and the tokens of the embedded sequence. We will see multiple examples of this in section 5.3.

5.3 A Survey of Large-Scale Image and Video Models

We conclude this section by briefly examining two large-scale generative models: *Stable Diffusion 3* for image generation and Meta's *Movie Gen Video* for video generation [7, 21]. As you will see, these models use the techniques we have described in this work along with additional architectural enhancements to both scale and accommodate richly structured conditioning modalities, such as text-based input.

5.3.1 Stable Diffusion 3

Stable Diffusion is a series of state-of-the-art image generation models. These models were among the first to use large-scale latent diffusion models for image generation. If you have not done so, we highly recommend testing it for yourself online (https://stability.ai/news/stable-diffusion-3).

Stable Diffusion 3 uses the same conditional flow matching objective that we study in this work (see algorithm 5).[3] As outlined in their paper, they extensively tested various flow and diffusion alternatives and found flow matching to perform best. For training, it uses classifier-free guidance training (with dropping class labels) as outlined above. Further, Stable Diffusion 3 follows the approach outlined in section 5.2 by training within the latent space of a pre-trained autoencoder. Training a good autoencoder was a big contribution of the first stable diffusion papers.

To enhance text conditioning, Stable Diffusion 3 makes use of both 3 different types of text embeddings (including CLIP embeddings as well as the sequential outputs produced by a pretrained instance of the encoder of Google's T5-XXL [23], and similar to approaches taken in [3, 26]). Whereas CLIP embeddings provide a coarse, overarching embedding of the input text, the T5 embeddings provide a more granular level of context, allowing for the possibility of the model attending to particular elements of the conditioning text. To accommodate these sequential context embeddings, the authors then propose to extend the diffusion transformer to attend not just to patches of the image, but to the text embeddings as well, thereby extending the conditioning capacity from the class-based scheme originally proposed for DiT to sequential context embeddings. This proposed modified DiT is referred to as a **multi-modal DiT** (MM-DiT), and is depicted in fig. 15. Their final, largest model has **8 billion parameters**. For sampling, they use 50 steps (i.e. they have to evaluate the network 50 times) using a Euler simulation scheme and a classifier-free guidance weight between 2.0-5.0.

5.3.2 Meta Movie Gen Video

Next, we discuss Meta's video generator, *Movie Gen Video* (https://ai.meta.com/research/movie-gen/). As the data are not images but *videos*, the data x lie in the space $\mathbb{R}^{T\times C\times H\times W}$ where T represents the new **temporal** dimension (i.e. the number of frames). As we shall see, many of the design choices made in this video setting can be seen as adapting existing techniques (e.g., autoencoders, diffusion transformers, etc.) from the image setting to handle this extra temporal dimension.

Movie Gen Video utilizes the conditional flow matching objective with the same CondOT path (see algorithm 5). Like Stable Diffusion 3, Movie Gen Video also operates in the latent space of frozen, pretrained autoencoder. Note that the autoencoder to reduce memory consumption is even more important for videos than for images - which is why most video generators right now are pretty limited in the length of the video they generate. Specifically, the authors propose to handle the added time dimension by introducing a **temporal autoencoder** (TAE) which maps a raw video $x'_t \in \mathbb{R}^{T'\times 3\times H\times W}$ to a latent $x_t \in \mathbb{R}^{T\times C\times H\times W}$, with $\frac{T'}{T} = \frac{H'}{H} = \frac{W'}{W} = 8$ [21]. To accomodate long videos, a temporal tiling procedure is proposed by which the video is chopped up into pieces, each piece is encoder separately, and the latents are sticthed together [21]. The model itself - that is, $u_t^\theta(x_t)$ - is given by a

[3] In their work, they use a different convention to condition on the noise. But this is only notation and the algorithm is the same.

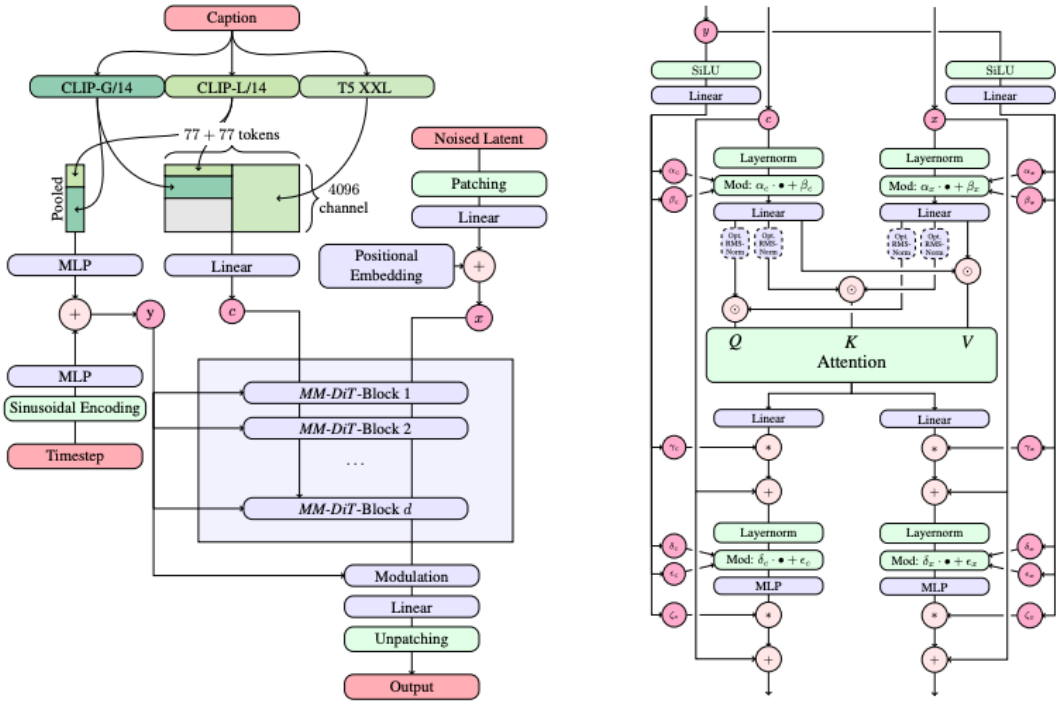

Figure 15: The architecture of the multi-modal diffusion transformer (MM-DiT) proposed in [7]. Figure also taken from [7].

DiT-like backbone in which x_t is patchified along the time and space dimensions. The image patches are then passed through a transformer employing both self-attention among the image patches, and cross-attention with language model embeddings, similar to the MM-DiT employed by Stable Diffusion 3. For text conditioning, Movie Gen Video employs three types of text embeddings: UL2 embeddings, for granular, text-based reasoning [33], ByT5 embeddings, for attending to character-level details (for e.g., prompts explicitly requesting specific text to be present) [36], and MetaCLIP embeddings, trained in a shared text-image embedding space [13, 21]. Their final, largest model has **30 billion parameters**. For a significantly more detailed and expansive treatment, we encourage the reader to check out the Movie Gen technical report itself [21].

6 Acknowledgements

This course would not have been possible without the generous support of many others. We would like to thank Tommi Jaakkola for serving as the advisor and faculty sponsor for this course, and for thoughtful feedback throughout the process. We would like to thank Christian Fiedler, Tim Griesbach, Benedikt Geiger, and Albrecht Holderrieth for value feedback on the lecture notes. Further, we thank Elaine Mello from MIT Open Learning for support with lecture recordings and Ashay Athalye from Students for Open and Universal Learning for helping to cut and process the videos. We would additionally like to thank Cameron Diao, Tally Portnoi, Andi Qu, Roger Trullo, Ádám Burián, Zewen Yang, and many others for their invaluable contributions to the labs. We would also like to thank Lisa Bella, Ellen Reid, and everyone else at MIT EECS for their generous support. Finally, we would like to thank all participants in the original course offering (MIT 6.S184/6.S975, taught over IAP 2025), as well as readers like you for your interest in this class. Thanks!

7 References

[1] Michael S Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. "Stochastic interpolants: A unifying framework for flows and diffusions". In: *arXiv preprint arXiv:2303.08797* (2023).

[2] Brian DO Anderson. "Reverse-time diffusion equation models". In: *Stochastic Processes and their Applications* 12.3 (1982), pp. 313–326.

[3] Yogesh Balaji et al. *eDiff-I: Text-to-Image Diffusion Models with an Ensemble of Expert Denoisers*. 2023. arXiv: 2211.01324 [cs.CV]. URL: https://arxiv.org/abs/2211.01324.

[4] Earl A Coddington, Norman Levinson, and T Teichmann. *Theory of ordinary differential equations*. 1956.

[5] Prafulla Dhariwal and Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. 2021. arXiv: 2105.05233 [cs.LG]. URL: https://arxiv.org/abs/2105.05233.

[6] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929.

[7] Patrick Esser et al. *Scaling Rectified Flow Transformers for High-Resolution Image Synthesis*. 2024. arXiv: 2403.03206 [cs.CV]. URL: https://arxiv.org/abs/2403.03206.

[8] Lawrence C Evans. *Partial differential equations*. Vol. 19. American Mathematical Society, 2022.

[9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[10] Jonathan Ho and Tim Salimans. *Classifier-Free Diffusion Guidance*. 2022. arXiv: 2207.12598 [cs.LG]. URL: https://arxiv.org/abs/2207.12598.

[11] Arieh Iserles. *A first course in the numerical analysis of differential equations*. Cambridge university press, 2009.

[12] Tero Karras et al. "Elucidating the design space of diffusion-based generative models". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 26565–26577.

[13] Samuel Lavoie et al. *Modeling Caption Diversity in Contrastive Vision-Language Pretraining*. 2024. arXiv: 2405.00740 [cs.CV]. URL: https://arxiv.org/abs/2405.00740.

[14] Yaron Lipman et al. "Flow matching for generative modeling". In: *arXiv preprint arXiv:2210.02747* (2022).

[15] Yaron Lipman et al. "Flow Matching Guide and Code". In: *arXiv preprint arXiv:2412.06264* (2024).

[16] Xingchao Liu, Chengyue Gong, and Qiang Liu. "Flow straight and fast: Learning to generate and transfer data with rectified flow". In: *arXiv preprint arXiv:2209.03003* (2022).

[17] Nanye Ma et al. "Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers". In: *arXiv preprint arXiv:2401.08740* (2024).

[18] Xuerong Mao. *Stochastic differential equations and applications*. Elsevier, 2007.

[19] William Peebles and Saining Xie. *Scalable Diffusion Models with Transformers*. 2023. arXiv: 2212.09748 [cs.CV]. URL: https://arxiv.org/abs/2212.09748.

[20] Lawrence Perko. *Differential equations and dynamical systems*. Vol. 7. Springer Science & Business Media, 2013.

[21] Adam Polyak et al. *Movie Gen: A Cast of Media Foundation Models*. 2024. arXiv: 2410.13720 [cs.CV]. URL: https://arxiv.org/abs/2410.13720.

[22] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV]. URL: https://arxiv.org/abs/2103.00020.

[23] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023. arXiv: 1910.10683 [cs.LG]. URL: https://arxiv.org/abs/1910.10683.

[24] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. arXiv: 2112.10752 [cs.CV]. URL: https://arxiv.org/abs/2112.10752.

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, pp. 234–241.

[26] Chitwan Saharia et al. *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*. 2022. arXiv: 2205.11487 [cs.CV]. URL: https://arxiv.org/abs/2205.11487.

[27] Simo Särkkä and Arno Solin. *Applied stochastic differential equations*. Vol. 10. Cambridge University Press, 2019.

[28] Jascha Sohl-Dickstein et al. "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International conference on machine learning*. PMLR. 2015, pp. 2256–2265.

[29] Yang Song and Stefano Ermon. "Generative modeling by estimating gradients of the data distribution". In: *Advances in neural information processing systems* 32 (2019).

[30] Yang Song et al. *Score-Based Generative Modeling through Stochastic Differential Equations*. 2021. arXiv: 2011.13456 [cs.LG]. URL: https://arxiv.org/abs/2011.13456.

[31] Yang Song et al. “Score-Based Generative Modeling through Stochastic Differential Equations”. In: *International Conference on Learning Representations (ICLR)*. 2021.

[32] Yang Song et al. “Score-based generative modeling through stochastic differential equations”. In: *arXiv preprint arXiv:2011.13456* (2020).

[33] Yi Tay et al. *UL2: Unifying Language Learning Paradigms*. 2023. arXiv: 2205.05131 [cs.CL]. URL: https://arxiv.org/abs/2205.05131.

[34] Arash Vahdat, Karsten Kreis, and Jan Kautz. “Score-based generative modeling in latent space”. In: *Advances in neural information processing systems* 34 (2021), pp. 11287–11302.

[35] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[36] Linting Xue et al. *ByT5: Towards a token-free future with pre-trained byte-to-byte models*. 2022. arXiv: 2105.13626 [cs.CL]. URL: https://arxiv.org/abs/2105.13626.

A A Reminder on Probability Theory

We present a brief overview of basic concepts from probability theory. This section was partially taken from [15].

A.1 Random vectors

Consider data in the d-dimensional Euclidean space $x = (x^1, \dots, x^d) \in \mathbb{R}^d$ with the standard Euclidean inner product $\langle x, y \rangle = \sum_{i=1}^d x^i y^i$ and norm $\|x\| = \sqrt{\langle x, x \rangle}$. We will consider random variables (RVs) $X \in \mathbb{R}^d$ with continuous probability density function (PDF), defined as a *continuous* function $p_X : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ providing event A with probability

$$\mathbb{P}(X \in A) = \int_A p_X(x) \mathrm{d}x, \tag{80}$$

where $\int p_X(x)\mathrm{d}x = 1$. By convention, we omit the integration interval when integrating over the whole space ($\int \equiv \int_{\mathbb{R}^d}$). To keep notation concise, we will refer to the PDF p_{X_t} of RV X_t as simply p_t. We will use the notation $X \sim p$ or $X \sim p(X)$ to indicate that X is distributed according to p. One common PDF in generative modeling is the d-dimensional isotropic Gaussian:

$$\mathcal{N}(x; \mu, \sigma^2 I) = (2\pi\sigma^2)^{-\frac{d}{2}} \exp\left(-\frac{\|x - \mu\|_2^2}{2\sigma^2}\right), \tag{81}$$

where $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}_{>0}$ stand for the mean and the standard deviation of the distribution, respectively.

The expectation of a RV is the constant vector closest to X in the least-squares sense:

$$\mathbb{E}[X] = \arg\min_{z \in \mathbb{R}^d} \int \|x - z\|^2 p_X(x)\mathrm{d}x = \int x p_X(x)\mathrm{d}x. \tag{82}$$

One useful tool to compute the expectation of *functions of RVs* is the *law of the unconscious statistician*:

$$\mathbb{E}[f(X)] = \int f(x) p_X(x)\mathrm{d}x. \tag{83}$$

When necessary, we will indicate the random variables under expectation as $\mathbb{E}_X f(X)$.

A.2 Conditional densities and expectations

Given two random variables $X, Y \in \mathbb{R}^d$, their joint PDF $p_{X,Y}(x, y)$ has marginals

$$\int p_{X,Y}(x, y)\mathrm{d}y = p_X(x) \text{ and } \int p_{X,Y}(x, y)\mathrm{d}x = p_Y(y). \tag{84}$$

See fig. 16 for an illustration of the joint PDF of two RVs in $\mathbb{R}$ ($d = 1$). The *conditional* PDF $p_{X|Y}$ describes the PDF of the random variable X when conditioned on an event $Y = y$ with density $p_Y(y) > 0$:

$$p_{X|Y}(x|y) \coloneqq \frac{p_{X,Y}(x, y)}{p_Y(y)}, \tag{85}$$

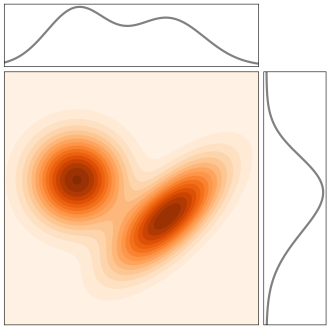

Figure 16: Joint PDF $p_{X,Y}$ (in shades) and its marginals p_X and p_Y (in black lines). Figure from [15]

.

and similarly for the conditional PDF $p_{Y|X}$. Bayes' rule expresses the conditional PDF $p_{Y|X}$ with $p_{X|Y}$ by

$$p_{Y|X}(y|x) = \frac{p_{X|Y}(x|y)p_Y(y)}{p_X(x)}, \tag{86}$$

for $p_X(x) > 0$.

The *conditional expectation* $\mathbb{E}[X|Y]$ is the best approximating *function* $g_\star(Y)$ to X in the least-squares sense:

$$\begin{aligned} g_\star &\coloneqq \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \mathbb{E}\left[\|X - g(Y)\|^2\right] = \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \int \|x - g(y)\|^2 p_{X,Y}(x,y)\mathrm{d}x\mathrm{d}y \\ &= \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \int \left[\int \|x - g(y)\|^2 p_{X|Y}(x|y)\mathrm{d}x\right] p_Y(y)\mathrm{d}y. \end{aligned} \tag{87}$$

For $y \in \mathbb{R}^d$ such that $p_Y(y) > 0$ the conditional expectation function is therefore

$$\mathbb{E}[X|Y = y] \coloneqq g_\star(y) = \int x p_{X|Y}(x|y)\mathrm{d}x, \tag{88}$$

where the second equality follows from taking the minimizer of the inner brackets in eq. (87) for $Y = y$, similarly to eq. (82). Composing $g_\star$ with the random variable Y, we get

$$\mathbb{E}[X|Y] \coloneqq g_\star(Y), \tag{89}$$

which is a random variable in $\mathbb{R}^d$. Rather confusingly, both $\mathbb{E}[X|Y = y]$ and $\mathbb{E}[X|Y]$ are often called *conditional expectation*, but these are different objects. In particular, $\mathbb{E}[X|Y = y]$ is a function $\mathbb{R}^d \to \mathbb{R}^d$, while $\mathbb{E}[X|Y]$ is a random variable assuming values in $\mathbb{R}^d$. To disambiguate these two terms, our discussions will employ the notations introduced here.

The *tower property* is an useful property that helps simplify derivations involving conditional expectations of two RVs X and Y:

$$\mathbb{E}[\mathbb{E}[X|Y]] = \mathbb{E}[X] \tag{90}$$

Because $\mathbb{E}[X|Y]$ is a RV, itself a function of the RV Y, the outer expectation computes the expectation of $\mathbb{E}[X|Y]$. The tower property can be verified by using some of the definitions above:

$$\begin{aligned} \mathbb{E}[\mathbb{E}[X|Y]] &= \int \left(\int x p_{X|Y}(x|y)\mathrm{d}x\right) p_Y(y)\mathrm{d}y \\ &\overset{(85)}{=} \int\int x p_{X,Y}(x,y)\mathrm{d}x\mathrm{d}y \\ &\overset{(84)}{=} \int x p_X(x)\mathrm{d}x \\ &= \mathbb{E}[X]. \end{aligned}$$

Finally, consider a helpful property involving two RVs $f(X,Y)$ and Y, where X and Y are two arbitrary RVs. Then, by using the Law of the Unconscious Statistician with (88), we obtain the identity

$$\mathbb{E}[f(X,Y)|Y = y] = \int f(x,y) p_{X|Y}(x|y)\mathrm{d}x. \tag{91}$$

B A Proof of the Fokker-Planck equation

In this section, we give here a self-contained proof of the Fokker-Planck equation (theorem 15) which includes the continuity equation as a special case (theorem 12). We stress that **this section is not necessary to understand the remainder of this document** and is mathematically more advanced. If you desire to understand where the Fokker-Planck equation comes from, then this section is for you.

We start by showing that the Fokker-Planck is a necessary condition, i.e. if $X_t \sim p_t$, then the Fokker-Planck equation is fulfilled. The trick for the proof is to use **test functions** f, i.e. functions $f : \mathbb{R}^d \to \mathbb{R}$ that are infinitely differentiable ("smooth") and are only non-zero within a bounded domain (compact support). We use the fact that for arbitrary integrable functions $g_1, g_2 : \mathbb{R}^d \to \mathbb{R}$ it holds that

$$g_1(x) = g_2(x) \text{ for all } x \in \mathbb{R}^d \quad \Leftrightarrow \quad \int f(x)g_1(x)\mathrm{d}x = \int f(x)g_2(x)\mathrm{d}x \text{ for all test functions } f \tag{92}$$

In other words, we can express the pointwise equality as equality of taking integrals. The useful thing about test functions is that they are smooth, i.e. we can take gradients and higher-order derivatives. In particular, we can use **integration by parts** for arbitrary test functions f_1, f_2:

$$\int f_1(x)\frac{\partial}{\partial x_i}f_2(x)\mathrm{d}x = -\int f_2(x)\frac{\partial}{\partial x_i}f_1(x)\mathrm{d}x \tag{93}$$

By using this together with the definition of the divergence and Laplacian (see eq. (23)), we get the identities:

$$\int \nabla f_1^T(x) f_2(x)\mathrm{d}x = -\int f_1(x)\mathrm{div}(f_2)(x)\mathrm{d}x \quad (f_1 : \mathbb{R}^d \to \mathbb{R}, f_2 : \mathbb{R}^d \to \mathbb{R}^d) \tag{94}$$

$$\int f_1(x)\Delta f_2(x)\mathrm{d}x = \int f_2(x)\Delta f_1(x)\mathrm{d}x \quad (f_1 : \mathbb{R}^d \to \mathbb{R}, f_2 : \mathbb{R}^d \to \mathbb{R}) \tag{95}$$

Now let's proceed to the proof. We use the stochastic update of SDE trajectories as in eq. (6):

$$X_{t+h} = X_t + hu_t(X_t) + \sigma_t(W_{t+h} - W_t) + hR_t(h) \tag{96}$$

$$\approx X_t + hu_t(X_t) + \sigma_t(W_{t+h} - W_t) \tag{97}$$

where for now we simply ignore the error term $R_t(h)$ for readability as we will take $h \to 0$ anyway. We can then make the following calculation:

$$\begin{aligned}
&f(X_{t+h}) - f(X_t) \\
&\overset{(97)}{=} f(X_t + hu_t(X_t) + \sigma_t(W_{t+h} - W_t)) - f(X_t) \\
&\overset{(i)}{=} \nabla f(X_t)^T \left(hu_t(X_t) + \sigma_t(W_{t+h} - W_t)\right) \\
&\quad + \frac{1}{2}\left(hu_t(X_t) + \sigma_t(W_{t+h} - W_t)\right)^T \nabla^2 f(X_t) \left(hu_t(X_t) + \sigma_t(W_{t+h} - W_t)\right) \\
&\overset{(ii)}{=} h\nabla f(X_t)^T u_t(X_t) + \sigma_t \nabla f(X_t)^T (W_{t+h} - W_t) \\
&\quad + \frac{1}{2}h^2 u_t(X_t)^T \nabla^2 f(X_t) u_t(X_t) + h\sigma_t u_t(X_t)^T \nabla^2 f(X_t)(W_{t+h} - W_t) + \frac{1}{2}\sigma_t^2 (W_{t+h} - W_t)^T \nabla^2 f(X_t)(W_{t+h} - W_t)
\end{aligned}$$

where in (i) we used a 2nd Taylor approximation of f around X_t and in (ii) we used the fact that the Hessian $\nabla^2 f$ is a symmetric matrix. Note that $\mathbb{E}[W_{t+h} - W_t | X_t] = 0$ and $W_{t+h} - W_t | X_t \sim \mathcal{N}(0, hI_d)$. Therefore

$$\begin{aligned}
&\mathbb{E}[f(X_{t+h}) - f(X_t)|X_t] \\
=&h\nabla f(X_t)^T u_t(X_t) + \frac{1}{2}h^2 u_t(X_t)^T \nabla^2 f(X_t) u_t(X_t) + \frac{h}{2}\sigma_t^2 \mathbb{E}_{\epsilon_t \sim \mathcal{N}(0,I_d)}[\epsilon_t^T \nabla^2 f(X_t)\epsilon_t] \\
\overset{(i)}{=}&h\nabla f(X_t)^T u_t(X_t) + \frac{1}{2}h^2 u_t(X_t)^T \nabla^2 f(X_t) u_t(X_t) + \frac{h}{2}\sigma_t^2 \text{trace}(\nabla^2 f(X_t)) \\
\overset{(ii)}{=}&h\nabla f(X_t)^T u_t(X_t) + \frac{1}{2}h^2 u_t(X_t)^T \nabla^2 f(X_t) u_t(X_t) + \frac{h}{2}\sigma_t^2 \Delta f(X_t)
\end{aligned}$$

where in (i) we used the fact that $\mathbb{E}_{\epsilon_t \sim \mathcal{N}(0,I_d)}[\epsilon_t^T A \epsilon_t] = \text{trace}(A)$ and in (ii) we used the definition of the Laplacian and the Hessian matrix. With this, we get that

$$\begin{aligned}
&\partial_t \mathbb{E}[f(X_t)] \\
=&\lim_{h\to 0}\frac{1}{h}\mathbb{E}[f(X_{t+h}) - f(X_t)] \\
=&\lim_{h\to 0}\frac{1}{h}\mathbb{E}[\mathbb{E}[f(X_{t+h}) - f(X_t)|X_t]] \\
=&\mathbb{E}[\lim_{h\to 0}\frac{1}{h}\left(h\nabla f(X_t)^T u_t(X_t) + \frac{1}{2}h^2 u_t(X_t)^T \nabla^2 f(X_t) u_t(X_t) + \frac{h}{2}\sigma_t^2 \Delta f(X_t)\right)] \\
=&\mathbb{E}[\nabla f(X_t)^T u_t(X_t) + \frac{1}{2}\sigma_t^2 \Delta f(X_t)] \\
\overset{(i)}{=}&\int \nabla f(x)^T u_t(x) p_t(x) \mathrm{d}x + \int \frac{1}{2}\sigma_t^2 \Delta f(x) p_t(x) \mathrm{d}x \\
\overset{(ii)}{=}&-\int f(x) \text{div}(u_t p_t)(x) \mathrm{d}x + \int \frac{1}{2}\sigma_t^2 f(x) \Delta p_t(x) \mathrm{d}x \\
=&\int f(x)\left(-\text{div}(u_t p_t)(x) + \frac{1}{2}\sigma_t^2 \Delta p_t(x)\right)\mathrm{d}x
\end{aligned}$$

where in (i) we used the assumption that p_t as the distribution of X_t and in (ii) we used eq. (94) and eq. (95). Therefore, it holds that

$$\partial_t \mathbb{E}[f(X_t)] = \int f(x)\left(-\text{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2}\Delta p_t(x)\right)\mathrm{d}x \quad \text{(for all } f \text{ and } 0 \le t \le 1) \tag{98}$$

$$\overset{(i)}{\Leftrightarrow} \quad \partial_t \int f(x) p_t(x) \mathrm{d}x = \int f(x)\left(-\text{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2}\Delta p_t(x)\right)\mathrm{d}x \quad \text{(for all } f \text{ and } 0 \le t \le 1) \tag{99}$$

$$\overset{(ii)}{\Leftrightarrow} \quad \int f(x)\partial_t p_t(x) \mathrm{d}x = \int f(x)\left(-\text{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2}\Delta p_t(x)\right)\mathrm{d}x \quad \text{(for all } f \text{ and } 0 \le t \le 1) \tag{100}$$

$$\overset{(iii)}{\Leftrightarrow} \quad \partial_t p_t(x) = -\text{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2}\Delta p_t(x) \quad \text{(for all } x \in \mathbb{R}^d, 0 \le t \le 1) \tag{101}$$

where in (i) we used the assumption that $X_t \sim p_t$, in (ii) we swapped the derivative with the integral and (iii) we used eq. (92) . This completes the proof that the Fokker-Planck equation is a necessary condition.

Finally, we explain why it is also a sufficient condition. The Fokker-Planck equation is a partial differential equation (PDE). More specifically, it is a so-called *parabolic partial differential equation*. Similar to theorem 3,

such differential equations have a unique solution given fixed initial conditions (see e.g. [8, Chapter 7]). Now, if eq. (30) holds for p_t, we just shown above that it must also hold for true distribution q_t of X_t (i.e. $X_t \sim q_t$) - in other words, both p_t and q_t are solutions to the parabolic PDE. Further, we know that the initial conditions are the same, i.e. $p_0 = q_0 = p_{\text{init}}$ by construction of an interpolating probability path (see **??**). Hence, by uniqueness of the solution of the differential equation, we know that $p_t = q_t$ for all $0 \leq t \leq 1$ - which means $X_t \sim q_t = p_t$ and which is what we wanted to show.