4. Diffusion Models in Vision

In computer vision, diffusion models refer to a class of generative models that define a parameterized transformation from a simple prior to the data distribution. An important application is image synthesis, where samples are generated by iteratively denoising Gaussian noise. Although diffusion models have roots in non-equilibrium thermodynamics and variational inference, they can be elegantly interpreted using differential equations. As discussed in previous sections, residual networks represent a simple Euler discretization of ODEs, whereas neural ODEs model the continuous-time dynamics directly. Diffusion models link these concepts. Conceptually, they define a continuous-time evolution (similar to neural ODEs/SDEs), yet in practice, they are often optimized and deployed using discrete time steps, akin to deep residual networks.

In this section, we begin with a problem formulation of generative modeling by considering a dual process of forward and backward transformations. Then we introduce two perspectives for modeling the generation problem: the stochastic perspective and the deterministic perspective. The former is based on **stochastic differential equations** (SDEs), which frame generation as stochastic processes of adding and removing noise. The latter is based on **probability flows**, which treat generation as a deterministic process. Finally, we discuss methods for improving the efficiency of diffusion models.

Diffusion models have been extensively discussed in many papers [Croitoru et al., 2023; Yang et al., 2023] and blogs [Song, 2021; Dieleman, 2023]. This section does not aim to cover every detail of these models. Instead, we focus on discussing them from the differential equation perspective. Occasionally, we extend our discussion a bit to ensure completeness.

4.1 Problem Statement

Prior to introducing diffusion models, we provide a formal definition of the generative modeling problem.

4.1.1 Generative Modeling as Probability Transport

In the standard setting of generative modeling, the goal is to learn the underlying probability distribution of a dataset, denoted as $p_{\text{data}}(\cdot)$, and to generate new samples from it. For instance, if we knew the true distribution of natural images, we could generate a novel image $\mathbf{x}$ simply by sampling from it. However, since the true distribution is unknown, the typical approach is to approximate it by learning from a set of observed images.

To achieve this, we usually rely on the concept of **probability transport**. Since directly modeling and sampling from the complex data distribution $p_{\text{data}}(\mathbf{x})$ is difficult, we instead introduce a latent variable $\mathbf{z}$ sampled from a tractable prior distribution $p_{\text{prior}}(\mathbf{z})$, such as a standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The generation task is then reduced to finding a deterministic or stochastic mapping $\Phi : \mathbb{R}^d \to \mathbb{R}^d$ that transforms the latent variable $\mathbf{z}$ into a data sample $\mathbf{x} \approx \Phi(\mathbf{z})$.

From the perspective of differential equations, this mapping is not instantaneous but is induced by a continuous-time dynamic process. In this process, $\mathbf{x}$ and $\mathbf{z}$ are viewed as the states of a system at the start and end of a trajectory. While neural ODEs (discussed in Section 3) generally model dynamics over an arbitrary interval $[t_0, t_1]$, diffusion models specifically formulate this transport

as a continuous-time evolution over a fixed interval $t \in [0, T]$. So we follow the convention that the process starts from the data distribution at $t = 0$ and evolves to the prior distribution at $t = T$, or vice versa.

Formally, we can view this dynamic evolution as a flow. Let $\Phi_{0\to t} : \mathbb{R}^d \to \mathbb{R}^d$ denote the flow map that transports a sample from its state at time 0 to its state at time t. While this map operates on individual samples, it induces a corresponding transformation on the probability distribution itself. This relationship is mathematically described by the **pushforward** operation. Specifically, if we start with the data distribution $p_0(\mathbf{x})$, the distribution $p_t(\mathbf{x})$ at any intermediate time t is the pushforward of p_0 by the flow map, denoted as

$$p_t = (\Phi_{0\to t})_{\#} p_0. \tag{1}$$

This equality implies that probability mass is conserved during the transport: if a random variable $\mathbf{x}$ follows p_0, then the transformed variable $\Phi_{0\to t}(\mathbf{x})$ follows p_t. In general, the flow is assumed to be a diffeomorphism, and thus this operation is fully invertible. Then, we can define the generation process as the reverse pushforward $p_0 = (\Phi_{T\to 0})_{\#} p_T$, which transports the simple prior p_T back to the complex data distribution p_0.

To understand the pushforward operation and the evolution of probability distributions, consider the analogy of dropping colored dye into flowing water. In this example, the probability distribution corresponds to the concentration of the dye, and the data samples correspond to individual dye molecules. The flow map $\Phi_{0\to t}$ can be thought of as the current of the water, which determines the precise trajectory of every molecule over time. As these molecules follow the current, the overall shape of the dye cloud changes. The pushforward operation is the mathematical tool that helps us determine the density of the dye at any specific location and time, based on the movements of the molecules. See Figure 1 for an illustration.

Note that, although the local density $p_t(\mathbf{x})$ changes as the fluid expands or contracts, the total amount of dye remains invariant. This means that the integral of the probability density over the entire space is always equal to 1. Thus, by defining the flow of the "particles" (samples), we implicitly and uniquely define the evolution of the "cloud" (distribution).

Now, we have our task of generative modeling: we learn a reverse flow $\Phi_{T\to 0}$, as well as the corresponding pushforward operation $(\Phi_{T\to 0})_{\#} p_T$, such that we can pushforward samples from the prior noise to the data distribution. The final output is a sample drawn from this data distribution. As $\Phi_{T\to 0}$ is a flow, we can model it using ODEs or related tools in differential equations. In the rest of this section, we will see several methods for modeling probability transport, which can be well interpreted using differential equations.

Although a flow can be seen as a single, continuous-time evolution of system states, in practice, we typically discretize it into a number of simpler steps to facilitate modeling and learning. Instead of attempting to learn a monolithic mapping that directly transports samples between the noise and data distributions, we decompose the process into a sequence of incremental transitions. Specifically, let N be the number of discretization steps and $0 = t_0 < t_1 < \cdots < t_N = T$ be the time points. We represent the trajectory from $\mathbf{z}$ to $\mathbf{x}$ as $\mathbf{z} = \mathbf{x}(t_N) \to \mathbf{x}(t_{N-1}) \to \cdots \to \mathbf{x}(t_1) \to \mathbf{x}(t_0) = \mathbf{x}$. In image generation, this means that we start with a purely noisy

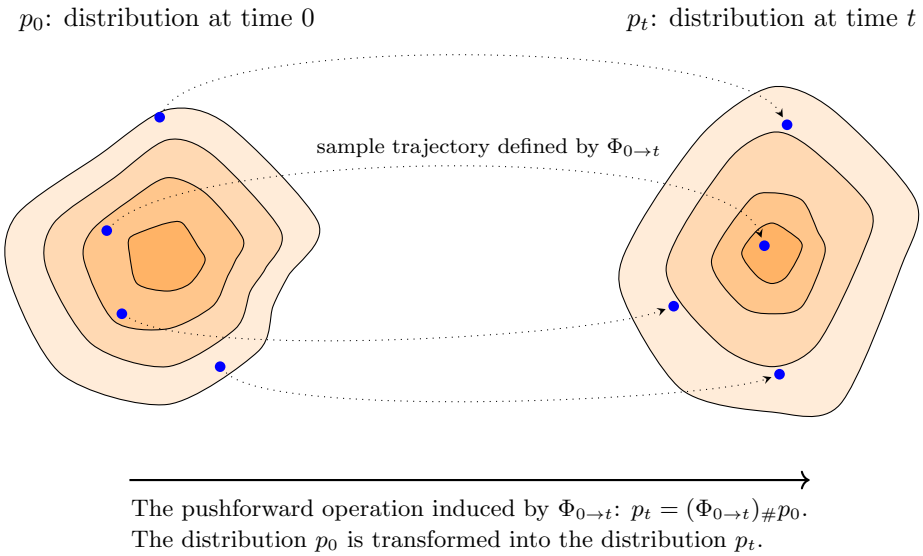


Figure 1: Illustration of the pushforward operation. The distributions at time 0 and time t are visualized as the evolution of dye density in a fluid. The map $\Phi_{0\to t}$ describes the trajectories of individual dye molecules over time. As these molecules move to new positions, the local density varies, yet the total mass remains conserved. The pushforward operation formally defines this transformation, mapping the initial density distribution p_0 to the resulting distribution p_t based on the collective trajectories defined by Φ.

image, and remove the noise step by step to generate a realistic image. At any specific step k, the transformation required to map the distribution p_{t_k} to the subsequent distribution $p_{t_{k-1}}$ is simple. By decomposing the global transport problem into these local transitions, the model is only required to learn the direction of the flow (i.e., the vector field) at each step.

4.1.2 The Duality of Forward and Backward Processes

As discussed above, central to the design of diffusion models is the division of the task into two processes: a process that destroys data and a process that creates it. We refer to these as the **forward process** and the **backward process**, respectively.

The forward process defines the transformation from the data distribution to the prior distribution as time evolves from $t = 0$ to T. This process can be formulated either stochastically or deterministically. This leads to two different perspectives in designing diffusion models, as will be presented later in this section. As a simple example, here we describe the forward process using deterministic differential equations via the perspective of ODEs. In this view, we model the sample trajectories via an ODE

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t), \tag{2}$$

where $f(\mathbf{x}(t), t)$ is a vector field which is typically pre-defined (e.g., Gaussian noise injection or linear interpolation). Given the flow defined by this ODE, we can transform the distribution of the samples from time 0 to any time t using the pushforward operation (see Eq. (1)).

The backward process is the reverse of the forward process. If we take the ODE view, it corresponds to the time-reversal of the dynamics, evolving from $t = T$ back to 0. Given the forward ODE above, the generative trajectories are defined by the reverse-time ODE

$$\frac{d\mathbf{x}(t)}{dt} = -f(\mathbf{x}(t), t), \tag{3}$$

which is solved backwards from T to 0. Note that the ideal backward vector field is simply the negation of the forward field, which we can approximate using a neural network.

This bi-directional framework naturally aligns with the architecture of autoencoders [Kingma and Welling, 2019; Bank et al., 2023], as illustrated in Figure 2. The forward process corresponds to the encoder, which maps the observed data $\mathbf{x}$ (i.e., $\mathbf{x}(0)$) to the latent variable $\mathbf{z}$ (i.e., $\mathbf{x}(T)$) through a sequence of intermediate states $\{\mathbf{x}(t)\}$. The backward process corresponds to the decoder, which reconstructs the data $\mathbf{x}$ from the latent variable $\mathbf{z}$ through the reverse sequence.

However, there are some differences between diffusion models and traditional autoencoders:

- First, regarding dimensionality, standard autoencoders typically perform dimensionality reduction: the dimension of the latent variable $\mathbf{z}$ is smaller than that of the input $\mathbf{x}$. In contrast, diffusion models generally maintain the same dimensionality throughout the process. The input data $\mathbf{x}$, the latent variable $\mathbf{z}$, and all intermediate states share the same dimension $\mathbb{R}^d$. This aligns with the task objective where we transform samples rather than changing their dimensionality.

- Second, regarding the functional relationship between the forward and backward paths, the general autoencoder framework does not require the encoder and decoder to share the same model architecture or parameters. They can be parameterized by distinct neural networks that are optimized jointly but operate as separate functions. In contrast, in the context of probability flow ODEs for diffusion models, the forward and backward processes are intrinsically coupled. As shown in Eqs. (2) and (3), the backward process is governed by the negation of the forward vector field. This implies a mathematical symmetry: defining the dynamics of the forward evolution implicitly defines the dynamics of the backward generation process.

Despite these differences, the theoretical foundations of autoencoders provide valuable insights for training diffusion models [Luo, 2022; Kingma and Gao, 2023]. For example, the **evidence lower bound** (ELBO), which is a commonly used loss function in **variational autoencoders** (VAEs), is closely related to the objective used in training diffusion models[1]. In this section, we

1. By maximizing the ELBO [Kingma and Welling, 2013], we minimize the divergence (typically the Kullback-Leibler divergence) between the forward and backward transition distributions at every time step. This formulation links the physical intuition of "reversing the flow" with the statistical objective of "maximizing data likelihood". As a result, the problem of generative modeling is reduced to a regression task: the model learns to estimate the vector field that generated the current state, thereby learning the reversal of the process.

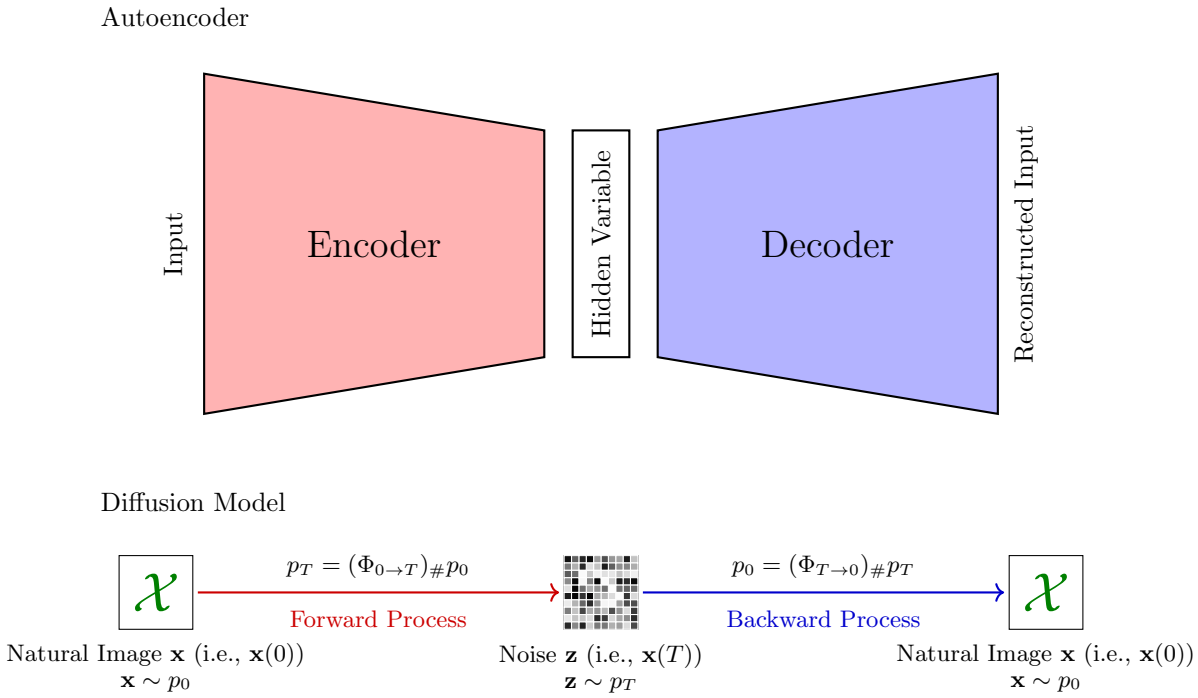


Figure 2: Comparison between autoencoders and diffusion models. The top panel depicts a standard autoencoder architecture, which maps inputs to a lower-dimensional hidden variable space via an encoder and reconstructs them via a decoder. The bottom panel depicts a diffusion model. In contrast to autoencoders, which rely on a bottleneck architecture to compress data, diffusion models define an invertible mapping between the data distribution and a noise distribution. The forward process pushes the data distribution p_0 to a simple noise prior p_T, while the backward process maps the distribution back to p_0. Here, $\mathbf{x}$ represents a natural image drawn from p_0, and $\mathbf{z}$ represents noise drawn from p_T. $(\Phi_{0\to T})_{\#}$ and $(\Phi_{T\to 0})_{\#}$ denote the pushforward operations for the forward and backward processes, respectively.

will not discuss the details of autoencoders, but will occasionally use them as a tool for interpreting diffusion models.

4.1.3 Training and Inference

While the forward and backward processes are theoretically symmetric, they play distinct roles. In the inference phase, our goal is to generate new samples. Thus, we only execute the backward process. We initialize the system with a latent variable $\mathbf{z}$ (i.e., $\mathbf{x}(T)$) sampled from the prior distribution, and apply the learned reverse dynamics to iteratively map the noise back to a data sample $\mathbf{x}$ (i.e., $\mathbf{x}(0)$).

A natural question arises: if the actual application only involves the backward process, why is it necessary to explicitly define the forward process? To answer this, consider the analogy of traversing a maze: if we have walked from the entrance to the exit, finding the way back is difficult

without observing valid paths. The forward process serves as the guide that constructs these training paths. By transforming a data sample $\mathbf{x}$ into noise $\mathbf{z}$ through a sequence of intermediate states, the forward process provides the necessary supervision signals. It tells the model, for any given time t, what the correct velocity (or direction) should be to return towards the data distribution.

Therefore, the training of diffusion models typically involves two steps. First, we execute the forward process to generate the trajectories from data to noise, thereby creating the ground truth paths. Then, we optimize the parameters of the backward process to accurately match this ground truth flow.

4.2 The Stochastic Perspective: SDEs

While we have seen that ODEs provide a deterministic view of generative modeling, it is important to recognize that the underlying mechanism of diffusion is inherently random. From this perspective, the forward process is modeled not as a deterministic flow, but as a stochastic process that gradually corrupts the input sample with noise. Indeed, the canonical formulation of these models relies on stochastic differential equations (SDEs) to describe the dynamics of data corruption and reconstruction [Oksendal, 2013]. In what follows, we introduce SDE-based diffusion models.

4.2.1 Forward and Reverse SDEs

An SDE is a differential equation in which one or more terms are stochastic processes. Typically, it describes the evolution of a system that is subject to both deterministic forces (predictable trends) and random noise (unpredictable fluctuations).

To understand this, let us first recall the standard ODE describing the evolution of a state vector $\mathbf{x}(t)$ over time t: $\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t)$. Here, $f(\cdot)$ is a deterministic vector field that describes the velocity of the state. Given an initial condition $\mathbf{x}(0)$, the future trajectory is entirely determined.

However, in diffusion models, we introduce random noise to the data continuously. To formulate this, we must add a stochastic term to the equation. Since standard Brownian motion is nowhere differentiable, we cannot simply write $\frac{d\mathbf{x}(t)}{dt} = f + \text{noise}$. Instead, we write the equation in differential form involving a random component

$$d\mathbf{x}(t) \quad = \quad \underbrace{f(\mathbf{x}(t), t)dt}_{\text{deterministic}} + \underbrace{g(t)d\mathbf{w}(t)}_{\text{stochastic}}. \tag{4}$$

This is a generic Itô SDE[2]. The components of this equation are defined as follows

- The drift coefficient $f(\mathbf{x}(t), t) \in \mathbb{R}^d$ represents the deterministic part of the evolution, similar to the vector field in an ODE. It determines the general trend of the evolution.

2. Stochastic calculus requires a specific interpretation of the integral $\int g(t)d\mathbf{w}(t)$. The **Itô interpretation** evaluates the integrand at the beginning of each time interval (i.e., the left endpoint) during discretization [Karatzas and Shreve, 2014]. This choice is standard in diffusion models because it ensures that the noise increment $d\mathbf{w}$ is independent of the current state, thereby preserving the martingale property and simplifying the computation of expectations.

- The diffusion coefficient $g(t)$ is typically a scalar function that controls the magnitude of the stochastic noise injected into the system at time t.
- The Wiener process (also called Brownian motion) $\mathbf{w}(t) \in \mathbb{R}^d$ represents the source of randomness. The increment $d\mathbf{w}(t)$ can be thought of as an infinitesimal Gaussian noise vector with mean $\mathbf{0}$ and covariance $dt \cdot \mathbf{I}$.

Eq. (4) is called the forward SDE. By using this SDE, we define a forward process that gradually adds noise to a data sample $\mathbf{x}(0) \sim p_{\text{data}}$ as time t flows from 0 to T. During this process, the distribution $p_t(\mathbf{x}(t))$ diffuses, and the distribution of $\mathbf{x}(T)$ will eventually converge to a known prior distribution (usually $\mathcal{N}(\mathbf{0}, \mathbf{I})$), as illustrated in Figure 3.

The remarkable property of SDEs, derived by Anderson [1982], is that for any forward SDE, there exists a corresponding reverse-time SDE. If we simulate the process backwards in time from $t = T$ to 0, we can generate samples from the data distribution. The reverse SDE is given by

$$d\mathbf{x}(t) \quad = \quad \left[f(\mathbf{x}(t), t) - g(t)^2 \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))\right] dt + g(t) d\bar{\mathbf{w}}(t), \tag{5}$$

where dt represents an infinitesimal negative time increment, $d\bar{\mathbf{w}}(t)$ is a standard Wiener process in the reverse time, and $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$ is the gradient of the log-probability density with respect to the data, also called the **score function**. Although this equation appears a bit complicated, its application is straightforward. Given a forward SDE (Eq. (4)), if we can access the score function, we can use the reverse SDE (Eq. (5)) to transport samples from the prior noise p_T back to the data distribution p_0.

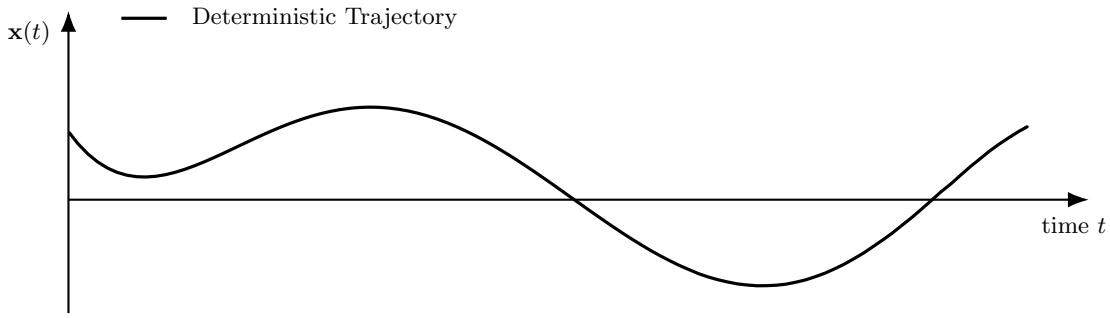
There are two fundamental issues when applying the reverse SDE to generative modeling.

- **How to obtain the score function in the reverse SDE?** The backward process depends on the gradient of the log-density $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$. However, the intermediate marginal distribution $p_t(\mathbf{x}(t))$ is obtained by marginalizing over all possible data samples and diffusion paths, making it analytically intractable. We only have access to samples from the data distribution p_0, not an explicit expression for p_t.
- **How to define the forward SDE?** We need to specify the drift coefficient $f(\mathbf{x}(t), t)$ and the diffusion coefficient $g(t)$. These functions determine the trajectory of the diffusion process. They must be carefully designed so that the complex data distribution p_0 is effectively transformed into a simple prior distribution p_T at the end of the process.

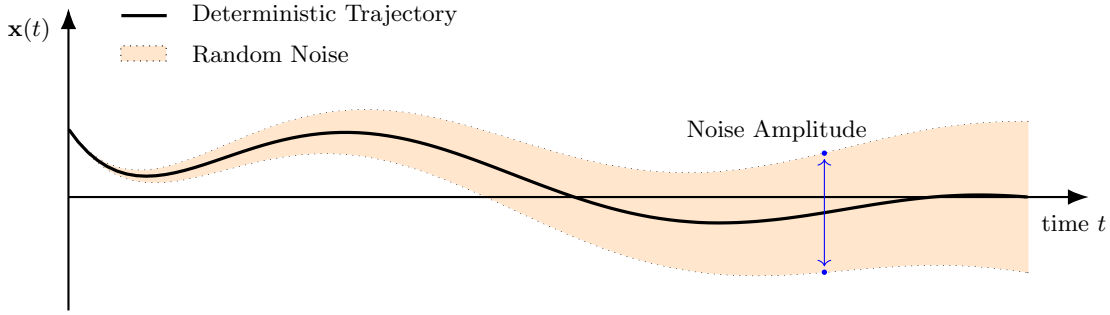
In what follows, we address these issues by introducing the **score matching** technique and discussing SDE formulations commonly used in the literature.

4.2.2 Learning the Score Function

As mentioned above, since the marginal distribution $p_t(\mathbf{x}(t))$ is generally intractable, we cannot compute the gradient $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$ directly. Instead, we resort to score matching [Hyvärinen and Dayan, 2005], in particular the **denoising score matching** technique [Vincent, 2011].



(a) ODE (deterministic trajectory)



(b) SDE (with random noise)

Figure 3: Deterministic vs. stochastic trajectories. For comparison, subfigure (a) shows a deterministic trajectory governed by an ODE. Subfigure (b) shows the trajectory of the SDE defined by $d\mathbf{x}(t) = f(\mathbf{x}(t), t)dt + g(t)d\mathbf{w}(t)$. The term $f(\mathbf{x}(t), t)dt$ represents the deterministic drift (analogous to the ODE), and $g(t)d\mathbf{w}(t)$ introduces Gaussian noise. In this example, the drift coefficient $f(\mathbf{x}(t), t)$ gradually decays to zero, whereas the diffusion coefficient $g(t)$ increases over time. At the terminal time T, we have $f(\mathbf{x}(T), T) = 0$ and $g(T) = 1$, and the state converges to a standard Gaussian distribution. This behavior is analogous to the forward process of a diffusion model: as time progresses, more noise is injected while the deterministic signal diminishes.

The core idea is that while the score of the marginal distribution is unknown, the score of the conditional transition distribution $p(\mathbf{x}(t)|\mathbf{x}(0))$ is often easy to compute. Specifically, consider a transition kernel where Gaussian noise $\boldsymbol{\epsilon}$ is added to a data point $\mathbf{x}(0)$ such that $\mathbf{x}(t) \sim \mathcal{N}(\mathbf{x}(0), \sigma(t)^2\mathbf{I})$, that is,

$$\mathbf{x}(t) \quad = \quad \mathbf{x}(0) + \boldsymbol{\epsilon}. \tag{6}$$

The score of this conditional distribution is given by

$$\begin{aligned} \nabla_{\mathbf{x}(t)} \log p(\mathbf{x}(t)|\mathbf{x}(0)) &= -\frac{\mathbf{x}(t) - \mathbf{x}(0)}{\sigma(t)^2} \\ &= -\frac{\boldsymbol{\epsilon}}{\sigma(t)^2}. \end{aligned} \tag{7}$$

Vincent [2011] proved that training a neural network $s_\theta(\mathbf{x}(t), t)$ to approximate this conditional score (which is proportional to the added noise $\boldsymbol{\epsilon}$) is equivalent to learning the marginal score $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$. Thus, the objective of generative modeling can be effectively reduced to a denoising regression problem.

Formally, this intuition leads to the following objective function. We seek to minimize the expected squared Euclidean distance between the estimated score and the true conditional score:

$$\mathcal{L}(\theta) = \mathbb{E}_{t,\mathbf{x}(0),\boldsymbol{\epsilon}} \left[\lambda(t) \left\| s_\theta(\mathbf{x}(t), t) - \left(-\frac{\boldsymbol{\epsilon}}{\sigma(t)^2} \right) \right\|^2 \right], \tag{8}$$

where $\lambda(t)$ is a positive weighting function. A common choice is $\lambda(t) = \sigma(t)^2$, which balances the magnitude of the score across different noise levels.

By substituting this weighting and rearranging the terms, the objective simplifies to

$$\mathcal{L}(\theta) = \mathbb{E}_{t,\mathbf{x}(0),\boldsymbol{\epsilon}} \left[\frac{1}{\sigma(t)^2} \left\| \boldsymbol{\epsilon} + \sigma(t)^2 s_\theta(\mathbf{x}(0) + \boldsymbol{\epsilon}, t) \right\|^2 \right]. \tag{9}$$

This formulation highlights that optimizing the score model is mathematically equivalent to training a network to predict the noise $\boldsymbol{\epsilon}$ added to the sample. In practice, the weighting factor $1/\sigma(t)^2$ is often discarded to prioritize perceptual quality, which leads to a simple noise prediction objective.

It is worth noting that the above approach shares some theoretical connections with **energy-based models** (EBMs) [LeCun et al., 2006]. In statistical physics, a probability distribution is often expressed via an energy function $E_\theta(\mathbf{x})$ as $p_\theta(\mathbf{x}) = \frac{\exp(-E_\theta(\mathbf{x}))}{Z_\theta}$, where Z_θ is the normalization constant or partition function.

An advantage of modeling the score function, rather than the likelihood directly, is that it can bypass the computation of the intractable normalization constant Z_θ. For a model to be normalized, the integral of the density over the entire space must equal 1. For example, language models use a Softmax function to normalize logits, which requires a summation over the vocabulary. However, for high-dimensional continuous data like images, calculating $Z_\theta = \int \exp(-E_\theta(\mathbf{x}))d\mathbf{x}$ is computationally intractable. Score-based models avoid this issue because the gradient of the log-partition function with respect to the data is zero, i.e., $\nabla_{\mathbf{x}} \log Z_\theta = 0$. Thus, we have

$$\begin{aligned} \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) &= \nabla_{\mathbf{x}} \left(-E_\theta(\mathbf{x}) - \log Z_\theta \right) \\ &= -\nabla_{\mathbf{x}} E_\theta(\mathbf{x}). \end{aligned} \tag{10}$$

This property allows us to train unnormalized models without ever evaluating the partition function.

Once the score network $s_\theta(\mathbf{x}, t)$ is trained, we can use it to generate new samples. Specifically, we substitute our learned approximation $s_\theta(\mathbf{x}(t), t) = \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$ into Eq. (5), and obtain

$$d\mathbf{x}(t) = \left[f(\mathbf{x}(t), t) - g(t)^2 s_\theta(\mathbf{x}(t), t)\right] dt + g(t) d\bar{\mathbf{w}}(t). \tag{11}$$

At test time, we initialize $\mathbf{x}(T)$ from the prior distribution (e.g., $\mathcal{N}(\mathbf{0}, \mathbf{I})$) and integrate this equation backwards from $t = T$ to $t = 0$ using numerical solvers such as the Euler-Maruyama method [Kloeden and Pearson, 1977]. By gradually removing the noise, this procedure transforms the initial noisy sample into a sample from the target data distribution.

4.2.3 The VE and VP SDEs

We now turn to the problem of defining the forward SDE, that is, we define the drift coefficient $f(\mathbf{x}(t), t)$ and the diffusion coefficient $g(t)$ in Eq. (4). There are many ways to construct a forward SDE. Here we consider two commonly used methods. They correspond to the continuous-time limits of two discrete-time models: **noise conditional score network** (NCSN) [Song and Ermon, 2019] and **denoising diffusion probabilistic models** (DDPM) [Ho et al., 2020]. Song et al. [2021] unified these methods under the SDE framework, and termed them **variance exploding** (VE) and **variance preserving** (VP) SDEs.

In the NCSN framework, the forward process perturbs the data with a sequence of increasing Gaussian noise levels $\{\sigma_1, \sigma_2, \ldots, \sigma_N\}$, where $\sigma_1 < \sigma_2 < \cdots < \sigma_N$. Consider a discrete process where the state at step i, denoted by $\mathbf{x}_i$, evolves from $\mathbf{x}_{i-1}$ by adding the necessary amount of noise to increase the total variance from σ_{i-1}^2 to σ_i^2. Then, the update rule is given by

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2}\boldsymbol{\epsilon}_i, \tag{12}$$

where $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is the incremental noise at step i. This model ensures that if we start with data $\mathbf{x}_0$, the distribution at step i has a variance of σ_i^2[3].

Now, let us move to the continuous limit. We define a continuous variance function $\sigma(t)$ with respect to time t. As the number of steps approaches infinity ($N \to \infty$), the discrete time interval approaches zero ($\Delta t \to 0$). The difference in variance between adjacent steps becomes a differential

$$\sigma_i^2 - \sigma_{i-1}^2 \approx \frac{d\sigma^2(t)}{dt}\Delta t. \tag{13}$$

Recall that in stochastic calculus, the increment of a Wiener process $d\mathbf{w}$ scales with $\sqrt{\Delta t}$ (i.e., $\sqrt{\Delta t}\boldsymbol{\epsilon}_i \approx d\mathbf{w}(t)$). We can rewrite the update rule by substituting the variance difference

$$\mathbf{x}_i - \mathbf{x}_{i-1} \approx \sqrt{\frac{d\sigma^2(t)}{dt}\Delta t}\,\boldsymbol{\epsilon}_i. \tag{14}$$

3. More precisely, since the initial state $\mathbf{x}_0$ is given, we should say that the conditional distribution given $\mathbf{x}_0$ has a variance of σ_i^2

Taking the limit $\Delta t \to 0$, we obtain the continuous-time SDE

$$d\mathbf{x}(t) \quad = \quad \sqrt{\frac{d\sigma^2(t)}{dt}} d\mathbf{w}(t). \tag{15}$$

In this SDE, the drift coefficient is defined as $f(\mathbf{x}(t), t) = \mathbf{0}$, and the diffusion coefficient is defined as $g(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}$. Since the variance of the distribution increases monotonically as $t \to T$, this SDE maps the data distribution to a noise distribution with diverging variance. Hence, it is referred to as the variance exploding SDE.

Alternatively, the DDPM framework adopts a different noise injection strategy. Instead of simply adding noise with increasing variance, it rescales the data at each step to keep the variance bounded. In the discrete formulation, the forward process is defined by a variance schedule $0 < \beta_1 < \beta_2 < \cdots < \beta_N < 1$. The transition from $\mathbf{x}_{i-1}$ to $\mathbf{x}_i$ is given by

$$\mathbf{x}_i = \sqrt{1 - \beta_i}\mathbf{x}_{i-1} + \sqrt{\beta_i}\boldsymbol{\epsilon}_i, \tag{16}$$

where $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Unlike the NCSN formulation, the coefficient $\sqrt{1 - \beta_i}$ shrinks the mean of the previous state. This design ensures that if the input data follows a standard normal distribution, the marginal distribution remains a standard normal distribution at every step (i.e., the variance is preserved).

Then, the increment $\mathbf{x}_i - \mathbf{x}_{i-1}$ can be expressed as

$$\mathbf{x}_i - \mathbf{x}_{i-1} \quad = \quad (\sqrt{1 - \beta_i} - 1)\mathbf{x}_{i-1} + \sqrt{\beta_i}\boldsymbol{\epsilon}_i. \tag{17}$$

To obtain the continuous-time SDE, we again let $N \to \infty$ and introduce a continuous function $\beta(t)$ such that $\beta_i \approx \beta(t)\Delta t$. Using the Taylor expansion approximation $\sqrt{1 - x} \approx 1 - \frac{1}{2}x$ for small x, we have $\sqrt{1 - \beta(t)\Delta t} - 1 \approx -\frac{1}{2}\beta(t)\Delta t$. Substituting this back into Eq. (17) and applying the scaling rule $\sqrt{\beta(t)\Delta t}\boldsymbol{\epsilon}_i \approx \sqrt{\beta(t)}d\mathbf{w}(t)$, we obtain the limit

$$d\mathbf{x}(t) \quad = \quad -\frac{1}{2}\beta(t)\mathbf{x}(t)dt + \sqrt{\beta(t)}d\mathbf{w}(t). \tag{18}$$

In this SDE, the drift coefficient is $f(\mathbf{x}(t), t) = -\frac{1}{2}\beta(t)\mathbf{x}(t)$, and the diffusion coefficient is $g(t) = \sqrt{\beta(t)}$. The drift term serves as a restoring force that pulls the state towards the origin (zero mean), and so counteracts the variance injected by the diffusion term. As a result, if the initial distribution has unit variance, the variance remains fixed at unity throughout the process. For this reason, Eq. (18) is called the variance preserving SDE.

Here, for brevity, we do not explicitly define the functional forms of the parameters $\{\sigma_i\}$ (or $\sigma(t)$) and $\{\beta_i\}$ (or $\beta(t)$). It is worth noting that in DDPMs, auxiliary variables (such as $\alpha_i = 1 - \beta_i$ and $\bar{\alpha}_i = \prod_{s=1}^{i} \alpha_s$) are often introduced to simplify the derivation and implementation. The careful design of the scheduling parameters can make the resulting model theoretically sound. We refer interested readers to the original papers for detailed discussions on the selection of these scheduling parameters [Song and Ermon, 2019; Ho et al., 2020].

Entry	Variance Exploding (NCSN)	Variance Preserving (DDPM)
Drift coefficient $f(\mathbf{x}(t), t)$	$\mathbf{0}$	$-\frac{1}{2}\beta(t)\mathbf{x}(t)$
Diffusion coefficient $g(t)$	$\sqrt{\frac{d\sigma^2(t)}{dt}}$	$\sqrt{\beta(t)}$
Continuous-time SDE	$d\mathbf{x}(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}d\mathbf{w}(t)$	$d\mathbf{x}(t) = -\frac{1}{2}\beta(t)\mathbf{x}(t)dt + \sqrt{\beta(t)}d\mathbf{w}(t)$
Iterative diffusion (discrete)	$\mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2}\boldsymbol{\epsilon}_i$	$\mathbf{x}_i = \sqrt{1-\beta_i}\mathbf{x}_{i-1} + \sqrt{\beta_i}\boldsymbol{\epsilon}_i$
Single-step diffusion (discrete)	$\mathbf{x}_i = \mathbf{x}_0 + \sigma_i\boldsymbol{\epsilon}$	$\mathbf{x}_i = \sqrt{\bar{\alpha}_i}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_i}\boldsymbol{\epsilon}$

Table 1: Continuous-time and discrete-time forms of the VP and VE SDEs [Song et al., 2021]. $\mathbf{x}(t)$, $\mathbf{w}(t)$, $\sigma(t)$, and $\beta(t)$ are continuous functions of time t. $\mathbf{x}_i$ is the state at discrete time step i, and β_i, σ_i, and $\bar{\alpha}_i$ are corresponding parameters. Here β_i and σ_i are specified in advance, and $\bar{\alpha}_i$ is defined as $\prod_{s=1}^{i}(1-\beta_s)$, which determines the relative weight of the signal $\mathbf{x}_0$ in the single-step diffusion process. Both $\boldsymbol{\epsilon}_i$ and $\boldsymbol{\epsilon}$ are standard Gaussian noise. The former represents the incremental noise at step i, and the latter represents the total noise perturbing $\mathbf{x}_0$ directly to $\mathbf{x}_i$.

While we focus on the continuous-time SDE formulation in this subsection to establish a more unified theoretical foundation, it is important to note that in practice, diffusion models are generally implemented via discretization steps (see Table 1). The discrete variables $\mathbf{x}_i$ introduced in the NCSN and DDPM methods essentially serve as numerical approximations of the continuous process. In actual implementation, we simulate the evolution of the SDE by iterating through these discrete time steps $\{\mathbf{x}_i\}$. This approach reflects a common paradigm in applying differential equations to deep learning: we use continuous differential equations for their theoretical completeness and expressive power, while relying on discrete numerical approximations for efficient training and inference.

4.2.4 Numerical Solvers

To generate samples from a trained diffusion model, we need to simulate the reverse-time SDE (Eq. (5)) starting from a sample $\mathbf{x}(T) \sim p_T$ and evolving backwards to $t = 0$. Since analytical solutions are rarely available for complex data distributions, we resort to numerical solvers to approximate the continuous trajectory.

The simplest and most common approach is the Euler-Maruyama method, which generalizes the Euler method to SDEs [Kloeden and Platen, 1992]. We first discretize the time interval $[0, T]$ into N steps: $t_N = T > t_{N-1} > \cdots > t_0 = 0$. For a small time step $\Delta t = t_{i+1} - t_i$, the reverse SDE can be approximated by

$$\mathbf{x}_{t_i} \approx \mathbf{x}_{t_{i+1}} - \left[f(\mathbf{x}_{t_{i+1}}, t_{i+1}) - g(t_{i+1})^2 s_\theta(\mathbf{x}_{t_{i+1}}, t_{i+1})\right]\Delta t + g(t_{i+1})\sqrt{\Delta t}\,\boldsymbol{\epsilon}_{i+1}, \tag{19}$$

Note that because we are integrating backwards in time, the computation step in the numerical solver corresponds to a negative time increment in the physical time t. This formulation recovers

the sampling steps used in the original NCSN and DDPM papers, and provides a theoretical justification for their heuristic sampling algorithms[4].

An advantage of the SDE framework is the ability to combine numerical integration with **Markov Chain Monte Carlo** (MCMC) methods [Robert et al., 1999]. For instance, by using the predictor-corrector method, the numerical discretization can be decoupled from the score-based correction. In the predictor step, a numerical solver (such as Euler-Maruyama or a higher-order solver) estimates the state at the next time step $\mathbf{x}_{t_i}$ based on the current state $\mathbf{x}_{t_{i+1}}$. This step evolves the process backward in time. After the predictor step, an MCMC method (such as Langevin dynamics) is applied to $\mathbf{x}_{t_i}$ for several iterations. Since the score function $s_\theta(\mathbf{x}_{t_i}, t_i)$ defines the distribution p_{t_i}, running Langevin dynamics moves the predicted sample towards the high-density regions of the marginal distribution p_{t_i} without changing the time t_i.

4.3 The Deterministic Perspective: Probability Flow ODEs

An alternative perspective on generative modeling frames the generation task as a probability flow defined by ODEs. Unlike the stochastic perspective, this approach provides a deterministic formulation of probability transport, which enables a bidirectional mapping between the data distribution and the prior distribution through continuous-time ODE trajectories. In this subsection, we first demonstrate the theoretical connection between diffusion SDEs and these deterministic flows. We then discuss methods for learning these deterministic flows directly, such as **flow matching**.

4.3.1 From SDEs to ODEs

An interesting theoretical result in score-based generative modeling is that the stochastic process described by the SDE has an equivalent deterministic representation. Song et al. [2021] demonstrated that for any diffusion SDE, there exists an associated deterministic ODE whose trajectories induce the exact same marginal probability densities $\{p_t(\mathbf{x}(t))\}_{t=0}^T$ as the SDE.

This ODE is referred to as the probability flow ODE, given by

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t)). \tag{20}$$

The derivation relies on the Fokker-Planck equation (also known as the Kolmogorov forward equation), which describes the time evolution of the probability density function $p_t(\mathbf{x})$ under the dynamics of an SDE. It can be shown that the continuity equation induced by the ODE in Eq. (20) is identical to the Fokker-Planck equation of the SDE in Eq. (4) (see Appendix C for a more detailed derivation). As a result, if we sample $\mathbf{x}(T) \sim p_T$ and solve this ODE backwards in time (from T to 0), the resulting sample $\mathbf{x}(0)$ will be distributed according to the data distribution p_0, identical to the result of the reverse SDE.

4. For instance, applying the Euler-Maruyama discretization to the reverse VP SDE (from t_{i+1} to t_i) yields $\mathbf{x}_{t_i} \approx (1 + \frac{1}{2}\beta_{i+1})\mathbf{x}_{t_{i+1}} + \beta_{i+1} s_\theta(\mathbf{x}_{t_{i+1}}, t_{i+1}) + \sqrt{\beta_{i+1}}\boldsymbol{\epsilon}_{i+1}$. By substituting the score parameterization $s_\theta(\mathbf{x}, t) \approx -\boldsymbol{\epsilon}_\theta(\mathbf{x}, t)/\sqrt{1-\bar{\alpha}_t}$ (where indices correspond to t_{i+1}) and recognizing that $1 + \frac{1}{2}\beta_{i+1}$ corresponds to the first-order Taylor expansion of the scaling term $(1-\beta_{i+1})^{-1/2}$ used in DDPM, the derived update rule becomes identical to the standard DDPM sampling algorithm.

This equivalence provides an important link between the stochastic and deterministic perspectives. Note that Eq. (20) depends on the score function $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$. This means that once we have trained a score model $s_\theta(\mathbf{x}(t), t)$, we can immediately use it to construct the probability flow ODE

$$\frac{d\mathbf{x}(t)}{dt} \approx f(\mathbf{x}(t), t) - \frac{1}{2} g(t)^2 s_\theta(\mathbf{x}(t), t). \tag{21}$$

In practice, using the ODE approach for sampling offers several advantages over the SDE approach. First, given a fixed initial noise vector $\mathbf{x}(T)$, the generation process is deterministic. Thus, we have a unique flow map, and can invert a real image into its corresponding latent noise representation by integrating the ODE forward in time. Second, ODE trajectories are generally smoother and easier to solve than SDEs. We can employ advanced black-box ODE solvers (such as Runge-Kutta methods) with adaptive step sizes to generate high-quality samples in fewer steps compared to the Euler-Maruyama method required for SDEs. Third, using the instantaneous change of variables formula, one can compute the exact log-likelihood of data under the probability flow ODE, which facilitates quantitative evaluation of model performance.

4.3.2 Flow Matching

While the probability flow ODE derived from SDEs provides a deterministic process for generation, it is essentially a by-product of the stochastic diffusion process. The vector field $f(\mathbf{x}(t), t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$ in Eq. (20) is determined by the specific choice of the forward diffusion SDE (e.g., variance exploding or variance preserving). This raises a question: can we learn a vector field for the probability flow ODE directly, without relying on the intermediate diffusion formulation? Moreover, can we design this vector field to have desirable properties, such as straighter trajectories, which would be easier to integrate numerically?

This motivates the framework of flow matching [Lipman et al., 2023; Liu et al., 2023; Albergo and Vanden-Eijnden, 2023]. Flow matching is a simulation-free approach to training continuous normalizing flows that allows for flexible definitions of the probability path.

The core idea of flow matching is to directly regress a neural network vector field $v(\mathbf{x}(t), \theta, t)$ (or $v_\theta(\mathbf{x}(t), t)$) against a target vector field $u_t(\mathbf{x}(t))$ that generates a desired probability path $p_t(\mathbf{x}(t))$. Here, we compare the flow matching-based approach with the score-based approach, as follows:

- **Score-based Approach**. The forward process is a fixed Gaussian diffusion. As shown in the previous subsections, the corresponding probability flow ODE has a vector field composed of a linear drift and a score term. The resulting trajectories are often fluctuating because the diffusion process destroys information according to a rigid schedule (e.g., exponentially decaying signal). As a result, solving the reverse ODE requires many steps or complex solvers to trace these fluctuating paths accurately.

- **Flow Matching Approach**. Instead of having the dynamics determined by an SDE, flow matching allows us to design the path. We explicitly define how a sample from the noise distribution maps to a sample from the data distribution. For instance, the simplest

transport map between two points is a straight line. By enforcing straight trajectories, the vector field becomes constant in time along the path. During inference, such an ODE is easier to solve.

Figure 4 illustrates this difference. While diffusion models find a curved path between noise and data, flow matching aims to find a more efficient path that can reach the target data more directly.

To align with the notation used in this paper, we adapt the standard flow matching formulation (typically defined from $t = 0$ as noise to $t = 1$ as data, as in [Liu et al., 2023]) to our time setting where $t = 0$ corresponds to the data distribution p_0 and $t = T$ corresponds to the noise prior p_T.

In the SDE framework, the probability path $p_t(\mathbf{x}(t))$ is implicitly defined by the diffusion process. We do not know the closed-form expression of an intermediate sample $\mathbf{x}(t)$ without simulating the SDE or computing complex marginals. Flow matching flips this logic: we construct the path explicitly. The key insight of flow matching is to work with conditional probability paths. Instead of modeling the intractable marginal vector field $u_t(\mathbf{x}(t))$ directly, we condition the path on a specific data sample $\mathbf{x}(0) \sim p_0$ and a specific noise sample $\mathbf{x}(T) \sim p_T$. We define a conditional flow $\psi_t(\mathbf{x}(t)|\mathbf{x}(0), \mathbf{x}(T))$ that creates a bridge between them. A simple and effective choice is the linear interpolation

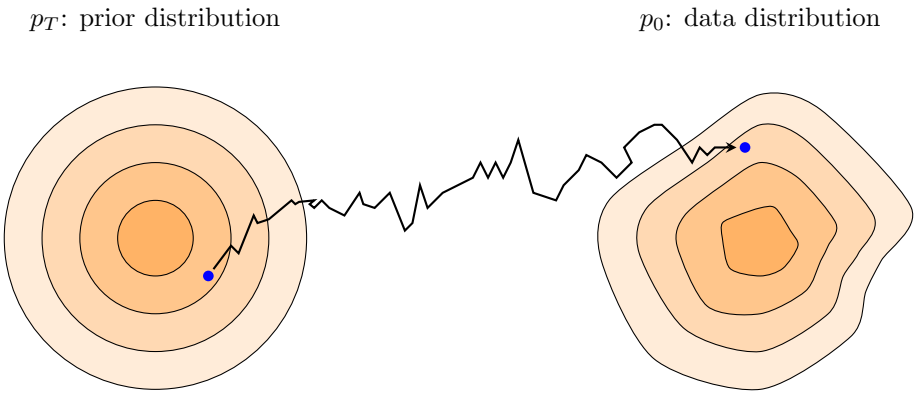
$$\begin{aligned} \mathbf{x}(t) &= \psi_t(\mathbf{x}(t)|\mathbf{x}(0), \mathbf{x}(T)) \\ &= \left(1 - \frac{t}{T}\right)\mathbf{x}(0) + \frac{t}{T}\mathbf{x}(T), \end{aligned} \tag{22}$$

This equation defines a forward process purely through geometry rather than stochastic diffusion steps. We can interpret this as drawing a straight line connecting a data point and a noise point in the high-dimensional space. The velocity of this path is simply the time derivative of $\mathbf{x}(t)$, given by

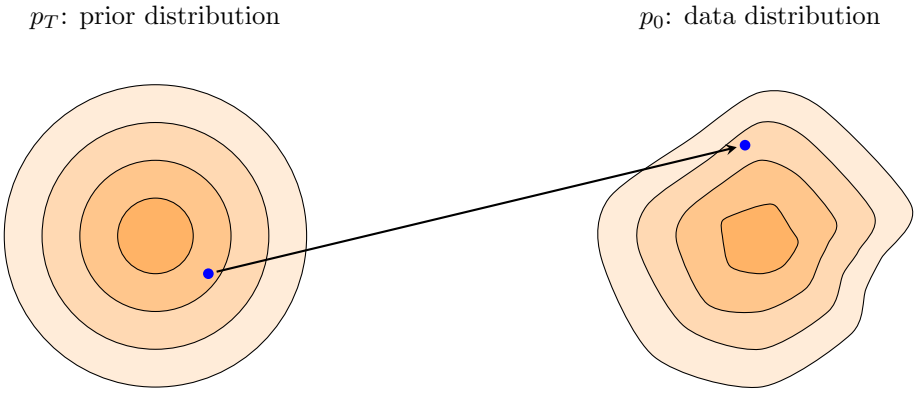
$$\begin{aligned} u_t(\mathbf{x}(t)|\mathbf{x}(0), \mathbf{x}(T)) &= \frac{d\mathbf{x}(t)}{dt} \\ &= \frac{1}{T}(\mathbf{x}(T) - \mathbf{x}(0)). \end{aligned} \tag{23}$$

Note that this target vector field is constant with respect to time for a specific pair $(\mathbf{x}(0), \mathbf{x}(T))$, which implies a straight trajectory. This path design is often referred to as the **optimal transport** (OT) conditional vector field, as it represents the most efficient (shortest) path to transport mass from one distribution to another under squared Euclidean cost.

From a formal perspective, this modeling approach offers a significant advantage: it eliminates the dependency on a stochastic diffusion process to define the forward dynamics. Instead, the forward trajectory from $t = 0$ to T is obtained directly via the analytic interpolation in Eq. (22). This allows us to efficiently generate intermediate states $\mathbf{x}(t)$ at any arbitrary time t without the computational overhead of iterative simulation. These interpolated samples, along with their corresponding target velocities, serve directly as the ground truth for training the "backward"



(a) Trajectory of the score-based approach



(b) Trajectory of the flow matching approach (straight path transport)

Figure 4: Comparison of generative trajectories from the prior noise (p_T) to the data distribution (p_0). In subfigure (a), the score-based approach (via SDE) generates a stochastic trajectory. The process involves random noise injection, resulting in a jagged path that typically requires many discretization steps. In subfigure (b), the flow matching approach constructs a vector field that transports samples along a straight path. Here we show the ideal case where generation is performed in a single step. Although more generation steps are generally required in practice, this linearity makes the generation significantly more efficient compared to SDE paths.

model. Specifically, the neural network vector field $v_\theta(\mathbf{x}(t), t)$ is optimized to regress against this target velocity u_t, thereby learning to reverse the flow for generation.

Given the above formulation, we can learn the global vector field $v_\theta(\mathbf{x}, t)$ by regressing it against the conditional vector fields. Lipman et al. [2023] proved that minimizing the difference between the model output and the conditional vector field is equivalent to minimizing the difference against the true marginal vector field.

The training objective, known as **conditional flow matching** (CFM), is defined as

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,\mathbf{x}(0),\mathbf{x}(T)} \left[\left\| v_\theta(\mathbf{x}(t), t) - u_t(\mathbf{x}(t)|\mathbf{x}(0), \mathbf{x}(T)) \right\|^2 \right]. \tag{24}$$

In practice, the training procedure is quite simple:

Training a Flow Matching Model:

1. Sample a data point $\mathbf{x}(0) \sim p_{\text{data}}$ and a noise point $\mathbf{x}(T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
2. Sample a time $t \sim \mathcal{U}[0, T]$.
3. Compute the intermediate state $\mathbf{x}(t)$ via linear interpolation (Eq. (22)).
4. Compute the target velocity $u_t = (\mathbf{x}(T) - \mathbf{x}(0))/T$.
5. Update the network v_θ to minimize $\|v_\theta(\mathbf{x}(t), t) - u_t\|^2$.

Intuitively, during training, the model sees many crossing straight lines connecting data and noise. By considering all these conditional paths, the network learns a coherent global vector field that pushes noise towards data regions (and vice versa) in the most direct way possible. This leads us to an important theoretical insight regarding the nature of the learned vector field. Since the objective function employs the mean squared error, the optimized model v_θ does not memorize individual trajectories. Instead, it converges to the conditional expectation of the target vector fields given the current state. Mathematically, the optimal vector field $v^*(\mathbf{x}, t)$, referred to as the **marginal vector field**, satisfies

$$v^*(\mathbf{x}, t) = \mathbb{E}_{p(\mathbf{x}(0),\mathbf{x}(T)|\mathbf{x}(t))} \left[u_t(\mathbf{x}(t)|\mathbf{x}(0), \mathbf{x}(T)) \right]. \tag{25}$$

It implies that at any specific location $\mathbf{x}(t)$ and time t, the learned vector is the **statistical average** of the instantaneous velocities of all possible straight-line trajectories passing through this point, as illustrated in Figure 5. As a result, the learned global vector field generates the marginal probability path $p_t(\mathbf{x})$, which transports the entire noise distribution to the data distribution.

Once the model $v_\theta(\mathbf{x}, t)$ is trained to approximate the vector field pointing from $t = 0$ (data) to $t = T$ (noise), generation is performed by solving the ODE backwards in time. We start with pure noise $\mathbf{x}(T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and integrate to $t = 0$. Since we integrate backwards, the numerical update uses the negative direction of the field: $\mathbf{x}(t - \Delta t) \approx \mathbf{x}(t) - v_\theta(\mathbf{x}(t), t)\Delta t$.

The primary advantage of this approach over standard diffusion models is the trajectory shape. Flow matching with the optimal transport path encourages the learned global vector field to be as straight as possible. The resulting straight trajectories are much easier for ODE solvers to track, and we can adopt larger step sizes Δt. As a result, flow matching models can often generate high-quality samples in fewer steps compared to SDE-based diffusion models.

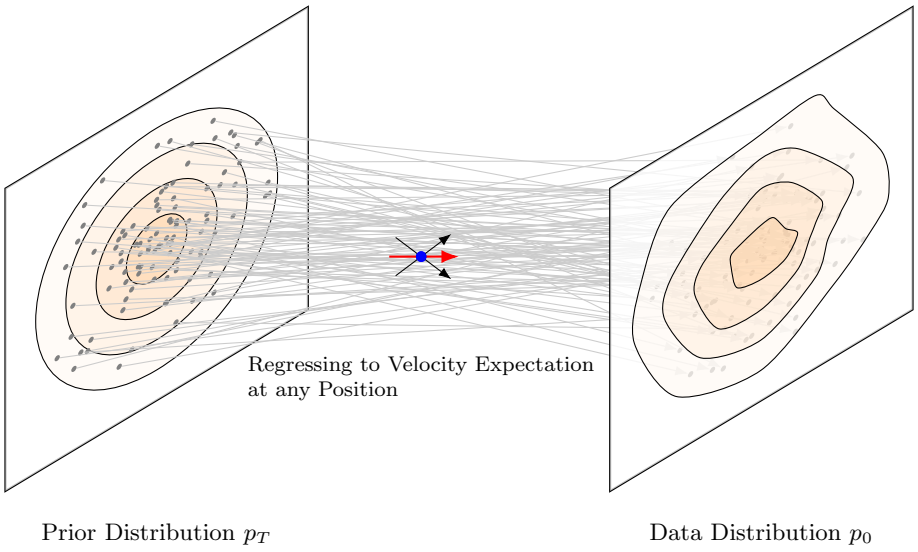


Figure 5: Illustration of the marginal vector field learning in flow matching. The gray lines represent linear interpolation trajectories connecting noise samples $\mathbf{x}(T)$ to data samples $\mathbf{x}(0)$. During training, at a specific location $\mathbf{x}(t)$ (blue dot), multiple trajectories may pass through the same point. Since the training objective minimizes the mean squared error, the learned vector field $v_\theta(\mathbf{x}, t)$ does not memorize individual paths but instead converges to the conditional expectation of the target velocities (red arrow).

4.4 Towards More Efficient Generation

A bottleneck in deploying diffusion models for real-world applications is their high computational cost during inference. Efficient diffusion modeling is itself a broad topic [Shen et al., 2025]. Here we briefly outline recent methods for efficient generation, including high-order numerical solvers, trajectory rectification, and consistency models.

4.4.1 High-Order and Specialized Solvers

One of the most straightforward approaches to acceleration is to use high-order numerical solvers. However, standard solvers like the first-order Euler method (e.g., DDIM sampling in the diffusion context) require many steps to minimize discretization errors when trajectories are curved. To address this, high-order solvers have been used to exploit the specific mathematical structure of diffusion ODEs.

For example, DPM-Solver and DPM-Solver++ make use of the semi-linear structure of diffusion ODEs to analytically compute the linear part of the solution, thus simplifying the non-linear component into an exponentially weighted integral [Lu et al., 2022]. The core principle lies in the variation of constants formula, which separates the ODE into a linear drift term and a non-linear score-based term. By analytically solving the linear part, the solver only needs to approximate the non-linear score function using $(k-1)$-th order polynomials. DPM-Solver++ further improves stability by using a data-prediction parameterization and dynamic thresholding to prevent the

solution from drifting outside the valid data range [Lu et al., 2025]. Most recently, Zheng et al. [2023] introduced DPM-Solver-v3, featuring a predictor-corrector framework that refines the integral approximation using an inner predictor step. By combining this with coefficients optimized via empirical model statistics, it achieves superior performance in few-step regimes (e.g., < 10 steps) without additional function evaluations.

Beyond general numerical methods, several distinct approaches have been developed from geometric and mathematical perspectives. For instance, the approximate mean-direction solver (AMED-Solver) exploits the observation that sampling trajectories often reside in low-dimensional subspaces [Zhou et al., 2024]. By learning to predict the mean direction of the flow, it achieves convergence in fewer steps. Similarly, based on exponential integrators, the diffusion exponential integrator sampler (DEIS) reduces discretization error by fitting polynomials in a transformed log-rho space [Zhang and Chen, 2022]. This transformation linearizes the curvature of the sampling trajectory more effectively than standard time-steps. Alternatively, the unified predictor-corrector method (UniPC) introduces a free corrector step that reuses the noise prediction already evaluated during the predictor step [Zhao et al., 2023]. By reusing model outputs, this method increases the order of accuracy without requiring additional neural function evaluations. Pseudo numerical methods for diffusion models (PNDM) treat the denoising process as solving a differential equation on a manifold, which represents the high-density region of the data. They decompose the numerical step into a gradient part and a nonlinear transfer part. The transfer part ensures that even if the trajectory is curved, the resulting state x_t remains constrained to the data manifold [Liu et al., 2022]. In addition, the elucidating diffusion model (EDM) uses a 2nd-order Heun solver [Karras et al., 2022]. By scaling network inputs, outputs, and losses to unit variance, it stabilizes training dynamics and improves score estimation.

4.4.2 Trajectory Rectification

As presented previously, flow matching aims to learn a vector field $v_\theta(\mathbf{x}(t), t)$ that induces a probability flow capable of transforming a simple prior distribution p_T into a complex data distribution p_0. The core objective is to minimize the difference between the learned velocity field and the ground-truth conditional velocities that transport samples along a chosen path (Eq. (24)). However, a problem arises from the coupling between the data distribution p_0 and the noise distribution p_T.

In the initial training phase, frequently referred to as 1-rectified flow, the data points $\mathbf{x}(0)$ and noise points $\mathbf{x}(T)$ are paired randomly from their respective distributions. This random pairing leads to a phenomenon where trajectories in the high-dimensional latent space frequently intersect. When multiple straight trajectories cross at the same point $\mathbf{x}(t)$ in spacetime, the learned marginal vector field $v_\theta(\mathbf{x}(t), t)$ (i.e., the expectation of these various intersecting velocities) becomes curved. Consequently, the actual paths followed during the integration of the ODE are not straight, which leads to more discretization errors when adopting large step sizes in numerical solvers [Zhang et al., 2025; Yang et al., 2025]. In the rectified flow framework, one can improve the coupling between the noise and data distributions by reducing the interference between conditional paths. First, a 1-rectified flow v_θ^1 is trained using random pairings $(\mathbf{x}(0), \mathbf{x}(T))$. Then, the trained model v_θ^1 is used to generate a new synthetic dataset. Specifically, for a set of noise samples $\{\mathbf{x}_i(T)\}$, the

ODE $\frac{d\mathbf{x}(t)}{dt} = -v_\theta^1(\mathbf{x}, t)$ is solved to obtain the corresponding generated data samples $\{\hat{\mathbf{x}}_i(0)\}$. This creates a set of deterministic, rewired pairs $(\hat{\mathbf{x}}_i(0), \mathbf{x}_i(T))$. Finally, a new model v_θ^2 (called 2-rectified flow) is trained using these aligned pairs. Because $\hat{\mathbf{x}}_i(0)$ was generated from $\mathbf{x}_i(T)$ via the dynamics of the first model, these pairs are highly aligned in the latent space. Thus the resulting vector field is significantly more linear.

The search for straight trajectories is rooted in the principles of optimal transport and the minimization of the Wasserstein distance. An important result in this domain is that for a rectified flow from a Gaussian to a mixture of two Gaussians, a few rectifications are sufficient to achieve straight flows. This insight underpins the empirical success of models like InstaFlow and Stable Diffusion, which make use of the straightening properties of rectified flow to reduce the number of function evaluations while maintaining sample quality [Esser et al., 2024]. The success of these commercialized models also demonstrates that diffusion models with rectified flows can be well scaled up for high-quality image generation.

The straightness of the trajectory is also linked to the Lipschitz constant of the learned vector field. If the 1-rectified flow is L-Lipschitz, nearby noise samples are prevented from mapping to arbitrarily distant data points. However, in practice, L can be large, leading to significant bending of the transport paths in the early stages of generation. This bending degrades the accuracy of one-step Euler sampling, which may require further rectification or more advanced sampling schedules.

The practical implementation of trajectory rectification also involves architectural improvements. Researchers have replaced simple U-Net architectures with sophisticated multimodal diffusion Transformers (MMDiT). For example, Stable Diffusion 3 introduced the MMDiT architecture [Esser et al., 2024], while Flux.1 further scaled the rectified flow transformer architecture to 12 billion parameters [Greenberg, 2025]. This shows that diffusion models can be scaled to very large sizes while maintaining high sampling efficiency.

To further enhance efficiency and stability, it is also possible to apply the divide-and-conquer strategy to rectified flows. For example, the piecewise rectified flow (PeRFlow) partitions the generative trajectory into several discrete time windows [Yan et al., 2024]. Instead of straightening the global flow, it straightens segments within each interval individually, and this localization reduces the computational cost of simulating training trajectories and minimizes numerical errors. Furthermore, researchers have addressed the stability of conditional generation through advanced guidance techniques. For example, Rectified-CFG++ employs an adaptive predictor-corrector scheme [Saini et al., 2025]. Each generation step first executes a conditional update to anchor the sample near the learned transport path, then applies a manifold-aware correction based on the difference between conditional and unconditional velocities. This ensures that sampling trajectories remain within a stable tubular neighborhood of the data manifold.

4.4.3 Consistency Models

Consistency modeling represents a shift away from iterative integration toward a direct mapping paradigm. Instead of traversing a path, a consistency model is designed to learn a function that maps any point at any time step on a probability flow ODE trajectory directly to its origin [Song et al., 2023]. This self-consistency property ensures that all points residing on the same path result

in the same output image, and so we can generate high-quality results using one-step/few-step generation.

To achieve this, consistency models are generally trained via two mechanisms: consistency training and consistency distillation. While consistency training allows training from scratch, consistency distillation has proven more practical for large-scale applications by distilling the knowledge of a pre-trained standard diffusion model (the teacher) into a consistency model (the student). The distillation loss minimizes the discrepancy between the prediction of the student at the current time step t and its prediction at the next step. This compresses the multi-step trajectory of the teacher into a single jump. To further improve generation quality, especially for one-step sampling, adversarial loss functions have been introduced to ensure the predicted samples remain indistinguishable from real data distributions [Sauer et al., 2024].

An extension of this paradigm is the latent consistency model (LCM), which applies consistency distillation to the latent space of high-resolution image synthesis models like Stable Diffusion [Luo et al., 2023a]. By distilling the probability flow of the latent ODE, LCMs drastically reduce inference latency while retaining the high-resolution capabilities of the parent model. Furthermore, to address the computational overhead of retraining large backbones, LCM-LoRA introduces a parameter-efficient fine-tuning strategy. This approach trains a lightweight low-rank adaptation (LoRA) module that acts as a universal accelerator. Thus, various fine-tuned diffusion models can be used to acquire consistency properties without altering their pre-trained parameters [Luo et al., 2023b].

While early LCMs achieved significant speedups, they often suffered from degraded quality as the number of sampling steps increased. To address this, trajectory consistency distillation (TCD) reformulates the consistency function in a semi-linear framework inspired by exponential integrators to allow mapping to arbitrary intermediate steps [Zheng et al., 2024]. Moreover, phased consistency models (PCM) partition the trajectory into multiple sub-trajectories to enable deterministic multi-step sampling without stochasticity error accumulation [Wang et al., 2024]. Recent continuous-time models like simplified consistency models (sCM) also introduce the TrigFlow formulation to stabilize training [Lu and Song, 2025].

Note that, although consistency models are optimized for single-step generation, they also support multistep consistency sampling. By taking 2 to 4 steps and re-encoding the intermediate outputs, the model can iteratively correct discretization errors. This offers a flexible trade-off between inference speed and sample fidelity.

Appendix C. Deriving Probability Flow ODEs from SDEs

Here, we provide the detailed derivation of the probability flow ODE introduced in Section 4.3. We demonstrate that for a given diffusion SDE, there exists a deterministic ODE whose trajectories induce exactly the same marginal probability densities $\{p_t\}_{t=0}^{T}$.

Consider the forward SDE defined as

$$d\mathbf{x}(t) \quad = \quad f(\mathbf{x}(t), t)dt + g(t)d\mathbf{w}(t), \tag{26}$$

where $f(\mathbf{x}(t), t)$ is the drift coefficient, $g(t)$ is the diffusion coefficient, and $\mathbf{w}(t)$ is the standard Wiener process. In the following we omit the argument t in $\mathbf{x}(t)$ to keep the notation uncluttered.

The time evolution of the probability density function $p_t(\mathbf{x})$ of the state variable $\mathbf{x}$ is governed by the Fokker-Planck equation (also known as the Kolmogorov forward equation):

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = -\nabla \cdot [f(\mathbf{x}, t)p_t(\mathbf{x})] + \frac{1}{2}g(t)^2 \Delta p_t(\mathbf{x}), \tag{27}$$

where $\nabla\cdot$ denotes the divergence operator and $\Delta = \nabla \cdot \nabla$ is the Laplace operator. The first term on the right-hand side represents the advection of probability mass due to the drift, while the second term represents the diffusion caused by the stochastic noise.

To derive the equivalent ODE, we rewrite Eq. (27) in the form of a continuity equation, which describes the density evolution of a deterministic system (also see Appendix B). A continuity equation involves only first-order derivatives and takes the form

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = -\nabla \cdot [\tilde{f}(\mathbf{x}, t)p_t(\mathbf{x})], \tag{28}$$

where $\tilde{f}(\mathbf{x}, t)$ corresponds to the vector field of the deterministic ODE.

The key step is to express the diffusion term $\Delta p_t(\mathbf{x})$ (which involves second-order derivatives) as the divergence of a first-order term. We use the identity $\nabla p_t(\mathbf{x}) = p_t(\mathbf{x})\nabla \log p_t(\mathbf{x})$ to rewrite the Laplacian as

$$\begin{aligned} \Delta p_t(\mathbf{x}) &= \nabla \cdot (\nabla p_t(\mathbf{x})) \\ &= \nabla \cdot (p_t(\mathbf{x})\nabla \log p_t(\mathbf{x})). \end{aligned} \tag{29}$$

Substituting this identity back into the Fokker-Planck Eq. (27), we can group the terms under a single divergence operator, as follows

$$\begin{aligned} \frac{\partial p_t(\mathbf{x})}{\partial t} &= -\nabla \cdot [f(\mathbf{x}, t)p_t(\mathbf{x})] + \frac{1}{2}g(t)^2 \nabla \cdot [p_t(\mathbf{x})\nabla \log p_t(\mathbf{x})] \\ &= -\nabla \cdot \left[f(\mathbf{x}, t)p_t(\mathbf{x}) - \frac{1}{2}g(t)^2 p_t(\mathbf{x})\nabla \log p_t(\mathbf{x}) \right] \\ &= -\nabla \cdot \left[\left(f(\mathbf{x}, t) - \frac{1}{2}g(t)^2 \nabla \log p_t(\mathbf{x}) \right) p_t(\mathbf{x}) \right]. \end{aligned} \tag{30}$$

Comparing this result with the continuity equation in Eq. (28), we find that the equations match if we define the effective deterministic vector field $\tilde{f}(\mathbf{x}, t)$ as:

$$\tilde{f}(\mathbf{x}, t) = f(\mathbf{x}, t) - \frac{1}{2}g(t)^2 \nabla \log p_t(\mathbf{x}). \tag{31}$$

Therefore, the deterministic ODE yields identical same marginal distributions $p_t(\mathbf{x})$ as the stochastic process defined by the original SDE. This ODE is given by

$$\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, t) - \frac{1}{2}g(t)^2 \nabla \log p_t(\mathbf{x}). \tag{32}$$

References

Michael Samuel Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In The Eleventh International Conference on Learning Representations, 2023.

Brian D.O. Anderson. Reverse-time diffusion equation models. Stochastic Processes and their Applications, 12(3):313–326, 1982.

Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. Machine learning for data science handbook: data mining and knowledge discovery handbook, pages 353–374, 2023.

Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. IEEE transactions on pattern analysis and machine intelligence, 45(9): 10850–10869, 2023.

Sander Dieleman. Perspectives on diffusion, 2023. URL https://sander.ai/2023/07/20/perspectives.html.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In Forty-first international conference on machine learning, 2024.

Or Greenberg. Demystifying flux architecture. arXiv preprint arXiv:2507.09595, 2025.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. Advances in neural information processing systems, 33:6840–6851, 2020.

Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. Journal of Machine Learning Research, 6(4), 2005.

Ioannis Karatzas and Steven Shreve. Brownian motion and stochastic calculus. springer, 2014.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. Advances in neural information processing systems, 35:26565–26577, 2022.

Diederik P Kingma and Ruiqi Gao. Understanding diffusion objectives as the ELBO with simple data augmentation. In Thirty-seventh Conference on Neural Information Processing Systems, 2023.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.

Diederik P Kingma and Max Welling. An introduction to variational autoencoders. Foundations and Trends® in Machine Learning, 12(4):307–392, 2019.

Peter E Kloeden and RA Pearson. The numerical solution of stochastic differential equations. The ANZIAM Journal, 20(1):8–12, 1977.

Peter E. Kloeden and Eckhard Platen. Numerical Solution of Stochastic Differential Equations. Springer Berlin, 1992.

Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, Fujie Huang, et al. A tutorial on energy-based learning. Predicting structured data, 1, 2006.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In The Eleventh International Conference on Learning Representations, 2023.

Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. In International Conference on Learning Representations, 2022.

Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In The Eleventh International Conference on Learning Representations, 2023.

Cheng Lu and Yang Song. Simplifying, stabilizing and scaling continuous-time consistency models. In The Thirteenth International Conference on Learning Representations, 2025.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. Advances in neural information processing systems, 35:5775–5787, 2022.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. Machine Intelligence Research, pages 1–22, 2025.

Calvin Luo. Understanding diffusion models: A unified perspective. arXiv, 2022.

Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. arXiv preprint arXiv:2310.04378, 2023a.

Simian Luo, Yiqin Tan, Suraj Patil, Daniel Gu, Patrick Von Platen, Apolinário Passos, Longbo Huang, Jian Li, and Hang Zhao. Lcm-lora: A universal stable-diffusion acceleration module. arXiv preprint arXiv:2311.05556, 2023b.

Bernt Oksendal. Stochastic differential equations: an introduction with applications. Springer Science & Business Media, 2013.

Christian P Robert, George Casella, and George Casella. Monte Carlo statistical methods, volume 2. Springer, 1999.

Shreshth Saini, Shashank Gupta, and Alan C Bovik. Rectified-cfg++ for flow based models. arXiv preprint arXiv:2510.07631, 2025.

Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation. In European Conference on Computer Vision, pages 87–103. Springer, 2024.

Hui Shen, Jingxuan Zhang, Boning Xiong, Rui Hu, Shoufa Chen, Zhongwei Wan, Xin Wang, Yu Zhang, Zixuan Gong, Guangyin Bao, Chaofan Tao, Yongfeng Huang, Ye Yuan, and Mi Zhang. Efficient diffusion models: A survey. arXiv preprint arXiv:2502.06805, 2025.

Yang Song. Generative modeling by estimating gradients of the data distribution, 2021. URL https://yang-song.net/blog/2021/score/.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. Advances in neural information processing systems, 32, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In International Conference on Learning Representations, 2021.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In International Conference on Machine Learning, pages 32211–32252. PMLR, 2023.

Pascal Vincent. A connection between score matching and denoising autoencoders. Neural computation, 23(7):1661–1674, 2011.

Fu-Yun Wang, Zhaoyang Huang, Alexander Bergman, Dazhong Shen, Peng Gao, Michael Lingelbach, Keqiang Sun, Weikang Bian, Guanglu Song, Yu Liu, et al. Phased consistency models. Advances in neural information processing systems, 37:83951–84009, 2024.

Hanshu Yan, Xingchao Liu, Jiachun Pan, Jun Hao Liew, qiang liu, and Jiashi Feng. PeRFlow: Piecewise rectified flow as universal plug-and-play accelerator. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024.

Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. ACM computing surveys, 56(4):1–39, 2023.

Xiaofeng Yang, Chen Cheng, Xulei Yang, Fayao Liu, and Guosheng Lin. Text-to-image rectified flow as plug-and-play priors. In The Thirteenth International Conference on Learning Representations, 2025.

Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. In NeurIPS 2022 Workshop on Score-Based Methods, 2022.

Xinxi Zhang, Shiwei Tan, Quang Nguyen, Quan Dao, Ligong Han, Xiaoxiao He, Tunyu Zhang, Alen Mrdovic, and Dimitris Metaxas. Flow straighter and faster: Efficient one-step generative modeling via meanflow on rectified trajectories. arXiv preprint arXiv:2511.23342, 2025.

Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. Advances in Neural Information Processing Systems, 36:49842–49869, 2023.

Jianbin Zheng, Minghui Hu, Zhongyi Fan, Chaoyue Wang, Changxing Ding, Dacheng Tao, and Tat-Jen Cham. Trajectory consistency distillation. CoRR, 2024.

Kaiwen Zheng, Cheng Lu, Jianfei Chen, and Jun Zhu. DPM-solver-v3: Improved diffusion ODE solver with empirical model statistics. In Thirty-seventh Conference on Neural Information Processing Systems, 2023. URL https://openreview.net/forum?id=9fWKExmKa0.

Zhenyu Zhou, Defang Chen, Can Wang, and Chun Chen. Fast ode-based sampling for diffusion models in around 5 steps. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 7777–7786, 2024.