LEARNING, SAMPLING AND INFERENCE WITH STOCHASTIC DIFFERENTIAL EQUATIONS

A Dissertation
Presented to
The Academic Faculty

By

Qinsheng Zhang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Robotics

Georgia Institute of Technology

December 2023

LEARNING, SAMPLING AND INFERENCE WITH STOCHASTIC DIFFERENTIAL EQUATIONS

Thesis committee:

Professor Yongxin Chen, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Professor Danfei Xu
School of Interactive Computing
Georgia Institute of Technology

Professor Humphrey Shi
School of Interactive Computing
Georgia Institute of Technology

Professor Molei Tao
Department of Mathematics
Georgia Institute of Technology

Professor Zsolt Kira
School of Interactive Computing
Georgia Institute of Technology

Date approved: November 30 2023

ACKNOWLEDGMENTS

During my Ph.D. years, I have been fortunate to have many people who have helped me in my research. I wish to express my sincere appreciation and gratitude to the many individuals who have contributed to my Ph.D. studies and the completion of this dissertation. Without them, I would not be where I am today.

Firstly, I would like to express my heartfelt gratitude to my esteemed supervisor, Professor Yongxin Chen. His outstanding guidance, supervision, and mentorship have been invaluable throughout this challenging journey. Professor Chen's dedication extended far beyond the usual responsibilities of a supervisor, providing steadfast support, insightful feedback, constructive criticism, and constant encouragement. These contributions were crucial in helping me navigate and overcome the numerous challenges and obstacles encountered along the way. The weekly discussions with Professor Yongxin Chen, whether held in front of his expansive whiteboard or over interactive Notebooks on iPads, consistently stand out as the happiest times in my academic journey. I would also like to express my sincere appreciation to the members of my thesis committee, Professor Zsolt Kira, Humphrey Shi, Danfei Xu and Molei Tao, for their valuable insights, feedback, and suggestions that have enhanced the quality of my work.

I am immensely grateful to my labmates for their unwavering support and encouragement throughout my PhD journey. Dr. Rahul Singh has been a guiding force and a big brother figure, generously providing me with help and advice whenever I needed it. Hiking and traveling with Hongzhe Yu, and Zishun Liu have been memorable experiences that I cherish. The insightful discussions with Haotian Xue and Utkarsh A. Mishra have broadened my research horizon and provided me with new perspectives. I would also like to express my heartfelt gratitude to every member of FLAIR Lab. Your support, guidance, and encouragement have been instrumental in shaping my research methodology and approach.

I am profoundly grateful for the support and guidance received from my colleagues at the Nvidia Deep Imagination Research group. Special thanks are due to my manager, Dr. Ming-yu Liu, whose unparalleled resources and insightful wisdom have been pivotal in my research journey. I owe a debt of gratitude to my mentors, Dr. Jiaming Song and Dr. Xun Huang, for their instrumental roles in shaping my development as a researcher. I also extend my heartfelt appreciation to other team members, with whom I have had the pleasure of building remarkable generative models, sharing enriching discussions, and enjoying daily camaraderie and gatherings. My sincere thanks to Dr. Zekun Hao, Dr. Max Li, Xiaohui Zeng, Dr. Tsung-Yi Lin, Dr. Seungjun Nah, Dr. Yogesh Balaji, and Dr. Ting-chun Wang and other colleagues for their invaluable contributions to my experience and growth.

Finally, I would like to thank my family for their unconditional love and unwavering support. My parents' sacrifices and my sister's support have been crucial in enabling me to pursue my academic goals. To my partner Jiaojiao, thank you for being by my side and sharing this journey with me. Your love, companionship, and encouragement have made this journey more exciting and colorful, and I am looking forward to sharing many more adventures with you in the future.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Summary

Stochastic differential equations (SDEs) constitute a formidable tool for modeling the dynamics of continuous-time stochastic processes and offer a natural framework for the probabilistic modeling of high-dimensional data. Consequently, they have garnered increasing attention in generative machine learning. Despite their promise, the applications of SDEs in machine learning have been limited due to the lack of scalable learning approaches that can train flexible neural networks to approximate stochastic processes, and the difficulty of conducting tractable inference and sampling caused by inefficient SDE solvers.

In my thesis, I outline my efforts to develop novel computational models capable of efficient and scalable learning, sampling, and inference from SDEs. Specifically, I introduce several approaches to learning SDEs for probabilistic modeling, including fitting non-linear forward and backward SDEs with neural networks and learning with limited data. Next, I present a novel deep model designed to learn SDE dynamics while satisfying given constraints on the marginal probability of the SDE. Furthermore, I developed an efficient algorithm for drawing samples from high-dimensional SDEs, which proves effective in generating diverse and high-fidelity data, such as realistic images and videos.

Although recent probability models have demonstrated remarkable achievements, the majority of these models have primarily focused on generative tasks. Few studies have explored or verified that knowledge acquired from such models could have the potential to benefit other fields, particularly with respect to various downsampling inference tasks. Additionally, this study outlines several limitations of current generative probability models that are founded on neural SDEs and explore several promising directions to overcome these limitations and further advance the field.

CHAPTER 1
INTRODUCTION

Stochastic differential equations (SDEs), which model the random dynamics subject to drift and diffusion noise, are an ideal framework for describing the evolution of probability distributions in various fields. The use of neural networks to model SDEs is commonly referred to as neural SDEs [1]. Although the initial motivation behind neural SDEs was to provide a flexible tool for time series modeling and Bayesian variational inference [1], practitioners have leveraged neural SDEs to solve a variety of machine learning problems, either explicitly or implicitly [2, 3, 4, 5]. This thesis will mainly focus on the application of SDEs in probability modeling and its applications.

There are four main components of probability modeling: *representation, learning, sampling, and inference*. While SDEs are widely used to connect an interested intractable distribution to a tractable distribution, such as the connection between the distribution of real images and Gaussian distributions, SDEs give a natural framework for probability representation. The learning of SDEs aims to fit an SDE such that certain properties of the joint or marginal distribution are desired. One common desired property is that the end distribution of the SDE is close to the target distribution. To learn SDEs, we may be presented with empirical samples from the target distribution, or we may only access the unnormalized density of the target distribution. The first case is also known as density estimation [6]. The second case is closely related to variational inference (VI) and Monte Carlo (MC) methods. Sampling is the process of generating samples from the interested target distribution, which can be achieved via simulating the SDE. From learned models of the target distribution, we can further perform inference to derive logical conclusions about the target distribution.

We start with generative tasks, which ask for generating non-trivial samples given empirical samples from the target distribution. Arguably, the most popular application of SDEs in machine learning is in diffusion generative models [7], also known as score-based generative models [4]. These models consist of a forward SDE that transforms clean data points into random noise by continuously injecting noise and a backward denoising SDE that iteratively removes noise until recovering clean data points from the noise. In recent years, diffusion models have achieved impressive performances on various tasks, including unconditional image generation [7, 4, 8, 9], text-conditioned image generation [10, 11], text generation [12, 13], 3D point cloud generation [14], and inverse problems [15, 16].

From an algorithmic design perspective, the impressive performance of diffusion models relies on two fundamental components. Firstly, the forward stochastic differential equation (SDE) is typically a linear SDE that allows for a closed-form expression of the transition probability. This property facilitates the fast drawing of noised samples along SDE trajectories for training denoising networks. Nevertheless, the rigid structure of linear SDEs may constrain the expressiveness of the forward noising dynamics when compared to more general SDEs. To overcome this limitation, diffusion models necessitate long-duration noising dynamics, requiring multiple fine-grained denoising steps. This constitutes the second fundamental component for the success of diffusion models. The backward denoising SDE, which is a time-reverse of the forward SDE, iteratively removes noise to generate images, leading to more expressive generative models and reducing the computational burden on the generative network compared to directly generating images from noise. However, the iterative nature of the denoising process is computationally expensive, particularly given that the evaluation of denoising networks employed in diffusion models is usually costly due to their large neural networks, consisting of hundreds of millions or even billions of parameters. For example, the Denoising Diffusion Probabilistic Model (DDPM) [7] requires 1000 steps to generate one sample, with each step requiring evaluation of the neural network once. This is considerably slower than Generative Adversarial Networks (GANs) [17, 18].

In this thesis, we first revisit the choice of linear SDEs and argue that general SDEs can be utilized to learn more efficient data-adaptive noising dynamics and generate samples with comparable quality but greater efficiency. We provide non-trivial training schemes to scale up the neural SDE models for generation tasks. Secondly, we revisit the previous denoising scheme and argue that solving differential equations by treating the entire SDE as a black box is not the most efficient approach for generating samples. Instead, we propose to draw samples by solving a family of marginal-equivalent SDEs/ODEs, demonstrating that less random noise during sampling can significantly accelerate the process. Additionally, we leverage the inherent structure of SDEs/ODEs to design a novel sampling scheme to further expedite the sampling process.

The success of diffusion models [7, 4] also largely be attributed to their scalability. With large-scale datasets and computing resources, practitioners can usually train high-capacity models that are able to produce high-fidelity samples. The recent generative AI revolution led by large-scale text-to-image diffusion models is a great example [11, 19, 20]. Unfortunately, the scalability of diffusion models is not available for every application. There are many applications where a large-scale dataset of the target content does not exist or is prohibitively expensive to collect, but individual pieces of the content are available in great quantities. 360-degree panorama images are such an example. While 360-degree panorama images are considered niche image content and only exist in small quantities, there are a large number of normal perspective images available on the Internet, each of which can be treated as a piece of a 360-degree panorama image. Another example is generating images of extreme aspect ratios. Each of the extreme-aspect-ratio images can be considered as the stitching of multiple images with normal aspect ratios. For such applications, while we cannot afford to collect a large-scale dataset of the target content to train a diffusion model, we wish to synthesize high-quality target content with a diffusion model trained on smaller pieces that are readily available. As an effect to tackle the data-hungry issue, we propose a compositional diffusion model that can be trained on small pieces of the target content and then be used to synthesize high-quality target content.

In the last, I also investigate learning SDE with only access to the unnormalized density of the target distribution. We learn a neural SDE that is able to transform from an original point into a sample from the target distribution by solving the SDE. Different from MCMC methods, our method is customized for sampling from a specific distribution. The task-specific sampling method enjoys better efficiency compared with existing approaches.

1.1 Thesis statement

Stochastic differential equations provide an expressive and efficient representation for probabilistic modeling in high-dimensional data, such as images and video, allowing for the investigation of the learning, sampling, and inference of underlying distributions.

1.2 Thesis outline

This dissertation is comprised of three thematic parts

Part I focuses on efficient and scalable techniques for learning SDEs parameterized by neural networks, with the goal to minimize the one marginal distribution of learned SDE close to the target distribution.

Learning neural SDEs with empirical samples The starting point of my journey starting to derive a new neural SDE that connects data distribution and gaussian distribution. To this end, we turn neural SDE into a generative model by solving neural SDE with initial points starting from white noise. We further such an approach can be scaled to medium size image datasets and achieve better performance than the state-of-the-art generative models based on neural ODE.

The new generative modeling algorithm resembles and unifies both the flow-based models and the diffusion models. It extends the normalizing flow method by gradually adding noise to the sampling trajectories to make them stochastic. It extends the diffusion model by making the forward noising process learnable. When added noise shrinks to zero, the model converges to the normalizing flow model. When the forward process is fixed to some specific type of diffusion, our algorithm reduces to a diffusion model. The work provides a unified framework for diffusion models and continuous normalizing flow. This chapter was previously published as [5].

Learning generative models with pieces data The second problem addressed in my thesis work is how to build generative models that large content generation where target data is expensive or even infeasible to collect. To this end, I propose a generic algorithm that synthesizes large content by merging the results generated by diffusion models trained on small pieces of the large content. Our approach can synthesize large content efficiently by generating pieces in parallel. Besides, the new algorithm can work out of the box when pre-trained diffusion models on different pieces are available.

We conduct extensive experimental results on benchmark datasets to show the effectiveness and versatility of the proposed approaches on various tasks, including generating infinite long images, complex motions, and 360 images. This chapter was previously published as [21].

Learning neural SDEs with unnormalized density We present Path Integral Sampler (PIS), a novel algorithm to draw samples from unnormalized probability density functions. The PIS is built on the Schrödinger bridge problem which aims to recover the most likely evolution of a diffusion process given its initial distribution and terminal distribution. The PIS draws samples from the initial distribution and then propagates the samples through the Schrödinger bridge to reach the terminal distribution. Applying the Girsanov theorem, with a simple prior diffusion, we formulate the PIS as a stochastic optimal control problem whose running cost is the control energy and whose terminal cost is chosen according to the target distribution. By modeling the control as a neural network, we establish a sampling algorithm that can be trained end-to-end. We provide theoretical justifications for the sampling quality of PIS in terms of Wasserstein distance when sub-optimal control is used. Moreover, the path integrals theory is used to compute the importance weights of the samples to compensate for the bias induced by the sub-optimality of the controller and time-discretization. We experimentally demonstrate the advantages of PIS compared with other start-of-the-art sampling methods on a variety of tasks. This chapter was previously published as [22].

Part II investigates the challenges of drawing sampling from learned SDE. We specifically focus on diffusion generative models, where accelerating sampling is crucial due to the expensive cost of running a large generative neural network. We develop principle and efficient algorithms to accelerate sampling for isotropic and non-tropic diffusion models.

Accelerating sampling for isotropic diffusion models In this section, we democratize diffusion models by accelerating their generation. We start with a family of marginal equivalent SDEs/ODEs associated with diffusion models and investigate numerical error sources, which include fitting error and discretization error. We observe that even with the same trained model, different discretization schemes can have dramatically different performances in terms of discretization error. We then carry out a sequence of experiments to systematically investigate the influences of different factors on the discretization error. We find out that the *Exponential Integrator (EI)* [23] that utilizes the semilinear structure of the backward diffusion has a minimum error. To further reduce the discretization error, we propose to either use high-order polynomials to approximate the nonlinear term in the ODE or employ Runge Kutta methods on a transformed ODE. The resulting algorithms termed *Diffusion Exponential Integrator Sampler (DEIS)*, achieve the best sampling quality with limited computational resources. This chapter was previously published as [24].

Accelerating sampling for non-isotropic diffusion models Our goal is to extend the denoising diffusion implicit model (DDIM) to general diffusion models besides isotropic diffusions. Instead of constructing a non-Markov noising process as in the original DDIM, we examine the mechanism of DDIM from a numerical perspective. We discover that the DDIM can be obtained by using some specific approximations of the score when solving the corresponding stochastic differential equation. We present an interpretation of the accelerating effects of DDIM that also explains the advantages of a deterministic sampling scheme over the stochastic one for fast sampling. Building on this insight, we extend DDIM to general DMs, coined generalized DDIM (gDDIM), with a small but delicate modification in parameterizing the scoring network. We validate gDDIM in two non-isotropic DMs: Blurring diffusion model (BDM) [25, 26] and Critically-damped Langevin diffusion model (CLD) [27]. This chapter was previously published as [28].

1.3 Contributions

My dissertation makes the following contributions:

- I show neural SDEs can be medium-scale generative models given empirical samples. The data-driven noising approach of learned SDEs has more flexibility compared with diffusion models. We also demonstrate the method enjoys better representation ability than generative models based on neural ODE.
- I introduce a new method to learn diffusion models for large content generation given only pieces of content. The method can generate large-scale content in parallel.
- I introduce a learning algorithm to learn SDE given only given an unnormalized target density, which customizes the sampling process of neural SDE to generate high-quality samples.
- I design efficient solvers that can accelerate the sampling of various diffusion models, including isotropic diffusion models and non-isotropic diffusion models.

CHAPTER 2
RELATED WORKS

We set out to provide some background knowledge on probability modeling using stochastic differential equations (SDEs). To begin with, we introduce the concept of learning probability models with either empirical samples or density that is known up to a normalizing constant. Subsequently, we provide a succinct overview of SDEs and their relevance in learning neural SDEs. We also introduce the concept of diffusion generative models and their relation to SDEs. Furthermore, we explore the prevalent approaches proposed in the current literature to tackle the challenges of learning in diverse settings.

2.1 Probability modeling

In this thesis, we investigate methods for modeling probability distributions, with a particular focus on high-dimensional spaces. This exploration is primarily driven by two prevalent tasks in machine learning when dealing with high-dimensional data: generation and inference. Generation refers to the process of producing novel samples of interest, while inference involves deriving logical conclusions about the model. Based on the availability of probability density, probability models can be classified into two categories: explicit and implicit models.

Explicit models, such as autoregressive models [29, 30, 31] and normalizing flow [6, 32], enable us to query the density of data samples. In contrast, implicit models, including generative adversarial networks (GANs)[17] and variational autoencoders (VAEs) [33], do not allow us to directly query the exact density of data samples, despite being able to draw samples from them.

To learn probability models, we can either learn the model from data or learn it from a given density function. In this regard, we first formulate the problem of learning from data.

Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \cdots, \boldsymbol{x}_N$ denote a dataset $\mathcal{D}$ of size N. A widely accepted assumption in machine learning is that the data points $\boldsymbol{x}_i$ are independent and identically distributed (i.i.d.) from an underlying probability distribution $p_{\text{data}}(\boldsymbol{x})$.

To learn a probability model, we first construct a parametric model $p_\theta(\boldsymbol{x})$, where $\theta \in \Theta$ denotes the parameters of the model and Θ represents all possible parameter values. Our objective is to learn a probability model $p_{\theta^*}(\boldsymbol{x})$ that approximates $p_{\text{data}}(\boldsymbol{x})$ as closely as possible, given the dataset $\mathcal{D}$. This enables us to generate samples from $p_{\theta^*}(\boldsymbol{x})$ and infer properties of $p_{\text{data}}(\boldsymbol{x})$ via the proxy $p_{\theta^*}(\boldsymbol{x})$. Such probability models are also known as generative models in machine learning literature.

Alternatively, we may encounter a scenario where we are provided with a density function $\hat{p}(\boldsymbol{x}) = Z p_{\text{data}}(\boldsymbol{x})$ known up to a normalizing constant Z. Although it is possible to evaluate the probability value $\hat{p}(\boldsymbol{x}')$ of any arbitrary data point $\boldsymbol{x}'$, generating asymptotically unbiased samples from such an unnormalized distribution can be challenging [34]. We aim to learn a probability model $p_{\theta^*}(\boldsymbol{x})$ that closely approximates the target distribution but is also easy to sample from and infer its properties. In the machine learning literature, the technique of constructing a proxy distribution to draw samples is known as variational inference [35, 36].

There are various representation choices for probability modeling. In this thesis, we focus on techniques based on SDEs. We include existing alternative methods and a more in-depth discussion and comparison in Sec. 2.4.

2.2 Stochastic differential equations

A stochastic differential equation (SDE) with D-dimensional states has the form [37]

$$d\boldsymbol{x} = \boldsymbol{F}(\boldsymbol{x}, t)dt + \boldsymbol{G}(\boldsymbol{x}, t)d\boldsymbol{w}, \tag{2.1}$$

where $\boldsymbol{F}(\boldsymbol{x}, t) : \boldsymbol{R}^D \times \boldsymbol{R} \rightarrow \boldsymbol{R}^D$ denotes the drift term, $\boldsymbol{G}(\boldsymbol{x}, t) : \boldsymbol{R}^D \times \boldsymbol{R} \rightarrow \boldsymbol{R}^D$ denotes the diffusion term, and $\boldsymbol{w}$ is a standard Wiener process. SDEs are commonly used to model systems that evolve with uncertainty and randomness, and to transform one probability distribution into another. In recent years, there has been growing interest among machine learning researchers in incorporating neural networks into SDEs to model complex stochastic processes [2, 38, 39]. In this approach, the drift and diffusion functions of the SDEs are defined by neural networks. However, in most existing works, it is necessary to solve the SDEs during training, which makes the gradient calculation of fitting neural SDEs notoriously difficult and challenging. Memory and computation time of simply differentiating through the operation of SDE solvers [40] scales linearly with the number of time steps of SDE solvers. Another attempt based on forward pathwise methods [41, 42] scales poorly in computational time with the number of parameters and states in the model. Inspired by the success of Neural Ordinary Differential Equations (Neural ODEs) [32], [1] generalized the scalable adjoint sensitivity method to SDEs. The adjoint method allows time-efficient and constant-memory computation of the gradient of the loss function with respect to the parameters of the neural network. To retrace trajectories, authors cached the randomness of SDEs in a Brownian Tree. Taking advantage of splittable pseudorandom number generators [43], the adjoint method can reconstruct the Brownian Tree in an online fashion by only storing a random seed. The adjoint method is further improved by an efficient data structure, named Brownian Interval [44]. Despite recent advancements, training neural SDEs remains challenging and existing works mainly focus on toy problems and low dimensional data [1, 44].

2.3 Diffusion generative models as SDEs

Diffusion generative models [45] is a highly successful class of probability models that learn empirically and are capable of generating high-fidelity samples through iterative refinement. These models fit a target distribution through the use of two Markov chains.

The first Markov chain is a forward process that gradually converts observed data $\boldsymbol{x}_0$ into noisy data $\boldsymbol{x}_T$ through a series of random transformations $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ so that $\boldsymbol{x}_T$ is close to a simple prior distribution, typically $\boldsymbol{x}_T \sim \boldsymbol{N}(0, \boldsymbol{I})$. The second Markov chain is a reverse process that reconstructs the original data $\boldsymbol{x}_0$ from the noisy data $\boldsymbol{x}_T$ using a series of decoders $p_\theta(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})$ such that the generated $\boldsymbol{x}_0$ is close to the observed data distribution $p_{\text{data}}(\boldsymbol{x})$.

In a similar vein, another closely related area of research involves score-based generative models that estimate the score, $\nabla \log q_t(\boldsymbol{x})$, which is the gradient of the log-likelihood, at each noise scale. These models use Langevin dynamics to sample from a sequence of decreasing noise scales [46, 47].

It has been demonstrated that diffusion generative models and score-based generative models can be unified into a single framework by utilizing different parameterizations of stochastic differential equations (SDEs) [4]. Specifically, this framework consists of a forward SDE and a backward SDE. The forward SDE is a simple linear SDE without learnable parameters, given by:

$$d\boldsymbol{x} = \boldsymbol{F}_t \boldsymbol{x} dt + \boldsymbol{G}_t d\boldsymbol{w}, \tag{2.2}$$

where $\boldsymbol{F}_t \in \mathbb{R}^{D\times D}$ denotes the linear drift coefficient and $\boldsymbol{G}_t \in \mathbb{R}^{D\times D}$ denotes the diffusion coefficient.

The diffusion model given by Eq. (2.2) is initialized at the training data and simulated over a fixed time window $[0, T]$. Let $p_t(\boldsymbol{x}_t)$ denote the marginal distribution of $\boldsymbol{x}_t$ and $p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)$ denote the conditional distribution from $\boldsymbol{x}_0$ to $\boldsymbol{x}_t$. Here, $p_0(\boldsymbol{x}_0)$ represents the underlying distribution of the training data $p_{\text{data}}(\boldsymbol{x})$. The simulated trajectories are represented by $\boldsymbol{x}_t 0 \leq t \leq T$. The parameters $\boldsymbol{F}_t$ and $\boldsymbol{G}_t$ are chosen such that the conditional marginal distribution $p0t(\boldsymbol{x}_t|\boldsymbol{x}_0)$ is a simple Gaussian distribution denoted as $\mathcal{N}(\mu_t \boldsymbol{x}_0, \Sigma_t)$. Additionally, the distribution $\pi(\boldsymbol{x}_T) := p_T(\boldsymbol{x}_T)$ is chosen to be easy to sample from. The denoising diffusion probabilistic models [45](DDPM) and Noise Conditional Score Networks(NCSN)[46] can be formulated by choosing the parameterization shown inTab. 2.1,

Table 2.1: Two popular SDEs, variance preserving SDE (VPSDE) and variance exploding SDE (VESDE). The parameter α_t is decreasing with $\alpha_0 \approx 1, \alpha_T \approx 0$, while σ_t is increasing.

SDE	$\boldsymbol{F}_t$	$\boldsymbol{G}_t$	μ_t	Σ_t
VPSDE (DDPM) [7, 4]	$\frac{1}{2}\frac{d\log\alpha_t}{dt}\boldsymbol{I}$	$\sqrt{-\frac{d\log\alpha_t}{dt}}\boldsymbol{I}$	$\sqrt{\alpha_t}\boldsymbol{I}$	$(1-\alpha_t)\boldsymbol{I}$
VESDE (NCSN) [46, 4]	0	$\sqrt{\frac{d[\sigma_t^2]}{dt}}\boldsymbol{I}$	$\boldsymbol{I}$	$\sigma_t^2\boldsymbol{I}$

which unifies the diffusion generative models and score-based generative models based on SDEs.

Under mild assumptions [48, 4], the forward diffusion process given by Eq. (2.2) is associated with a reverse-time diffusion process, described by

$$d\boldsymbol{x} = [\boldsymbol{F}_t\boldsymbol{x}dt - \boldsymbol{G}_t\boldsymbol{G}_t^T\nabla\log p_t(\boldsymbol{x})]dt + \boldsymbol{G}_t d\boldsymbol{w}, \tag{2.3}$$

where $\boldsymbol{w}$ denotes a standard Wiener process in the reverse-time direction. The distribution of trajectories generated by Eq. (2.3) with terminal distribution $\boldsymbol{x}_T \sim \pi$ is identical to that generated by Eq. (2.2) with initial distribution $\boldsymbol{x}_0 \sim p_0$. This means that Eq. (2.3) matches Eq. (2.2) in probability law. Eq. (2.3) represents an ideal generative model for the data distribution $p_{\text{data}}(\boldsymbol{x})$ and can be used to generate samples from the data distribution by solving Eq. (2.3) backward in time.

The key concept in training diffusion models is the use of a time-dependent network $\boldsymbol{s}_\theta(\boldsymbol{x}, t)$, called a score network, to approximate the score $\nabla\log p_t(\boldsymbol{x})$. This is achieved through score matching techniques [49, 50], where the score network $\boldsymbol{s}_\theta$ is trained by minimizing the denoising score matching loss defined as

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\text{Unif}[0,T]}\mathbb{E}_{p(\boldsymbol{x}_0)p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)}[\|\nabla\log p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0) - \boldsymbol{s}_\theta(\boldsymbol{x}_t, t)\|_{\Lambda_t}^2], \tag{2.4}$$

where $\nabla\log p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)$ has a closed-form expression since $p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)$ is a simple Gaussian distribution. Here, Λ_t is a time-dependent weight. This loss can be evaluated using empirical samples through Monte Carlo methods, allowing for the use of standard stochastic optimization algorithms for training.

Unlike previous neural SDE works [1], diffusion models aim to match the distribution of the backward SDE to that of the forward SDE, rather than specific data trajectories. Thanks to the simple linear SDE, the training objective in Eq. (2.4) is straightforward as a supervised regression problem, making it highly scalable.

2.4 Existing methods

In this section, we provide an overview of popular alternative methods for probability modeling and discuss their advantages and disadvantages.

2.4.1 Autoregressive models

Representation Autoregressive models [29, 30, 31] are a widely used approach for explicit probability modeling. Autoregressive models are constructed based on the probability *chain rule*, which states that the probability of a data point $\boldsymbol{x}$ can be decomposed into the product of the probabilities of each dimension conditioned on the previous dimensions. More formally, this can be written as:

$$p_\theta(\boldsymbol{x}) = \prod_{i=1}^{D} p_\theta(x_i | x_{<i}), \tag{2.5}$$

where D is the dimensionality of the data $\boldsymbol{x}$, x_i denotes the i-th element of $\boldsymbol{x}$, and $x_{<i}$ denotes the elements before the i-th element of $\boldsymbol{x}$, i.e., $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_{i-1}$. Eq. (2.5) provides a natural way to model the data distribution by modeling the conditional distribution $p_\theta(x_i|x_{<i})$ for each dimension i, which is further modeled by using exponential family distributions, such as multinomial or mixture of Gaussian, on top of neural networks.

Learning As autoregressive models are based on the chain rule for probability modeling, practitioners can use maximum likelihood estimation to learn the parameters θ of the model.

The maximum likelihood estimation is achieved by maximizing the following likelihood function:

$$\theta^* = \arg\max_{\theta} \sum_{i=1}^{D} \log p_\theta(x_i | x_{<i}). \tag{2.6}$$

Sampling Sampling from autoregressive models is a straightforward process. Firstly, we draw samples for $\boldsymbol{x}_1$ from $p_\theta(\boldsymbol{x}_1)$. Next, we sample the next element $\boldsymbol{x}_2$ by sampling from $p_\theta(x_2 | x_{<2} = \boldsymbol{x}_1)$, and so on. This process can be repeated for $p_\theta(\boldsymbol{x}_i | \boldsymbol{x}_{<i})$ until we have generated a sample $\boldsymbol{x}$ of the same dimension as the data.

Discussion The training objective of autoregressive models given in Eq. (2.6) is highly scalable and is employed to train large generative models, such as the ones used in language modeling with billions of parameters [51].

While autoregressive models are popular for explicit probability modeling, they require a particular order of data dimensions. However, not all data domains have a natural ordering of dimensions. For instance, in the image domain, it is unclear what a good ordering of pixels would be. Moreover, the autoregressive properties and strict ordering limit the design space of neural networks to specific architectures, such as masked convolutional networks [31] and causal attention [52].

2.4.2 Variational autoencoders

Representation A variational autoencoder (VAE) [33] is a latent variable model by augmenting the data distribution $p(\boldsymbol{x})$ with a prior distribution $p(\boldsymbol{z})$, which is usually a simple distribution to draw samples and evaluating density, such as a Gaussian distribution. To model probability, a VAE replies on an encoder $p_\theta(\boldsymbol{z}|\boldsymbol{x})$. The encoder is a neural network that maps the data $\boldsymbol{x}$ to a latent variable $\boldsymbol{z}$. Therefore, the probability model can be formulated as

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) d\boldsymbol{z}, \tag{2.7}$$

which can be interpreted as an infinite mixture of Gaussian distributions.

Learning Although Eq. (2.7) provides a possibility to evaluate the probability of data, it is intractable in practice since the integral Eq. (2.7) is intractable. Instead of maximizing the probability of data exactly, VAEs maximize the evidence lower bound (ELBO) of the data distribution $p(\boldsymbol{x})$, which reads

$$\begin{aligned}
\log p_\theta(\boldsymbol{x}) &= \log \int p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z} \\
&= \log \int q_\phi(\boldsymbol{z}|\boldsymbol{x})\frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}d\boldsymbol{z} \\
&\geq \int q_\phi(\boldsymbol{z}|\boldsymbol{x})\log\frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}d\boldsymbol{z} \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z}) + \log p(\boldsymbol{z}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - \mathcal{D}_{KL}\left[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})\right], \qquad (2.8)
\end{aligned}$$

where $q_\phi(\boldsymbol{z}|\boldsymbol{z})$ is the encoder, which is particularly helpful for evaluating the lower bound of the data likelihood.

Sampling Sampling from Variational Autoencoders (VAEs) is a simple process owing to the manageable prior distribution $p(\boldsymbol{z})$ and the decoder $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. First, samples are drawn from the prior distribution $p(\boldsymbol{z})$, after which the latent variable $\boldsymbol{z}$ is decoded into the corresponding data $\boldsymbol{x}$ using the decoder $p_\theta(\boldsymbol{x}|\boldsymbol{z})$.

Discussions VAEs exhibit a high degree of versatility in modeling intricate data distributions, and the architecture of neural networks used to build them is flexible. Nevertheless, VAEs have several limitations that hinder their scalability to high-dimensional data with good quality. Firstly, the optimal decoder for a given encoder q_θ is a weighted average of the data, expressed as

$$\mathbb{E}_{p_{\text{data}}(x)}[\frac{q_\theta(z|x)}{\mathbb{E}_{p_{\text{data}}}[q_\theta(z|x)]}x],$$

which inevitably results in a blurry reconstruction unless the encoder is a delta function. Secondly, it has been observed that VAE encoders may neglect the data and instead concentrate solely on the prior distribution $p(\boldsymbol{z})$, resulting in posterior collapse. Researchers have found that utilizing a richer and more expressive prior distribution $p(\boldsymbol{z})$ can alleviate this issue. However, training deep VAEs with complex prior distributions remains a challenging and active area of research [53, 54].

2.4.3 Normalizing Flows

Representation Normalizing flows are a popular choice for explicit probability modeling. They connect a simple distribution $\pi(\boldsymbol{z})$ to a complex distribution $p(\boldsymbol{x})$ through an invertible transformation $\boldsymbol{x} = f_\theta(\boldsymbol{z})$, where θ is the parameter of the transformation. The key component of a normalizing flow is the continuously differentiable transformation $f_\theta : \mathbb{R}^D \rightarrow \mathbb{R}^D$ implemented using a deep neural network. By applying the *change of variable formula*, a normalizing flow models the probability of data as

$$p_\theta(\boldsymbol{x}) = \frac{\pi(f_\theta^{-1}(\boldsymbol{x}))}{\left|\det \frac{\partial f_\theta(\boldsymbol{z})}{\partial \boldsymbol{z}}\right|}. \tag{2.9}$$

Learning In order to fit a normalizing flow to data, we aim to maximize the log-likelihood of the data, which can be written as

$$\theta^* = \arg\max_\theta \log p_\theta(\boldsymbol{x}). \tag{2.10}$$

Sampling Sampling from a normalizing flow follows a similar process to that of VAEs. Firstly, samples are drawn from the prior distribution $\pi(\boldsymbol{z})$, which are then transformed into the data space $\boldsymbol{x}$ using $f_\theta(\boldsymbol{z})$.

Discussions The use of a bijective transformation f_θ in normalizing flows allows for efficient evaluation of data probabilities and sampling from the model. However, learning such a transformation is a challenging task, as the bijective property places restrictions on the design space of neural network architectures. Moreover, normalizing flows require the determinant of the Jacobian matrix to be tractable, which further limits the space of possible neural network designs.

2.4.4 Generative adversarial networks

Representation Generative adversarial networks (GANs) offer a direct approach to modeling the sampling process, circumventing the challenges associated with modeling a distribution and density querying. GANs operate by first drawing a sample $\boldsymbol{z}$ from a prior distribution $\pi(\boldsymbol{z})$. Then, a generator function $G_\theta : \mathbb{R}^m \to \mathbb{R}^n$ transforms the sample $\boldsymbol{z}$ into the generated data $\boldsymbol{x}$. Unlike other generative models, GANs implicitly model the distribution $p_\theta(\boldsymbol{x})$ through $\pi(\boldsymbol{z})$ and $G_\theta(\boldsymbol{z})$.

Learning To fit a GAN to data, we resort to an auxiliary model critic $D_\phi(\boldsymbol{x})$, which is a binary classifier that is learned to distinguish between real data $\boldsymbol{x}$ and fake data $G_\theta(\boldsymbol{z})$. Therefore, the critic D_ϕ and deterministic mapping G_θ are trained in an adversarial manner, resulting in a two-player game. Specifically, the training objective is given by

$$\min_\theta \max_\phi \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log D_\phi(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z}\sim\pi(\boldsymbol{z})}\left[\log(1 - D_\phi(G_\theta(\boldsymbol{z})))\right]. \tag{2.11}$$

Discussions GANs enjoys several merits. First, the sampling procedure of GANs is efficient. Second, GANs have a flexible design space. GANs can be built with flexible deep neural networks and show better sampling performance than VAEs and normalizing flows. However, GANs can not estimate the density of given data samples and it is difficult to infer latent variables from data samples. Moreover, the training objective in Eq. (2.11) is not stable and demands careful hyperparameter tuning and advanced optimization techniques.

CHAPTER 3
LEARNING NEURAL SDES WITH EMPIRICAL SAMPLES

3.1 Introduction

Generative model is a class of machine learning models used to estimate data distributions and sometimes generate new samples from the distributions [55, 56, 57, 58, 59]. Many generative models learn the data distributions by transforming a latent variable $\boldsymbol{z}$ with a tractable prior distribution to the data space [55, 56, 60]. To generate new samples, one can sample from the latent space and then follow the transformation to the data space. There exist a large class of generative models where the latent space and the data space are of the same dimension. The latent variable and the data are coupled through trajectories in the same space. These trajectories serve two purposes: in the forward direction $\boldsymbol{x} \to \boldsymbol{z}$, the trajectories infer the posterior distribution in the latent space associated with a given data sample $\boldsymbol{x}$, and in the backward direction $\boldsymbol{z} \to \boldsymbol{x}$, it generates new samples by simulating the trajectories starting from the latent space. This type of generative model can be roughly divided into two categories, depending on whether these trajectories connecting the latent space and the data space are deterministic or stochastic.

When deterministic trajectories are used, these generative models are known as flow-based models. The latent space and the data space are connected through an invertible map, which could either be realized by the composition of multiple invertible maps [56, 55, 61] or a differential equation [62, 63]. In these models, the probability density at each data point can be evaluated explicitly using the change of variable theorem, and thus the training can be carried out by minimizing the negative log-likelihood (NLL) directly. One limitation of the flow-based model is that the invertible map parameterized by neural networks used in it imposes topological constraints on the transformation from $\boldsymbol{z}$ to $\boldsymbol{x}$. Such limitation affects the performance significantly when the prior distribution on z is a simple unimodal distribution such as Gaussian while the target data distribution is a well-separated multi-modal distribution, i.e., its support has multiple isolated components. In [64], it is shown that there are some fundamental issues of using well-conditioned invertible functions to approximate such complicated multi-modal data distributions.

When stochastic trajectories are used, the generative models are often known as the diffusion model [65]. In a diffusion model, a prespecified stochastic forward process gradually adds noise into the data to transform the data samples into simple random variables. A separate backward process is trained to revert this process to gradually remove the noise from the data to recover the original data distributions. When the forward process is modeled by a stochastic differential equation (SDE), the optimal backward SDE [66] can be retrieved by learning the score function [67, 68, 7, 69]. When the noise is added to the data sufficiently slow in the forward process, the backward diffusion can often revert the forward one reasonably well and is able to generate high fidelity samples. However, this also means that the trajectories have to be sufficiently long with a large number of time-discretization steps, which leads to slow training and sampling. In addition, since the forward process is fixed, the way noise is added is independent of the data distribution. As a consequence, the learned model may miss some complex but important details in the data distribution, as we will explain later.

In this work, we present a new generative modeling algorithm that resembles both the flow-based models and the diffusion models. It extends the normalizing flow method by gradually adding noise to the sampling trajectories to make them stochastic. It extends the diffusion model by making the forward process from x to z trainable. Our algorithm is thus termed *Diffusion Normalizing Flow (DiffFlow)*. The comparisons and relations among DiffFlow, normalizing flow, and diffusion models are shown in Figure Figure 3.1. When the noise in DiffFlow shrinks to zero, DiffFlow reduces to a standard normalizing flow. When the forward process is fixed to some specific type of diffusion, DiffFlow reduces to a diffusion model.

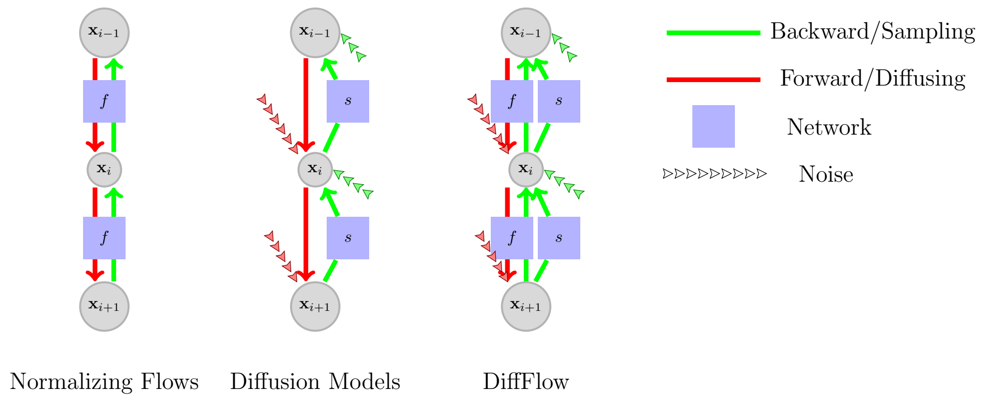


Figure 3.1: The schematic diagram for normalizing flows, diffusion models, and DiffFlow. In normalizing flow, both the forward and the backward processes are deterministic. They are the inverse of each other and thus collapse into a single process. The diffusion model has a fixed forward process and trainable backward process, both are stochastic. In DiffFlow, both the forward and the backward processes are trainable and stochastic.

In DiffFlow, the forward and backward diffusion processes are trained simultaneously by minimizing the distance between the forward and the backward process in terms of the Kullback-Leibler (KL) divergence of the induced probability measures [70]. This cost turns out to be equivalent to (see Section section 3.3 for a derivation) the (amortized) negative evidence lower bound (ELBO) widely used in variational inference [71]. One advantage to use the KL divergence directly is that it can be estimated with no bias using sampled trajectories of the diffusion processes. The KL divergence in the trajectory space also bounds the KL divergence of the marginals, providing an alternative method to bound the likelihood (see Section section 3.3 for details). To summarize, we have made the following contributions.

1. We propose a novel density estimation model termed diffusion normalizing flow (DiffFlow) that extends both the flow-based models and the diffusion models. The added stochasticity in DiffFlow boosts the expressive power of the normalizing flow and results in better performance in terms of sampling quality and likelihood. Compared with diffusion models, DiffFlow is able to learn a forward diffusion process to add noise to the data adaptively and more efficiently. This avoids adding noise to regions where noise is not so

desirable. The learnable forward process also shortens the trajectory length, making the sampling much faster than standard diffusion models (We observe a 20 times speedup over diffusion models without decreasing sampling quality much).

2. We develop a stochastic adjoint algorithm to train the DiffFlow model. This algorithm evaluates the objective function and its gradient sequentially along the trajectory. It avoids storing all the intermediate values in the computational graph, making it possible to train DiffFlow for high-dimensional problems.

3. We apply the DiffFlow model to several generative modeling tasks with both synthetic and real datasets, and verify the performance of DiffFlow and its advantages over other methods.

3.2 Background

Below we provide a brief introduction to normalizing flows and diffusion models. In both of these models, we use $\tau = \{\boldsymbol{x}(t), 0 \leq t \leq T\}$ to denote trajectories from the data space to the latent space in the continuous-time setting, and $\tau = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ in the discrete-time setting.

Normalizing Flows: The trajectory in normalizing flows is modeled by a differential equation

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, t, \theta), \tag{3.1}$$

parameterized by θ. This differential equation starts from random $\boldsymbol{x}(0) = \boldsymbol{x}$ and ends at $\boldsymbol{x}(T) = \boldsymbol{z}$. Denote by $p(\boldsymbol{x}(t))$ the probability distribution of $\boldsymbol{x}(t)$, then under mild assumptions, it evolves following [62]

$$\frac{\partial \log p(\boldsymbol{x}(t))}{\partial t} = -\mathrm{tr}(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}). \tag{3.2}$$

Using this relation equation Equation 3.1 equation Equation 3.2 one can compute the likelihood of the model at any data point $\boldsymbol{x}$.

In the discrete-time setting, the map from $\boldsymbol{x}$ to $\boldsymbol{z}$ in normalizing flows is a composition of a collection of bijective functions as $F = F_N \circ F_{N-1} \cdots F_2 \circ F_1$. The trajectory $\tau = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ satisfies

$$\boldsymbol{x}_i = F_i(\boldsymbol{x}_{i-1}, \theta), \quad \boldsymbol{x}_{i-1} = F_i^{-1}(\boldsymbol{x}_i, \theta) \tag{3.3}$$

for all i. Similar to Equation equation Equation 3.2, based on the rule for change of variable, the log-likelihood of any data samples $\boldsymbol{x}_0 = \boldsymbol{x}$ can be evaluated as

$$\log p(\boldsymbol{x}_0) = \log p(\boldsymbol{x}_N) - \sum_{i=1}^{N} \log |\det(\frac{\partial F_i^{-1}(\boldsymbol{x}_i)}{\partial \boldsymbol{x}_i})|. \tag{3.4}$$

Since the exact likelihood is accessible in normalizing flows, these models can be trained by minimizing the negative log-likelihood directly.

Diffusion Models: The trajectories in diffusion models are modeled by stochastic differential equations. More explicitly, the forward process is of the form

$$d\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t)dt + g(t)d\mathbf{w}, \tag{3.5}$$

where the *drift* term $\boldsymbol{f} : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is a vector-valued function, and the *diffusion* coefficient $g : \mathbb{R} \to \mathbb{R}$ is a scalar function (in fact, g is often chosen to be a constant). Here $\mathbf{w}$ denotes the standard Brownian motion. The forward process is normally a simple linear diffusion process [65, 57]. The forward trajectory τ can be sampled using equation Equation 3.5 initialized with the data distribution. Denote by p_F the resulting probability distribution over the trajectories. With a slight abuse of notation, we also use p_F to denote the marginal distribution of the forward process.

The backward diffusion from $\boldsymbol{z} = \boldsymbol{x}(T)$ to $\boldsymbol{x} = \boldsymbol{x}(0)$ is of the form

$$d\boldsymbol{x} = [\boldsymbol{f}(\boldsymbol{x}, t) - g^2(t)\boldsymbol{s}(\boldsymbol{x}, t, \theta)]dt + g(t)d\mathbf{w}. \tag{3.6}$$

It is well-known that when $\boldsymbol{s}$ coincides with the score function $\nabla \log p_F$, and $\boldsymbol{x}(T)$ in the forward and backward processes share the same distribution, the distribution p_B induced by the backward process equation Equation 3.6 is equal to p_F [66],[72, Chapter 13]. To train the score network $\boldsymbol{s}(\boldsymbol{x}, t, \theta)$, one can use the KL divergence between p_F and p_B as an objective function to reduce the difference between p_F and p_B. When the difference is sufficiently small, p_F and p_B should have similar distribution over $\boldsymbol{x}(0)$, and one can then use the backward diffusion equation Equation 3.6 to sample from the data distribution.

In the discrete setting, the trajectory distributions can be more explicitly written as

$$p_F(\tau) = p_F(\boldsymbol{x}_0) \prod_{i=1}^{N} p_F(\boldsymbol{x}_i | \boldsymbol{x}_{i-1}), \quad p_B(\tau) = p_B(\boldsymbol{x}_T) \prod_{i=1}^{N} p_B(\boldsymbol{x}_{i-1} | \boldsymbol{x}_i). \tag{3.7}$$

The KL divergence between p_F and p_B can be decomposed according to this expression equation Equation 3.7. Most diffusion models use this decomposition, and meanwhile take advantage of the simple structure of the forward process equation Equation 3.5, to evaluate the objective function in training [67, 68, 73].

3.3 Diffusion normalizing flow

We next present our diffusion normalizing flow models. Similar to diffusion models, the DiffFlow models also has a forward process

$$d\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t, \theta) dt + g(t) d\mathbf{w}, \tag{3.8}$$

and a backward process

$$d\boldsymbol{x} = [\boldsymbol{f}(\boldsymbol{x}, t, \theta) - g^2(t) \boldsymbol{s}(\boldsymbol{x}, t, \theta)] dt + g(t) d\mathbf{w}. \tag{3.9}$$

The major difference is that, instead of being a fixed linear function as in most diffusion models, the drift term $\boldsymbol{f}$ is also learnable in DiffFlow. The forward process is initialized with the data samples at $t = 0$ and the backward process is initialized with a given noise distribution at $t = T$. Our goal is to ensure the distribution of the backward process at time $t = 0$ is close to the real data distribution. That is, we would like the difference between $p_B(\boldsymbol{x}(0))$ and $p_F(\boldsymbol{x}(0))$ to be small.

To this end, we use the KL divergence between $p_B(\tau)$ and $p_F(\tau)$ over the trajectory space as the training objective function. Since

$$KL(p_F(\boldsymbol{x}(t))|p_B(\boldsymbol{x}(t))) \leq KL(p_F(\tau)|p_B(\tau)) \tag{3.10}$$

for any $0 \leq t \leq T$ by data processing inequality, small difference between $p_B(\tau)$ and $p_F(\tau)$ implies small difference between $p_B(\boldsymbol{x}(0))$ and $p_F(\boldsymbol{x}(0))$ in terms of KL divergence.

3.3.1 Implementation

In real implementation, we discretize the forward process equation Equation 3.8 and the backward process equation Equation 3.9 as

$$\begin{aligned} \boldsymbol{x}_{i+1} &= \boldsymbol{x}_i + \boldsymbol{f}_i(\boldsymbol{x}_i)\Delta t_i + g_i \delta_i^F \sqrt{\Delta t_i} && (3.11) \\ \boldsymbol{x}_i &= \boldsymbol{x}_{i+1} - [\boldsymbol{f}_{i+1}(\boldsymbol{x}_{i+1}) - g_{i+1}^2 \boldsymbol{s}_{i+1}(\boldsymbol{x}_{i+1})]\Delta t_i + g_{i+1}\delta_i^B \sqrt{\Delta t_i}, && (3.12) \end{aligned}$$

where $\delta_i^F, \delta_i^B \sim \mathcal{N}(0, \mathbf{I})$ are unit Gaussian noise, $\{t_i\}_{i=0}^N$ are the discretization time points, and $\Delta t_i = t_{i+1} - t_i$ is the step size at the i-th step. Here we have dropped the dependence on the parameter θ to simplify the notation. With this discretization, the KL divergence between trajectory distributions becomes

$$KL(p_F(\tau)|p_B(\tau)) = \underbrace{\mathbb{E}_{\tau\sim p_F}[\log p_F(\boldsymbol{x}_0)]}_{L_0} + \underbrace{\mathbb{E}_{\tau\sim p_F}[-\log p_B(\boldsymbol{x}_N)]}_{L_N} + \sum_{i=1}^{N-1} \underbrace{\mathbb{E}_{\tau\sim p_F}[\log \frac{p_F(\boldsymbol{x}_i|\boldsymbol{x}_{i-1})}{p_B(\boldsymbol{x}_{i-1}|\boldsymbol{x}_i)}]}_{L_i}. \tag{3.13}$$

The term L_0 in equation Equation 3.13 is a constant determined by entropy of the dataset as

$$\mathbb{E}_{\tau\sim p_F}[\log p_F(\boldsymbol{x}_0)] = \mathbb{E}_{\boldsymbol{x}_0\sim p_F}[\log p_F(\boldsymbol{x}_0)] =: -H(p_F(x(0))).$$

The term L_N is easy to calculate since $p_B(x_N)$ is a simple distribution, typically standard Gaussian distribution.

To evaluate $L_{1:N-1}$, we estimate it over sampled trajectory from the forward process p_F. For a given trajectory τ sampled from $p_F(\tau)$, we need to calculate $p_B(\tau)$ along the same trajectory. To this end, a specific group of $\{\delta_i^B\}$ is chosen such that the same trajectory can be reconstructed from the backward process. Thus, δ_i^B satisfies

$$\delta_i^B(\tau) = \frac{1}{g_{i+1}\sqrt{\Delta t}}\left[\boldsymbol{x}_i - \boldsymbol{x}_{i+1} + [\boldsymbol{f}_{i+1}(\boldsymbol{x}_{i+1}) - g_{i+1}^2\boldsymbol{s}_{i+1}(\boldsymbol{x}_{i+1})]\Delta t\right]. \tag{3.14}$$

Since δ_i^B is a Gaussian noise, the negative log-likelihood term $p_B(\boldsymbol{x}_i|\boldsymbol{x}_{i+1})$ is equal to $\frac{1}{2}(\delta_i^B(\tau))^2$ after dropping constants. In view of the fact that the expectation of $\sum_i \frac{1}{2}(\delta_i^F(\tau))^2$ remains a constant, minimizing Equation equation Equation 3.13 is equivalent to minimizing the following loss:

$$L := \mathbb{E}_{\tau\sim p_F}[-\log p_B(\boldsymbol{x}_N) + \sum_i \frac{1}{2}(\delta_i^B(\tau))^2] = \mathbb{E}_{\delta^F;\boldsymbol{x}_0\sim p_0}[-\log p_B(\boldsymbol{x}_N) + \sum_i \frac{1}{2}(\delta_i^B(\tau))^2], \tag{3.15}$$

where the last equality is based on a reparameterization trick [71]. We can minimize the loss in Equation equation Equation 3.15 with Monto Carlo gradient estimation as in Algorithm Algorithm 1.

3.3.2 Stochastic Adjoint method

One challenge in training DiffFlow is memory consumption. When a naive backpropagation strategy is used, the memory consumption explodes quickly. Indeed, differentiating through the operations of the forward pass requires unrolling networks N times and caching

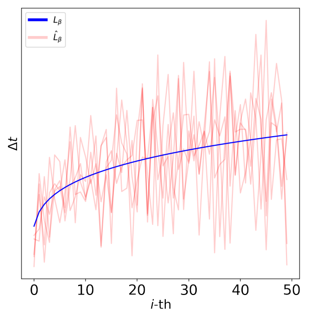
Algorithm 1 Training

```
repeat
    $\boldsymbol{x}_0 \sim$ Real data distribution
    $\delta^F_{1:N} \sim \mathcal{N}(0, \mathbf{I})$
    Discrete timestamps: $t^N_{i=0}$
    Sample: $\tau = \{\boldsymbol{x}_i\}^N_{i=0}$ based on $\delta^F_{1:N}$
    Gradient descent step $\nabla_\theta[-\log p_B(\boldsymbol{x}_N) + \sum_i \frac{1}{2}(\delta^B_i(\tau))^2]$
until converged
```

all network intermediate values for every step, which prevents this naive implementation of DiffFlow from being applied in high dimensional applications. Inspired by the adjoint method in Neural ODE [62], we propose the adjoint variable $\frac{\partial L}{\partial \boldsymbol{x}_i}$ and a stochastic adjoint algorithm that allows training the DiffFlow model with reduced memory consumption. In this adjoint method, we cache intermediate states $\boldsymbol{x}_i$ and, based on these intermediate states, reproduce the whole process, including δ^F_i, δ^B_i as well as f_i, s_i exactly.

We note another similar approach [1] of training SDEs caches random noise $d\boldsymbol{w}$ and further takes advantage of the pseudo-random generator to save memory for the intermediate noises, resulting in constant memory consumption However, the approach can not reproduce exact trajectories due to time discretization error and requires extra computation to recover $d\boldsymbol{w}$ from the pseudo-random generator. We found that in our image experiments in Section section 3.4, the cached $\{\boldsymbol{x}_i\}$ consumes only about 2% memory out of all the memory being used during training. Due to the accuracy and acceptable memory overhead, the introduced stochastic adjoint approach is a better choice for DiffFlow. We summarize the method in Algorithm Algorithm 2 and Figure Figure 3.2. We also include the PyTorch [74] implementation in the supplemental material.

Figure 3.3: Δt_i of L_β and $\hat{L}_\beta$



Algorithm 2 Stochastic Adjoint Algorithm for DiffFlow

```
1: Input: Forward trajectory $\{\boldsymbol{x}_i\}_{i=0}^N$
2: $\frac{\partial L}{\partial \boldsymbol{x}_N} = \frac{1}{2}\frac{\partial(\delta_N^B(\tau))^2}{\partial \boldsymbol{x}_N} - \frac{\partial \log p_B(\boldsymbol{x}_N)}{\boldsymbol{x}_N}$
3: $\frac{\partial L}{\partial \theta} = 0$
4: for $i = N, N-1, \cdots, 1$ do
5:     $\frac{\partial L}{\partial \boldsymbol{x}_{i-1}} = (\frac{\partial L}{\partial \boldsymbol{x}_i} + \frac{1}{2}\frac{\partial(\delta_i^B(\tau))^2}{\partial \boldsymbol{x}_i})\frac{\partial \boldsymbol{x}_i}{\partial \boldsymbol{x}_{i-1}} + \frac{1}{2}\frac{\partial(\delta_i^B(\tau))^2}{\partial \boldsymbol{x}_{i-1}}$
6:     $\frac{\partial L}{\partial \theta} += \frac{1}{2}\frac{\partial(\delta_i^B(\tau))^2}{\partial \theta} + (\frac{\partial L}{\partial \boldsymbol{x}_i} + \frac{1}{2}\frac{\partial(\delta_i^B(\tau))^2}{\partial \boldsymbol{x}_i})\frac{\partial \boldsymbol{x}_i}{\partial \theta}$
7: end for
```

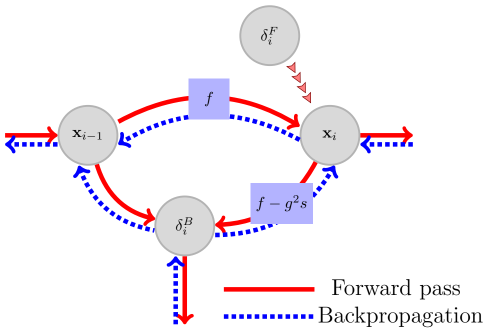


Figure 3.2: Gradient Flowchart.

3.3.3 Time discretization and progressive training

We propose two time discretization schemes for training DiffFlow: fixed timestamps L_β and flexible timestamps $\hat{L}_\beta$. For fixed timestamps, the time interval $[0, T]$ is discretized with fixed schedule $t_i = (\frac{i}{N})^\beta T$. With such fixed time discretization over batches, we denote the loss function as L_β. Empirically, we found $\beta = 0.9$ works well. This choice of β increases stepsize Δt_i when the forward process approaches $\mathbf{z} = \mathbf{x}_N$ and provides higher resolution when the backward process is close to $\mathbf{x}_0$. We found such discretization generates samples with good quality and high fidelity details. The choice of polynomial function is arbitrary; other functions of similar sharps may work as well.

In the flexiable timestamps scheme, we train different batches with different time discretization points. Specifically, t_i is sampled uniformly from the interval $[(\frac{i-1}{N-1})^\beta T, (\frac{i}{N-1})^\beta T]$ for each batch. We denote the training objective function with such random time discretization as $\hat{L}_\beta$. We empirically found such implementation results in lower loss and better stability when we conduct progressive training where we increase N gradually as training progresses. In progressive training, we refine the forward and backward processes as N increase. This training scheme can significantly save training time compared with the other method that uses a fixed large N during the whole process. Empirically, we found that progressive training can speed up the training up to 16 times.

To understand why such random time discretization scheme is more stable, we hypothesis that this choice encourages a smoother $\boldsymbol{f}, \boldsymbol{s}$ since it seeks functions $\boldsymbol{f}, \boldsymbol{s}$ to reduce objective loss under different sets of $\{t_i\}$ instead of a specific $\{t_i\}$. We illustrate fixed timestamps in L_β and a realization of random discretization in $\hat{L}_\beta$ in Figure Figure 3.3 with $\beta = 0.9$.

3.3.4 Learnable forward process

The forward process not only is responsible for driving data into latent space, but also provides enough supervised information to learning backward process. Thanks to bijective property, NFs can reconstruct data exactly but there is no guarantee that it can reach the standard Gaussian. At the other extreme, Denoising diffusion probabilistic models (DDPM) [7] adopt a data-invariant forward diffusing schema, ensuring that $\boldsymbol{x}_N$ is Gaussian. DDPM can even reach Gaussian in one step with $N = 1$, which output noise disre-

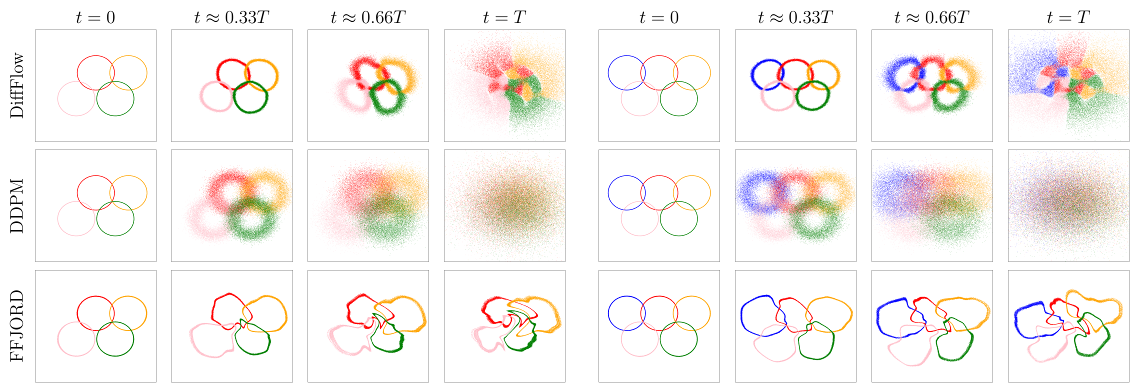


Figure 3.4: Illustration of forwarding trajectories of DiffFlow, DDPM, and FFJORD. Each row shows two trajectories of transforming data distributions, four rings, and Olympics rings, to a base distribution. Different modes of densities are in different colors. Though FFJORD adjusts forward process based on data, its bijective property prevents the approach from expanding density support to the whole space. DDPM can transform data distributions into Gaussian distribution but a data-invariant way of adding noise can corrupt the details of densities, e.g., the densities at the intersections of the rings. DiffFlow not only transforms data into base distribution but also keeps the topological information of the original datasets. Points from the same ring are transformed into continental plates instead of being distributed randomly.

garding data samples. However, backward process will be difficult to learn if data is destroyed in one step. Therefore, DDPM adds noise slowly and often needs more than one thousand steps for diffusion.

The forward module of DiffFlow is a combination of normalizing flow and diffusion models. We show the comparision in fitting toy 2D datasets in Figure Figure 3.4. We are especially interested in data with well-separated modes and sharp density boundaries. Those properties are believed to appear in various datasets. As stated by manifold hypothesis [75], real-world data lie on low-dimensional manifold [76] embedded in a high-dimensional space. To construct the distributions in Figure Figure 3.4, we rotate the 1-d Gaussian distribution $\mathcal{N}(1, 0.001^2)$ around the center to form a ring and copy the rings to different locations.

As a bijective model, FFJORD [63] struggles to diffuse the concentrated density mass into a Gaussian distribution. DiffFlow overcomes expressivity limitations of the bijective constraint by adding noise. As added noise shrinks to zero, the DiffFlow has no stochasticity and degrades to a flow-based model. Based on this fact, we present the following theorem.

Theorem 1. *As diffusion coefficients $g_i \to 0$, DiffFlow reduces to Normalizing Flow. Moreover, minimizing the objective function in Equation equation Equation 3.13 is equivalent to minimizing the negative log-likelihood as in Equation equation Equation 3.4.*

DDPM [7] uses a fixed noising transformation. Due to the data-invariant approach $p(\boldsymbol{x}_T|\boldsymbol{x}_0) = p(\boldsymbol{x}_T)$, points are diffused in the same way even though they may appear in different modes or different datasets. We observe that sharp details are destroyed quickly in DDPM diffusion, such as the intersection regions between rings. However, with the help of learnable transformation, DiffFlow diffuses in a much efficient way. The data-dependent approach shows different diffusion strategies on different modes and different datasets. Meanwhile, similar to NFs, it keeps some topological information for learning backward processes. We include more details about the toy sample in Section section 3.4.

3.4 Experiments

We evaluate the performance of DiffFlow in sample quality and likelihood on test data. To evaluate the likelihood, we adopt the marginals distribution equivalent SDEs

$$d\boldsymbol{x} = [\boldsymbol{f}(\boldsymbol{x}, t, \theta) - \frac{1+\lambda^2}{2} g^2(t)\boldsymbol{s}(\boldsymbol{x}, t, \theta)]dt + \lambda g(t) d\boldsymbol{w}, \tag{3.16}$$

with $\lambda \geq 0$. When $\lambda = 0$, it reduces to probability ODE [73]. The ODE provides an efficient way to evaluate the density and negative log-likelihood. For any $0 \leq \lambda \leq 1$, the above SDE can be used for sampling. Empirically, we found $\lambda = 1$ has the best performance.

3.4.1 Synthetic 2D examples

We compare the performance of DiffFlow and existing diffusion models and NFs on estimating the density of 2-dimensional data. We compare the forward trajectories of DiffFlow, DDPM [7] and FFJORD [63][1] in Figure Figure 3.4 and its sampling performance in Figure Figure 3.5. To make a fair comparison, we build models with comparable network sizes, around 90k learnable parameters.

All three algorithms have good performance on datasets whose underlying distribution has smooth density, such as 2 spirals. However, when we shrink the support of samples or add complex patterns, performance varies significantly. We observe that FFJORD leaks many samples out of the main modes and datasets with complex details and sharp density exacerbates the disadvantage.

DDPM has higher sample quality but blurs density details, such as intersections between rings, areas around leaves of the Fractal tree, and boxes in Sierpiński Carpet. The performance is within expectation given that details are easy to be destroyed and ignored with the data-invariant noising schema. On the less sharp dataset, such as 2 Spirals and Checkerboard, its samples align with data almost perfectly.

DiffFlow produces the best samples (according to a human observer). We owe the performance to the flexible noising forward process. As illustrated in Figure Figure 3.4, DiffFlow provides more clues and retains detailed patterns longer for learning its reverse process. DiffFlow has a much lower negative likelihood, especially on sharp datasets.

3.4.2 Density estimation on real data

We perform density estimation experiments on five tabular datasets [77]. We employ the probability flow to evaluate the negative log-likelihood. We find that our algorithm has better performance in most datasets than most approaches trained by directly minimizing negative log-likelihood, including NFs and autoregressive models. DiffFlow outperforms

[1]Implementation of FFJORD and DDPM are based on official codebases.

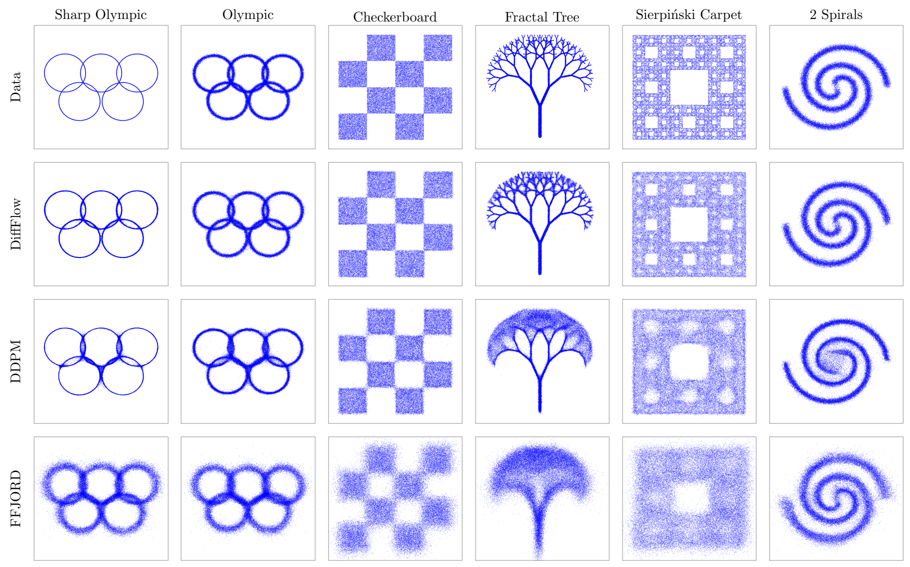


Figure 3.5: Samples from DiffFlow, DDPM and FFJORD on 2-D datasets. All three models have reasonable performance on datasets that have smooth underlying distributions. But only DiffFlow is capable to capture complex patterns and provides sharp samples when dealing with more challenging datasets.

FFJORD by a wide margin on all datasets except HEPMASS. Compared with autoregressive models, it excels NAF [78] on all but GAS. Those models require $\mathcal{O}(d)$ computations to sample from. Meanwhile, DiffFlow is quite effective in achieving such performance with MLPs that have less than 5 layers.

3.4.3 Image generation

In this section, we report the quantitative comparison and qualitative performance of our method and existing methods on common image datasets, MNIST [81] and CIFAR-10 [82]. We use the same unconstrained U-net style model as used successfully by [7] for drift and score network. We reduce the network size to half of the original DDPM network so that the total number of trainable parameters of DiffFlow and DDPM are comparable. We use small $N = 10$ at the beginning of training and slowly increase to large N as training proceeds. The schedule of N reduces the training time greatly compared with using large N all the time. We use constants $g_i = 1$ and $T = 0.05$ for MNIST and CIFAR10, and $N = 30$ for sampling MNIST data and $N = 100$ for sampling CIFAR10. As it is reported

Table 3.1: Average negative log-likelihood (in nats) on tabular datasets [77] for density estimation (lower is better).

Dataset	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
RealNVP [55]	-0.17	-8.33	18.71	13.55	-153.28
FFJORD [63]	-0.46	-8.59	**14.92**	10.43	-157.40
DiffFlow (ODE)	**-1.04**	-10.45	15.04	**8.06**	**-157.80**
MADE [79]	3.08	-3.56	20.98	15.59	-148.85
MAF [77]	-0.24	-10.08	17.70	11.75	-155.69
TAN [80]	-0.48	-11.19	15.12	11.01	-157.03
NAF [78]	-0.62	**-11.96**	15.09	8.86	-157.73

by [83], adding noise at the last step will significantly lower sampling quality, we use one single denoising step at the end of sampling with Tweedie's formula [84].

We report negative log-likelihood (NLL) in bits per dimension or negative ELBO if NLL is unavailable. On MNIST, we achieve competitive performance on NLL as in Table Table 3.2. We show the uncurated samples from DiffFlow in Figure Figure 3.6 and Figure Figure 3.7. On CIFAR-10, DiffFlow also achieves competitive NLL performance as shown in Table Table 3.3. DiffFlow performs better than normalizing flows and DDPM models, but is slightly worse than DDPM++(sub, deep, sub-vp) and Improved DDPM. However, these approaches conduct multiple architectural improvements and use much deeper and wider networks. We also report the popular sample metric, Fenchel Inception Distance (FID) [85]. DiffFLow has a lower FID score than normalizing flows and has competitive performance compared with DDPM trained with unweighted variational bounds, DDPM and Improved DDPM. It is worse than DDPM trained with reweighted loss, DDPM (L_s), DDPM cont, and DDPM++ [7, 73, 86]. Besides, sampling quality with different sampling steps N are compared in Table Table 3.4 [2]. The advantage of DiffFlow is clear when we compare relative FIDs degeneracy ratio with $N = 100$ respectively. DiffFlow is able to retain better sampling quality when decreasing N.

[2]The performance of DDPM is evaluated based on the officially released checkpoint with L_s denotes for L_{simple} in the original paper.

Table 3.2: NLL on MNIST

Model	NLL (↓)
RealNVP [55]	1.06
Glow [61]	1.05
FFJORD [63]	0.99
ResFlow [87]	0.97
DiffFlow	0.93

Figure 3.6: MNIST Samples

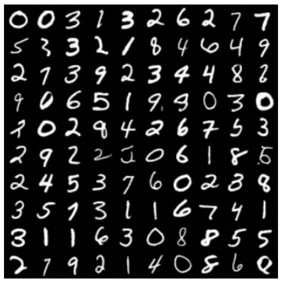


Figure 3.7: CIFAR10 Samples

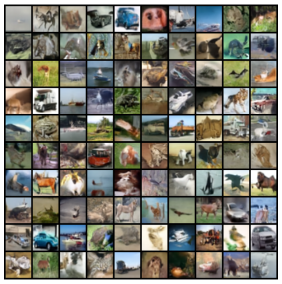


Table 3.3: NLLs and FIDs on CIFAR-10.

Model	NLL(↓)	FID (↓)
RealNVP [55]	3.49	-
Glow [61]	3.35	46.90
Flow++ [57]	3.29	-
FFJORD [63]	3.40	-
ResFlow [87]	3.28	46.37
DDPM (L) [7]	≤ 3.70	13.51
DDPM (L_s) [7]	≤ 3.75	3.17
DDPM (L_s)(ODE) [73]	3.28	3.37
DDPM cont. (sub-VP) [73]	3.05	3.56
DDPM++ (sub-VP) [73]	3.02	3.16
DDPM++ (deep, sub-VP) [73]	2.99	2.94
Improved DDPM [86]	$\leq$2.94	11.47
DiffFlow (L_β)	≤ 3.71	13.87
DiffFlow ($\hat{L}_\beta$)	≤ 3.67	13.43
DiffFlow ($\hat{L}_\beta$, ODE)	3.04	14.14

Table 3.4: FIDs with various N

N	DiffFlow	DDPM (L)	DDPM (L_s)	DDIM
5	28.31	373.51	370.23	44.69
10	22.56	364.64	365.12	18.62
20	17.98	138.84	135.44	10.89
50	14.72	47.12	34.56	7.01
100	13.43	22.23	10.04	5.63

Table 3.5: Relative FIDs degeneracy ratio

N	DiffFlow	DDPM (L)	DDPM (L_s)	DDIM
5	2.12	16.80	37.02	7.94
10	1.68	16.40	36.12	3.31
20	1.34	6.24	13.54	1.93
50	1.10	2.12	3.45	1.24
100	1.0	1.0	1.0	1.0

3.5 Related work

Normalizing flows [55, 56] have recently received lots of attention due to its exact density evaluation and ability to model high dimensional data [61, 88]. However, the bijective requirement poses limitations on modeling complex data, both empirically and theoretically [70, 64]. Some works attempt to relax the bijective requirement; discretely index flows [88] use domain partitioning with only locally invertible functions. Continuously indexed flows [64] extend discretely indexing to a continuously indexing approach. As pointed out in Stochastic Normalizing Flows (SNF) [70], stochasticity can effectively improve the expressive power of the flow-based model in low dimension applications. The architecture used in SNF, which requires known underlying energy models, presents challenges for density learning tasks; SNF is designed for sampling from unnormalized probability distribution instead of density estimation. Besides, even with ideal networks and infinite amount of data, due to the predefined stochstic block being used, SNF cannot find models with aligned forward and backward distribution as DiffFlow.

When it comes to stochastic trajectories, minimizing the distance between trajectory distributions has been explored in existing works. Denoising diffusion model [65] uses a fixed linear forward diffusion schema and reparameterizes the KL divergence such that minimizing loss is possible without computing whole trajectories. Diffusion models essentially corrupt real data iteratively and learn to remove the noise when sampling. Recently, Diffusion models have shown the capability to model high-dimensional data distribution, such as images [7, 68], shapes [89], text-to-speech [90]. Lately, the Score-based model [73] provides a unified framework for score-matching methods and diffusion models based on stochastic calculus. The diffusion processes and sampling processes can be viewed as forwarding SDE and reverse-time SDE. Thanks to the linear forward SDE being used in DDPM, the forward marginal distributions have a closed-form and are suitable for training score functions on large-scale datasets. Also, due to the reliance on fixed linear forward process, it takes thousands of steps to diffuse data and generate samples. DiffFlow considers general SDEs and nosing and sampling are more efficient.

Existing Neural SDE approaches suffer from poor scaling properties. Backpropagating through solver [91] has a linear memory complexity with the number of steps. The pathwise approach [92] scales poorly in computation complexity. Our stochastic adjoint approach shares a similar spirit with SDE adjoint sensitivity [93]. The choice of caching noise requires high resolution of time discretization and prevents the approach from scaling to high dimension applications. By caching the trajectory states, DiffFlow can use a coarser discretization and deploy on larger dimension problems and problems with more challenging densities. The additional memory footprint is negligible compared with the other network memory consumption in DiffFlow.

3.6 Limitations

While DiffFlow gains more flexibility due to the introduction of a learnable forward process, it loses the analytical form for $p_F(\boldsymbol{x}_t|\boldsymbol{x}_0)$ and thus the training less efficient compared with score-based loss [73]. Training DiffFlow relies on backpropagation through trajectories and is thus significantly slower than diffusion models with affine drift. Empirically, we found DiffFlow is about 6 times slower than DDPM in 2d toy examples, 55 times in MNIST, and 160 times in CIFAR10 without progressive training in Section subsection 3.3.3. Though the stochastic adjoint method and progressive training help save memory footprint and reduce training time, the training of DiffFlow is still more expensive than DDPM and its variants. On the other hand, compared with normalizing flows, the extra noise in DiffFlow boosts the expressive power of the model with little extra cost. Though DiffFlow trained based on SDE, its marginal distribution equivalent ODE Equation 3.2 shows much better performance than its counterpart trained with ODE [63]. It is interesting to investigate, both empirically and theoretically, the benefit in terms of expressiveness improvement caused by stochastic noise for training normalizing flows.

3.7 Conclusions

We proposed a novel algorithm, the diffusion normalizing flow (DiffFlow), for generative modeling and density estimation. The proposed method extends both the normalizing flow models and the diffusion models. Our DiffFlow algorithm has two trainable diffusion processes modeled by neural SDEs, one forward and one backward. These two SDEs are trained jointly by minimizing the KL divergence between them. Compared with most normalizing flow models, the added noise in DiffFlow relaxes the bijectivity condition in deterministic flow-based models and improves their expressive power. Compared with diffusion models, DiffFlow learns a more flexible forward diffusion that is able to transform data into noise more effectively and adaptively. In our experiments, we observed that DiffFlow is able to model distributions with complex details that are not captured by representative normalizing flow models and diffusion models, including FFJORD, DDPM. For CIFAR10 dataset, our DiffFlow method has worse performance than DDPM in terms of FID score. We believe our DiffFlow algorithm can be improved further by using different neural network architectures, different time discretizing method and different choices of time interval. We plan to explore these options in the near future.

Our algorithm is able to learn the distribution of high-dimensional data and then generate new samples from it. Like many other generative modeling algorithms, it may be potentially used to generate misleading data such as fake images or videos.

CHAPTER 4
LEARNING GENERATIVE MODELS WITH PIECES DATA

4.1 Introduction

The success of diffusion models [45, 4] can largely be attributed to their scalability. With large-scale datasets and computing resources, practitioners can usually train high-capacity models that are able to produce high-fidelity images. The recent generative AI revolution led by large-scale text-to-image diffusion models is a great example [11, 19, 20]. The same procedure, collecting a large dataset and using it to train a large-scale model, has been applied to various problems and achieved great success [94, 51].

In this paper, we are interested in extending the success of diffusion models to a wider class of data. We focus on applications where a large-scale dataset of the target content does not exist or is prohibitively expensive to collect, but individual pieces of the content are available in great quantities. 360-degree panorama images are such an example. While 360-degree panorama images are considered niche image content and only exist in small quantities, there are a large number of normal perspective images available on the Internet, each of which can be treated as a piece of a 360-degree panorama image. Another example is generating images of extreme aspect ratios, as shown in Fig. 4.1. Each of the extreme-aspect-ratio images can be considered as the stitching of multiple images with normal aspect ratios. For such applications, while we cannot afford to collect a large-scale dataset of the target content to train a diffusion model, we wish to synthesize high-quality target content with a diffusion model trained on smaller pieces that are readily available.

A popular solution to this class of problems is to first train a diffusion model on small pieces of the content and then generate the large content piece by piece in an autoregressive manner [95, 96]. However, such an autoregressive approach has three drawbacks. First, as

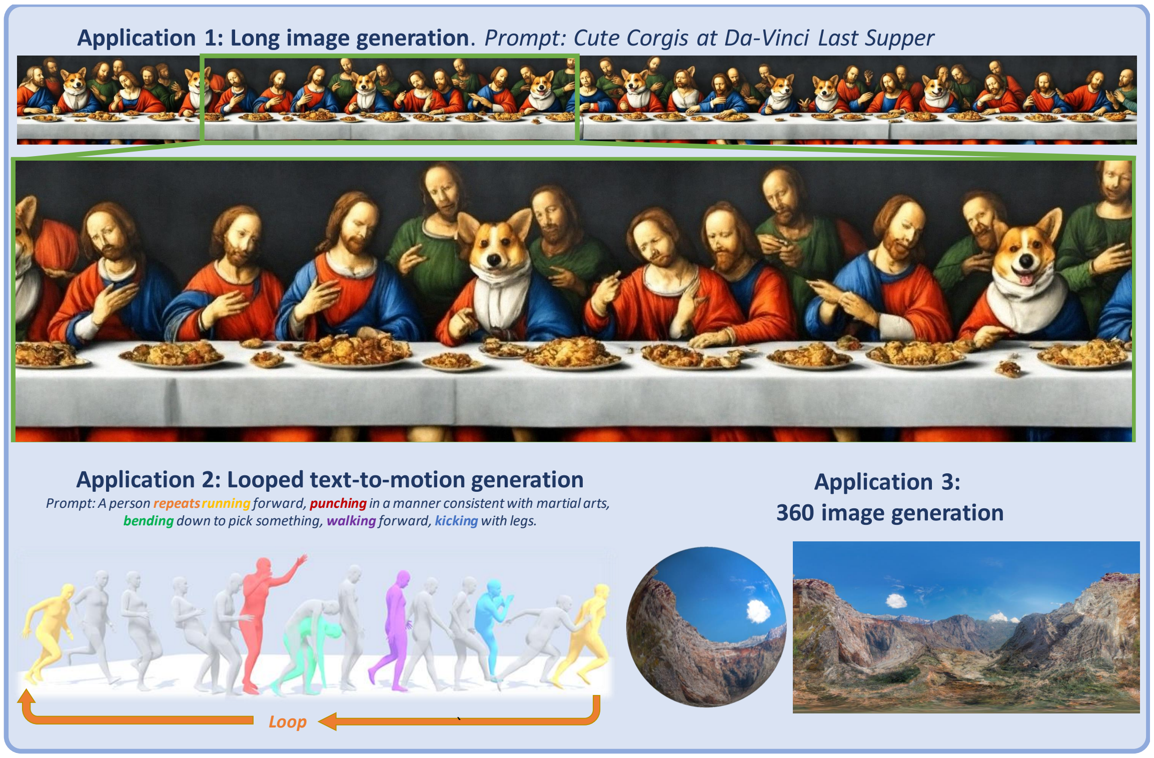

Application 1: Long image generation. Prompt: Cute Corgis at Da-Vinci Last Supper
Application 2: Looped text-to-motion generation
Prompt: A person repeats running forward, punching in a manner consistent with martial arts, bending down to pick something, walking forward, kicking with legs.
Loop
Application 3: 360 image generation

Figure 4.1: `DiffCollage`, a scalable probabilistic model that synthesizes large content, including long images, looped motions, and 360 images, with diffusion models only trained on pieces of the content.

pieces are generated sequentially, the later-generated pieces have no influence on the prior-generated ones. Such a sequential scheme could lead to sub-optimal results, especially when there is a circular structure in the data. For example, it is hard to enforce consistency between the start and end frames when generating looped videos autoregressively. Second, autoregressive methods may suffer from error accumulation since the model was conditioned on ground-truth data during training but is conditioned on its own prediction at test time. Lastly, the time consumption of autoregressive generation increases linearly with the size of the data and could become prohibitive when generating very large content.

To address the large content generation problem, we propose `DiffCollage`, a generic algorithm that synthesizes large content by merging the results generated by diffusion models trained on small pieces of the large content. Our approach is based on a factor graph formulation where a datum is modeled by a set of nodes and the edges connecting them.

In our formulation, each node represents a contiguous portion of the large content, and the portions of content in neighboring nodes have a small overlap. Each node is associated with a small diffusion model and each piece affects the generation of the other pieces. Our method generates multiple pieces of content in parallel, which can greatly accelerate sampling when a large pool of computation is available.

We evaluate our approach on multiple large content generation tasks, including infinity image generation, long-duration text-to-motion with complex actions, content with unusual structures such as looped motion, and 360-degree images. Experiment results show that our approach outperforms existing approaches by a wide margin.

In summary, we make the following contributions.

- We propose `DiffCollage`, a scalable probabilistic model that synthesizes large content by merging results generated by diffusion models trained on pieces of the large content. It can synthesize large content efficiently by generating pieces in parallel.
- `DiffCollage` can work out-of-the-box when pre-trained diffusion models on different pieces are available.
- Extensive experimental results on benchmark datasets show the effectiveness and versatility of the proposed approaches on various tasks.

4.2 Related Work

Diffusion models Diffusion models [97, 45, 4] have achieved great success in various problems, such as text-to-image generation [11, 19, 20], time series modeling [98], point cloud generation [99, 100, 101], natural language processing [102], image editing [103, 104, 105, 106], inpainting [107, 108, 109, 15], and adversarial defense [110]. Recently, impressive progress has been made in improving its quality [19, 8, 11, 20], controllability [111, 103, 112, 113, 114, 104], and efficiency [115, 24, 28, 116]. In this paper, we aim to enlarge the kind of data that diffusion models can generate.

Large content generation Generating large content with generative models trained on small pieces of large content has been explored by prior works. One class of methods relies on latent variable models, *e.g.* , GANs [17], that map a global latent code and a spatial latent code to an output image. The global latent code represents the holistic appearance of the image and the spatial latent code is typically computed from a coordinate system. Some works [117, 118] generate different patches using the same global code and merge them to obtain the full image. A discriminator can be used to ensure the coherence of the full image. Instead of generating patches independently, some recent works generate the full image in one shot using architectures that guarantee translation equivariance, such as padding-free generators [119, 120, 121] or implicit MLP-based generators [122, 123, 124].

Another popular approach to generating large content is to autoregressively apply "outpainting" to gradually enlarge the content. The outpainting could be implemented by a diffusion model [95, 107, 125, 108, 126, 127], an autoregressive transformer [96, 128, 129], or a masked transformer [130, 129, 131].

4.3 Preliminaries

Diffusion models consist of two processes: a forward diffusion process and a reverse process. The forward diffusion process progressively injects Gaussian noise into samples from the data distribution $q_0(\boldsymbol{u}_0)$ and results in a family of noised data distributions $q_t(\boldsymbol{u}_t)$. It can be shown that the distribution of $\boldsymbol{u}_t$ conditioned on the clean data $\boldsymbol{u}_0$ is also Gaussian: $q_{0t}(\boldsymbol{u}_t|\boldsymbol{u}_0) = \mathcal{N}(\boldsymbol{u}_0, \sigma_t^2 \boldsymbol{I})$. The standard deviation σ_t monotonically increases with respect to the forward diffusion time t. The reverse process is designed to iteratively remove the noise from the noised data to recover the clean data, which can be formulated as the following stochastic differential equation (SDE) [132, 28, 37]

$$d\boldsymbol{u} = -(1+\eta^2)\dot{\sigma}_t\sigma_t\nabla_{\boldsymbol{u}}\log q_t(\boldsymbol{u})dt + \eta\sqrt{2\dot{\sigma}_t\sigma_t}d\boldsymbol{w}, \tag{4.1}$$

where $\nabla_{\boldsymbol{u}}\log q_t(\boldsymbol{u})$ is the score function of a noised data distribution, $\boldsymbol{w}_t$ is the standard Wiener process, and $\eta \geq 0$ determines the amount of random noise injected during the denoising process. When $\eta = 1$, Eq. (4.1) is known as reverse-time SDE of the forward diffusion process[4, 48], from which ancestral sampling and samplers based on Euler-Maruyama can be employed [4, 45]. Eq. (4.1) reduces to a probability flow ODE when $\eta = 0$ [4]. In practice, the unknown score function $\nabla_{\boldsymbol{u}}\log q_t(\boldsymbol{u})$ is estimated using a neural network $\boldsymbol{s}_\theta(\boldsymbol{u}, t)$ by minimizing a weighted sum of denoising autoencoder (score matching [50]) objectives:

$$\arg\min_\theta \mathbb{E}_{t,\boldsymbol{u}_0}[\omega(t)\left\|\nabla_{\boldsymbol{u}_t}\log q_{0t}(\boldsymbol{u}_t|\boldsymbol{u}_0) - \boldsymbol{s}_\theta(\boldsymbol{u}_t, t)\right\|^2], \tag{4.2}$$

where $\omega(t)$ denotes a time-dependent weight.

4.4 Diffusion Collage

`DiffCollage` is an algorithm that can generate large content in parallel using diffusion models trained on data consisting of portions of the large content. We first introduce the data representation and then discuss training and sampling. For simplicity, we derive the formulation for unconditional synthesis throughout this section; the formulation can be easily extended to conditional synthesis.

4.4.1 Representation

A simple example A simple use case of `DiffCollage` is to generate a long image by assembling diffusion models trained on shorter images. An autoregressive solution to this problem is to first generate an initial square image and then perform outpainting condi-

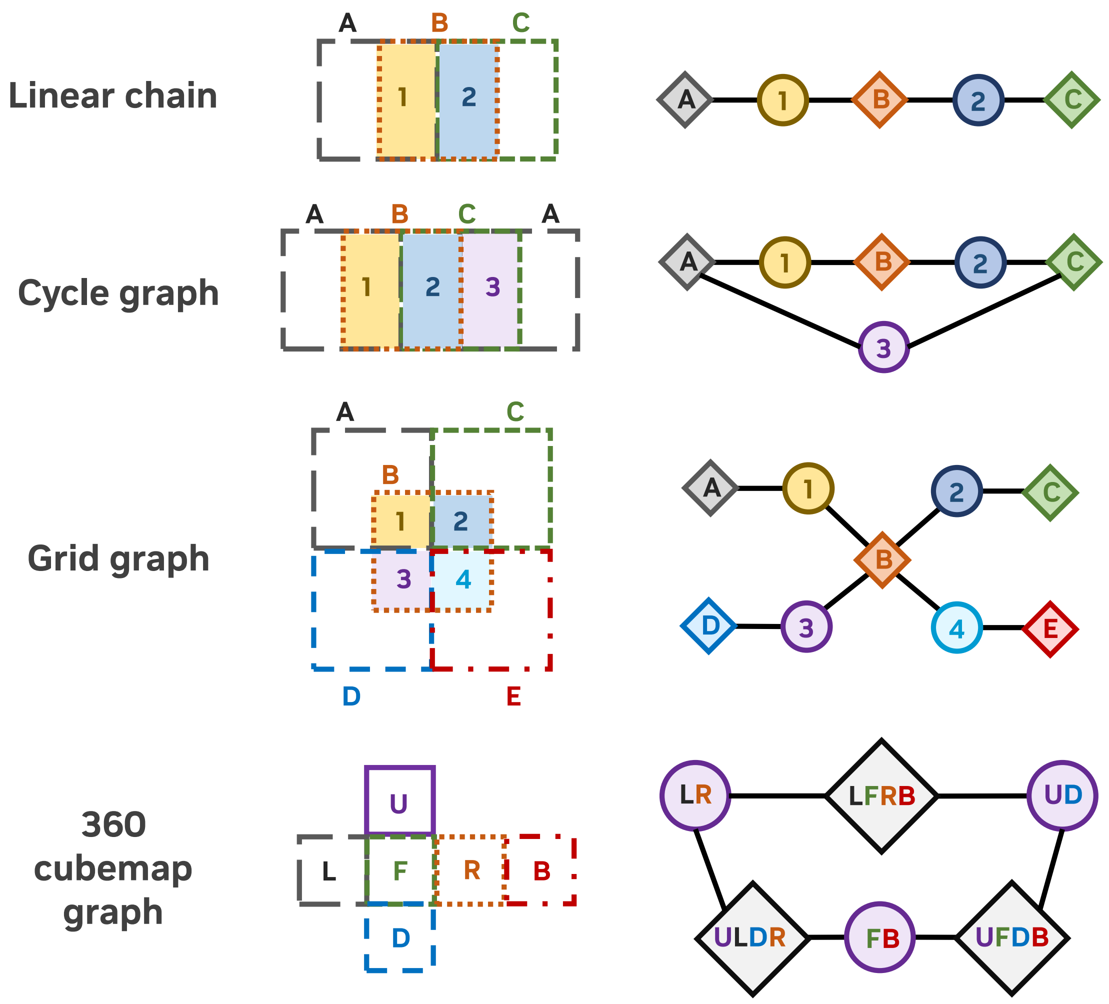
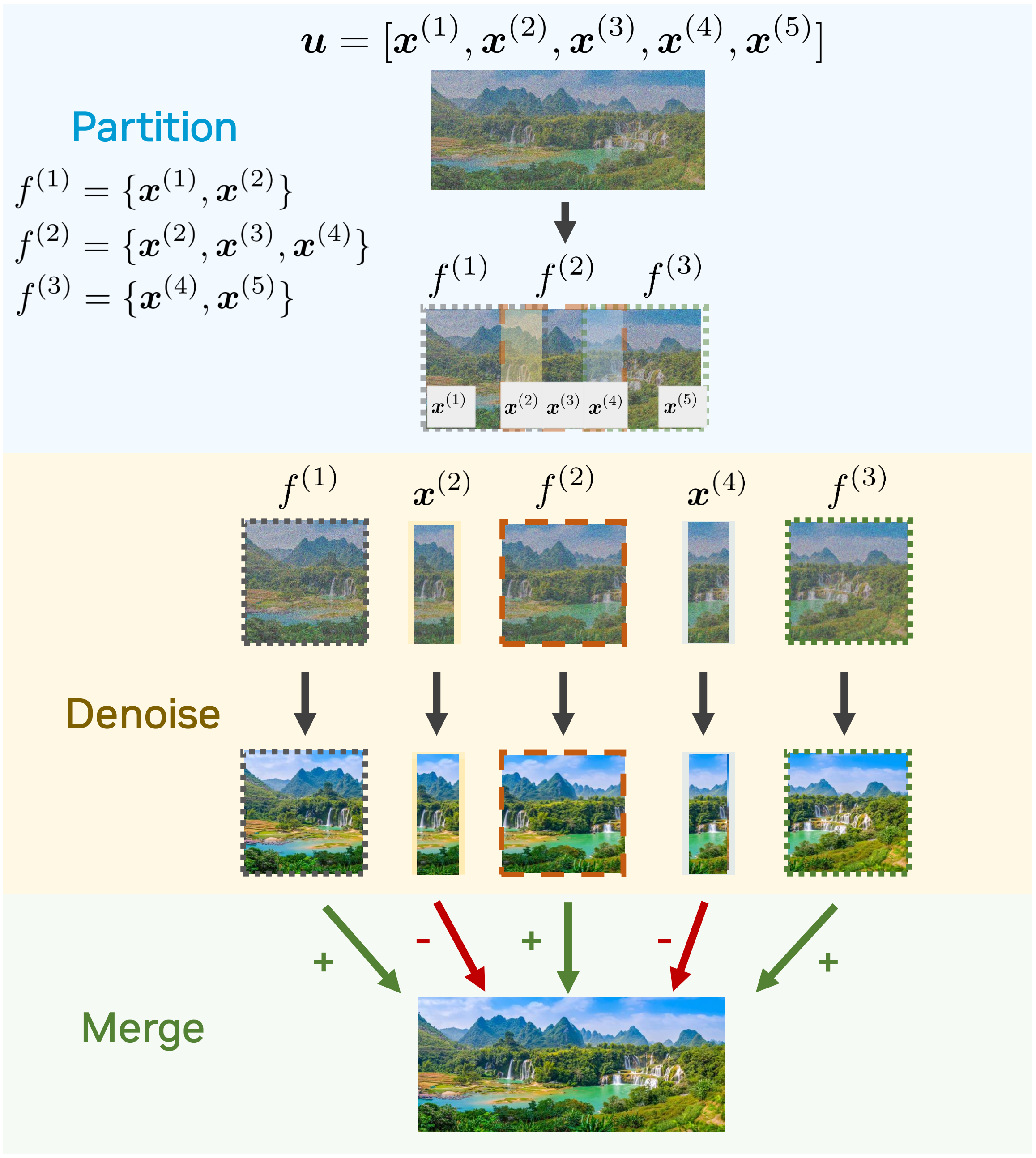


Figure 4.2: **Factor graphs for various applications.** From top to bottom: a linear chain for arbitrarily long sequences, a cycle graph for arbitrarily long loops, a grid graph for images of arbitrary height and width, and a complex factor graph for 360-degree panoramas.



$u = [x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}]$
Partition
$f^{(1)} = \{x^{(1)}, x^{(2)}\}$
$f^{(2)} = \{x^{(2)}, x^{(3)}, x^{(4)}\}$
$f^{(3)} = \{x^{(4)}, x^{(5)}\}$
$f^{(1)}$ $f^{(2)}$ $f^{(3)}$
$x^{(1)}$ $x^{(2)}$ $x^{(3)}$ $x^{(4)}$ $x^{(5)}$
Denoise
$f^{(1)}$ $x^{(2)}$ $f^{(2)}$ $x^{(4)}$ $f^{(3)}$
Merge
+
-
+
-
+

Figure 4.3: How `DiffCollage` synthesize long images. To calculate the score for each denoising step on the long image, we split the input based on the factor graph into regions of factor nodes and variables. We obtain the scores of individual regions by using the individual diffusion models. We then merge the scores to compute the score of the target diffusion model on the long image.

tioned on a part of the previously generated image [95], which results in a slightly larger output image. We denote this larger image as $\boldsymbol{u} = [\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}]$ where $[\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}]$ is the initial image and $\boldsymbol{x}^{(3)}$ is the outpainted region generated by the conditional model $\boldsymbol{x}^{(3)}|\boldsymbol{x}^{(2)}$. Notably, this procedure makes a conditional independence assumption: conditioned on $\boldsymbol{x}^{(2)}$, $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(3)}$ are independent, *i.e.*, $q(\boldsymbol{x}^{(3)}|\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) = q(\boldsymbol{x}^{(3)}|\boldsymbol{x}^{(2)})$. Therefore, the joint probability is

$$\begin{aligned} q(\boldsymbol{u}) &= q(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}) = q(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) q(\boldsymbol{x}^{(3)}|\boldsymbol{x}^{(2)}) \\ &= \frac{q(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) q(\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)})}{q(\boldsymbol{x}^{(2)})} \,. \end{aligned} \tag{4.3}$$

The score function of $q(\boldsymbol{u})$ can be represented as a sum over the scores of smaller images

$$\begin{aligned} \nabla \log q(\boldsymbol{u}) = \nabla \log q(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) &+ \nabla \log q(\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}) \\ &- \nabla \log q(\boldsymbol{x}^{(2)}) \,. \end{aligned} \tag{4.4}$$

Each individual score can be estimated using a diffusion model trained on smaller images. Unlike the autoregressive method, which generates content sequentially, `DiffCollage` can generate different pieces in parallel since all individual scores can be computed independently.

Generalization to arbitrary factor graphs Now, we generalize the above example to more complex scenarios. For a joint variable $\boldsymbol{u} = [\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(n)}]$, a factor graph [133] is a bipartite graph connecting variable nodes $\{\boldsymbol{x}^{(i)}\}_{i=1}^{n}$ and factor nodes $\{f^{(j)}\}_{j=1}^{m}$, where $f^{(j)} \subseteq \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(n)}\}$. An undirected edge between $\boldsymbol{x}^{(i)}$ and $f^{(j)}$ exists if and only if $\boldsymbol{x}^{(i)} \in f^{(j)}$. In the above example, there are two factors $f^{(1)} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\}$ and $f^{(2)} = \{\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}\}$. Given a factor graph that represents the factorization of the joint distribution $q(\boldsymbol{u})$[1], `DiffCollage` approximates the distribution as follows:

$$p(\boldsymbol{u}) := \frac{\prod_{j=1}^{m} q(f^{(j)})}{\prod_{i=1}^{n} q(\boldsymbol{x}^{(i)})^{d_i - 1}}\,, \tag{4.5}$$

where d_i is the degree of the variable node $\boldsymbol{x}^{(i)}$. It is easy to verify that Eq. (4.5) reduces to Eq. (4.3) in the simple case since the nodes for $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(3)}$ have a degree of one (connected to $f^{(1)}$ and $f^{(2)}$ respectively) and the node for $\boldsymbol{x}^{(2)}$ has a degree of two (connected to both $f^{(1)}$ and $f^{(2)}$). Similar to Eq. (4.4), we can approximate the score of $q(\boldsymbol{u})$ by adding the scores over factor nodes (*i.e.*, $q(f^{(j)})$) and subtracting the scores over non-leaf variable nodes (*i.e.*, $q(\boldsymbol{x}^{(i)})$)

$$\nabla \log p(\boldsymbol{u}) := \sum_{j=1}^{m} \nabla \log q(f^{(j)}) + \sum_{i=1}^{n} (1 - d_i) \nabla \log q(\boldsymbol{x}^{(i)})\,. \tag{4.6}$$

In fact, Eq. (4.5) is also known in the probabilistic graphical model literature as the seminal *Bethe approximation*, which approximates the joint distribution $q(\boldsymbol{u})$ by its marginals defined over factor and variable nodes [134, 135]. The approximation is exact, *i.e.*, $p(\boldsymbol{u}) = q(\boldsymbol{u})$, when the factor graph is an acyclic graph. For a general graph with cycles, the Bethe approximation is widely used in practice and obtains good performance [136, 133].

In practice, factor graphs are general enough to cover contents of arbitrary size and shape, such as those in Fig. 4.2:

- **An arbitrarily long sequence**: The factor graph is a linear chain in which each factor is connected to two variables, and each variable is connected to two factors (except for leaf variables). We show a detailed characterization in Fig. 4.3.
- **An arbitrarily long sequence with a loop**: Similar to the linear chain but with a variable node connecting the head and tail factor nodes.
- **An image of arbitrary height and width**: Here, each factor is an image patch

[1] In order words, the joint distribution can be written as a product of functions, each of which is a function of a single factor.

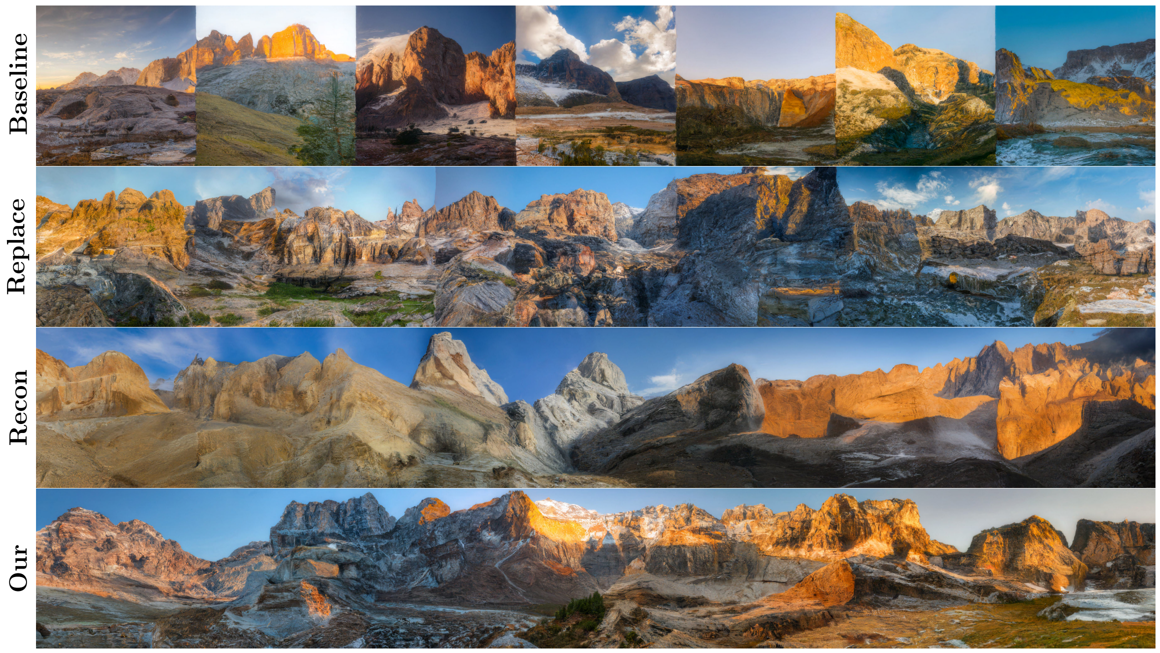


Figure 4.4: Long images generated by various approaches that only use diffusion models trained on smaller square images. For autoregressive approaches (Replace and Recon), we first generate the image in the middle then outpaint towards left and right. Replace and Recon introduce discontinuity artifacts while `DiffCollage` can generate high-fidelity images in parallel.

that overlaps with 4 other factors at the 4 corners. Each overlapping region is a variable. Thus, each factor node is connected to 4 variables, and each variables node is connected to 2 factors (except for edge cases).

- **A 360-degree image represented as a cubemap**: A cube consists of 6 faces: **F**ront, **B**ack, **L**eft, **R**ight, **U**p, **D**own. There are three cycles (LFRB, ULDR, UFDB) can be modeled via the cycle graph, and these cycles overlap one another by two faces. Intuitively, we can treat these cycles as factors and faces as variables.

4.4.2 Training and Sampling

Training `DiffCollage` is trained to estimate the score of noised data distributions $q_t(\boldsymbol{u})$. Similar to the Bethe approximation (Eqs. (4.5) and (4.6)) for clean data, we factorize the score of the time-dependent noised data distributions:

$$\nabla \log p_\theta(\boldsymbol{u}, t) = \sum_{j=1}^{m} \nabla \log p_\theta(f^{(j)}, t) + \sum_{i=1}^{n} (1 - d_i) \nabla \log p_\theta(\boldsymbol{x}^{(i)}, t). \tag{4.7}$$

To close the gap between our learned model and the Bethe approximation in Eq. (4.5), we optimize θ by performing denoising score matching between the marginal scores of real data $\{q(\boldsymbol{x}^{(i)}, t), q(f^{(j)}, t)\}$ and learned marginal scores $\{p_\theta(\boldsymbol{x}^{(i)}, t), p_\theta(f^{(j)}, t)\}$ (following Eq. (4.2)). This can be done by learning a diffusion model for each marginal distribution of real data; we list the detailed algorithm for training in the supplementary material.

It should be noted that even though we aim to approximate one joint distribution, learning a diffusion model for one marginal distribution is independent of learning other marginals. With such independence, diffusion models on different marginals can be learned in parallel. Practically, diffusion models on different marginals can be amortized where we employ one shared diffusion model with conditional signals $\boldsymbol{y}[f^{(j)}]$ from factor node $f^{(j)}$ and $\boldsymbol{y}[i]$ from variable node $\boldsymbol{x}^{(i)}$ to learn various marginals.

Sampling After training the diffusion models for each marginal, the score model of `DiffCollage` for $p_\theta(\boldsymbol{u}, t)$ is simply obtained via Eq. (4.7), and it is a diffusion model with a specific score approximation. Thus `DiffCollage` is sampler-agnostic, and we can leverage existing solvers for Eq. (4.1) to generate samples with the approximated score in Eq. (4.7), such as DDIM [115], DEIS [24], DPM-Solver [137] and gDDIM [28], all without any modifications. We emphasize that diffuson models on various marginals can be evaluated at the same time and generate different pieces of data $\{f^{(j)}, \boldsymbol{x}^{(i)}\}$ in parallel, unlike the conventional autoregressive approaches, so with advanced samplers, the number of iterations taken by `DiffCollage` could be much less than that of an autoregressive model.

4.5 Experiments

Here, we present quantitative and qualitative results to show the effectiveness and efficiency of `DiffCollage`. We perform experiments on various generation tasks, such as infinite image generation (Sec. 4.5.1), arbitrary-sized image translation (Sec. 4.5.2), motion synthesis (Sec. 4.5.3), and 360-degree panorama generation (Sec. 4.5.4).

4.5.1 Infinite image generation

We first evaluate `DiffCollage` in the infinite image generation task [118] where the goal is to generate images extended to infinity horizontally. We employ a linear chain as shown in Fig. 4.2 and use **the same** score network for all factor nodes and variable nodes since the marginal image distribution is shift-invariant.

We finetune a pre-trained GLIDE model [10], which is a two-stage diffusion model consisting of a 64×64 square generator and one $64 \rightarrow 256$ upsampler on an internal landscape dataset. We additionally finetune a pre-trained eDiff-I [20] $256 \rightarrow 1024$ upsampler. Combining them together, we have a score model for individual nodes that can generate images of resolution up to 1024×1024. To control the style of the output [20, 11], the base diffusion model is conditioned on CLIP [94] image embeddings.

Other diffusion-based approaches tackle this problem by performing outpainting autoregressively [95]. Specifically, it generates the first image using a standard diffusion model, then extends the image through repeated application of outpainting toward left and right. The outpainting problem can be treated as an inpainting problem with 50% of the content masked out. While there exist diffusion models specifically trained for inpainting [95], we only perform comparisons with other generic methods that work on any pre-trained diffusion models. We compare `DiffCollage` with two inpainting approaches. The first one is the "**replacement**" approach, where we constantly replace part of intermediate predictions with known pixels [107, 108, 109]. The second is the "**reconstruction**" approach, which uses the gradient of the reconstruction loss on known pixels to correct the unconditional samples [126, 127, 138]. This approach is slightly more computationally expensive since it needs to compute the gradient of the reconstruction loss w.r.t the intermediate predictions. We discuss more details in the supplementary.

To compare the generation quality of generated panorama images, we propose *FID Plus (FID+)*. We first generate 50k panorama images with spatial ratio $W/H = 6$ and randomly crop one square image $H \times H$ for each image. FID+ is the Frechet inception distance (FID, [139]) of the 50k randomly cropped images. We also include a **baseline** that naively concatenates independently generated images of $H \times H$ into a long image. Although each generated image is realistic, this approach has a bad FID+ because randomly cropped images may contain clear boundaries.

As shown in Tab. 4.1, `DiffCollage` outperforms other approaches in terms of sample quality evaluated by FID+. In Fig. 4.4, we show that the sample quality of autoregressive approaches deteriorates as the image grows due to error accumulation, while `DiffCollage` does not have this issue. Fig. 4.5 compares the latency of generating one panorama image with different image sizes. Thanks to its parallelization, `DiffCollage` is about $H/2W$ times and H/W times faster than replacement and reconstruction methods respectively when generating one $H \times W$ image with $H \geq 2W$. We further apply `DiffCollage` to eDiff-I [20], a recent large-scale text-to-image model in Fig. 4.1, to generate a wide image from the text prompt *"Cute Corgis at Da-Vinci Last Supper"*, which demonstrates the general applicability of `DiffCollage` to arbitrary diffusion models.

In addition, the score models for different nodes can be conditioned on different control signals. We illustrate this point by connecting any two landscape images of different styles. This is a challenging inpainting task where only pixels at two ends are given. The score models for intermediate nodes are conditioned on interpolated CLIP embeddings. As shown in Fig. 4.6, we are able to generate a long image that transitions from one style to another totally different one.

Table 4.1: Comaprison among diffusion-based methods for infinite image generation on an internal landscape dataset. Our method achieves the best image quality while also being the fastest since we can compute individual scores in parallel and do not require backpropagating through the diffusion model to obtain gradients.

Algo	Parallel	Gradients	FID+ ↓	Time ↓
Baseline	-	-	24.15	5.61
Replacement	No	Not required	10.25	14.99
Reconstruction	No	Required	8.97	26.43
Ours	Yes	Not required	**4.54**	**6.47**

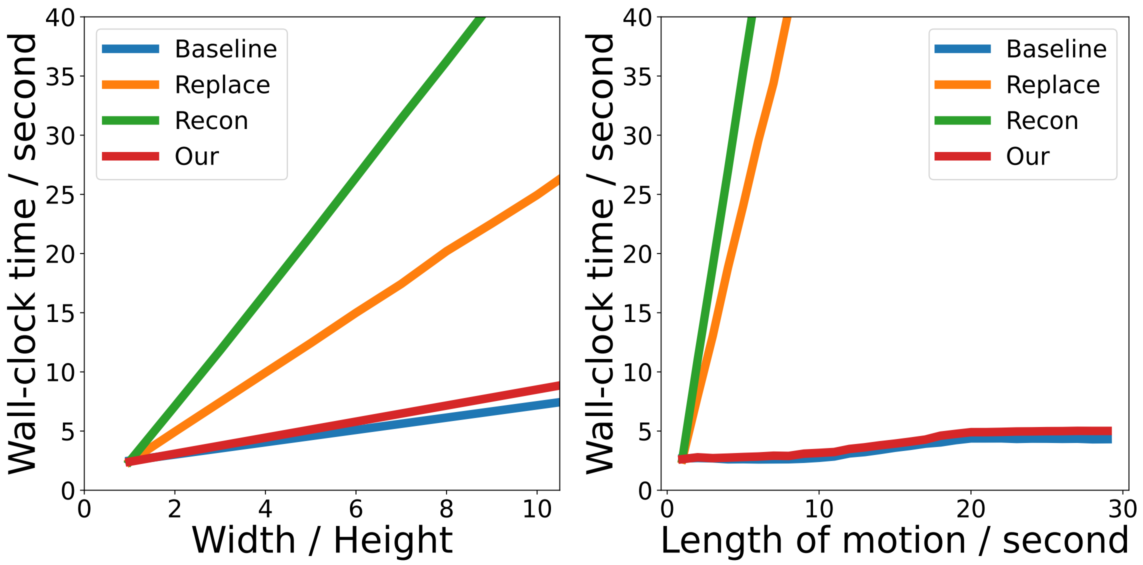


Figure 4.5: Wall-clock time for generating images with various lengths (Left) and motion sequences with various durations (Right).

We compare DiffCollage with other methods specifically designed for long image generation tasks on LHQ [118] and LSUN Tower [140], following the setting in Skorokhodov *et al.* [118]. We evaluate standard FID over images of size $H \times H$, as well as FID+ in Tab. 4.2. As shown in the table, even though our approach is never trained on long image generation, it achieves competitive results compared with methods that are tailored to the task and require problem-specific networks for the image dataset.

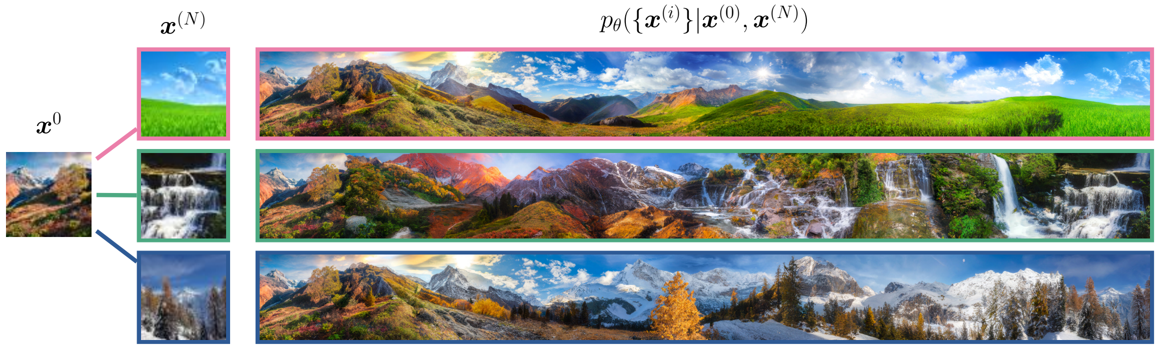

$\boldsymbol{x}^{(N)}$
$p_\theta(\{\boldsymbol{x}^{(i)}\}|\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(N)})$
$\boldsymbol{x}^0$

Figure 4.6: **Connecting real images.** Given a pair of 64×64 real images $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(N)}$, `DiffCollage` can generate a 1024×10752 image that transitions naturally from $\boldsymbol{x}^{(0)}$ into $\boldsymbol{x}^{(N)}$.

Table 4.2: Comparison against methods specifically designed for infinity image generation (Dark-colored rows). Our approach achieves higher quality despite being more general.

Dataset	LHQ 256^2		Tower 256^2	
	FID	FID+	FID	FID+
VQGAN [96]	58.27	62.12	45.18	47.32
ALIS [118]	12.60	14.27	11.85	15.27
Replacement	6.28	28.94	7.15	30.19
Reconstruction	6.28	18.37	7.15	19.56
Ours	6.28	**16.43**	7.15	**13.27**

4.5.2 Arbitrary-sized image translation

Our method can be applied to various image translation tasks where the size of the input image is different from what the diffusion model is trained on. We use `DiffCollage` to aggregate the scores of individual nodes and the score of each node can be estimated using methods that are developed for standard diffusion models, such as replacement [107, 125, 108] or reconstruction methods [126, 127] for inpainting, and SDEdit [103] for stroke-based image synthesis. To achieve these with existing methods, one could also split the large image into several smaller ones and apply the conditional generation methods independently. However, this fails to model the interactions between the split images, resulting in discontinuities in the final image; we illustrate this in Fig. 4.7 for the task of inpainting from sparse pixels. In contrast, `DiffCollage` can capture global information and

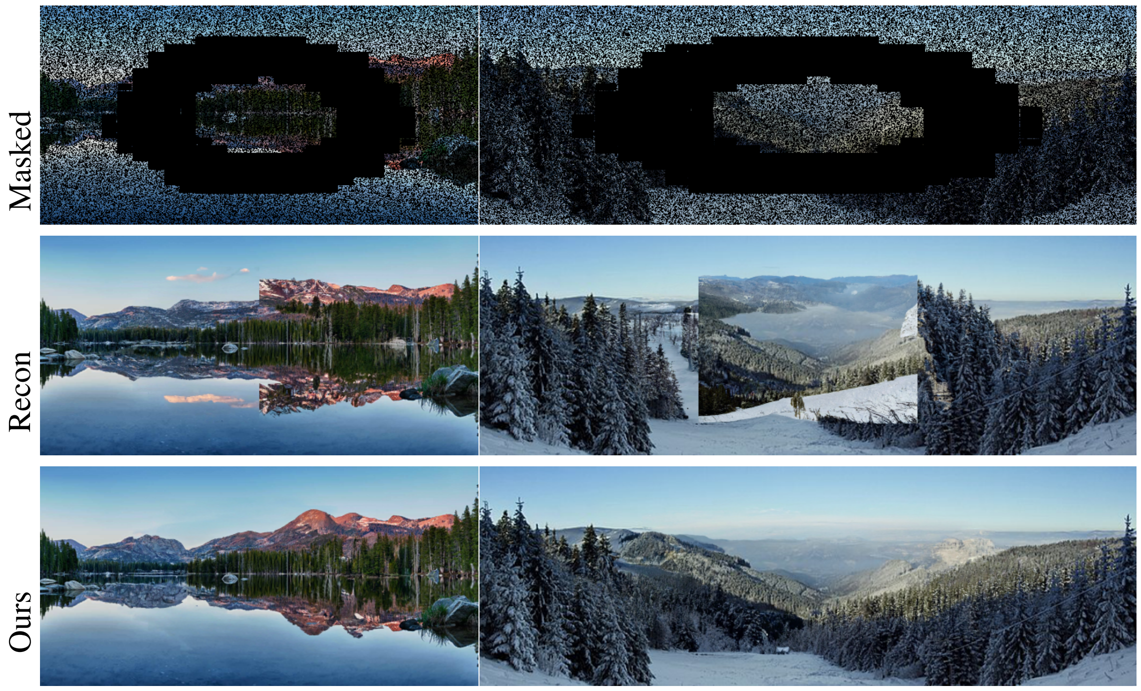


Figure 4.7: **Inpainting on non-square images.** The first row contains two masked images. The second row contains the inpainting results by splitting the input non-square images into a set of square images. In the left/right example, the input image is split into two/three square images. The Recon results in apparent boundary artifacts. The third row contains the inpainting results with `DiffCollage`.

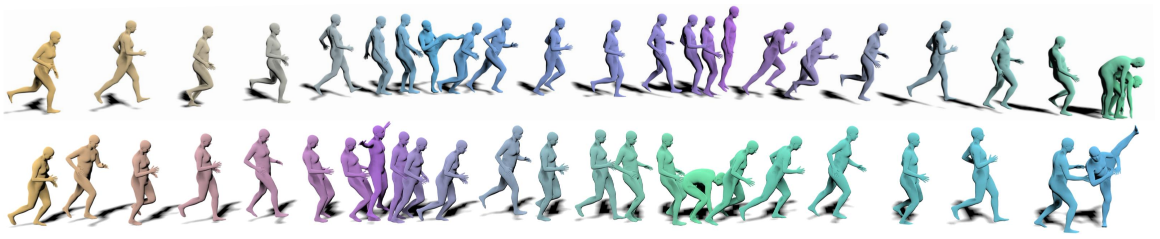


Figure 4.8: Complex motions synthesis. Though the pre-trained motion diffusion model [141] can only generate simple motions with one or two actions, `DiffCollage` can extend it to synthesize long sequences with an arbitrary number of actions. Prompts: (Top) *A person runs forward, then kicks his legs, then skips rope, then bends down to pick something up off the ground.* (Bottom) *A person runs forward, then skips rope, then bends down to pick something up off the ground, then kicks his legs.*

faithfully recover images from given sparse pixels.

4.5.3 Text-to-motion generation

Given a text description of the desired motion, the goal of this task is to synthesize a motion sequence corresponding to the description. In this section, we evaluate our approach on the popular benchmark HumanML3D [142, 143], using a pre-trained motion diffusion model [141]. The pre-trained diffusion model was trained with sequences of various lengths. As a result, it can use be used as the score estimator for both factor nodes and variable nodes in our formulation directly.

High-fidelity generated samples are expected to follow basic rules of physics and behave similarly to realistic human motions. Specifically, we adopt the set of metrics from Guo *et al.* [142], including *R-precision* and *Multimodal-Distance* that quantify the alignment between generated samples and the given prompt, *FID* that measures the distance between the distribution of ground truth motions and generated motions, and *Diversity* that measures the variability in samples generated by our methods.

Long-duration motion generation In HumanML3D, the average motion length is 7.1s, and the maximum duration is 10s. Our goal is to generate high-fidelity motion sequences that are much longer than what we have in the training data. To achieve this, we use a linear chain graph similar to the one used in infinite image generation. To evaluate our method, we generate a 24s motion for each text and randomly crop generated sequences, analogous to FID+ for images.

We compare our approach with several methods, including naively denoising a long sequence (Baseline) and autoregressive generation with replacement and reconstruction methods, respectively. The results in Tab. 4.3 show that `DiffCollage` outperforms other approaches in all evaluated metrics by a notable amount.

Compositing multiple actions The existing human motion generative model can only synthesize simple motions since there are only one or two actions for one motion sequence in the training dataset. With `DiffCollage`, we can augment the simple motion diffusion model with the ability to synthesize complex actions. We use the desired text prompts for

Table 4.3: Quantitative results of long-duration generation on the HumanML3D test set [142]. Dark-colored rows are the results of short-duration motion samples (for reference only), while other rows evaluate methods that generate 24 seconds of motion, which is around 4 times longer than the average length of training data. All methods are based on pre-trained MDM [141]. $\rightarrow$ means the results are better if the metric is closer to real data.

Method	R Precision (top 3)$\uparrow$	FID$\downarrow$	Multimodal Dist$\downarrow$	Diversity$\rightarrow$
Real data	0.798	0.001	2.960	9.471
MDM [141]	0.605	0.492	5.607	9.383
Baseline	0.298	10.690	7.512	6.764
Replacement	0.567	1.281	5.751	9.184
Reconstruction	0.585	1.012	5.716	9.175
Ours	**0.611**	**0.605**	**5.569**	**9.372**

the conditions of factors $\boldsymbol{y}[f_j]$, and the unconditional null token for that of variables $\boldsymbol{y}[i]$. As shown in Fig. 4.8, by constructing graphs with different marginal distributions specified by different conditions $\boldsymbol{y}[f_j], \boldsymbol{y}[i]$, we can generate complex motion sequences.

4.5.4 Generation with complex graphs

We further show that `DiffCollage` is able to generate data with a challenging dependency structure specified by a complex graph (such as the ones in Fig. 4.9). As shown in Fig. 4.9 (top), `DiffCollage` can generate a horizontal panorama by constructing a cycle graph. We also apply our method to generate a 360-degree panorama using a diffusion model trained only on normal perspective images conditioned on semantic segmentation maps (Fig. 4.9 bottom). This allows users to create beautiful panoramas from simple doodles, similar to some existing applications such as GauGAN [144] and GauGAN2 [145] but providing a more immersive experience to users.

4.6 Conclusion

In this work, we propose `DiffCollage`, a novel diffusion model that can synthesize large content via a collection of diffusion models trained on pieces of large content. `DiffCollage` is based on factor graph representation and inspired by Bethe approximation, both com-

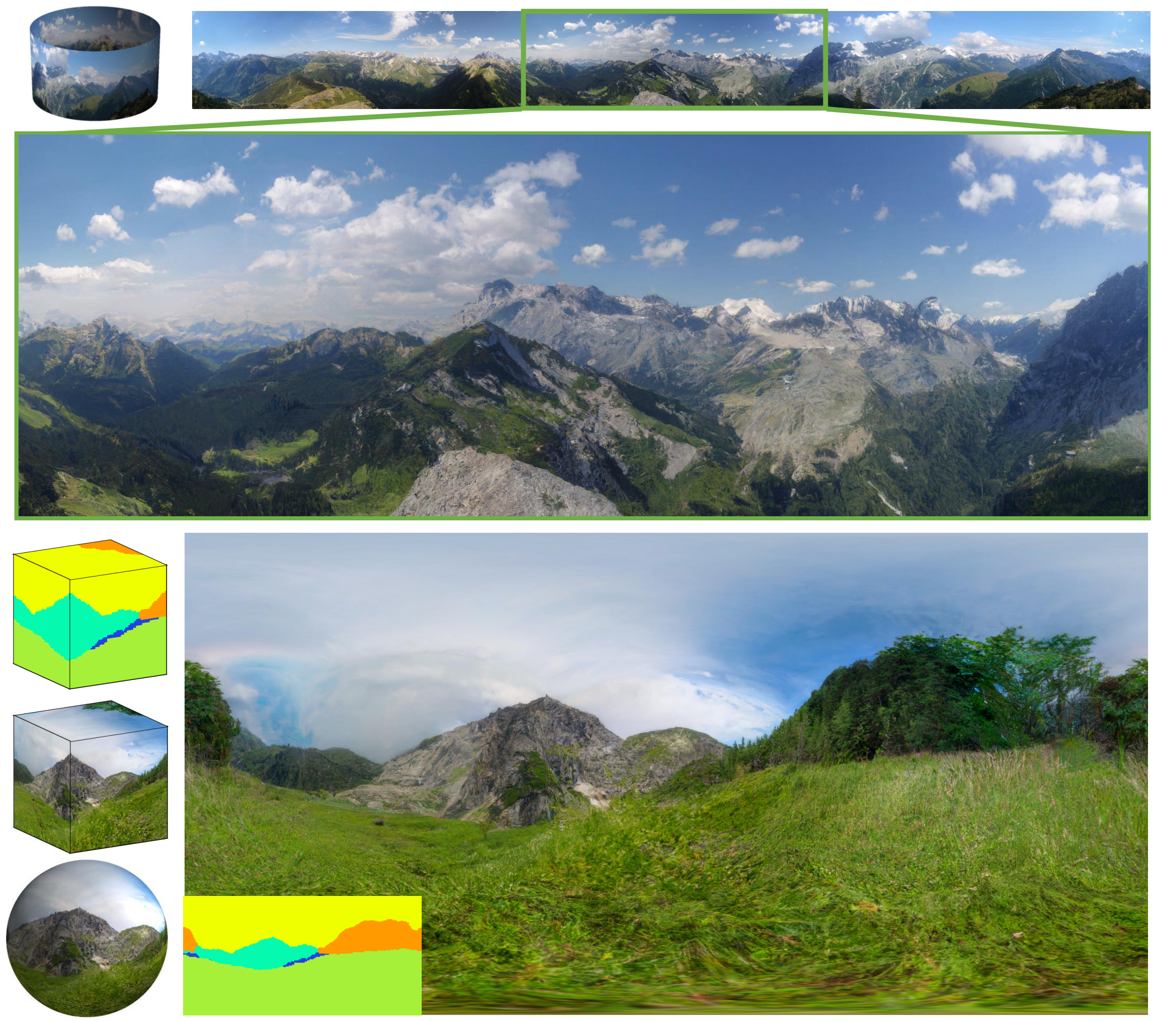


Figure 4.9: Top: a 1024×10240 horizontal panorama image. Bottom left: spherical/cube map representations of an input segmentation map and the output 360-degree panorama image. Each face of the cube is of size 1024×1024. Bottom right: equirectangular representation of the input segmentation and the output image.

monly used in probabilistic graphical models. `DiffCollage` is scalable; it allows different diffusion models trained only with samples from marginal distributions instead of joint data distribution, which are easier to obtain. `DiffCollage` is efficient; diffusion models for different marginals can be trained and sampled in parallel. Through `DiffCollage`, we enable large content generation with diffusion models.

CHAPTER 5
LEARNING NEURAL SDES WITH UNNORMALIZED DENSITY

5.1 Introduction

We are interested in drawing samples from a target density $\hat{\mu} = Z\mu$ known up to a normalizing constant Z. Although it has been widely studied in machine learning and statistics, generating asymptotically unbiased samples from such unnormalized distribution can still be challenging [34]. In practice, variational inference (VI) and Monte Carlo (MC) methods are two popular frameworks for sampling.

Variational inference employs a density model q, from which samples are easy and efficient to draw, to approximate the target density [56, 70]. Two important ingredients for variational inference sampling include a distance metric between q and $\hat{\mu}$ to identify good q and the importance weight to account for the mismatch between the two distributions. Thus, in variational inference, one needs to access the explicit density of q, which restricts the possible parameterization of q. Indeed, explicit density models that provide samples and probability density such as Autoregressive models and normalizing flow are widely used in density estimation [146, 147]. However, such models impose special structural constraints on the representation of q. For instance, the expressive power of normalizing flows [56] is constrained by the requirements that the induced map has to be bijective and its Jacobian needs to be easy-to-compute [64, 63, 5].

Most MC methods generate samples by iteratively simulating a well-designed Markov chain (MCMC) or sampling ancestrally [148]. Among them, Sequential Monte Carlo and its variants augmented with annealing trick are regarded as state-of-the-art in certain sampling tasks [149]. Despite its popularity, MCMC methods may suffer from long mixing time. The short-run performance of MCMC can be difficult to analyze and samples often get stuck in local minima [150, 151]. There are some recent works exploring the possibility of incorporating neural networks to improve MCMC [152, 153]. However, evaluating existing MCMC empirically, not to say designing an objective loss function to train network-powered MCMC, is difficult [154, 155]. Most existing works in this direction focus only on designing data-aware proposals [156, 157] and training such networks can be challenging without expertise knowledge in sampling.

In this work, we propose an efficient sampler termed Path Integral Sampler (PIS) to generate samples by simulating a stochastic differential equation (SDE) in finite steps. Our algorithm is built on the Schrödinger bridge problem [158, 159, 160, 161] whose original goal was to infer the most likely evolution of a diffusion given its marginal distributions at two time points. With a proper prior diffusion model, this Schrödinger bridge framework can be adopted for the sampling task. Moreover, it can be reformulated as a stochastic control problem [162] whose terminal cost depends on the target density $\hat{\mu}$ so that the diffusion under optimal control has terminal distribution $\hat{\mu}$. We model the control policy with a network and develop a method to train it gradually and efficiently. The discrepancy of the learned policy from the optimal policy also provides an evaluation metric for sampling performance. Furthermore, PIS can be made unbiased even with sub-optimal control policy via the path integral theorem to compute the importance weights of samples. Compared with VI that uses explicit density models, PIS uses an implicit model and has the advantage of free-form network design. The explicit density models have weaker expressive power and flexibility compared with implicit models, both theoretically and empirically [64, 87, 71, 163]. Compared with MCMC, PIS is more efficient and is able to generate high-quality samples with fewer steps. Besides, the behavior of MCMC over finite steps can be analyzed and quantified. We provide explicit sampling quality guarantee in terms of Wasserstein distance to the target density for any given sub-optimal policy.

Our algorithm is based on [39], where the authors establish the connections between generative models with latent diffusion and stochastic control and justify the expressive-

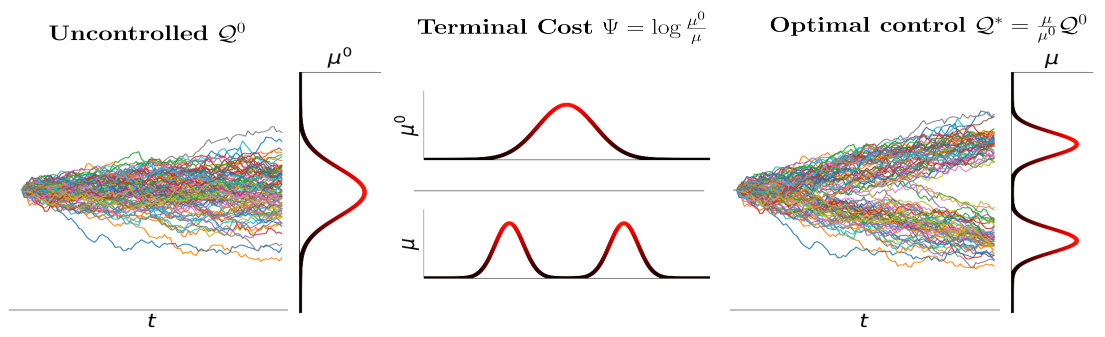


Figure 5.1: Illustration of Path Integral Sampler (PIS). The optimal policy of a specific stochastic control problem where a terminal cost function is chosen according to the given target density μ, can generate unbiased samples over a finite time horizon.

ness of such models theoretically. How to realize this model with networks and how the method performs on real datasets are unclear in [39]. Another closely related work is [70, 164], which extends Sequential Monte Carlo (SMC) by combining deterministic normalizing flow blocks with stochastic MCMC blocks. To be able to evaluate the importance weights efficiently, MCMC blocks need to be chosen based on annealed target distributions carefully. In contrast, in PIS one can design expressive architecture freely and train the model end-to-end without the burden of tuning MCMC kernels, resampling or annealing scheduling. We summarize our contributions as follows. 1). We propose Path Integral Sampler, a generic sampler that generates samples through simulating a target-dependent SDE which can be trained with free-form architecture network design. We derive performance guarantee in terms of the Wasserstein distance to the target density based on the optimality of the learned SDE. 2). An evaluation metric is provided to quantify the performance of learned PIS. By minimizing such evaluation metric, PIS can be trained end-to-end. This metric also provides an estimation of the normalization constants of target distributions. 3). PIS can generate samples without bias even with sub-optimal SDEs by assigning importance weights using path integral theory. 4). Empirically, PIS achieves the state-of-the-art sampling performance in several sampling tasks.

5.2 Sampling and stochastic control problems

We begin with a brief introduction to the sampling problem and the stochastic control problem. Throughout, we denote by $\tau = \{\mathbf{x}_t, 0 \le t \le T\}$ a continuous-time stochastic trajectory.

5.2.1 Sampling problems

We are interested in drawing samples from a target distribution $\mu(\mathbf{x}) = \hat{\mu}(\mathbf{x})/Z$ in $\boldsymbol{R}^d$ where Z is the normalization constant. Many sampling algorithms rely on constructing a stochastic process that drives the random particles from an initial distribution ν that is easy to sample from, to the target distribution μ.

In the variational inference framework, one seeks to construct a parameterized stochastic process to achieve this goal. Denote by $\Omega = C([0,T]; \boldsymbol{R}^d)$ the path space consisting of all possible trajectories and by $\mathcal{P}$ the measure over Ω induced by a stochastic process with terminal distribution μ at time T. Let $\mathcal{Q}$ be the measure induced by a parameterized stochastic and denote its marginal distribution at T by $\mu^{\mathcal{Q}}$. Then, by the data processing inequality, the Kullback-Leibler divergence (KL) between marginal distributions $\mu^{\mathcal{Q}}$ and μ can be bounded by

$$D_{\mathrm{KL}}(\mu^{\mathcal{Q}} \| \mu) \le D_{\mathrm{KL}}(\mathcal{Q} \| \mathcal{P}) := \int_{\Omega} \mathrm{d}\mathcal{Q} \log \frac{\mathrm{d}\mathcal{Q}}{\mathrm{d}\mathcal{P}}. \tag{5.1}$$

Thus, $D_{\mathrm{KL}}(\mathcal{Q} \| \mathcal{P})$ serves as a performance metric for the sampler, and a small $D_{\mathrm{KL}}(\mathcal{Q} \| \mathcal{P})$ value corresponds to a good sampler.

5.2.2 Stochastic control

Consider a model characterized by a special stochastic differential equation (SDE) [165]

$$\mathrm{d}\mathbf{x}_t = \mathbf{u}_t \mathrm{d}t + \mathrm{d}\mathbf{w}_t, \ \mathbf{x}_0 \sim \nu, \tag{5.2}$$

where $\mathbf{x}_t$, $\mathbf{u}_t$ denote state and control input respectively, and $\mathbf{w}_t$ denotes standard Brownian motion. In stochastic control, the goal is to find an feedback control strategy that minimizes a certain given cost function.

The standard stochastic control problem can be associated with any cost and any dynamics. In this work, we only consider cost of the form

$$\mathbb{E}\left[\int_0^T \frac{1}{2}\left\|\mathbf{u}_t\right\|^2 \mathrm{d}t + \Psi(\mathbf{x}_T) \mid \mathbf{x}_0 \sim \nu\right], \tag{5.3}$$

where Ψ represents the terminal cost. The corresponding optimal control problem can be solved via dynamic programming [166], which amounts to solving the Hamilton-Jacobi-Bellman (HJB) equation [167]

$$\frac{\partial V_t}{\partial t} - \frac{1}{2}\nabla V_t' \nabla V_t + \frac{1}{2}\Delta V_t = 0, \;\; V_T(\cdot) = \Psi(\cdot). \tag{5.4}$$

The space-time function $V_t(\mathbf{x})$ is known as *cost-to-go* function or *value function*. The optimal policy can be computed from $V_t(\mathbf{x})$ as [158]

$$\mathbf{u}_t^*(\mathbf{x}) = -\nabla V_t(\mathbf{x}). \tag{5.5}$$

5.3 Path Integral Sampler

It turns out that, with a proper choice of initial distribution ν and terminal loss function Ψ, the stochastic control problem coincides with sampling problem, and the optimal policy drives samples from ν to μ perfectly. The process under optimal control can be viewed as the posterior of uncontrolled dynamics conditioned on target distribution as illustrated in Fig. 5.1. Throughout, we denote by $\mathcal{Q}^u$ the path measure associated with control policy $\mathbf{u}$. We also denote by μ^0 the terminal distribution of the uncontrolled process $\mathcal{Q}^0$. For the ease of presentation, we begin with sampling from a normalized density μ, and then generalize the results to unnormalized $\hat{\mu}$ in Sec. 5.3.4.

5.3.1 Path Integral and value function

Thanks to the special cost structure, the nonlinear HJB Eq. (5.4) can be transformed into a linear partial differential equation (PDE)

$$\frac{\partial \phi_t}{\partial t} + \frac{1}{2}\Delta \phi_t = 0, \;\; \phi_T(\cdot) = \exp\{-\Psi(\cdot)\} \tag{5.6}$$

by logarithmic transformation [165] $V_t(\mathbf{x}) = -\log \phi_t(\mathbf{x})$. By the celebrated Feynman-Kac formula [168], the above has solution

$$\phi_t(\mathbf{x}) = \mathbb{E}_{\mathcal{Q}^0}[\exp(-\Psi(\mathbf{x}_T))|\mathbf{x}_t = \mathbf{x}]. \tag{5.7}$$

We remark that Eq. (5.7) implies that the optimal value function can be evaluated without knowing the optimal policy since the above expectation is with respect to the uncontrolled process $\mathcal{Q}^0$. This is exactly the Path Integral control theory [169, 170, 171]. Furthermore, the optimal control at $(t, \mathbf{x})$ is

$$\mathbf{u}_t^*(\mathbf{x}) = \nabla \log \phi_t(\mathbf{x}) = \lim_{s \searrow t} \frac{\mathbb{E}_{\mathcal{Q}^0}\{\exp\{-\Psi(\mathbf{x}_T)\} \int_t^s d\mathbf{w}_t \mid \mathbf{x}_t = \mathbf{x}\}}{(s-t)\mathbb{E}_{\mathcal{Q}^0}\{\exp\{-\Psi(\mathbf{x}_T)\} \mid \mathbf{x}_t = \mathbf{x}\}}, \tag{5.8}$$

meaning that $\mathbf{u}_t^*(\mathbf{x})$ can also be estimated by uncontrolled trajectories.

5.3.2 Sampling as a stochastic optimal control problem

There are infinite choices of control strategy $\mathbf{u}$ such that Eq. (5.2) has terminal distribution μ. We are interested in the one that minimizes the KL divergence to the prior uncontrolled process. This is exactly the Schrödinger bridge problem [158, 159, 162, 161], which has been shown to have a stochastic control formulation with cost being control efforts. In cases where ν is a Dirac distribution, it is the same as the stochastic control problem in Sec. 5.2.2 with a proper terminal cost as characterized in the following result [39].

Theorem 2. *When ν is a Dirac distribution and terminal loss is chosen as $\Psi(\mathbf{x}_T) = \log \frac{\mu^0(\mathbf{x}_T)}{\mu(\mathbf{x}_T)}$, the distribution $\mathcal{Q}^*$ induced by the optimal control policy is*

$$\mathcal{Q}^*(\tau) = \mathcal{Q}^0(\tau|\mathbf{x}_T)\mu(\mathbf{x}_T). \tag{5.9}$$

Moreover, $\mathcal{Q}^*(\mathbf{x}_T) = \mu(\mathbf{x}_T)$.

To gain more insight, consider the KL divergence

$$D_{\mathrm{KL}}(\mathcal{Q}^u(\tau)\|\mathcal{Q}^0(\tau|\mathbf{x}_T)\mu(\mathbf{x}_T)) = D_{\mathrm{KL}}(\mathcal{Q}^u(\tau)\|\mathcal{Q}^0(\tau)\frac{\mu(\mathbf{x}_T)}{\mu^0(\mathbf{x}_T)}) \tag{5.10}$$

$$= D_{\mathrm{KL}}(\mathcal{Q}^u\|\mathcal{Q}^0) + \mathbb{E}_{\mathcal{Q}^u}[\log \frac{\mu^0}{\mu}]. \tag{5.11}$$

Thanks to the Girsanov theorem [165],

$$\frac{\mathrm{d}\mathcal{Q}^u}{\mathrm{d}\mathcal{Q}^0} = \exp(\int_0^T \frac{1}{2}\|\mathbf{u}_t\|^2 \,\mathrm{d}t + \mathbf{u}_t' \mathrm{d}\mathbf{w}_t). \tag{5.12}$$

It follows that

$$D_{\mathrm{KL}}(\mathcal{Q}^u\|\mathcal{Q}^0) = \mathbb{E}_{\mathcal{Q}^u}[\int_0^T \frac{1}{2}\|\mathbf{u}_t\|^2 \,\mathrm{d}t]. \tag{5.13}$$

Plugging Eq. (5.13) into Eq. (5.10) yields

$$D_{\mathrm{KL}}(\mathcal{Q}^u(\tau)\|\mathcal{Q}^0(\tau|\mathbf{x}_T)\mu(\mathbf{x}_T)) = \mathbb{E}_{\mathcal{Q}^u}[\int_0^T \frac{1}{2}\|\mathbf{u}_t\|^2 \,\mathrm{d}t + \log \frac{\mu^0(\mathbf{x}_T)}{\mu(\mathbf{x}_T)}], \tag{5.14}$$

which is exactly the cost defined in Eq. (5.3) with $\Psi = \log \frac{\mu^0}{\mu}$. Theorem 2 implies that once the optimal control policy that minimizes this cost is found, it can also drive particles from $\mathbf{x}_0 \sim \nu$ to $\mathbf{x}_T \sim \mu$.

5.3.3 Optimal control policy and sampler

Optimal Policy Representation: Consider the sampling strategy from a given target density by simulating SDE in Eq. (5.2) under optimal control. Even though the optimal policy is characterized by Eq. (5.8), only in rare case (Gaussian target distribution) it has an analytic closed-form.

For more general target distributions, we can instead evaluate the value function Eq. (5.7) via empirical samples using Monte Carlo. The approach is essentially importance sampling whose proposal distribution is the uncontrolled dynamics. However, this approach has two drawbacks. First, it is known that the estimation variance can be intolerably high when the proposal distribution is not close enough to the target distribution [148]. Second, even if the variance is acceptable, without a good proposal, the required samples size increases exponentially with dimension, which prevents the algorithm from being used in high or even medium dimension settings [172].

To overcome the above shortcomings, we parameterize the control policy with a neural network $\mathbf{u}_\theta$. We seek a control policy that minimizes the cost

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \mathbb{E}_{Q^u}\left[\int_0^T \frac{1}{2}\left\|\mathbf{u}_t\right\|^2 \mathrm{d}t + \log\frac{\mu^0(\mathbf{x}_T)}{\mu(\mathbf{x}_T)}\right]. \tag{5.15}$$

The formula Eq. (5.15) also serves as distance metric between $\mathbf{u}_\theta$ and $\mathbf{u}^*$ as in Eq. (5.14).

Gradient-informed Policy Representation: It is believed that proper prior information can significantly boost the performance of neural network [173]. The score $\nabla \log \mu(\mathbf{x})$ has been used widely to improve the proposal distribution in MCMC [153, 174] and often leads to better results compared with proposals without gradient information. In the same spirit, we incorporate $\nabla \log \mu(\mathbf{x})$ and parameterize the policy as

$$\mathbf{u}_t(\mathbf{x}) = \mathrm{NN}_1(t, \mathbf{x}) + \mathrm{NN}_2(t) \times \nabla \log \mu(\mathbf{x}), \tag{5.16}$$

Algorithm 3 Training

```
Input: Vector: $\mathbf{x}_0 = 0$, Scalar: $y_0 = 0$
Output: $\mathbf{u}_t(\mathbf{x})$ parameterized by $\theta$
Define: SDE drift $\mathbf{f}(t, [\mathbf{x}_t, y_t]) = [\mathbf{u}_{\theta t}(\mathbf{x}_t), \frac{1}{2}\|\mathbf{u}_{\theta t}(\mathbf{x}_t)\|^2]$, diffusion $\mathbf{g}(t, [\mathbf{x}_t, y_t]) = [1, 0]$
loop epoches
    $\mathbf{x}_T, y_T$ = sdeint($\mathbf{f}, \mathbf{g}, [\mathbf{x}_0, y_0], [0, T]$)    # Integrate SDE from 0 to $T$ with Neural SDE
    Gradient descent step $\nabla_\theta [y_T + \log \frac{\mu^0(\mathbf{x}_T)}{\mu(\mathbf{x}_T)}]$    # Optimize control policy
done
```

where NN_1 and NN_2 are two neural networks. Empirically, we also found that the gradient information leads to faster convergence and smaller discrepancy $D_{\text{KL}}(\mathcal{Q}^u \| \mathcal{Q}^*)$. We remark that PIS with policy Eq. (5.16) can be viewed as a modulated Langevin dynamics [148] that achieves μ within finite time T instead of infinite time.

Optimize Policy: Optimizing $\mathbf{u}_\theta$ requires the gradient of loss in Eq. (5.15), which involves $\mathbf{u}_t$ and the terminal state $\mathbf{x}_T$. To calculate gradients, we rely on backpropagation through trajectories. We train the control policy with recent techniques of Neural SDEs [1, 44], which greatly reduce memory consumption during training. The gradient computation for Neural SDE is based on stochastic adjoint sensitivity, which generalizes the adjoint sensitivity method for Neural ODE [32]. Therefore, the backpropagation in Neural SDE is another SDE associated with adjoint states. Unlike the training of traditional deep MLPs which often runs into gradient vanishing/exploding issues, the training of Neural SDE/ODE is more stable and not sensitive the number of discretization steps [32, 44]. We augment the origin SDE with state $\int_0^t \frac{1}{2} \|\mathbf{u}_s\|^2 \, \mathrm{d}s$ such that the whole training can be conducted end to end. The full training procedure in provided in Algo 3.

Wasserstein distance bound: The PIS trained by Algo 3 can not generate unbiased samples from the target distribution μ for two reasons. First, due to the non-convexity of networks and randomness of stochastic gradient descent, there is no guarantee that the learned policy is optimal. Second, even if the learned policy is optimal, the time-discretization error in simulating SDEs is inevitable. Fortunately, the following theorem quantifies the Wasserstein distance between the sampler and the target density.

Theorem 3 (Informal)**.** *Under mild condition, with sampling step size Δt, if $\|\mathbf{u}_t^* - \mathbf{u}_t\|^2 \leq d\epsilon$ for any t, then*

$$W_2(\mathcal{Q}^u(\mathbf{x}_T), \mu(\mathbf{x}_T)) = \mathcal{O}(\sqrt{Td(\Delta t + \epsilon)}). \tag{5.17}$$

5.3.4 Importance Sampling

The training procedure for PIS does not guarantee its optimality. To compensate for the mismatch between the trained policy and the optimal policy, we introduce importance weight to calibrate generated samples. The importance weight can be calculated by

$$w^u(\tau) = \frac{\mathrm{d}\mathcal{Q}^*(\tau)}{\mathrm{d}\mathcal{Q}^u(\tau)} = \exp(\int_0^T -\frac{1}{2}\|\mathbf{u}_t\|^2 \mathrm{d}t - \mathbf{u}_t' \mathrm{d}\mathbf{w}_t - \Psi(\mathbf{x}_T)). \tag{5.18}$$

Algorithm 4 Sampling

```
**Input:** Vector: $\mathbf{x}_0 = 0$, Scalar: $y_0 = 0$
**Output:** Samples with weights
**for** $i \leftarrow 1$ to $N$ **do**
    $\Delta t = t_i - t_{i-1}, \Delta \mathbf{w} \sim \mathcal{N}(0, \Delta t \mathcal{I})$,
    $\mathbf{x}_i = \mathbf{x}_{i-1} + \mathbf{u}\Delta t + \Delta \mathbf{w}$
    $y_i = y_{i-1} + \mathbf{u}'\Delta \mathbf{w} + \frac{1}{2}\|\mathbf{u}\|^2 \Delta t$
**end for**
**Outputs:** $\mathbf{x}_N, \exp(-y_N - \log \frac{\mu^0(\mathbf{x}_N)}{\mu(\mathbf{x}_N)})$
```

We note Eq. (5.18) resembles training objective Eq. (5.15). Indeed, Eq. (5.15) is the average of logarithm of Eq. (5.18). If the trained policy is optimal, that is, $\mathcal{Q}^u = \mathcal{Q}^*$, all the particles share the same weight. We summarize the sampling algorithm in Algo 4.

Effective Sample Size: The Effective Sample Size (ESS), $\mathrm{ESS}^u = \frac{1}{\mathbb{E}_{Q^u}[(w^u)^2]}$, is a popular metric to measure the variance of importance weights. ESS is often accompanied by resampling trick [175] to mitigate deterioration of sample quality. ESS is also regarded as a metric for quantifying goodness of sampler based on importance sampling. Low ESS means that estimation or downstream tasks based on such sampling methods may suffer from a high variance. ESS of most importance samplers is decreasing along the time. Thanks to the adaptive control policy in PIS, we can quantify the ESS of PIS based on the optimality of learned policy.

Theorem 4 (Corollary 7 [171])**.** *If* $\max_{t,\mathbf{x}} \|\mathbf{u}_t(\mathbf{x}) - \mathbf{u}^*_t(\mathbf{x})\|^2 \leq \frac{\epsilon}{T}$, *then*

$$\frac{1}{\mathbb{E}_{Q^u}[(w^u)^2]} \geq 1 - \epsilon.$$

Estimation of normalization constants: In most sampling problems we only have access to the target density up to a normalization constant, denoted by $\hat{\mu} = Z\mu$. PIS can still generate samples following the same protocol with new terminal cost $\hat{\Psi} = \log \frac{\mu^0}{\hat{\mu}} = \Psi - \log Z$. The additional constant $-\log Z$ is independent of $\mathbf{x}_T$ and thus does not affect the optimal policy and the optimization of $\mathbf{u}_\theta$. As a byproduct, we can estimate the normalization constants.

Theorem 5. *For any given policy* $\mathbf{u}$, *the logarithm of normalization constant is bounded below by*

$$\mathbb{E}_{\tau\sim Q^u}[-\hat{S}^u(\tau)] \leq \log Z, \tag{5.19}$$

where $\hat{S}^u(\tau) = \int_0^T \frac{1}{2}\|\mathbf{u}_t(\mathbf{x}_t)\|^2 \mathrm{d}t + \mathbf{u}'_t(\mathbf{x}_t)\mathrm{d}\mathbf{w}_t + \hat{\Psi}(\mathbf{x}_T)$. *The equality holds only when* $\mathbf{u} = \mathbf{u}^*$. *Moreover, for any sub-optimal policy, an unbiased estimation of* Z *using importance sampling is*

$$Z = \mathbb{E}_{\tau\sim Q^u}[\exp(-\hat{S}^u(\tau))]. \tag{5.20}$$

5.4 Experiments

In this section, we present empirical evaluations of PIS and the comparisons to several baselines. We also provide details of practical implementations. Inspired by [164], we conduct experiments for tasks of Bayesian inference and normalization constant estimation.

We consider three types of relevant methods. The first category is gradient-guided MCMC methods without the annealing trick. It includes the Hamiltonian Monte Carlo (HMC) [148] and No-U-Turn Sampler (NUTS) [174]. The second is Sequential Monte Carlo with annealing trick (SMC), which is regarded as a state-of-the-art sampling algorithm [149] in terms of sampling quality. We choose a standard instance of SMC samplers and the recently proposed Annealed Flow Transport Monte Carlo (AFT) [164]. Both use default 10 temperature levels with a linear annealing scheme. We note that there are optimized SMC variants that achieve better performance [149, 176, 177]. Since the introduction of advanced tricks, we exclude the comparison with those variants for fair comparison purposes. We note PIS can also be augmented with an annealing trick, possible improvements for PIS can be explored in the future. Last, the variational normalizing flow (VI-NF) [56] is also included for comparison. We note that another popular line of sampling algorithms use Stein-Variational Gradient Descent (SVGD) or other particle-based variational inference approaches [178, 179]. We include the comparison and more discussions on SGVD due to its significant difference. In our experiments, the number of steps N of MCMC algorithms and the number of SDE time-discretization steps for PIS work as a proxy for benchmarking computation times.

We also investigate the effects of two different network architectures for Path Integral Sampler. The first one is a time-conditioned neural network without any prior information, which we denote as *PIS-NN*, while the second one incorporates the gradient information of the given energy function as in Eq. (5.16), denoted as *PIS-Grad*. When we have an analytical form for the ground truth optimal policy, the policy is denoted as *PIS-GT*. The

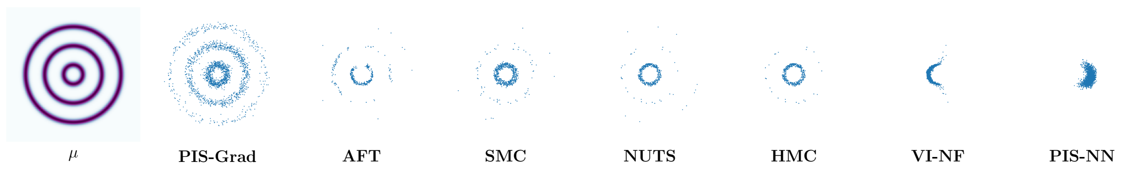


Figure 5.2: Sampling performance on rings-shape density function with 100 steps. The gradient information can help PIS-Grad and MCMC algorithm improve sampling performance.

subscript *RW* is to distinguish PIS with path integral importance weights Eq. (5.18) that use Eq. (5.20) to estimate normalization constants from the ones without importance weights that use the bound in Eq. (5.19) to estimate Z. For approaches without the annealing trick, we take default $N = 100$ unless otherwise stated. With annealing, N steps are the default for each temperature level, thus AFT and SMC rougly use 10 times more steps compared with HMC and PIS. We include more details about hyperparameters, training time, sampling efficiency, and more experiments with large N.

5.4.1 PIS-Grad vs PIS-NN: Importance of gradient guidance

We observed that the advantage of PIS-Grad over PIS-NN is clearer when the target density has multiple modes as in the toy example shown in Fig. 5.2. The objective $D_{\text{KL}}(\mathcal{Q}\|\mathcal{Q}^*)$ is known to have *zero forcing*. In particular, when the modes of the density are well separated and $\mathcal{Q}$ is not expressive enough, minimizing $D_{\text{KL}}(\mathcal{Q}\|\mathcal{Q}^*)$ can drive $\mathcal{Q}(\tau)$ to zero on some area, even if $\mathcal{Q}^*(\tau) > 0$ [180]. PIS-NN and VI-NF generate very similar samples that almost cover half the inner ring. The training objective function of VI-NF can also be viewed as minimizing KL divergence between two trajectory distributions [70]. The added noise during the process can encourage exploration but it is unlikely such noise only can overcome the local minima. On the other hand, the gradient information can help cover more modes and provide exploring directions.

5.4.2 Benchmarking datasets

Mode-separated mixture of Gaussian: We consider the mixture of Gaussian in 2-dimension. We notice that when the Gaussian modes are not far away from each other, all methods work well. However, when we reduce the variances of the Gaussian distributions and separate the modes of Gaussian, the advantage of PIS becomes clear even in this low dimension task. PIS generates samples that are visually indistinguishable from the target density.

Funnel distribution: We consider the popular testing distribution in MCMC literature [174, 181], the 10-dimensional Funnel distribution charaterized by

$$x_0 \sim \mathcal{N}(0, 9), \quad x_{1:9}|x_0 \sim \mathcal{N}(0, \exp(x_0)\mathbf{I}).$$

This distribution can be pictured as a funnel - with x_0 wide at the mouth of funnel, getting smaller as the funnel narrows.

	MG ($d = 2$)			Funnel ($d = 10$)			LGCP ($d = 1600$)		
	B	S	A	B	S	A	B	S	A
PIS$_{RW}$-GT	-0.012	0.013	0.018	-	-	-	-	-	-
PIS-NN	-1.691	0.370	1.731	-0.098	**5e-3**	0.098	-92.4	6.4	92.62
PIS-Grad	-0.440	0.024	0.441	-0.103	9e-3	0.104	-13.2	3.21	13.58
PIS$_{RW}$-NN	-1.192	0.482	1.285	-0.018	7e-3	0.02	-60.8	4.81	60.99
PIS$_{RW}$-Grad	**-0.021**	**0.030**	**0.037**	**-0.008**	9e-3	**0.012**	**-1.94**	**0.91**	**2.14**
AFT	-0.509	0.24	0.562	-0.208	0.193	0.284	-3.08	1.59	3.46
SMC	-0.362	0.293	0.466	-0.216	0.157	0.267	-435	14.7	436
NUTS	-1.871	0.527	1.943	-0.835	0.257	0.874	-1.3e3	8.01	1.3e3
HMC	-1.876	0.527	1.948	-0.835	0.257	0.874	-1.3e3	8.01	1.3e3
VI-NF	-1.632	0.965	1.896	-0.236	0.0591	0.243	-77.9	5.6	78.2

Table 5.1: Benchmarking on mode separated mixture of Gaussian (MG), Funnel distribution and Log Gaussian Cox Process (LGCP) for estimation log normalization constants. B and S stand for estimation bias and standard deviation among 100 runs and $A^2 = B^2 + S^2$.

Log Gaussian Cox Process: We further investigate the normalization constant estimation problem for the challenging log Gaussian Cox process (LGCP), which is designed for modeling the positions of Finland pine saplings. In LGCP [182], an underlying field λ of positive real values is modeled using an exponentially-transformed Gaussian process. Then

λ is used to parameterize Poisson points process to model the locations of pine saplings. The posterior density is

$$\lambda(\mathbf{x}) \sim \exp(-\frac{(\mathbf{x}-\mu)^T K^{-1}(\mathbf{x}-\mu)}{2}) \prod_{i\in d} \exp(x_i y_i - \alpha \exp x_i), \tag{5.21}$$

where d denotes the size of discretized grid and y_i denotes observation information. The modeling parameters, including normal distribution and α, follow [164].

Tab. 5.1 clearly shows the advantages of PIS for the above three datasets, and supports the claim that importance weight helps improve the estimation of log normalization constants, based on the comparison between PIS$_{RW}$ and PIS. We also found that PIS-Grad trained with gradient information outperforms PIS-NN. The difference is more obvious in datasets that have well-separated modes, such as MG and LGCP, and less obvious on unimodal distributions like Funnel.

In all cases, PIS$_{RW}$-Grad is better than AFT and SMC. Interestingly, even *without* annealing and gradient information of target density, PIS$_{RW}$-NN can outperform SMC with annealing trick and HMC kernel for the Funnel distribution.

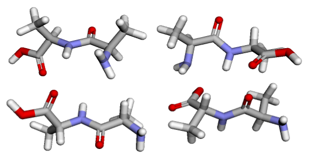
5.4.3 Advantage of the specialized sampling algorithm

From the perspective of particles dynamics, most existing MCMC algorithms are invariant to the target distribution. Therefore, particles are driven by gradient and random noise in a way that is independent of the given target distribution. In contrast, PIS learns different strategies to combine gradient information and noise for different target densities. The specialized sampling algorithm can generate samples more efficiently and shows better performance empirically in our experiments. The advantage can be showed in various datasets, from unimodal distributions like the Funnel distribution to multimodal distributions. The benefits and efficiency of PIS are more obvious in high dimensional settings as we have shown.

KL*	μ	ϕ	η_1	ψ	η_2	η_3
VI-NF	175.6 ±4.5	24.2± 4.1	3.1 ± 0.05	14.6± 6.4	7e-2±5e-3	**8.5e-2±3.5e-3**
SMC	183.3 ±2.3	18.3± 2.1	0.32 ± 0.08	9.6 ± 1.2	0.12±0.05	0.15 ± 9e-3
SNF	181.8 ±0.75	6.3± 0.71	**0.17±0.05**	1.58 ± 0.36	0.11± 0.03	8.8e-2 ±8e-3
PIS-NN	**171.3 ±0.61**	**5.2± 0.35**	0.32±0.03	**1.03 ± 0.23**	**5e-2±5e-3**	8.7e-2±3e-3

Table 5.2: KL-divergences comparison among variational approaches of generated density with target density in overall atom states distribution and five multimodal torsion angles. We emphasize KL* denote the KL divergence between unnormalized distribution due to lack of ground truth normalization constants. Mean and standard deviation are conducted with five different random seeds.

Figure 5.3: Sampled Alanine dipeptide molecules



5.4.4 Alanine dipeptide

Building on the success achieved by flow models in the generation of asymptotically unbiased samples from physics models [183], we investigate the applications in the sampling of molecular structure from a simulation of Alanine dipeptide as introduced in [70]. The target density of molecule is $\hat{\mu} = \exp(-E(\mathbf{x}_{[0:65]}) - \frac{1}{2}\left\|\mathbf{x}_{[66:131]}\right\|^2)$.

We compare PIS with popular variational approaches used in generating samples from the above model. More specifically, we consider VI-NF, and Stochastic Normalizing Flow (SNF) [70]. SNF is very close to AFT [164]. Both of them couple deterministic normalizing flow layers and MCMC blocks except SNF uses an amortized structure. We show a generated molecular in Fig. 5.3 and quantitative comparison in terms of KL divergence in Tab. 5.2, including overall atom states distribution and five multimodal torsion angles (backbone angles ϕ, ψ and methyl rotation angles η_1, η_2, η_3). We remark that unweighted samples are used to approximate the density of torsion angles and all approaches do not use gradient information. Clearly, PIS gives lower divergence.

Table 5.3: Estimation of $\log p_\theta(x)$ of a trained VAE.

	B	S	$\sqrt{\mathrm{B}^2+\mathrm{S}^2}$
VI-NF	-2.3	0.76	2.42
AFT	-1.7	0.95	1.96
SMC	-10.6	2.01	10.79
PIS$_{RW}$-NN	-1.9	0.81	2.06
PIS$_{RW}$-Grad	**-0.87**	**0.31**	**0.92**

5.4.5 Sampling in Variational Autoencoder latent space

In this experiment, we investigate sampling in the latent space of a trained Variational Autoencoder (VAE). VAE aims to minimize $D_{\mathrm{KL}}(q(\mathbf{x})q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z}))$, where $q_\phi(\mathbf{z}|\mathbf{x})$ represents encoder and p_θ for a decoder with latent variable $\mathbf{z}$ and data $\mathbf{x}$. We investigate the posterior distribution

$$\mathbf{z} \sim p(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z}). \tag{5.22}$$

The normalization constant of such target unnormalized density function $p(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z})$ is exactly the likelihood of data points $p_\theta(\mathbf{x})$, which serves as an evaluation metric for the trained VAE.

We investigate a vanilla VAE model trained with plateau loss on the binary MNIST [183] dataset. For each distribution, we regard the average estimation from 10 long-run SMC with 1000 temperature levels as the ground truth normalization constant. We choose 100 images randomly and run the various approaches on estimating normalization of those posterior distributions in Eq. (5.22) and report the average performance in Tab. 5.3. PIS has a lower bias and variance.

5.5 Conclusion

Contributions. In this work, we proposed a new sampling algorithm, Path Integral Sampler, based on the connections between sampling and stochastic control. The control can drive particles from a simple initial distribution to a target density perfectly when the policy is optimal for an optimal control problem whose terminal cost depends on the target distribution. Furthermore, we provide a calibration based on importance weights, ensuring sampling quality even with sub-optimal policies.

Limitations. Compared with most popular non-learnable MCMC algorithms, PIS requires training neural networks for the given distributions, which adds additional computational overhead, though this can be mitigated with amortization. Besides, the sampling quality of PIS in finite steps depends on the optimality of trained network. Improper choices of hyperparameters may lead to numerical issues and failure modes.

CHAPTER 6
ACCELERATING SAMPLING FOR ISOTROPIC DIFFUSION MODELS

6.1 Introduction

The Diffusion model (DM) [7] is a generative modeling method developed recently that relies on the basic idea of reversing a given simple diffusion process. A time-dependent score function is learned for this purpose and DMs are thus also known as score-based models [4]. Compared with other generative models such as generative adversarial networks (GANs), in addition to great scalability, the DM has the advantage of stable training is less hyperparameter sensitive [184, 185]. DMs have recently achieved impressive performances on a variety of tasks, including unconditional image generation [7, 4, 186, 187], text conditioned image generation [188, 11], text generation [12, 13], 3D point cloud generation [14], inverse problem [15, 16], etc.

However, the remarkable performance of DMs comes at the cost of slow sampling; it takes much longer time to produce high-quality samples compared with GANs. For instance, the Denoising Diffusion Probabilistic Model (DDPM) [7] needs 1000 steps to generate one sample and each step requires evaluating the learning neural network once; this is substantially slower than GANs [17, 18]. For this reason, there exist several studies aiming at improve the sampling speed for DMs. One category of methods modify/optimize the forward noising process such that backward denoising process can be more efficient [189, 4, 190, 191]. An important and effective instance is the Denoising Diffusion Implicit Model (DDIM) [192] that uses a non-Markovian noising process. Another category of methods speed up the numerical solver for stochastic differential equations (SDEs) or ordinary differential equations (ODEs) associated with the DMs [193, 4, 194]. In [4], blackbox ODE solvers are used to solve a marginal equivalent ODE known as the Probability Flow (PF),

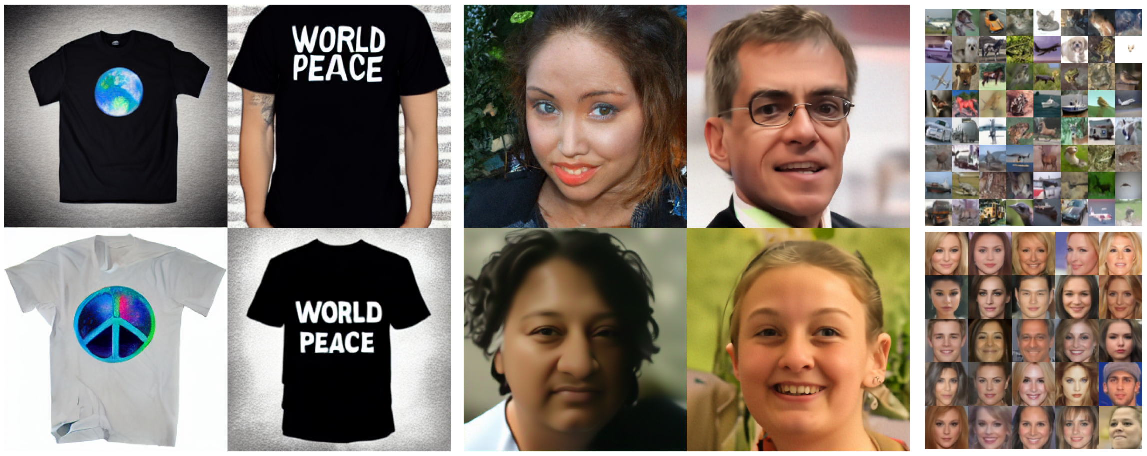


Figure 6.1: Generated images with various DMs. Latent diffusion [186] (Left), 256×256 image with text *A shirt with inscription "World peace"* (15 NFE). VE diffusion [4] (Mid), FFHQ 256×256 (12 NFE). VP diffusion [7] (Right), CIFAR10 (7 NFE) and CELEBA (5 NFE).

for fast sampling. In [195], the authors combine DDIM with high order methods to solve this ODE and achieve further acceleration. Note that the deterministic DDIM can also be viewed as a time discretization of the PF as it matches the latter in the continuous limit [192, 195]. However, it is unclear why DDIM works better than generic methods such as Euler.

The objective of this work is to establish a principled discretization scheme for the learned backward diffusion processes in DMs so as to achieve fast sampling. Since the most expensive part in sampling a DM is the evaluation of the neural network that parameterizes the backward diffusion, we seek a discretization method that requires a small number of network function evaluation (NFE). We start with a family of marginal equivalent SDEs/ODEs associated with DMs and investigate numerical error sources, which include fitting error and discretization error. We observe that even with the same trained model, different discretization schemes can have dramatically different performances in terms of discretization error. We then carry out a sequence of experiments to systematically investigate the influences of different factors on the discretization error. We find out that the *Exponential Integrator (EI)* [23] that utilizes the semilinear structure of the backward diffusion has minimum error. To further reduce the discretization error, we propose to either use high order polynomials to approximate the nonlinear term in the ODE or employ Runge

Kutta methods on a transformed ODE. The resulting algorithms, termed *Diffusion Exponential Integrator Sampler (DEIS)*, achieve the best sampling quality with limited NFEs.

Our contributions are summarized as follows: 1) We investigate a family of marginal equivalent SDEs/ODEs for fast sampling and conduct a systematic error analysis for their numerical solvers. 2) We propose DEIS, an efficient sampler that can be applied to any DMs to achieve superior sampling quality with a limited number of NFEs. DEIS can also accelerate data log-likelihood evaluation. 3) We prove that the deterministic DDIM is a special case of DEIS, justifying the effectiveness of DDIM from a discretization perspective. 4) We conduct comprehensive experiments to validate the efficacy of DEIS. For instance, with a pre-trained model [4], DEIS is able to reach 4.17 FID with 10 NFEs, and 2.86 FID with 20 NFEs on CIFAR10.

6.2 Background on Diffusion Models

A DM consists of a fixed forward diffusion (noising) process that adds noise to the data, and a learned backward diffusion (denoising) process that gradually removes the added noise. The backward diffusion is trained to match the forward one in probability law, and when this happens, one can in principle generate perfect samples from the data distribution by simulating the backward diffusion.

Forward noising diffusion: The forward diffusion of a DM for D-dimensional data is a linear diffusion described by the stochastic differential equation (SDE) [37]

$$d\boldsymbol{x} = \boldsymbol{F}_t \boldsymbol{x} dt + \boldsymbol{G}_t d\boldsymbol{w}, \tag{6.1}$$

where $\boldsymbol{F}_t \in \mathbb{R}^{D\times D}$ denotes the linear drift coefficient, $\boldsymbol{G}_t \in \mathbb{R}^{D\times D}$ denotes the diffusion coefficient, and $\boldsymbol{w}$ is a standard Wiener process. The diffusion Eq. (6.1) is initiated at the training data and simulated over a fixed time window $[0, T]$. Denote by $p_t(\boldsymbol{x}_t)$ the marginal distribution of $\boldsymbol{x}_t$ and by $p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)$ the conditional distribution from $\boldsymbol{x}_0$ to $\boldsymbol{x}_t$, then $p_0(\boldsymbol{x}_0)$

Table 6.1: Two popular SDEs, variance preserving SDE (VPSDE) and variance exploding SDE (VESDE). The parameter α_t is decreasing with $\alpha_0 \approx 1, \alpha_T \approx 0$, while σ_t is increasing.

SDE	$\boldsymbol{F}_t$	$\boldsymbol{G}_t$	μ_t	Σ_t
VPSDE [7]	$\frac{1}{2}\frac{d\log\alpha_t}{dt}\boldsymbol{I}$	$\sqrt{-\frac{d\log\alpha_t}{dt}}\boldsymbol{I}$	$\sqrt{\alpha_t}\boldsymbol{I}$	$(1-\alpha_t)\boldsymbol{I}$
VESDE [4]	0	$\sqrt{\frac{d[\sigma_t^2]}{dt}}\boldsymbol{I}$	$\boldsymbol{I}$	$\sigma_t^2\boldsymbol{I}$

represents the underlying distribution of the training data. The simulated trajectories are represented by $\{\boldsymbol{x}_t\}_{0\le t\le T}$. The parameters $\boldsymbol{F}_t$ and $\boldsymbol{G}_t$ are chosen such that the conditional marginal distribution $p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)$ is a simple Gaussian distribution, denoted as $\mathcal{N}(\mu_t\boldsymbol{x}_0, \Sigma_t)$, and the distribution $\pi(\boldsymbol{x}_T) := p_T(\boldsymbol{x}_T)$ is easy to sample from. Two popular SDEs in diffusion models [4] are summarized in Tab. 6.1. Here we use matrix notation for $\boldsymbol{F}_t$ and $\boldsymbol{G}_t$ to highlight the generality of our method. Our approach is applicable to any DMs, including the Blurring diffusion models (BDM) [25, 26] and the critically-damped Langevin diffusion (CLD) [196] where these coefficients are indeed non-diagonal matrices.

Backward denoising diffusion: Under mild assumptions [48, 4], the forward diffusion Eq. (6.1) is associated with a reverse-time diffusion process

$$d\boldsymbol{x} = [\boldsymbol{F}_t\boldsymbol{x}dt - \boldsymbol{G}_t\boldsymbol{G}_t^T\nabla\log p_t(\boldsymbol{x})]dt + \boldsymbol{G}_t d\boldsymbol{w}, \tag{6.2}$$

where $\boldsymbol{w}$ denotes a standard Wiener process in the reverse-time direction. The distribution of the trajectories of Eq. (6.2) with terminal distribution $\boldsymbol{x}_T \sim \pi$ coincides with that of Eq. (6.1) with initial distribution $\boldsymbol{x}_0 \sim p_0$, that is, Eq. (6.2) matches Eq. (6.1) in probability law. Thus, in principle, we can generate new samples from the data distribution p_0 by simulating the backward diffusion Eq. (6.2). However, to solve Eq. (6.2), we need to evaluate the score function $\nabla\log p_t(\boldsymbol{x})$, which is not accessible.

Training: The basic idea of DMs is to use a time-dependent network $\boldsymbol{s}_\theta(\boldsymbol{x}, t)$, known as a score network, to approximate the score $\nabla\log p_t(\boldsymbol{x})$. This is achieved by score matching techniques [197, 198] where the score network $\boldsymbol{s}_\theta$ is trained by minimizing the denoising score matching loss

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathrm{Unif}[0,T]}\mathbb{E}_{p(\boldsymbol{x}_0)p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)}[\|\nabla \log p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0) - \boldsymbol{s}_\theta(\boldsymbol{x}_t, t)\|^2_{\Lambda_t}]. \tag{6.3}$$

Here $\nabla \log p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)$ has a closed form expression as $p_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0)$ is a simple Gaussian distribution, and Λ_t denotes a time-dependent weight. This loss can be evaluated using empirical samples by Monte Carlo methods and thus standard stochastic optimization algorithms can be used for training. We refer the reader to [7, 4] for more details on choices of Λ_t and training techniques.

6.3 Fast Sampling with learned score models

Once the score network $\boldsymbol{s}_\theta(\boldsymbol{x}, t) \approx \nabla \log p_t(\boldsymbol{x})$ is trained, it can be used to generate new samples by solving the backward SDE Eq. (6.2) with $\nabla \log p_t(\boldsymbol{x})$ replaced by $\boldsymbol{s}_\theta(\boldsymbol{x}, t)$. It turns out there are infinitely many diffusion processes one can use. In this work, we consider a family of SDEs

$$d\hat{\boldsymbol{x}} = [\boldsymbol{F}_t\hat{\boldsymbol{x}} - \frac{1+\lambda^2}{2}\boldsymbol{G}_t\boldsymbol{G}_t^T\boldsymbol{s}_\theta(\hat{\boldsymbol{x}}, t)]dt + \lambda\boldsymbol{G}_t d\boldsymbol{w}, \tag{6.4}$$

parameterized by $\lambda \geq 0$. Here we use $\hat{\boldsymbol{x}}$ to distinguish the solution to the SDE associated with the learned score from the ground truth $\boldsymbol{x}$ in Eqs. (6.1) and (6.2). When $\lambda = 0$, Eq. (6.4) reduces to an ODE known as the *probability flow ODE* [4]. The reverse-time diffusion Eq. (6.2) with an approximated score is a special case of Eq. (6.4) with $\lambda = 1$. Denote the trajectories generated by Eq. (6.4) as $\{\hat{\boldsymbol{x}}^*_t\}_{0\leq t\leq T}$ and the marginal distributions as $\hat{p}^*_t$. The following Proposition [5] holds.

Proposition 1. *When $\boldsymbol{s}_\theta(\boldsymbol{x}, t) = \nabla \log p_t(\boldsymbol{x})$ for all $\boldsymbol{x}, t$, and $\hat{p}^*_T = \pi$, the marginal distribution $\hat{p}^*_t$ of Eq.* (6.4) *matches p_t of the forward diffusion Eq.* (6.1) *for all $0 \leq t \leq T$.*

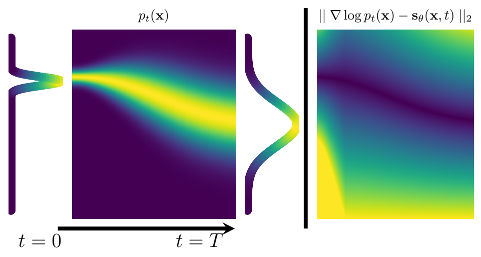


Figure 6.2: Fitting error on a toy demo. Lighter areas represent higher probability region (left) and larger fitting error (right).

The above result justifies the usage of Eq. (6.4) for generating samples. To generate a new sample, one can sample $\hat{\boldsymbol{x}}_T^*$ from π and solve Eq. (6.4) to obtain a sample $\hat{\boldsymbol{x}}_0^*$. However, in practice, exact solutions to Eq. (6.4) are not attainable and one needs to discretize Eq. (6.4) over time to get an approximated solution. Denote the approximated solution by $\hat{\boldsymbol{x}}_t$ and its marginal distribution by $\hat{p}_t$, then the error of the generative model, that is, the difference between $p_0(\boldsymbol{x})$ and $\hat{p}_0(\boldsymbol{x})$, is caused by two error sources, *fitting error* and *discretization error*. The fitting error is due to the mismatch between the learned score network $\boldsymbol{s}_\theta$ and the ground truth score $\nabla \log p_t(\boldsymbol{x})$. The discretization error includes all extra errors introduced by the discretization in numerically solving Eq. (6.4). To reduce discretization error, one needs to use smaller stepsize and thus larger number of steps, making the sampling less efficient.

The objective of this work is to investigate these two error sources and develop a more efficient sampling scheme from Eq. (6.4) with less errors. In this section, we focus on the ODE approach with $\lambda = 0$. All experiments in this section are conducted based on VPSDE over the CIFAR10 dataset unless stated otherwise.

6.3.1 Can we learn globally accurate score?

Since DMs demonstrate impressive empirical results in generating high-fidelity samples, it is tempting to believe that the learned score network is able to fit the score of data distribution very well, that is, $\boldsymbol{s}_\theta(\boldsymbol{x}, t) \approx \nabla \log p_t(\boldsymbol{x})$ for almost all $\boldsymbol{x} \in \mathbb{R}^D$ and $t \in [0, T]$. This is, however, not true; the fitting error can be arbitrarily large on some $\boldsymbol{x}, t$ as illustrated in a simple example below. In fact, the learned score models are not accurate for most $\boldsymbol{x}, t$.

Consider a generative modeling task over 1-dimensional space, i.e., $D = 1$. The data distribution is a Gaussian concentrated with a very small variance. We plot the fitting error[1] between a score model trained by minimizing Eq. (6.3) and the ground truth score in Fig. 6.2. As can be seen from the figure, the score model works well in the region where $p_t(\boldsymbol{x})$ is large but suffers from large error in the region where $p_t(\boldsymbol{x})$ is small. This observation can be explained by examining the training loss Eq. (6.3). In particular, the training data of Eq. (6.3) are sampled from $p_t(\boldsymbol{x})$. In regions with a low $p_t(\boldsymbol{x})$ value, the learned score network is not expected to work well due to the lack of training data. This phenomenon becomes even clearer in realistic settings with high-dimensional data. The region with high $p_t(\boldsymbol{x})$ value is extremely small since realistic data is often sparsely distributed in $\mathbb{R}^D$; it is believed real data such as images concentrate on an intrinsic low dimensional manifold [199, 200, 195].

As a consequence, to ensure $\hat{\boldsymbol{x}}_0$ is close to $\boldsymbol{x}_0$, we need to make sure $\hat{\boldsymbol{x}}_t$ stays in the high $p_t(\boldsymbol{x})$ region for all t. This makes fast sampling from Eq. (6.4) a challenging task as it prevents us from taking an aggressive step size that is likely to take the solution to the region where the fitting error of the learned score network is large. A good discretization scheme for Eq. (6.4) should be able to help reduce the impact of the fitting error of the score network during sampling.

6.3.2 Discretization error

We next investigate the discretization error of solving the probability flow ODE ($\lambda = 0$)

$$\frac{d\hat{\boldsymbol{x}}}{dt} = \boldsymbol{F}_t\hat{\boldsymbol{x}} - \frac{1}{2}\boldsymbol{G}_t\boldsymbol{G}_t^T\boldsymbol{s}_\theta(\hat{\boldsymbol{x}}, t). \tag{6.5}$$

The exact solution to this ODE is

[1]Because the fitting error explodes when $t \to 0$, we have scaled the fitting error for better visualization.

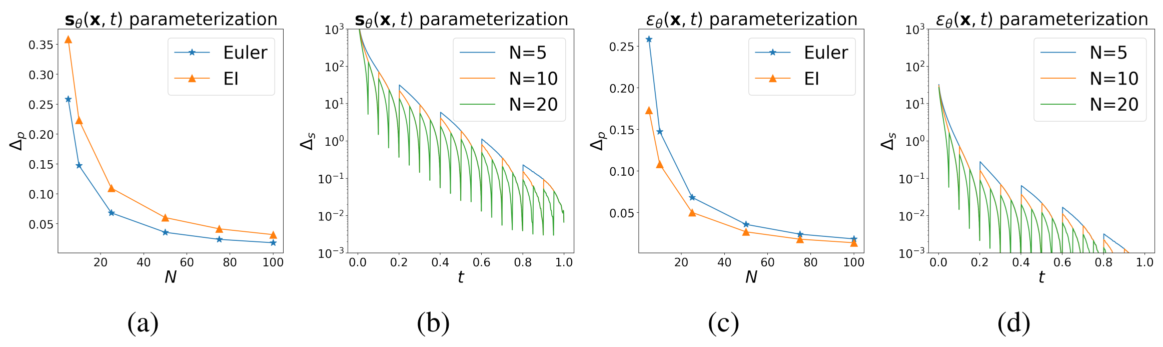


Figure 6.3: Fig. 6.3a shows average pixel difference Δ_p between ground truth $\hat{\boldsymbol{x}}_0^*$ and numerical solution $\hat{\boldsymbol{x}}_0$ from Euler method and EI method. Fig. 6.3b depicts approximation error $\Delta_{\boldsymbol{s}}$ along ground truth solutions. Fig. 6.3d shows Δ_s can be dramatically reduced if the parameterization $\epsilon_\theta(\boldsymbol{x}, t)$ instead of $\boldsymbol{s}_\theta(\boldsymbol{x}, t)$ is used. This parameterization helps the EI method outperform the Euler method in Fig. 6.3c.

$$\hat{\boldsymbol{x}}_t = \Psi(t, s)\hat{\boldsymbol{x}}_s + \int_s^t \Psi(t, \tau)[-\frac{1}{2}\boldsymbol{G}_\tau \boldsymbol{G}_\tau^T \boldsymbol{s}_\theta(\hat{\boldsymbol{x}}_\tau, \tau)]d\tau, \tag{6.6}$$

where $\Psi(t, s)$ satisfying $\frac{\partial}{\partial t}\Psi(t, s) = \boldsymbol{F}_t \Psi(t, s), \Psi(s, s) = \boldsymbol{I}$ is known as the transition matrix from time s to t associated with $\boldsymbol{F}_\tau$. Eq. (6.5) is a semilinear stiff ODE [23] that consists of a linear term $\boldsymbol{F}_t \hat{\boldsymbol{x}}$ and a nonlinear term $\boldsymbol{s}_\theta(\hat{\boldsymbol{x}}, t)$. There exist many different numerical solvers for Eq. (6.5) associated with different discretization schemes to approximate Eq. (6.6) [201]. As the discretization step size goes to zero, the solutions obtained from all these methods converge to that of Eq. (6.5). However, the performances of these methods can be dramatically different when the step size is large. On the other hand, to achieve fast sampling with Eq. (6.5), we need to approximately solve it with a small number of discretization steps, and thus large step size. This motivates us to develop an efficient discretizaiton scheme that fits with Eq. (6.5) best. In the rest of this section, we systematically study the discretization error in solving Eq. (6.5), both theoretically and empirically with carefully designed experiments. Based on these results, we develop an efficient algorithm for Eq. (6.5) that requires a small number of NFEs.

Ingredient 1: Exponential Integrator over Euler method. The Euler method is the most elementary explicit numerical method for ODEs and is widely used in numerical softwares [202]. When applied to Eq. (6.5), the Euler method reads

$$\hat{\boldsymbol{x}}_{t-\Delta t} = \hat{\boldsymbol{x}}_t - \ [\boldsymbol{F}_t \hat{\boldsymbol{x}}_t - \frac{1}{2}\boldsymbol{G}_t \boldsymbol{G}_t^T \boldsymbol{s}_\theta(\hat{\boldsymbol{x}}_t, t)]\Delta t. \tag{6.7}$$

This is used in many existing works in DMs [4, 196]. This approach however has low accuracy and is sometimes unstable when the stepsize is not sufficiently small. To improve the accuracy, we propose to use the *Exponential Integrator (EI)*, a method that leverages the semilinear structure of Eq. (6.5). When applied to Eq. (6.5), the EI reads

$$\hat{\boldsymbol{x}}_{t-\Delta t} = \Psi(t-\Delta t, t)\hat{\boldsymbol{x}}_t + \ [\int_t^{t-\Delta t} -\frac{1}{2}\Psi(t-\Delta t, \tau)\boldsymbol{G}_\tau \boldsymbol{G}_\tau^T d\tau]\boldsymbol{s}_\theta(\hat{\boldsymbol{x}}_t, t). \tag{6.8}$$

It is effective if the nonlinear term $\boldsymbol{s}_\theta(\hat{\boldsymbol{x}}_t, t)$ does not change much along the solution. In fact, for any given Δt, Eq. (6.8) solves Eq. (6.5) exactly if $\boldsymbol{s}_\theta(\hat{\boldsymbol{x}}_t, t)$ is constant over the time interval $[t-\Delta t, t]$.

To compare the EI Eq. (6.8) and the Euler method Eq. (6.7), we plot in Fig. 6.3a the *average pixel difference* Δ_p between the ground truth $\hat{\boldsymbol{x}}_0^*$ and the numerical solution $\hat{\boldsymbol{x}}_0$ obtained by these two methods for various number N of steps. Surprisingly, the EI method performs worse than the Euler method.

This observation suggests that there are other major factors that contribute to the error Δ_p. In particular, the condition that the nonlinear term $\boldsymbol{s}_\theta(\hat{\boldsymbol{x}}_t, t)$ does not change much along the solution assumed for the EI method does not hold. To see this, we plot the *score approximation error* $\Delta_{\boldsymbol{s}}(\tau) = ||\boldsymbol{s}_\theta(\boldsymbol{x}_\tau, \tau) - \boldsymbol{s}_\theta(\boldsymbol{x}_t, t)||_2, \tau \in [t-\Delta t, t]$ along the exact solution $\{\hat{\boldsymbol{x}}_t^*\}$ to Eq. (6.5) in Fig. 6.3b[2]. It can be seen that the approximation error grows rapidly as t approaches 0. This is not strange; the score of realistic data distribution $\nabla \log p_t(\boldsymbol{x})$ should change rapidly as $t \to 0$ [196].

Ingredient 2: $\epsilon_\theta(\boldsymbol{x}, t)$ over $\boldsymbol{s}_\theta(\boldsymbol{x}, t)$. The issues caused by rapidly changing score $\nabla \log p_t(\boldsymbol{x})$ do not only exist in sampling, but also appear in the training of DMs. To address these issues, a different parameterization of the score network is used. In particular,

[2]The $\{\hat{\boldsymbol{x}}_t^*\}$ are approximated by solving ODE with high accuracy solvers and sufficiently small step size. For better visualization, we have removed the time discretization points in Fig. 6.3b and Fig. 6.3d, since $\Delta_{\boldsymbol{s}} = 0$ at these points and becomes negative infinity in log scale.

it is found that the parameterization [7] $\nabla \log p_t(\boldsymbol{x}) \approx -\boldsymbol{L}_t^{-T}\epsilon_\theta(\boldsymbol{x}, t)$, where $\boldsymbol{L}_t$ can be any matrix satisfying $\boldsymbol{L}_t\boldsymbol{L}_t^T = \Sigma_t$, leads to significant improvements of accuracy. The rationale of this parameterization is based on a reformulation of the training loss Eq. (6.3) as [7]

$$\bar{\mathcal{L}}(\theta) = \mathbb{E}_{t\sim \text{Unif}[0,T]}\mathbb{E}_{p(\boldsymbol{x}_0),\epsilon\sim\mathcal{N}(0,\boldsymbol{I})}[\|\epsilon - \epsilon_\theta(\mu_t\boldsymbol{x}_0 + \boldsymbol{L}_t\epsilon, t)\|^2_{\bar{\Lambda}_t}] \tag{6.9}$$

with $\bar{\Lambda}_t = \boldsymbol{L}_t^{-1}\Lambda_t\boldsymbol{L}_t^{-T}$. The network ϵ_θ tries to follow ϵ which is sampled from a standard Gaussian and thus has a small magnitude. In comparison, the parameterization $\boldsymbol{s}_\theta = -\boldsymbol{L}_t^{-T}\epsilon_\theta$ can take large value as $\boldsymbol{L}_t \to 0$ as t approaches 0. It is thus better to approximate ϵ_θ than $\boldsymbol{s}_\theta$ with a neural network.

We adopt this parameterization and rewrite Eq. (6.5) as

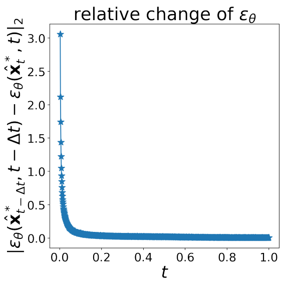
$$\frac{d\hat{\boldsymbol{x}}}{dt} = \boldsymbol{F}_t\hat{\boldsymbol{x}} + \frac{1}{2}\boldsymbol{G}_t\boldsymbol{G}_t^T\boldsymbol{L}_t^{-T}\epsilon_\theta(\hat{\boldsymbol{x}}, t). \tag{6.10}$$

Applying the EI to Eq. (6.10) yields

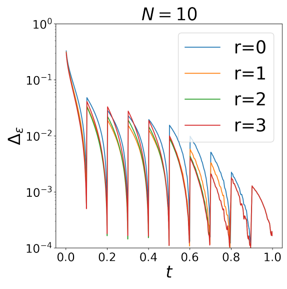
$$\hat{\boldsymbol{x}}_{t-\Delta t} = \Psi(t-\Delta t, t)\hat{\boldsymbol{x}}_t + \left[\int_t^{t-\Delta t}\frac{1}{2}\Psi(t-\Delta t, \tau)\boldsymbol{G}_\tau\boldsymbol{G}_\tau^T\boldsymbol{L}_\tau^{-T}d\tau\right]\epsilon_\theta(\hat{\boldsymbol{x}}_t, t). \tag{6.11}$$

Compared with Eq. (6.8), Eq. (6.11) employs $-\boldsymbol{L}_\tau^{-T}\epsilon_\theta(\boldsymbol{x}_t, t)$ instead of $\boldsymbol{s}_\theta(\boldsymbol{x}_t, t) = -\boldsymbol{L}_t^{-T}\epsilon_\theta(\boldsymbol{x}_t, t)$ to approximate the score $\boldsymbol{s}_\theta(\boldsymbol{x}_\tau, \tau)$ over the time interval $\tau \in [t-\Delta t, t]$. This modification from $\boldsymbol{L}_t^{-T}$ to $\boldsymbol{L}_\tau^{-T}$ turns out to be crucial; the coefficient $\boldsymbol{L}_\tau^{-T}$ changes rapidly over time. This is verified by Fig. 6.3d where we plot the score approximation error Δ_s when the parameterization ϵ_θ is used, from which we see the error Δ_s is greatly reduced compared with Fig. 6.3b. With this modification, the EI method significantly outperforms the Euler method as shown in Fig. 6.3c. Next we develop several fast sampling algorithms, all coined as the *Diffusion Exponential Integrator Sampler (DEIS)*, based on Eq. (6.11), for DMs.

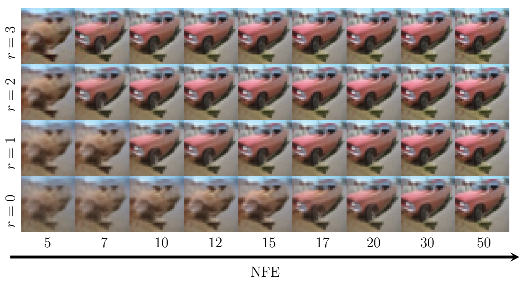
Interestingly, the discretization Eq. (6.11) based on EI coincides with the popular deterministic DDIM when the forward diffusion Eq. (6.1) is VPSDE [192] as summarized



(a)



(b)



(c)

Figure 6.4: Fig. 6.4a shows relative changes of $\epsilon_\theta(\hat{\boldsymbol{x}}^*_t, t)$ with respect to t are relatively small, especially when $t > 0.15$. Fig. 6.4b depicts the extrapolation error with $N = 10$. High order polynomial can reduce approximation error effectively. Fig. 6.4c illustrates effects of extrapolation. When N is small, higher order polynomial approximation leads to better samples.

below.

Proposition 2. *When the forward diffusion Eq.* (6.1) *is set to be VPSDE (*$\boldsymbol{F}_t, \boldsymbol{G}_t$ *are specified in Tab. 6.1), the EI discretization Eq.* (6.11) *becomes*

$$\hat{\boldsymbol{x}}_{t-\Delta t} = \sqrt{\frac{\alpha_{t-\Delta t}}{\alpha_t}}\hat{\boldsymbol{x}}_t + [\sqrt{1-\alpha_{t-\Delta t}} - \sqrt{\frac{\alpha_{t-\Delta t}}{\alpha_t}}\sqrt{1-\alpha_t}]\epsilon_\theta(\hat{\boldsymbol{x}}_t, t), \tag{6.12}$$

which coincides with the deterministic DDIM sampling algorithm.

Our result provides an alternative justification for the efficacy of DDIM for VPSDE from a numerical discretization point of view. Unlike DDIM, our method Eq. (6.11) can be applied to any diffusion SDEs to improve the efficiency and accuracy of discretizations.

In the discretization Eq. (6.11), we use $\epsilon_\theta(\hat{\boldsymbol{x}}_t, t)$ to approximate $\epsilon_\theta(\hat{\boldsymbol{x}}_\tau, \tau)$ for all $\tau \in [t-\Delta t, t]$, which is a zero order approximation. Comparing Eq. (6.11) and Eq. (6.6) we see that this approximation error largely determines the accuracy of discretization. One natural question to ask is whether it is possible to use a better approximation of $\epsilon_\theta(\hat{\boldsymbol{x}}_\tau, \tau)$ to further improve the accuracy? We answer this question affirmatively below with an improved algorithm.

Ingredient 3: Polynomial extrapolation of ϵ_θ. Before presenting our algorithm, we investigate how $\epsilon_\theta(\boldsymbol{x}_t, t)$ evolves along a ground truth solution $\{\hat{\boldsymbol{x}}_t\}$ from $t = T$ to $t = 0$. We plot the relative change in 2-norm of $\epsilon_\theta(\boldsymbol{x}_t, t)$ in Fig. 6.4a. It reveals that for most time instances the relative change is small. This motivates us to use previous (backward) evaluations of ϵ_θ up to t to extrapolate $\epsilon_\theta(\boldsymbol{x}_\tau, \tau)$ for $\tau \in [t - \Delta t, t]$.

Inspired by the high-order polynomial extrapolation in linear multistep methods, we propose to use high-order polynomial extrapolation of ϵ_θ in our EI method. To this end, consider time discretization $\{t_i\}_{i=0}^N$ where $t_0 = 0, t_N = T$. For each i, we fit a polynomial $\boldsymbol{P}_r(t)$ of degree r with respect to the interpolation points $(t_{i+j}, \epsilon_\theta(\hat{\boldsymbol{x}}_{t_{i+j}}, t_{i+j})), 0 \leq j \leq r$. This polynomial $\boldsymbol{P}_r(t)$ has explicit expression

$$\boldsymbol{P}_r(t) = \sum_{j=0}^{r} [\prod_{k \neq j} \frac{t - t_{i+k}}{t_{i+j} - t_{i+k}}] \epsilon_\theta(\hat{\boldsymbol{x}}_{t_{i+j}}, t_{i+j}). \tag{6.13}$$

We then use $\boldsymbol{P}_r(t)$ to approximate $\epsilon_\theta(\boldsymbol{x}_\tau, \tau)$ over the interval $[t_{i-1}, t_i]$. For $i > N - r$, we need to use polynomials of lower order to approximate ϵ_θ. To see the advantages of this approximation, we plot the approximate error $\Delta_\epsilon(t) = ||\epsilon_\theta(\boldsymbol{x}_t, t) - \boldsymbol{P}_r(t)||_2$ of $\epsilon_\theta(\boldsymbol{x}_t, t)$ by $\boldsymbol{P}_r(t)$ along ground truth trajectories $\{\hat{\boldsymbol{x}}_t^*\}$ in Fig. 6.4b. It can be seen that higher order polynomials can reduce approximation error compared with the case $r = 0$ which uses zero order approximation as in Eq. (6.11).

As in the EI method Eq. (6.11) that uses a zero order approximation of the score in Eq. (6.6), the update step of order r is obtained by plugging the polynomial approximation Eq. (6.13) into Eq. (6.6). It can be written explicitly as

$$\hat{\boldsymbol{x}}_{t_{i-1}} = \Psi(t_{i-1}, t_i)\hat{\boldsymbol{x}}_{t_i} + \sum_{j=0}^{r} [\mathrm{C}_{ij} \epsilon_\theta(\hat{\boldsymbol{x}}_{t_{i+j}}, t_{i+j})] \tag{6.14}$$

$$\mathrm{C}_{ij} = \int_{t_i}^{t_{i-1}} \frac{1}{2} \Psi(t_{i-1}, \tau) \boldsymbol{G}_\tau \boldsymbol{G}_\tau^T \boldsymbol{L}_\tau^{-T} \prod_{k \neq j} [\frac{\tau - t_{i+k}}{t_{i+j} - t_{i+k}}] d\tau. \tag{6.15}$$

We remark that the update in Eq. (6.14) is a linear combination of $\hat{\boldsymbol{x}}_{t_i}$ and $\epsilon_\theta(\hat{\boldsymbol{x}}_{t_{i+j}}, t_{i+j})$, where the weights $\Psi(t_{i-1}, t_i)$ and C_{ij} are calculated once for a given forward diffusion Eq. (6.1) and time discretization, and can be reused across batches. For some diffusion Eq. (6.1), $\Psi(t_{i-1}, t_i), \mathrm{C}_{ij}$ have closed form expression. Even if analytic formulas are not available, one can use high accuracy solver to obtain these coefficients. In DMs (e.g., VPSDE and VESDE), Eq. (6.15) are normally 1-dimensional or 2-dimensional integrations and are thus easy to evaluate numerically. This approach resembles the classical Adams–Bashforth [23] method, thus we term it tAB-DEIS. Here we use t to differentiate it from other DEIS algorithms we present later in Sec. 6.4 based on a time-scaled ODE.

The tAB-DEIS algorithm is summarized in Algo 5. Note that the deterministic DDIM is a special case of tAB-DEIS for VPSDE with $r = 0$. The polynomial approximation used in DEIS improves the sampling quality significantly when sampling steps N is small, as shown in Fig. 6.4c.

Algorithm 5 tAB-DEIS

```
Input: $\{t_i\}_{i=0}^N, r$
Instantiate: $\hat{\boldsymbol{x}}_{t_N}$, Empty $\epsilon$-buffer
Calculate weights $\Psi, \boldsymbol{C}$ based on Eq. (6.15)
for $i$ in $N, N-1, \cdots, 1$ do
    $\epsilon$-buffer.append($\epsilon_\theta(\hat{\boldsymbol{x}}_{t_i}, t_i)$)
    $\hat{\boldsymbol{x}}_{t_{i-1}} \leftarrow$ Eq. (6.14) with $\Psi, \boldsymbol{C}, \epsilon$-buffer
end for
```

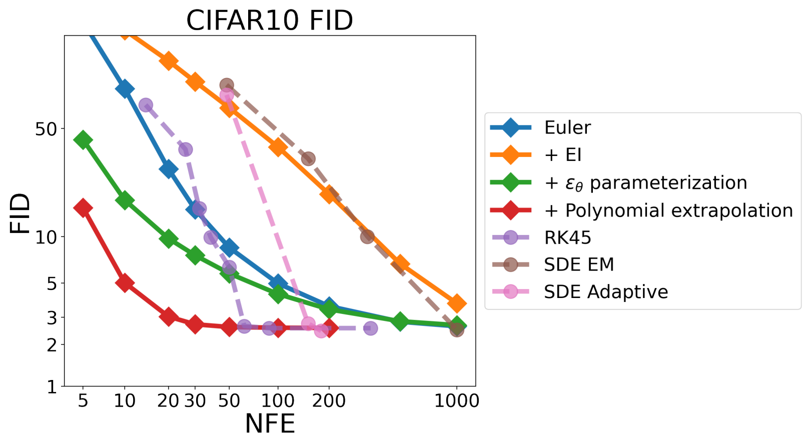


Figure 6.5: Ablation study and comparison with other samplers. We notice switching from Euler to Exponential Integrator worsens FID, which we explore and explain Ingredient 2 in Sec. 6.3. With EI, ϵ_θ, polynomial extrapolation and optimizing timestamps can significantly improve the sampling quality. Compared with other samplers, ODE sampler based on RK45 [4], SDE samplers based on Euler-Maruyama (EM) [4] and SDE adaptive step size solver [193], DEIS can converge much faster.

6.4 Exponential Integrator: simplify probability Flow ODE

Next we present a different perspective to DEIS based on ODE transformations. The probability ODE Eq. (6.10) can be transformed into a simple non-stiff ODE, and then off-the-shelf ODE solvers can be applied to solve the ODE effectively. To this end, we introduce variable $\hat{\boldsymbol{y}}_t := \Psi(t, 0)\hat{\boldsymbol{x}}_t$ and rewrite Eq. (6.10) into

$$\frac{d\hat{\boldsymbol{y}}}{dt} = \frac{1}{2}\Psi(t, 0)\boldsymbol{G}_t\boldsymbol{G}_t^T\boldsymbol{L}_t^{-T}\epsilon_\theta(\Psi(0, t)\hat{\boldsymbol{y}}, t). \tag{6.16}$$

Note that, departing from Eq. (6.10), Eq. (6.16) does not possess semi-linear structure. Thus, we can apply off-the-shelf ODE solvers to Eq. (6.16) without accounting for the semi-linear structure in algorithm design. This transformation Eq. (6.16) can be further improved by taking into account the analytical form of $\Psi, \boldsymbol{G}_t, \boldsymbol{L}_t$. Here we present treatment for VPSDE; the results can be extended to other (scalar) DMs such as VESDE.

Proposition 3. *For the VPSDE, with* $\hat{\boldsymbol{y}}_t = \sqrt{\frac{\alpha_0}{\alpha_t}}\hat{\boldsymbol{x}}_t$ *and the time-scaling* $\beta(t) = \sqrt{\alpha_0}(\sqrt{\frac{1-\alpha_t}{\alpha_t}} -$

$\sqrt{\frac{1-\alpha_0}{\alpha_0}}$), *Eq.* (6.10) *can be transformed into*

$$\frac{d\hat{\boldsymbol{y}}}{d\rho} = \epsilon_\theta(\sqrt{\frac{\alpha_{\beta^{-1}(\rho)}}{\alpha_0}}\hat{\boldsymbol{y}}, \beta^{-1}(\rho)), \quad \rho \in [\beta(0), \beta(T)]. \tag{6.17}$$

After transformation, the ODE becomes a black-box ODE that can be solved by generic ODE solvers efficiently since the stiffness caused by the semi-linear structure is removed. This is the core idea of the variants of DEIS we present next.

Based on the transformed ODE Eq. (6.17) and the above discussions, we propose two variants of the DEIS algorithm: ρ**RK-DEIS** when applying classical RK methods, and ρ**AB-DEIS** when applying Adams-Bashforth methods. We remark that the difference between tAB-DEIS and ρAB-DEIS lies in the fact that tAB-DEIS fits polynomials in t which may not be polynomials in ρ. Thanks to simplified ODEs, DEIS enjoys the convergence order guarantee as its underlying RK

6.5 Experiments

Abalation study: As shown in Fig. 6.5, ingredients introduced in Sec. 6.3.2 can significantly improve sampling efficiency on CIFAR10. Besides, DEIS outperforms standard samplers by a large margin.

DEIS variants: We include performance evaluations of various DEIS with VPSDE on CIFAR10 in Tab. 6.2, including DDIM, ρRK-DEIS, ρAB-DEIS and tAB-DEIS. For ρRK-DEIS, we find Heun's method works best among second-order RK methods, denoted as ρ2Heun, Kutta method for third order, denoted as ρ3Kutta, and classic fourth-order RK denoted as ρ4RK. For Adam-Bashforth methods, we consider fitting $1, 2, 3$ order polynomial in t, ρ, denoted as tAB and ρAB respectively. We observe that almost all DEIS algorithms can generate high-fidelity images with small NFE. Also, note that DEIS with high-order polynomial approximation can significantly outperform DDIM; the latter coincides with the zero-order polynomial approximation. We also find the performance of high

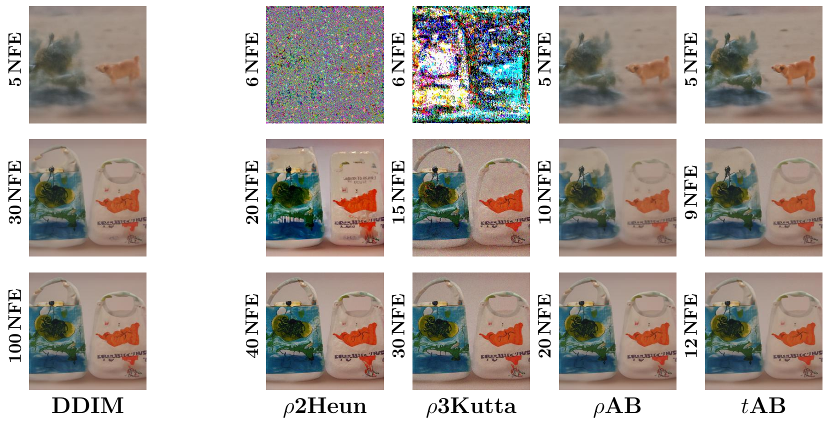


Figure 6.6: Generated samples of DDIM and DEIS with unconditional 256×256 ImageNet pretrained model [187]

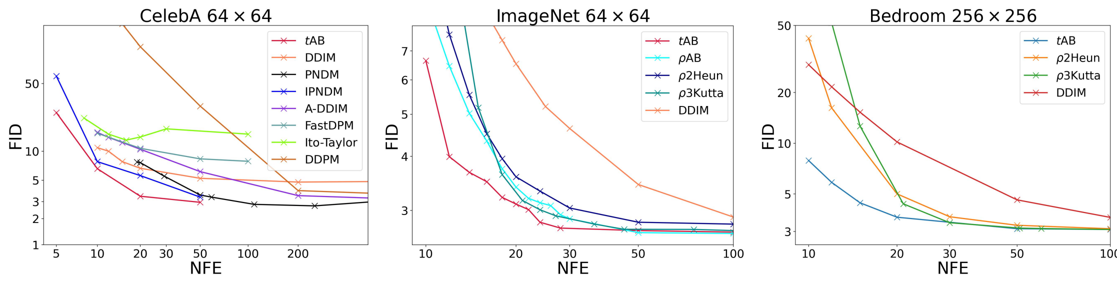


Figure 6.7: Sample quality measured by FID ↓ of different sampling algorithms with pre-trained DMs.

order ρRK-DEIS is not satisfying when NFE is small but competitive as NFE increases. It is within expectation as high order methods enjoy smaller local truncation error and total accumulated error when small step size is used and the advantage is vanishing as we reduce the number of steps.

More comparisons: We conduct more comparisons with popular sampler for DMs, including DDPM, DDIM, PNDM [203], A-DDIM [191], FastDPM [204], and Ito-Taylor [194]. We further propose Improved PNDM (iPNDM) that avoids the expensive warming start, which leads to better empirical performance. We conduct comparison on image datasets, including 64×64 CelebA [205] with pre-trained model from [192], class-conditioned 64×64 ImageNet [199] with pre-trained model [187], 256×256 LSUN Bedroom [140] with pre-trained model [187]. We compare DEIS with selected baselines in Fig. 6.7 quantitatively, and show empirical samples in Fig. 6.6.

Table 6.2: More results of DEIS for VPSDE on CIFAR10 with limited NFE. For ρRK-DEIS, the upper right number indicates extra NFEs used. Bold numbers denote the best performance achieved with similar NFE budgets. For a fair comparison, we report numbers based on their best time discretization for different algorithms with different NFE. †: The concurrent work [132] applies Heun method to a rescaled DM. This is a special case of ρ2Heun.

	FID for various DEIS									
NFE	DDIM	ρ2Heun†	ρ3Kutta	ρ4RK	ρAB1	ρAB2	ρAB3	tAB1	tAB2	tAB3
5	26.91	108^{+1}	185^{+1}	193^{+3}	22.28	21.53	21.43	19.72	16.31	**15.37**
10	11.14	14.72	13.19^{+2}	28.65^{+2}	7.56	6.72	6.50	6.09	4.57	**4.17**
15	7.06	4.89^{+1}	5.88	6.88^{+1}	4.69	4.16	3.99	4.29	3.57	**3.37**
20	5.47	3.50	2.97^{+1}	3.92	3.70	3.32	3.17	3.54	3.05	**2.86**
50	3.27	2.60	$\mathbf{2.55}^{+1}$	2.57^{+2}	2.70	2.62	2.59	2.67	2.59	2.57

6.6 Conclusion

In this work, we consider fast sampling problems for DMs. We present the diffusion exponential integrator sampler (DEIS), a fast sampling algorithm for DMs based on a novel discretization scheme of the backward diffusion process. In addition to its theoretical elegance, DEIS also works efficiently in practice; it is able to generate high-fidelity samples with less than 10 NFEs. Exploring better extrapolation may further improve sampling quality.

CHAPTER 7
ACCELERATING SAMPLING FOR NON-ISOTROPIC DIFFUSION MODELS

7.1 Introduction

Generative models based on diffusion models (DMs) have experienced rapid developments in the past few years and show competitive sample quality compared with generative adversarial networks (GANs) [187, 11, 186], competitive negative log likelihood compared with autoregressive models in various domains and tasks [206, 15]. Besides, DMs enjoy other merits such as stable and scalable training, and mode-collapsing resiliency [206, 189]. However, slow and expensive sampling prevents DMs from further application in more complex and higher dimension tasks. Once trained, GANs only forward pass neural networks once to generate samples, but the vanilla sampling method of DMs needs 1000 or even 4000 steps [189, 7, 4] to pull noise back to the data distribution, which means thousands of neural networks forward evaluations. Therefore, the generation process of DMs is several orders of magnitude slower than GANs.

How to speed up sampling of DMs has received significant attention. Building on the seminal work by [4] on the connection between stochastic differential equations (SDEs) and diffusion models, a promising strategy based on *probability flows* [4] has been developed. The probability flows are ordinary differential equations (ODE) associated with DMs that share equivalent marginal with SDE. Simple plug-in of off-the-shelf ODE solvers can already achieve significant acceleration compared to SDEs-based methods [4]. The arguably most popular sampling method is *denoising diffusion implicit model (DDIM)* [192], which includes both deterministic and stochastic samplers, and both show tremendous improvement in sampling quality compared with previous methods when only a small number of steps is used for the generation.

Although significant improvements of the DDIM in sampling efficiency have been observed empirically, the understanding of the mechanism of the DDIM is still lacking. First, why does solving probability flow ODE provide much higher sample quality than solving SDEs, when the number of steps is small? Second, it is shown that stochastic DDIM reduces to marginal-equivalent SDE [207], but its discretization scheme and mechanism of acceleration are still unclear. Finally, can we generalize DDIMs to other DMs and achieve similar or even better acceleration results?

In this work, we conduct a comprehensive study to answer the above questions, so that we can generalize and improve DDIM. We start with an interesting observation that the DDIM can solve corresponding SDEs/ODE **exactly** without any discretization error in finite or even one step when the training dataset consists of only one data point. For deterministic DDIM, we find that the added noise in perturbed data along the diffusion is constant along an exact solution of probability flow ODE (see Prop 4). Besides, provided only one evaluation of log density gradient (a.k.a. score), we are already able to recover accurate score information for any datapoints, and this explains the acceleration of stochastic DDIM for SDEs (see Prop 6). Based on this observation, together with the manifold hypothesis, we present one possible interpretation to explain why the discretization scheme used in DDIMs is effective on realistic datasets (see Fig. 7.2). Equipped with this new interpretation, we extend DDIM to general DMs, which we coin *generalized DDIM (gDDIM)*. With only a small but delicate change of the score model parameterization during sampling, gDDIM can accelerate DMs based on general diffusion processes. Specifically, we verify the sampling quality of gDDIM on Blurring diffusion models (BDM) [25, 26] and critically-damped Langevin diffusion (CLD) [196] in terms of Fréchet inception distance (FID) [139].

To summarize, we have made the following contributions: 1) We provide an interpretation for the DDIM and unravel its mechanism. 2) The interpretation not only justifies the numerical discretization of DDIMs but also provides insights into why ODE-based samplers are preferred over SDE-based samplers when NFE is low. 3) We propose gDDIM, a generalized DDIM that can accelerate a large class of DMs deterministically and stochastically. 4) We show by extensive experiments that gDDIM can drastically improve sampling quality/efficiency almost for free. Specifically, when applied to CLD, gDDIM can achieve an FID score of 2.86 with only 27 steps and 2.26 with 50 steps. gDDIM has more than 20 times acceleration on BDM compared with the original samplers.

The rest of this paper is organized as follows. In Sec. 7.2 we provide a brief inntroduction to diffusion models. In Sec. 7.3 we present an interpretation of the DDIM that explains its effectiveness in practice. Built on this interpretation, we generalize DDIM for general diffusion models in Sec. 7.4.

7.2 Background

In this section, we provide a brief introduction to diffusion models (DMs). Most DMs are built on two diffusion processes in continuous-time, one forward diffusion known as the noising process that drives any data distribution to a tractable distribution such as Gaussian by gradually adding noise to the data, and one backward diffusion known as the denoising process that sequentially removes noise from noised data to generate realistic samples. The continuous-time noising and denoising processes are modeled by stochastic differential equations (SDEs) [37].

In particular, the forward diffusion is a linear SDE with state $\boldsymbol{u}(t) \in \mathbb{R}^D$

$$d\boldsymbol{u} = \boldsymbol{F}_t \boldsymbol{u} dt + \boldsymbol{G}_t d\boldsymbol{w}, t \in [0, T] \tag{7.1}$$

where $\boldsymbol{F}_t, \boldsymbol{G}_t \in \mathbb{R}^{D \times D}$ represent the linear drift coefficient and diffusion coefficient respectively, and $\boldsymbol{w}$ is a standard Wiener process. When the coefficients are piece-wise continuous, Eq. (7.1) admits a unique solution [208]. Denote by $p_t(\boldsymbol{u})$ the distribution of the solutions $\{\boldsymbol{u}(t)\}_{0 \leq t \leq T}$ (simulated trajectories) to Eq. (7.1) at time t, then p_0 is determined by

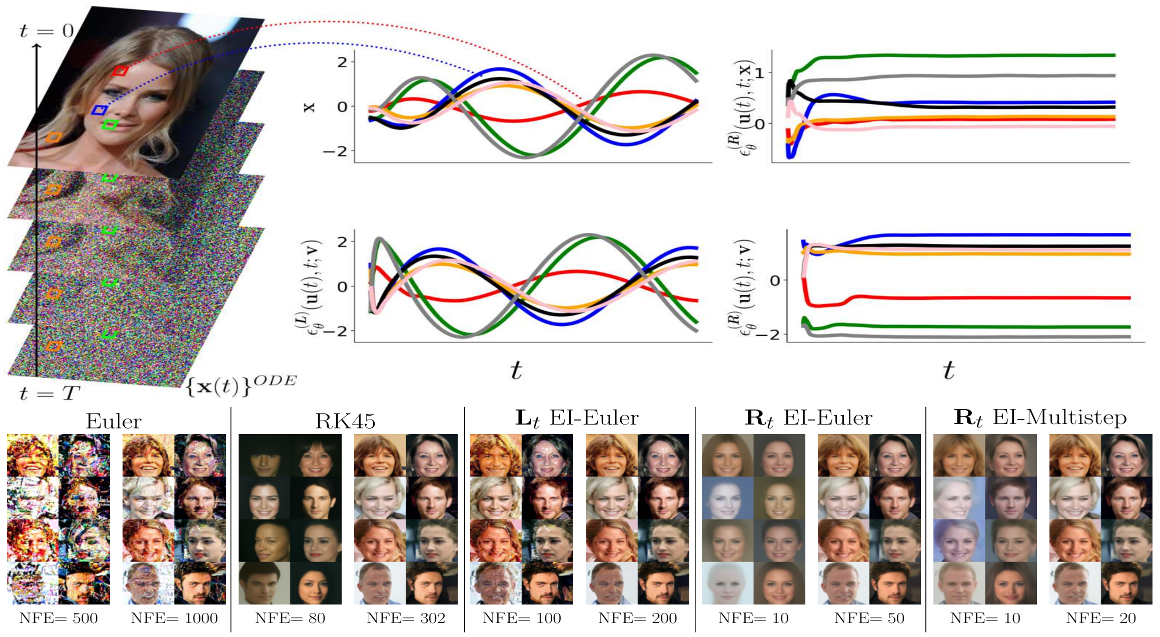


Figure 7.1: Importance of $\boldsymbol{K}_t$ for score parameterization $\boldsymbol{s}_\theta(\boldsymbol{u}, t) = -\boldsymbol{K}_t^{-T}\epsilon_\theta(\boldsymbol{u}, t)$ and acceleration of diffusion sampling with probability flow ODE. Trajectories of probability ODE for CLD [196] at random pixel locations (Left). Pixel value and output of ϵ_θ in $\boldsymbol{v}$ channel with choice $\boldsymbol{K}_t = \boldsymbol{L}_t$ [196] along the trajectory (Mid). Output of ϵ_θ in $\boldsymbol{x}, \boldsymbol{v}$ channels with our choice $\boldsymbol{R}_t$ (Right). The smooth network output along trajectories enables large stepsize and thus sampling acceleration. gDDIM based on the proper parameterization of $\boldsymbol{K}_t$ can accelerate more than 50 times compared with the naive Euler solver (Lower row).

the data distribution and p_T is a (approximate) Gaussian distribution. That is, the forward diffusion Eq. (7.1) starts as a data sample and ends as a Gaussian random variable. This can be achieved with properly chosen coefficients $\boldsymbol{F}_t, \boldsymbol{G}_t$. Thanks to linearity of Eq. (7.1), the transition probability $p_{st}(\boldsymbol{u}(t)|\boldsymbol{u}(s))$ from $\boldsymbol{u}(s)$ to $\boldsymbol{u}(t)$ is a Gaussian distribution. For convenience, denote $p_{0t}(\boldsymbol{u}(t)|\boldsymbol{u}(0))$ by $\mathcal{N}(\mu_t \boldsymbol{u}(0), \Sigma_t)$ where $\mu_t, \Sigma_t \in \mathbb{R}^{D\times D}$.

The backward process from $\boldsymbol{u}(T)$ to $\boldsymbol{u}(0)$ of Eq. (7.1) is the denoising process. It can be characterized by the backward SDE simulated in reverse-time direction [4, 48]

$$d\boldsymbol{u} = [\boldsymbol{F}_t \boldsymbol{u} dt - \boldsymbol{G}_t \boldsymbol{G}_t^T \nabla \log p_t(\boldsymbol{u})]dt + \boldsymbol{G}_t d\bar{\boldsymbol{w}}, \tag{7.2}$$

where $\bar{\boldsymbol{w}}$ denotes a standard Wiener process running backward in time. Here $\nabla \log p_t(\boldsymbol{u})$ is known as the score function. When Eq. (7.2) is initialized with $\boldsymbol{u}(T) \sim p_T$, the distribution of the simulated trajectories coincides with that of the forward diffusion Eq. (7.1). Thus, $\boldsymbol{u}(0)$ of these trajectories are unbiased samples from p_0; the backward diffusion Eq. (7.2) is an ideal generative model.

In general, the score function $\nabla \log p_t(\boldsymbol{u})$ is not accessible. In diffusion-based generative models, a time-dependent network $\boldsymbol{s}_\theta(\boldsymbol{u}, t)$, known as the score network, is used to fit the score $\nabla \log p_t(\boldsymbol{u})$. One effective approach to train $\boldsymbol{s}_\theta(\boldsymbol{u}, t)$ is the denoising score matching (DSM) technique [4, 7, 198] that seeks to minimize the DSM loss

$$\mathbb{E}_{t\sim\mathcal{U}[0,T]}\mathbb{E}_{\boldsymbol{u}(0),\boldsymbol{u}(t)|\boldsymbol{u}(0)}[\|\nabla \log p_{0t}(\boldsymbol{u}(t)|\boldsymbol{u}(0)) - \boldsymbol{s}_\theta(\boldsymbol{u}(t), t)\|^2_{\Lambda_t}], \tag{7.3}$$

where $\mathcal{U}[0, T]$ represents the uniform distribution over the interval $[0, T]$. The time-dependent weight Λ_t is chosen to balance the trade-off between sample fidelity and data likelihood of learned generative model [206]. It is discovered in [7] that reparameterizing the score network by

$$\boldsymbol{s}_\theta(\boldsymbol{u}, t) = -\boldsymbol{K}_t^{-T}\epsilon_\theta(\boldsymbol{u}, t) \tag{7.4}$$

with $\boldsymbol{K}_t\boldsymbol{K}_t^T = \Sigma_t$ leads to better sampling quality. In this parameterization, the network tries to predict directly the noise added to perturb the original data. Invoking the expression $\mathcal{N}(\mu_t\boldsymbol{u}(0), \Sigma_t)$ of $p_{0t}(\boldsymbol{u}(t)|\boldsymbol{u}(0))$, this parameterization results in the new DSM loss

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}[0,T]}\mathbb{E}_{\boldsymbol{u}(0)\sim p_0,\epsilon\sim\mathcal{N}(0,\boldsymbol{I}_D)}[\|\epsilon - \epsilon_\theta(\mu_t\boldsymbol{u}(0) + \boldsymbol{K}_t\epsilon, t)\|^2_{\boldsymbol{K}_t^{-1}\Lambda_t\boldsymbol{K}_t^{-T}}]. \tag{7.5}$$

Sampling: After the score network $\boldsymbol{s}_\theta$ is trained, one can generate samples via the backward SDE Eq. (7.2) with a learned score, or the marginal equivalent SDE/ODE [4, 5, 207]

$$d\boldsymbol{u} = [\boldsymbol{F}_t\boldsymbol{u} - \frac{1+\lambda^2}{2}\boldsymbol{G}_t\boldsymbol{G}_t^T\boldsymbol{s}_\theta(\boldsymbol{u}, t)]dt + \lambda\boldsymbol{G}_t d\boldsymbol{w}, \tag{7.6}$$

where $\lambda \geq 0$ is a free parameter. Regardless of the value of λ, the exact solutions to

Eq. (7.6) produce unbiased samples from $p_0(\boldsymbol{u})$ if $\boldsymbol{s}_\theta(\boldsymbol{u},t) = \nabla \log p_t(\boldsymbol{u})$ for all $t, \boldsymbol{u}$. When $\lambda = 1$, Eq. (7.6) reduces to reverse-time diffusion in Eq. (7.2). When $\lambda = 0$, Eq. (7.6) is known as the *probability flow ODE* [4]

$$d\boldsymbol{u} = [\boldsymbol{F}_t \boldsymbol{u} - \frac{1}{2}\boldsymbol{G}_t \boldsymbol{G}_t^T \boldsymbol{s}_\theta(\boldsymbol{u}, t)]dt. \tag{7.7}$$

Isotropic diffusion and DDIM: Most existing DMs are isotropic diffusions. A popular DM is *Denoising diffusion probabilistic modeling (DDPM)* [7]. For a given data distribution $p_{\text{data}}(\boldsymbol{x})$, DDPM has $\boldsymbol{u} = \boldsymbol{x} \in \mathbb{R}^d$ and sets $p_0(\boldsymbol{u}) = p_{\text{data}}(\boldsymbol{x})$. Though originally proposed in the discrete-time setting, it can be viewed as a discretization of a continuous-time SDE with parameters

$$\boldsymbol{F}_t := \frac{1}{2}\frac{d \log \alpha_t}{dt}\boldsymbol{I}_d, \quad \boldsymbol{G}_t := \sqrt{-\frac{d \log \alpha_t}{dt}}\boldsymbol{I}_d \tag{7.8}$$

for a decreasing scalar function α_t satisfying $\alpha_0 = 1, \alpha_T = 0$. Here $\boldsymbol{I}_d$ represents the identity matrix of dimension d. For this SDE, $\boldsymbol{K}_t$ is always chosen to be $\sqrt{1-\alpha_t}\boldsymbol{I}_d$.

The sampling scheme proposed in DDPM is inefficient; it requires hundreds or even thousands of steps, and thus number of score function evaluations (NFEs), to generate realistic samples. A more efficient alternative is the *Denoising diffusion implicit modeling (DDIM)* proposed in [192]. It proposes a different sampling scheme over a grid $\{t_i\}$

$$\boldsymbol{x}(t_{i-1}) = \sqrt{\frac{\alpha_{t_{i-1}}}{\alpha_{t_i}}}\boldsymbol{x}(t_i) + (\sqrt{1-\alpha_{t_{i-1}} - \sigma_{t_i}^2} - \sqrt{1-\alpha_{t_i}}\sqrt{\frac{\alpha_{t_{i-1}}}{\alpha_{t_i}}})\epsilon_\theta(\boldsymbol{x}(t_i), t_i) + \sigma_{t_i}\epsilon, \tag{7.9}$$

where $\{\sigma_{t_i}\}$ are hyperparameters and $\epsilon \sim \mathcal{N}(0, \boldsymbol{I}_d)$. DDIM can generate reasonable samples within 50 NFEs. For the special case where $\sigma_{t_i} = 0$, it is recently discovered in [207] that Eq. (7.9) coincides with the numerical solution to Eq. (7.7) using an advanced discretization scheme known as the *exponential integrator (EI)* that utilizes the semi-linear structure of Eq. (7.7).

CLD and BDM: [196] propose *critically-dampled Langevin diffusion (CLD)*, a DM based on an augmented diffusion with an auxiliary velocity term. More specifically, the state of the diffusion in CLD is of the form $\boldsymbol{u}(t) = [\boldsymbol{x}(t), \boldsymbol{v}(t)] \in \mathbb{R}^{2d}$ with velocity variable $\boldsymbol{v}(t) \in \mathbb{R}^d$. The CLD employs the forward diffusion Eq. (7.1) with coefficients

$$\boldsymbol{F}_t := \begin{bmatrix} 0 & \beta M^{-1} \\ \beta & -\Gamma\beta M^{-1} \end{bmatrix} \otimes \boldsymbol{I}_d, \quad \boldsymbol{G}_t := \begin{bmatrix} 0 & 0 \\ 0 & -\Gamma\beta M^{-1} \end{bmatrix} \otimes \boldsymbol{I}_d. \tag{7.10}$$

Here $\Gamma > 0, \beta > 0, M > 0$ are hyperparameters. Compared with most other DMs such as DDPM that inject noise to the data state $\boldsymbol{x}$ directly, the CLD introduces noise to the data state $\boldsymbol{x}$ through the coupling between $\boldsymbol{v}$ and $\boldsymbol{x}$ as the noise only affects the velocity component $\boldsymbol{v}$ directly. Another interesting DM is *Blurring diffusion model* (BDM) [25]. It can be shown the forward process in BDM can be formulated as a SDE with

$$\boldsymbol{F}_t := \frac{d \log[\boldsymbol{V}\boldsymbol{\alpha}_t\boldsymbol{V}^T]}{dt}, \quad \boldsymbol{G}_t := \sqrt{\frac{d\boldsymbol{\sigma}_t^2}{dt} - \boldsymbol{F}_t\boldsymbol{\sigma}_t^2 - \boldsymbol{\sigma}_t^2\boldsymbol{F}_t}, \tag{7.11}$$

where $\boldsymbol{V}^T$ denotes a Discrete Cosine Transform (DCT) and $\boldsymbol{V}$ denotes the Inverse DCT. Diagonal matrices $\boldsymbol{\alpha}_t, \boldsymbol{\sigma}_t$ are determined by frequencies information and dissipation time. Though it is argued that inductive bias in CLD and BDM can benefit diffusion model [196, 25], non-isotropic DMs are not easy to accelerate. Compared with DDPM, CLD introduces significant oscillation due to $\boldsymbol{x}$-$\boldsymbol{v}$ coupling while only inefficient ancestral sampling algorithm supports BDM [25].

7.3 Revisit DDIM: Gap between the exact solution and numerical solution

The complexity of sampling from a DM is proportional to the NFEs used to numerically solve Eq. (7.6). To establish a sampling algorithm with a small NFEs, we ask the bold question:

Can we generate samples ***exactly*** *from a DM with finite steps if the score function is precise?*

To gain some insights into this question, we start with the simplest scenario where the training dataset consists of only one data point $\boldsymbol{x}_0$. It turns out that accurate sampling from diffusion models on this toy example is not that easy, even if the exact score function is accessible. Most well-known numerical methods for Eq. (7.6), such as Runge Kutta (RK) for ODE, Euler-Maruyama (EM) for SDE, are accompanied by discretization error and cannot recover the single data point in the training set unless an infinite number of steps are used. Surprisingly, DDIMs can recover the single data point in this toy example in one step.

Built on this example, we show how the DDIM can be obtained by solving the SDE/ODE Eq. (7.6) with proper approximations. The effectiveness of DDIM is then explained by justifying the usage of those approximations for general datasets at the end of this section.

ODE sampling We consider the deterministic DDIM, that is, Eq. (7.9) with $\sigma_{t_i} = 0$. In view of Eq. (7.8), the score network Eq. (7.4) is $\boldsymbol{s}_\theta(\boldsymbol{u}, t) = -\frac{\epsilon_\theta(\boldsymbol{u}, t)}{\sqrt{1-\alpha_t}}$. To differentiate between the learned score and the real score, denote the ground truth version of ϵ_θ by ϵ_{GT}. In our toy example, the following property holds for ϵ_{GT}.

Proposition 4. *Assume $p_0(\boldsymbol{u})$ is a Dirac distribution. Let $\boldsymbol{u}(t)$ be an arbitrary solution to the probability flow ODE Eq.* (7.7) *with coefficient Eq.* (7.8) *and the ground truth score, then $\epsilon_{\text{GT}}(\boldsymbol{u}(t), t) = -\sqrt{1-\alpha_t}\nabla \log p_t(\boldsymbol{u}(t))$ remains constant, which is $\nabla \log p_T(\boldsymbol{u}(T))$, along $\boldsymbol{u}(t)$.*

We remark that even though $\epsilon_{\text{GT}}(\boldsymbol{u}(t), t)$ remains constant along an exact solution, the score $\nabla \log p_t(\boldsymbol{u}(t))$ is time-varying. This underscores the advantage of the parameterization ϵ_θ over $\boldsymbol{s}_\theta$. Inspired by Prop 4, we devise a sampling algorithm as follows that can recover the exact data point in one step for our toy example. This algorithm turns out to coincide with the deterministic DDIM.

Proposition 5. *With the parameterization $\boldsymbol{s}_\theta(\boldsymbol{u}, \tau) = -\frac{\epsilon_\theta(\boldsymbol{u}, \tau)}{\sqrt{1-\alpha_\tau}}$ and the approximation $\epsilon_\theta(\boldsymbol{u}, \tau) \approx \epsilon_\theta(\boldsymbol{u}(t), t)$ for $\tau \in [t-\Delta t, t]$, the solution to the probability flow ODE Eq.* (7.7) *with coefficient Eq.* (7.8) *is*

$$\boldsymbol{u}(t-\Delta t)=\sqrt{\frac{\alpha_{t-\Delta t}}{\alpha_t}}\boldsymbol{u}(t)+(\sqrt{1-\alpha_{t-\Delta t}}-\sqrt{1-\alpha_t}\sqrt{\frac{\alpha_{t-\Delta t}}{\alpha_t}})\epsilon_\theta(\boldsymbol{u}(t),t), \tag{7.12}$$

which coincides with deterministic DDIM.

When $\epsilon_\theta = \epsilon_{\mathrm{GT}}$ as is the case in our toy example, there is no approximation error in Prop 5 and Eq. (7.12) is precise. This implies that deterministic DDIM can recover the training data in one step in our example. The update Eq. (7.12) corresponds to a numerical method known as the exponential integrator to the probability flow ODE Eq. (7.7) with coefficient Eq. (7.8) and parameterization $\boldsymbol{s}_\theta(\boldsymbol{u},\tau) = -\frac{\epsilon_\theta(\boldsymbol{u},\tau)}{\sqrt{1-\alpha_\tau}}$. This strategy is used and developed recently in [207]. Prop 4 and toy experiments in Fig. 7.2 provide sights on why such a strategy should work.

SDE sampling The above discussions however do not hold for stochastic cases where $\lambda > 0$ in Eq. (7.6) and $\sigma_{t_i} > 0$ in Eq. (7.9). Since the solutions to Eq. (7.6) from $t = T$ to $t = 0$ are stochastic, neither $\nabla \log p_t(\boldsymbol{u}(t))$ nor $\epsilon_{\mathrm{GT}}(\boldsymbol{u}(t), t)$ remains constant along sampled trajectories; both are affected by the stochastic noise. The denoising SDE Eq. (7.6) is more challenging compared with the probability ODE since it injects additional noise to $\boldsymbol{u}(t)$. The score information needs to remove not only noise presented in $\boldsymbol{u}(T)$ but also injected noise along the diffusion. In general, one evaluation of $\epsilon_\theta(\boldsymbol{u}, t)$ can only provide the information to remove noise in the current state $\boldsymbol{u}$; it cannot predict the future injected noise. Can we do better? The answer is affirmative on our toy dataset. Given only one score evaluation, it turns out that score at any point can be recovered.

Proposition 6. *Assume SDE coefficients Eq.* (7.8) *and that* $p_0(\boldsymbol{u})$ *is a Dirac distribution. Given any evaluation of the score function* $\nabla \log p_s(\boldsymbol{u}(s))$, *one can recover* $\nabla \log p_t(\boldsymbol{u})$ *for* *any* $t, \boldsymbol{u}$ *as*

$$\nabla \log p_t(\boldsymbol{u}) = \frac{1-\alpha_s}{1-\alpha_t}\sqrt{\frac{\alpha_t}{\alpha_s}}\nabla \log p_s(\boldsymbol{u}(s)) - \frac{1}{1-\alpha_t}(\boldsymbol{u} - \sqrt{\frac{\alpha_t}{\alpha_s}}\boldsymbol{u}(s)). \tag{7.13}$$

The major difference between Prop 6 and Prop 4 is that Eq. (7.13) retains the dependence of the score over the state $\boldsymbol{u}$. This dependence is important in canceling the injected noise in the denoising SDE Eq. (7.6). This approximation Eq. (7.13) turns out to lead to a numerical scheme for Eq. (7.6) that coincide with the stochastic DDIM.

Theorem 6. *Given the parameterization* $\boldsymbol{s}_\theta(\boldsymbol{u}, \tau) = -\frac{\epsilon_\theta(\boldsymbol{u}, \tau)}{\sqrt{1-\alpha_\tau}}$ *and the approximation* $\boldsymbol{s}_\theta(\boldsymbol{u}, \tau) \approx \frac{1-\alpha_t}{1-\alpha_\tau}\sqrt{\frac{\alpha_\tau}{\alpha_t}}\boldsymbol{s}_\theta(\boldsymbol{u}(t), t) - \frac{1}{1-\alpha_\tau}(\boldsymbol{u} - \sqrt{\frac{\alpha_\tau}{\alpha_t}}\boldsymbol{u}(t))$ *for* $\tau \in [t-\Delta t, t]$*, the exact solution* $\boldsymbol{u}(t-\Delta t)$ *to Eq.* (7.6) *with coefficient Eq.* (7.8) *is*

$$\boldsymbol{u}(t-\Delta t) \sim \mathcal{N}(\sqrt{\frac{\alpha_{t-\Delta t}}{\alpha_t}}\boldsymbol{u}(t) + \left[-\sqrt{\frac{\alpha_{t-\Delta t}}{\alpha_t}}\sqrt{1-\alpha_t} + \sqrt{1-\alpha_{t-\Delta t}-\sigma_t^2}\right]\epsilon_\theta(\boldsymbol{u}(t), t), \sigma_t^2 \boldsymbol{I}_d) \tag{7.14}$$

with $\sigma_t = (1-\alpha_{t-\Delta t})\left[1 - \left(\frac{1-\alpha_{t-\Delta t}}{1-\alpha_t}\right)^{\lambda^2}\left(\frac{\alpha_t}{\alpha_{t-\Delta t}}\right)^{\lambda^2}\right]$*, which is the same as the DDIM Eq.* (7.9).

Note that theorem 6 with $\lambda = 0$ agrees with Prop 5; both reproduce the deterministic DDIM but with different derivations. In summary, DDIMs can be derived by utilizing local approximations.

Justification of Dirac approximation While Prop 4 and Prop 6 require the strong assumption that the data distribution is a Dirac, DDIMs in Prop 5 and theorem 6 work very effectively on realistic datasets, which may contain millions of datapoints [188]. Here we present one possible interpretation based on the manifold hypothesis [209].

It is believed that real-world data lie on a low-dimensional manifold [210] embedded in a high-dimensional space and the data points are well separated in high-dimensional data space. For example, realistic images are scattered in pixel space and the distance between

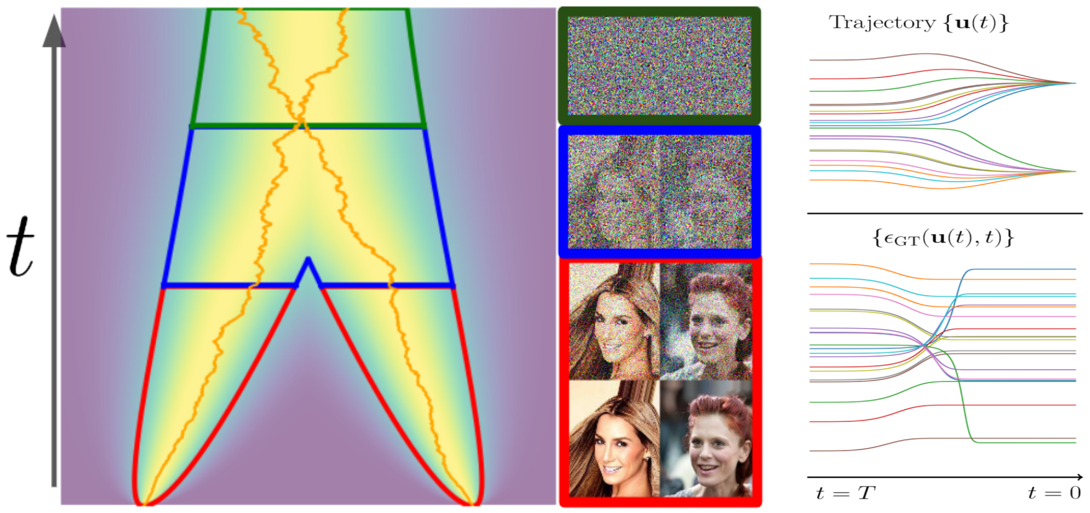

t
Trajectory $\{\mathbf{u}(t)\}$
$\{\epsilon_{\mathrm{GT}}(\mathbf{u}(t), t)\}$
$t = T$
$t = 0$

Figure 7.2: Manifold hypothesis and Dirac distribution assumption. We model an image dataset as a mixture of well-separated Dirac distribution and visualize the diffusion process on the left. Curves in **red** indicate high density area spanned by $p_{0t}(\boldsymbol{u}(t)|\boldsymbol{u}(0))$ by different mode and region surrounded by them indicates the phase when $p_t(\boldsymbol{u})$ is dominated by one mode while region surrounded by **blue** one is for the mixing phase, and **green** region indicates fully mixed phase. On the right, sampling trajectories depict smoothness of ϵ_{GT} along ODE solutions, which justifies approximations used in DDIM and partially explains its empirical acceleration.

every two images can be very large if measured in pixel difference even if they are similar perceptually. To model this property, we consider a dataset consisting of M datapoints $\{\boldsymbol{u}^{(m)}\}_{m=1}^{M}$. The exact score is

$$\nabla \log p_t(\boldsymbol{u}) = \sum_m w_m \nabla \log p_{0t}(\boldsymbol{u}|\boldsymbol{u}^{(m)}), \quad w_m = \frac{p_{0t}(\boldsymbol{u}|\boldsymbol{u}^{(m)})}{\sum_m p_{0t}(\boldsymbol{u}|\boldsymbol{u}^{(m)})}, \tag{7.15}$$

which can be interpreted as a weighted sum of M score functions associated with Dirac distributions. This is illustrated in Fig. 7.2. In the red color region where the weights $\{w_m\}$ are dominated by one specific data $\boldsymbol{u}^{(m^*)}$ and thus $\nabla \log p_t(\boldsymbol{u}) \approx \nabla \log p_{0t}(\boldsymbol{u}|\boldsymbol{u}^{(m^*)})$. Moreover, in the green region different modes have similar $\nabla \log p_{0t}(\boldsymbol{u}|\boldsymbol{u}^{(m)})$ as all of them are close to Gaussian and can be approximated by any condition score of any mode. The $\{\epsilon_{\mathrm{GT}}(\boldsymbol{u}(t), t)\}$ trajectories in Fig. 7.2 validate our hypothesis as we have very smooth curves at the beginning and ending period. The phenomenon that score of realistic datasets can be locally approximated by the score of one datapoint partially justifies the Dirac distribution assumption in Prop 4 and 6 and the effectiveness of DDIMs.

7.4 Generalize and improve DDIM

The DDIM is specifically designed for DDPMs. Can we generalize it to other DMs? With the insights in Prop 4 and 6, it turns out that with a carefully chosen $\boldsymbol{K}_\tau$, we can generalize DDIMs to any DMs with general drift and diffusion. We coin the resulted algorithm the *Generalized DDIM (gDDIM)*.

7.4.1 Deterministic gDDIM

Toy dataset: Motivated by Prop 4, we ask whether there exists an ϵ_{GT} that remains constant along a solution to the probability flow ODE Eq. (7.7). We start with a special case with initial distribution $p_0(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}_0, \Sigma_0)$. It turns out that any solution to Eq. (7.7) is of the form

$$\boldsymbol{u}(t) = \Psi(t, 0)\boldsymbol{u}_0 + \boldsymbol{R}_t\epsilon \tag{7.16}$$

with a constant ϵ and a time-varying parameterization coefficients $\boldsymbol{R}_t \in \mathbb{R}^{D\times D}$ that satisfies $\boldsymbol{R}_0\boldsymbol{R}_0^T = \Sigma_0$ and

$$\frac{d\boldsymbol{R}_t}{d\boldsymbol{t}} = (\boldsymbol{F}_t + \frac{1}{2}\boldsymbol{G}_t\boldsymbol{G}_t^T\Sigma_t^{-1})\boldsymbol{R}_t. \tag{7.17}$$

Here $\Psi(t, s)$ is the transition matrix associated with $\boldsymbol{F}_\tau$; it is the solution to $\frac{\partial\Psi(t,s)}{\partial t} = \boldsymbol{F}_t\Psi(t, s), \Psi(s, s) = \boldsymbol{I}_D$. Interestingly, $\boldsymbol{R}_t$ satisfies $\boldsymbol{R}_t\boldsymbol{R}_t^T = \Sigma_t$ like $\boldsymbol{K}_t$ in Eq. (7.4). We remark $\boldsymbol{K}_t = \sqrt{1-\alpha_t}\boldsymbol{I}_d$ is a solution to Eq. (7.17) when the DM is specialized to DDPM. Based on Eq. (7.16) and Eq. (7.17), we extend Prop 4 to more general DMs.

Proposition 7. *Assume the data distribution $p_0(\boldsymbol{u})$ is $\mathcal{N}(\boldsymbol{u}_0, \Sigma_0)$. Let $\boldsymbol{u}(t)$ be an arbitrary solution to the probability flow ODE Eq.* (7.7) *with the ground truth score, then $\epsilon_{\mathrm{GT}}(\boldsymbol{u}(t), t) := -\boldsymbol{R}_t^T\nabla\log p_t(\boldsymbol{u}(t))$ remains constant along $\boldsymbol{u}(t)$.*

Note that Prop 7 is slightly more general than Prop 4 in the sense that the initial distribution p_0 is a Gaussian instead of a Dirac. Diffusion models with augmented states such as CLD use a Gaussian distribution on the velocity channel for each data point. Thus, when there is a single data point, the initial distribution is a Gaussian instead of a Dirac distribution. A direct consequence of Prop 7 is that we can conduct accurate sampling in one step in the toy example since we can recover the score along any simulated trajectory given its value at $t = T$, if $\boldsymbol{K}_t$ in Eq. (7.4) is set to be $\boldsymbol{R}_t$. This choice $\boldsymbol{K}_t = \boldsymbol{R}_t$ will make a huge difference in sampling quality as we will show later. The fact provides guidance to design an efficient sampling scheme for realistic data.

Realistic dataset: As the accurate score is not available for realistic datasets, we need to use learned score $\boldsymbol{s}_\theta(\boldsymbol{u}, t)$ for sampling. With our new parameterization $\epsilon_\theta(\boldsymbol{u}, t) = -\boldsymbol{R}_t^T \boldsymbol{s}_\theta(\boldsymbol{u}, t)$ and the approximation $\tilde{\epsilon}_\theta(\boldsymbol{u}, \tau) = \epsilon_\theta(\boldsymbol{u}(t), t)$ for $\tau \in [t - \Delta t, t]$, we reach the update step for deterministic gDDIM by solving probability flow with approximator $\tilde{\epsilon}_\theta(\boldsymbol{u}, \tau)$ exactly as

$$\boldsymbol{u}(t-\Delta t) = \Psi(t-\Delta t, t)\boldsymbol{u}(t) + [\int_t^{t-\Delta t} \frac{1}{2}\Psi(t-\Delta t, \tau)\boldsymbol{G}_\tau \boldsymbol{G}_\tau^T \boldsymbol{R}_\tau^{-T} d\tau]\epsilon_\theta(\boldsymbol{u}(t), t), \quad (7.18)$$

Multistep predictor-corrector for ODE: Inspired by [207], we further boost the sampling efficiency of gDDIM by combining Eq. (7.18) with multistep methods [23, 207, 195]. We derive multistep *predictor-corrector* methods to reduce the number of steps while retaining accuracy [211, 212]. Empirically, we found that using more NFEs in predictor leads to better performance when the total NFE is small. Thus, we only present multistep predictor for deterministic gDDIM. For time discretization grid $\{t_i\}_{i=0}^N$ where $t_0 = 0, t_N = T$, the q-th step predictor from t_i to t_{i-1} in term of ϵ_θ parameterization reads

$$\boldsymbol{u}(t_{i\text{-}1}) = \Psi(t_{i-1}, t_i)\boldsymbol{u}(t_i) + \sum_{j=0}^{q-1}[\mathrm{C}_{ij}\epsilon_\theta(\boldsymbol{u}(t_{i+j}), t_{i+j})], \tag{7.19a}$$

$$\mathrm{C}_{ij} = \int_{t_i}^{t_{i-1}} \frac{1}{2}\Psi(t_{i-1}, \tau)\boldsymbol{G}_\tau \boldsymbol{G}_\tau^T \boldsymbol{R}_\tau^{-T} \prod_{k \neq j}[\frac{\tau - t_{i+k}}{t_{i+j} - t_{i+k}}]d\tau. \tag{7.19b}$$

We note that coefficients in Eqs. (7.18) and (7.19b) for general DMs can be calculated efficiently using standard numerical solvers if closed-form solutions are not available.

7.4.2 Stochastic gDDIM

Following the same spirits, we generalize Prop 6

Proposition 8. *Assume the data distribution* $p_0(\boldsymbol{u})$ *is* $\mathcal{N}(\boldsymbol{u}_0, \Sigma_0)$. *Given any evaluation of the score function* $\nabla \log p_s(\boldsymbol{u}(s))$, *one can recover* $\nabla \log p_t(\boldsymbol{u})$ *for any* $t, \boldsymbol{u}$ *as*

$$\nabla \log p_t(\boldsymbol{u}) = \Sigma_t^{-1}\Psi(t, s)\Sigma_s \nabla \log p_s(\boldsymbol{u}(s)) - \Sigma_t^{-1}[\boldsymbol{u} - \Psi(t, s)\boldsymbol{u}(s)]. \tag{7.20}$$

Prop 8 is not surprising; in our example, the score has a closed form. Eq. (7.20) not only provides an accurate score estimation for our toy dataset, but also serves as a score approximator for realistic data.

Realistic dataset: Based on Eq. (7.20), with the parameterization $\boldsymbol{s}_\theta(\boldsymbol{u}, \tau) = -\boldsymbol{R}_\tau^{-T}\epsilon_\theta(\boldsymbol{u}, \tau)$, we propose the following gDDIM approximator $\tilde{\epsilon}_\theta(\boldsymbol{u}, \tau)$ for $\epsilon_\theta(\boldsymbol{u}, \tau)$

$$\tilde{\epsilon}_\theta(\boldsymbol{u}, \tau) = \boldsymbol{R}_\tau^{-1}\Psi(\tau, s)\boldsymbol{R}_s\epsilon_\theta(\boldsymbol{u}(s), s) + \boldsymbol{R}_\tau^{-1}[\boldsymbol{u} - \Psi(\tau, s)\boldsymbol{u}(s)]. \tag{7.21}$$

Proposition 9. *With the parameterization* $\epsilon_\theta(\boldsymbol{u}, t) = -\boldsymbol{R}_t^T \boldsymbol{s}_\theta(\boldsymbol{u}, t)$ *and the approximator* $\tilde{\epsilon}_\theta(\boldsymbol{u}, \tau)$ *in Eq.* (7.21), *the solution to Eq.* (7.6) *satisfies*

$$\boldsymbol{u}(t) \sim \mathcal{N}(\Psi(t, s)\boldsymbol{u}(s) + [\hat{\Psi}(t, s) - \Psi(t, s)]\boldsymbol{R}_s\epsilon_\theta(\boldsymbol{u}(s), s), \boldsymbol{P}_{st}), \tag{7.22}$$

where $\hat{\Psi}(t, s)$ *is the transition matrix associated with* $\hat{\boldsymbol{F}}_\tau := \boldsymbol{F}_\tau + \frac{1+\lambda^2}{2}\boldsymbol{G}_\tau \boldsymbol{G}_\tau^T \Sigma_\tau^{-1}$ *and the covariance matrix* $\boldsymbol{P}_{st}$ *solves*

$$\frac{d\boldsymbol{P}_{s\tau}}{d\tau} = \hat{\boldsymbol{F}}_\tau \boldsymbol{P}_{s\tau} + \boldsymbol{P}_{s\tau} \hat{\boldsymbol{F}}_\tau^T + \lambda^2 \boldsymbol{G}_\tau \boldsymbol{G}_\tau^T, \quad \boldsymbol{P}_{ss} = 0. \tag{7.23}$$

Our stochastic gDDIM then uses Eq. (7.22) for update. Though the stochastic gDDIM and the deterministic gDDIM look quite different from each other, there exists a connection between them.

Proposition 10. *Eq.* (7.22) *in stochastic gDDIM reduces to Eq.* (7.18) *in deterministic gDDIM when* $\lambda = 0$.

7.5 Experiments

As gDDIM reduces to DDIM for VPSDE and DDIM proves very successful, we validate the generation and effectiveness of gDDIM on CLD and BDM. We design experiments to answer the following questions. How to verify Prop 7 and 8 empirically? Can gDDIM improve sampling efficiency compared with existing works? What differences do the choice of λ and $\boldsymbol{K}_t$ make?

Choice of $\boldsymbol{K}_t$: A key of gDDIM is the special choice $\boldsymbol{K}_t = \boldsymbol{R}_t$ which is obtained via solving Eq. (7.17). In CLD, [196] choose $\boldsymbol{K}_t = \boldsymbol{L}_t$ based on Cholesky decomposition of Σ_t and it does not obey Eq. (7.17). As it is shown in Fig. 7.1, on real datasets with a trained score model, we randomly pick pixel locations and check the pixel value and ϵ_θ output along the solutions to the probability flow ODE produced by the high-resolution ODE solver. With the choice $\boldsymbol{K}_t = \boldsymbol{L}_t$, $\epsilon_\theta^{(\boldsymbol{L})}(\boldsymbol{u}, t; \boldsymbol{v})$ suffers from oscillation like $\boldsymbol{x}$ value along time. However, $\epsilon_\theta^{(\boldsymbol{R})}(\boldsymbol{u}, t)$ is much more flat. We further compare samples generated by $\boldsymbol{L}_t$ and $\boldsymbol{R}_t$ parameterizaiton in Tab. 7.1, where both use the multistep exponential solver in Eq. (7.19).

Choice of λ: We further conduct a study with different λ values. Note that polynomial

Table 7.1: $\boldsymbol{L}_t$ vs $\boldsymbol{R}_t$ (Our) on CLD

	FID at different NFE			
$\boldsymbol{K}_t$	20	30	40	50
$\boldsymbol{L}_t$	368	167	4.12	3.31
$\boldsymbol{R}_t$	3.90	2.64	2.37	2.26

Table 7.2: λ and integrators choice with NFE=50

	FID at different λ					
Method	0.0	0.1	0.3	0.5	0.7	1.0
gDDIM	5.17	5.51	12.13	33	41	49
EM	346	168	137	89	45	57

extrapolation in Eq. (7.19) is not used here even when $\lambda = 0$. As it is shown in Tab. 7.2, increasing λ deteriorates the sample quality, demonstrating our claim that deterministic DDIM has better performance than its stochastic counterpart when a small NFE is used. We also find stochastic gDDIM significantly outperforms EM, which indicates the effectiveness of the approximation Eq. (7.21).

Accelerate various DMs: We present a comparison among various DMs and various sampling algorithms. To make a fair comparison, we compare three DMs with similar size networks while retaining other hyperparameters from their original works. We make two modifications to DDPM, including continuous-time training [4] and smaller stop sampling time [132], which help improve sampling quality empirically. For BDM, we note [25] only supports the ancestral sampling algorithm, a variant of EM algorithm. With reformulated noising and denoising process as SDE Eq. (7.11), we can generate samples by solving corresponding SDE/ODEs. The sampling quality of gDDIM with 50 NFE can outperform the original ancestral sampler with 1000 NFE, more than 20 times acceleration.

7.6 Conclusions and limitations

Contribution: The more structural knowledge we leverage, the more efficient algorithms we obtain. In this work, we provide a clean interpretation of DDIMs based on the manifold hypothesis and the sparsity property on realistic datasets. This new perspective unboxes

Table 7.3: Acceleration on various DMs with similar training pipelines and architecture. For RK45, we tune its tolerance hyperparameters so that the real NFE is close but not equal to the given NFE. †: pre-trained model from [4]. ††: [132] apply Heun method in rescaled DM, which is essentially a variant of DEIS [207]

		FID (↓) under different NFE				
DM	Sampler	10	20	50	100	1000
DDPM†	EM	>100	>100	31.2	12.2	2.64
	Prob.Flow, RK45	>100	52.5	6.62	2.63	**2.56**
	2nd Heun††	66.25	6.62	2.65	2.57	**2.56**
	gDDIM	**4.17**	**3.03**	**2.59**	**2.56**	**2.56**
BDM	Ancestral sampling	>100	>100	29.8	9.73	2.51
	Prob.Flow, RK45	>100	68.2	7.12	2.58	**2.46**
	gDDIM	**4.52**	**2.97**	**2.49**	**2.47**	**2.46**
CLD	EM	>100	>100	57.72	13.21	2.39
	Prob.Flow, RK45	>100	>100	31.7	4.56	**2.25**
	gDDIM	**13.41**	**3.39**	**2.26**	**2.26**	**2.25**

the numerical discretization used in DDIM and explains the advantage of ODE-based sampler over SDE-based when NFE is small. Based on this interpretation, we extend DDIMs to general diffusion models. The new algorithm, gDDIM, only requires a tiny but elegant modification to the parameterization of the score model and improves sampling efficiency drastically. We conduct extensive experiments to validate the effectiveness of our new sampling algorithm. **Limitation**: There are several promising future directions. First, though gDDIM is designed for general DMs, we only verify it on three DMs. It is beneficial to explore more efficient diffusion processes for different datasets, in which we believe gDDIM will play an important role in designing sampling algorithms. Second, more investigations are needed to design an efficient sampling algorithm by exploiting more structural knowledge in DMs. The structural knowledge can originate from different sources such as different modalities of datasets, and mathematical structures presented in specific diffusion processes.

CHAPTER 8
CONCLUSION AND FUTURE WORKS

8.1 Conclusion

In this thesis, I investigate the *representation, learning, sampling, and inference* tasks of probability modeling, which involve modeling a probabilistic distribution for specific samples of interest. Specifically, we delve into problems built upon representations via stochastic differential equations (SDEs), aiming to explore efficient and scalable algorithms for high-dimensional data.

We initially provide a scalable learning framework, named *Diffusion Normalizing Flow*, which enables learning to generate based on neural SDEs for medium-sized tasks. This approach offers manageable training costs and reduced inference time compared to the standard diffusion model's stochastic sampling approach.

Regarding scaling up with respect to data, we aim to learn generators that are more adaptable to data, which do not require complete data during training. To this end, we propose 'Diffusion Collage,' a novel algorithm capable of flexibly generating large content, yet trained only on incomplete data. Built upon conditional independence and the Bethe approximation, Diffusion Collage approximates the score of the joint distribution by combining scores of multiple marginal distributions. This method demonstrates its versatility in generating infinite images, various image editing tasks, long-duration motion generation, and 360-degree image synthesis. Additionally, we introduce the Path Integral Sampler, which enables learning neural SDEs to drive samples from a direct point to the target distribution, accessing only the density information of the target distribution.

To scale up with computation and build a computation-friendly approach toward probability modeling, we introduce a more efficient sampling algorithm to draw novel samples from pre-trained diffusion generators. By leveraging exponential integrators and appropriate parameterization, we can circumvent unnecessary discretization and reduce the number of function evaluations while retaining sampling quality. We further enhance sampling performance by introducing polynomial extrapolation. This extrapolation comes at zero cost by leveraging function evaluations from previous network assessments.

As a breakthrough realization of probability modeling, diffusion-based generative models have revolutionized content generation, especially in the field of visual content. I am convinced that diffusion-based probability modeling marks merely the commencement of a broader exploration. Numerous unknown aspects in this field persist, presenting ample opportunities for future investigation and discovery.

8.2 Toward more efficient and scalable probability modeling

Despite the significant advancements achieved in multiple fields through SDE-based probability modeling, several challenges continue to present themselves. Additionally, numerous theoretical and empirical research questions remain open, warranting further exploration. In this section, I outline a few of these persistent challenges that are particularly deserving of deeper investigation.

Representations for probability modeling In the realm of probabilistic modeling algorithms, the development and effectiveness of most learning, sampling, and inference algorithms are typically tailored to and most effective for specific representation approaches. For instance, transferring techniques from diffusion models to enhance GAN (Generative Adversarial Network) performance is non-trivial and rarely straightforward. The notable success of diffusion models over previous generative models can be largely attributed to their expressive approach in representing probability distributions and scalable training built upon the representation. Consequently, it becomes a compelling endeavor to explore approaches that might surpass the current capabilities of diffusion models in probabilistic modeling. Several limitations of diffusion generative models stem from their representation; for example, the slow generation speed is a consequence of their iterative denoising scheme. Several recent studies, such as those by Song *et al* [213] and Liu *et al* [214], have explored better coupling between Gaussian noise and samples from the target distribution. However, these approaches often result in faster sampling speeds at the cost of reduced generation quality. Furthermore, given the prevailing belief that realistic data resides within a low-dimensional manifold, it seems promising to investigate the potential of more meaningful and compact latent spaces, as opposed to relying solely on pure noise.

Maximum likelihood estimation and metrics Maximum Likelihood Estimation (MLE) is a widely employed method in learning probability models, encompassing neural SDEs and diffusion models. However, MLE does not always yield the optimal quality in the resulting models. For example, diffusion models trained using MLE have demonstrated inferior sampling quality in image generation tasks compared to those trained with weighted likelihood approaches, as noted by Song *et. al.* [206]. Furthermore, models like Normalizing Flow and VAEs, when trained using MLE, often exhibit unsatisfactory sampling quality. A commonly cited explanation for this is that the visual texture in images, which is highly redundant, requires significantly fewer bits for compression. In contrast, MLE-like loss functions are the default choice for language modeling during the pre-training stage.

The inadequacy of MLE loss in training visual generative models presents a significant challenge: the absence of reliable metrics to compare various models and quantify improvements, as well as to guide principled optimization of neural network parameters. For instance, in diffusion-based image generative models, researchers cannot rely on a perplexity loss, as in language modeling, and instead must depend on empirical metrics such as FID and KID for performance evaluation. These metrics, however, are sometimes misaligned with human judgment and can be sensitive to nuances in engineering implementation [215]. Moving forward, finding reliable and efficient metrics to measure improvement is critical for making measurable progress in probabilistic modeling.

Efficient and versatile architecture Large-scale probability models, such as 1k image diffusion models, are typically trained with large neural networks, which are computationally expensive for both training and inference. For instance, the current state-of-the-art diffusion model [20] consists of 9.1 billion parameters and takes over 30 seconds to generate a single image on an Nvidia A100.

A promising research direction is to investigate network architecture to reduce the computational costs of generators while maintaining the quality of the learned model. Additionally, the generative network architectures differ across various modalities and require extensive architecture search. Given recent works demonstrating the applicability of transformer networks across various tasks [216], exploring the feasibility of designing a general architecture that can be used across different modalities is worthwhile.

The scaling law and generalization Diffusion models, akin to large language models, are increasingly being utilized as foundational frameworks for large-scale content generation. However, there exists a notable disparity in research: while language models have been extensively empirically studied [217, 218], there is a paucity of studies exploring the scaling laws and generalization capabilities of diffusion models, particularly in computer vision. This gap is particularly surprising given the importance of understanding the scaling law in this domain. When it comes to visual content generation, how the quality of samples generated by diffusion models and their generalization ability are influenced by the model's parameter count and the volume of training data remains an unresolved question. For the development of large foundational models, the ability to finely tune and control a model, and to scale it predictably and effectively, is of paramount importance.

REFERENCES

[1] X. Li, T.-K. L. Wong, R. T. Q. Chen, and D. Duvenaud, “Scalable gradients for stochastic differential equations,” *arXiv preprint arXiv:2001.01328*, Jan. 2020. arXiv: 2001.01328 `[cs.LG]`.

[2] B. Tzen and M. Raginsky, “Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit,” *arXiv preprint arXiv:1905.09883*, 2019.

[3] P. Kidger, J. Morrill, J. Foster, and T. Lyons, “Neural controlled differential equations for irregular time series,” *arXiv preprint arXiv:2005.08926*, May 2020. arXiv: 2005.08926 `[cs.LG]`.

[4] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, “Score-Based Generative Modeling through Stochastic Differential Equations,” *ArXiv preprint*, vol. abs/2011.13456, 2020.

[5] Q. Zhang and Y. Chen, “Diffusion normalizing flow,” *arXiv preprint arXiv:2110.07579*, Oct. 2021. arXiv: 2110.07579 `[cs.LG]`.

[6] L. Dinh, J. Sohl-Dickstein, and S. Bengio, “Density estimation using real NVP,” *arXiv preprint arXiv:1605.08803*, May 2016. arXiv: 1605.08803 `[cs.LG]`.

[7] J. Ho, A. Jain, and P. Abbeel, “Denoising diffusion probabilistic models,” in *Advances in Neural Information Processing Systems*, vol. 2020-Decem, 2020, ISBN: 2006.11239v2. arXiv: 2006.11239.

[8] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 684–10 695.

[9] P. Dhariwal and A. Nichol, “Diffusion models beat GANs on image synthesis,” *arXiv preprint arXiv:2105.05233*, May 2021. arXiv: 2105.05233 `[cs.LG]`.

[10] A. Nichol *et al.*, “Glide: Towards photorealistic image generation and editing with text-guided diffusion models,” *arXiv preprint arXiv:2112.10741*, 2021.

[11] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, *Hierarchical text-conditional image generation with clip latents*, 2022.

[12] E. Hoogeboom, D. Nielsen, P. Jaini, P. Forré, and M. Welling, “Argmax flows and multinomial diffusion: Learning categorical distributions,” *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[13] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. van den Berg, "Structured denoising diffusion models in discrete State-Spaces," *arXiv preprint arXiv:2107.03006*, Jul. 2021. arXiv: 2107.03006 `[cs.LG]`.

[14] Z. Lyu, Z. Kong, X. Xu, L. Pan, and D. Lin, "A conditional point diffusion-refinement paradigm for 3d point cloud completion," *arXiv preprint arXiv:2112.03530*, 2021.

[15] B. Kawar, G. Vaksman, and M. Elad, "SNIPS: Solving noisy inverse problems stochastically," *arXiv preprint arXiv:2105.14951*, May 2021. arXiv: 2105.14951 `[eess.IV]`.

[16] Y. Song, L. Shen, L. Xing, and S. Ermon, "Solving inverse problems in medical imaging with score-based generative models," *arXiv preprint arXiv:2111.08005*, 2021.

[17] I. Goodfellow *et al.*, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[18] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

[19] C. Saharia *et al.*, "Photorealistic text-to-image diffusion models with deep language understanding," *arXiv preprint arXiv:2205.11487*, 2022.

[20] Y. Balaji *et al.*, "Ediffi: Text-to-image diffusion models with an ensemble of expert denoisers," *arXiv preprint arXiv:2211.01324*, 2022.

[21] Q. Zhang, J. Song, X. Huang, Y. Chen, and M.-y. Liu, "Diffcollage: Parallel generation of large content with diffusion models," in *CVPR*, 2023.

[22] Q. Zhang and Y. Chen, "Path integral sampler: A stochastic control approach for sampling," in *International Conference on Learning Representations*, 2022.

[23] M. Hochbruck and A. Ostermann, "Exponential integrators," *Acta Numerica*, vol. 19, pp. 209–286, 2010.

[24] Q. Zhang and Y. Chen, "Fast sampling of diffusion models with exponential integrator," *arXiv preprint arXiv:2204.13902*, Apr. 2022. arXiv: 2204.13902 `[cs.LG]`.

[25] E. Hoogeboom and T. Salimans, "Blurring diffusion models," *arXiv preprint arXiv:2209.05557*, 2022.

[26] S. Rissanen, M. Heinonen, and A. Solin, "Generative modelling with inverse heat dissipation," *arXiv preprint arXiv:2206.13397*, 2022.

[27] T. Dockhorn, A. Vahdat, and K. Kreis, "Score-Based generative modeling with Critically-Damped langevin diffusion," *arXiv preprint arXiv:2112.07068*, Dec. 2021. arXiv: 2112.07068 `[stat.ML]`.

[28] Q. Zhang, M. Tao, and Y. Chen, "Gddim: Generalized denoising diffusion implicit models," *arXiv preprint arXiv:2206.05564*, Jun. 2022.

[29] Y. Bengio and S. Bengio, "Modeling high-dimensional discrete data with multi-layer neural networks," *Advances in Neural Information Processing Systems*, vol. 12, 1999.

[30] M. Germain, K. Gregor, I. Murray, and H. Larochelle, "Made: Masked autoencoder for distribution estimation," *arXiv preprint arXiv:1502.03509*, Feb. 2015. arXiv: 1502.03509 `[cs.LG]`.

[31] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *International conference on machine learning*, PMLR, 2016, pp. 1747–1756.

[32] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *arXiv preprint arXiv:1806.07366*, Jun. 2018. arXiv: 1806.07366 `[cs.LG]`.

[33] D. P. Kingma and M. Welling, "Auto-Encoding variational bayes," *arXiv preprint arXiv:1312.6114v10*, Dec. 2013. arXiv: 1312.6114v10 `[stat.ML]`.

[34] K. Talwar, "Computational separations between sampling and optimization," *arXiv preprint arXiv:1911.02074*, 2019.

[35] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," *arXiv preprint arXiv:1505.05770*, May 2015. arXiv: 1505.05770 `[stat.ML]`.

[36] H. Wu, J. Köhler, and F. Noé, "Stochastic normalizing flows," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5933–5944, 2020.

[37] S. Särkkä and A. Solin, *Applied stochastic differential equations*. Cambridge University Press, 2019, vol. 10.

[38] X. Liu, T. Xiao, S. Si, Q. Cao, S. Kumar, and C.-J. Hsieh, "Neural sde: Stabilizing neural ode networks with stochastic noise," *arXiv preprint arXiv:1906.02355*, 2019.

[39] B. Tzen and M. Raginsky, "Theoretical guarantees for sampling and inference in generative models with latent diffusions," in *Conference on Learning Theory*, PMLR, 2019, pp. 3084–3114.

[40] M. Giles and P. Glasserman, "Smoking adjoints: Fast monte carlo greeks," *Risk*, vol. 19, no. 1, pp. 88–92, 2006.

[41] E. Gobet and R. Munos, "Sensitivity analysis using itô–malliavin calculus and martingales, and application to stochastic optimal control," *SIAM Journal on control and optimization*, vol. 43, no. 5, pp. 1676–1713, 2005.

[42] J. Yang and H. J. Kushner, "A monte carlo method for sensitivity analysis and parametric optimization of nonlinear stochastic systems," *SIAM journal on control and optimization*, vol. 29, no. 5, pp. 1216–1249, 1991.

[43] K. Claessen and M. H. Pałka, "Splittable pseudorandom number generators using cryptographic hashing," *ACM SIGPLAN Notices*, vol. 48, no. 12, pp. 47–58, 2013.

[44] P. Kidger, J. Foster, X. C. Li, and T. Lyons, "Efficient and accurate gradients for neural sdes," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 747–18 761, 2021.

[45] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *arXiv preprint arXiv:2006.11239*, Jun. 2020. arXiv: 2006.11239 `[cs.LG]`.

[46] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," *arXiv preprint arXiv:1907.05600*, Jul. 2019. arXiv: 1907.05600 `[cs.LG]`.

[47] Y. Song and S. Ermon, "Improved techniques for training Score-Based generative models," *arXiv preprint arXiv:2006.09011*, Jun. 2020. arXiv: 2006.09011 `[cs.LG]`.

[48] B. D. Anderson, "Reverse-time diffusion equation models," *Stochastic Process. Appl.*, vol. 12, no. 3, pp. 313–326, May 1982.

[49] A. Hyvärinen and P. Dayan, "Estimation of non-normalized statistical models by score matching.," *Journal of Machine Learning Research*, vol. 6, no. 4, 2005.

[50] P. Vincent, "A connection between score matching and denoising autoencoders," *Neural computation*, vol. 23, no. 7, pp. 1661–1674, Jul. 2011.

[51] T. B. Brown *et al.*, "Language models are Few-Shot learners," *arXiv preprint arXiv:2005.14165*, May 2020. arXiv: 2005.14165 `[cs.CL]`.

[52] A. Vaswani *et al.*, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, Jun. 2017. arXiv: 1706.03762 `[cs.CL]`.

[53] R. Child, "Very deep VAEs generalize autoregressive models and can outperform them on images," *arXiv preprint arXiv:2011.10650*, Nov. 2020. arXiv: 2011.10650 [cs.LG].

[54] A. Vahdat and J. Kautz, "NVAE: A deep hierarchical variational autoencoder," *arXiv preprint arXiv:2007.03898*, Jul. 2020. arXiv: 2007.03898 [stat.ML].

[55] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[56] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International Conference on Machine Learning*, PMLR, 2015, pp. 1530–1538.

[57] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, "Flow++: Improving flow-based generative models with variational dequantization and architecture design," in *International Conference on Machine Learning*, PMLR, 2019, pp. 2722–2730.

[58] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.

[59] Y. Deng, A. Bakhtin, M. Ott, A. Szlam, and M. Ranzato, "Residual energy-based models for text generation," *arXiv preprint arXiv:2004.11714*, 2020.

[60] A. v. d. Oord *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[61] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," *arXiv preprint arXiv:1807.03039*, 2018.

[62] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *arXiv preprint arXiv:1806.07366*, 2018.

[63] W. Grathwohl, R. T. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "Ffjord: Free-form continuous dynamics for scalable reversible generative models," *arXiv preprint arXiv:1810.01367*, 2018.

[64] R. Cornish, A. Caterini, G. Deligiannidis, and A. Doucet, "Relaxing bijectivity constraints with continuously indexed normalising flows," in *International Conference on Machine Learning*, PMLR, 2020, pp. 2133–2143.

[65] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International Conference on Machine Learning*, PMLR, 2015, pp. 2256–2265.

[66] B. D. Anderson, “Reverse-time diffusion equation models,” *Stochastic Processes and their Applications*, vol. 12, no. 3, pp. 313–326, 1982.

[67] Y. Song and S. Ermon, “Generative modeling by estimating gradients of the data distribution,” in *Advances in Neural Information Processing Systems*, 2019, pp. 11 895–11 907.

[68] Y. Song and S. Ermon, “Improved techniques for training score-based generative models,” *arXiv preprint arXiv:2006.09011*, 2020.

[69] F. Bordes, S. Honari, and P. Vincent, “Learning to generate samples from noise through infusion training,” *arXiv preprint arXiv:1703.06975*, 2017.

[70] H. Wu, J. Köhler, and F. Noe, “Stochastic normalizing flows,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 5933–5944.

[71] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[72] E. Nelson, *Dynamical theories of Brownian motion*. Princeton university press, 2020.

[73] Y. Song, J. Sohl-Dickstein, D. Kingma, A. Kumar, S. Ermon, and B. Poole, “Score-based generative modeling through stochastic differential equations,” in *International Conference on Learning Representations*, 2021.

[74] A. Paszke *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” *arXiv preprint arXiv:1912.01703*, 2019.

[75] S. T. Roweis and L. K. Saul, “Nonlinear dimensionality reduction by locally linear embedding,” *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[76] H. Narayanan and S. Mitter, “Sample complexity of testing the manifold hypothesis,” in *Proceedings of the 23rd International Conference on Neural Information Processing Systems-Volume 2*, 2010, pp. 1786–1794.

[77] G. Papamakarios, T. Pavlakou, and I. Murray, “Masked autoregressive flow for density estimation,” *arXiv preprint arXiv:1705.07057*, 2017.

[78] C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville, “Neural autoregressive flows,” in *International Conference on Machine Learning*, PMLR, 2018, pp. 2078–2087.

[79] M. Germain, K. Gregor, I. Murray, and H. Larochelle, "Made: Masked autoencoder for distribution estimation," in *International Conference on Machine Learning*, PMLR, 2015, pp. 881–889.

[80] J. Oliva *et al.*, "Transformation autoregressive networks," in *International Conference on Machine Learning*, PMLR, 2018, pp. 3898–3907.

[81] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[82] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[83] A. Jolicoeur-Martineau, R. Piché-Taillefer, R. T. d. Combes, and I. Mitliagkas, "Adversarial score matching and improved sampling for image generation," *arXiv preprint arXiv:2009.05475*, 2020.

[84] B. Efron, "Tweedie's formula and selection bias," *Journal of the American Statistical Association*, vol. 106, no. 496, pp. 1602–1614, 2011.

[85] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *arXiv preprint arXiv:1706.08500*, 2017.

[86] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," *arXiv preprint arXiv:2102.09672*, 2021.

[87] R. T. Q. Chen, J. Behrmann, D. Duvenaud, and J. Jacobsen, "Residual flows for invertible generative modeling," in *Advances in Neural Information Processing Systems*, 2019.

[88] L. Dinh, J. Sohl-Dickstein, H. Larochelle, and R. Pascanu, "A rad approach to deep mixture models," *arXiv preprint arXiv:1903.07714*, 2019.

[89] R. Cai *et al.*, "Learning gradient fields for shape generation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[90] Z. Kong, W. Ping, J. Huang, K. Zhao, and B. Catanzaro, "Diffwave: A versatile diffusion model for audio synthesis," *arXiv preprint arXiv:2009.09761*, 2020.

[91] M. Giles and P. Glasserman, "Smoking adjoints: Fast monte carlo greeks," *Risk*, vol. 19, no. 1, pp. 88–92, 2006.

[92] E. Gobet and R. Munos, "Sensitivity analysis using itô–malliavin calculus and martingales, and application to stochastic optimal control," *SIAM Journal on control and optimization*, vol. 43, no. 5, pp. 1676–1713, 2005.

[93] X. Li, T.-K. L. Wong, R. T. Chen, and D. Duvenaud, "Scalable gradients for stochastic differential equations," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 3870–3882.

[94] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," *arXiv preprint arXiv:2103.00020*, Feb. 2021. arXiv: 2103.00020 [cs.CV].

[95] C. Saharia *et al.*, "Palette: Image-to-image diffusion models," in *ACM SIGGRAPH 2022 Conference Proceedings*, 2022, pp. 1–10.

[96] P. Esser, R. Rombach, and B. Ommer, "Taming transformers for high-resolution image synthesis," in *CVPR*, 2021.

[97] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *ICML*, 2015.

[98] Y. Tashiro, J. Song, Y. Song, and S. Ermon, "Csdi: Conditional score-based diffusion models for probabilistic time series imputation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24 804–24 816, 2021.

[99] X. Zeng *et al.*, "Lion: Latent point diffusion models for 3d shape generation," in *NeurIPS*, 2022.

[100] M. Ye, L. Wu, and Q. Liu, "First hitting diffusion models," *arXiv preprint arXiv:2209.01170*, 2022.

[101] L. Zhou, Y. Du, and J. Wu, "3D shape generation and completion through Point-Voxel diffusion," *arXiv preprint arXiv:2104.03670*, Apr. 2021. arXiv: 2104.03670 [cs.CV].

[102] X. L. Li, J. Thickstun, I. Gulrajani, P. Liang, and T. B. Hashimoto, "Diffusion-lm improves controllable text generation," *arXiv preprint arXiv:2205.14217*, 2022.

[103] C. Meng, Y. Song, J. Song, J. Wu, J.-Y. Zhu, and S. Ermon, "Sdedit: Image synthesis and editing with stochastic differential equations," *arXiv preprint arXiv:2108.01073*, 2021.

[104] B. Kawar *et al.*, "Imagic: Text-based real image editing with diffusion models," *arXiv preprint arXiv:2210.09276*, 2022.

[105] G. Couairon, J. Verbeek, H. Schwenk, and M. Cord, “DiffEdit: Diffusion-based semantic image editing with mask guidance,” *arXiv preprint arXiv:2210.11427*, 2022.

[106] D. Valevski, M. Kalman, Y. Matias, and Y. Leviathan, “UniTune: Text-driven image editing by fine tuning an image generation model on a single image,” *arXiv preprint arXiv:2210.09477*, 2022.

[107] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool, “Repaint: Inpainting using denoising diffusion probabilistic models,” in *CVPR*, 2022.

[108] B. Kawar, M. Elad, S. Ermon, and J. Song, “Denoising diffusion restoration models,” *arXiv preprint arXiv:2201.11793*, 2022.

[109] J. Choi, S. Kim, Y. Jeong, Y. Gwon, and S. Yoon, “ILVR: Conditioning method for denoising diffusion probabilistic models,” *arXiv preprint arXiv:2108.02938*, Aug. 2021. arXiv: 2108.02938 `[cs.CV]`.

[110] W. Nie, B. Guo, Y. Huang, C. Xiao, A. Vahdat, and A. Anandkumar, “Diffusion models for adversarial purification,” in *ICML*, 2022.

[111] J. Ho and T. Salimans, “Classifier-free diffusion guidance,” *arXiv preprint arXiv:2207.12598*, 2022.

[112] N. Ruiz, Y. Li, V. Jampani, Y. Pritch, M. Rubinstein, and K. Aberman, “Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation,” *arXiv preprint arxiv:2208.12242*, 2022.

[113] R. Gal *et al.*, “An image is worth one word: Personalizing text-to-image generation using textual inversion,” *arXiv preprint arXiv:2208.01618*, 2022.

[114] A. Hertz, R. Mokady, J. Tenenbaum, K. Aberman, Y. Pritch, and D. Cohen-Or, “Prompt-to-prompt image editing with cross attention control,” *arXiv preprint arXiv:2208.01626*, 2022.

[115] J. Song, C. Meng, and S. Ermon, “Denoising diffusion implicit models,” in *International Conference on Learning Representations*, 2021.

[116] B. Jing, G. Corso, R. Berlinghieri, and T. Jaakkola, “Subspace diffusion generative models,” *arXiv preprint arXiv:2205.01490*, 2022.

[117] C. H. Lin, C.-C. Chang, Y.-S. Chen, D.-C. Juan, W. Wei, and H.-T. Chen, “COCO-GAN: Generation by parts via conditional coordinating,” *arXiv preprint arXiv:1904.00284*, Mar. 2019. arXiv: 1904.00284 `[cs.LG]`.

[118] I. Skorokhodov, G. Sotnikov, and M. Elhoseiny, “Aligning latent and image spaces to connect the unconnectable,” in *ICCV*, 2021.

[119] C. H. Lin, H.-Y. Lee, Y.-C. Cheng, S. Tulyakov, and M.-H. Yang, “InfinityGAN: Towards infinite-pixel image synthesis,” *arXiv preprint arXiv:2104.03963*, 2021.

[120] E. Ntavelis, M. Shahbazi, I. Kastanis, R. Timofte, M. Danelljan, and L. Van Gool, “Arbitrary-scale image synthesis,” in *CVPR*, 2022.

[121] Ł. Struski, S. Knop, P. Spurek, W. Daniec, and J. Tabor, “LocoGAN—locally convolutional GAN,” *CVIU*, 2022.

[122] I. Anokhin, K. Demochkin, T. Khakhulin, G. Sterkin, V. Lempitsky, and D. Korzhenkov, “Image generators with conditionally-independent pixel synthesis,” in *CVPR*, 2021.

[123] I. Skorokhodov, S. Ignatyev, and M. Elhoseiny, “Adversarial generation of continuous images,” in *CVPR*, 2021.

[124] L. Chai, M. Gharbi, E. Shechtman, P. Isola, and R. Zhang, “Any-resolution training for high-resolution image synthesis,” in *ECCV*, 2022.

[125] H. Chung, B. Sim, and J. C. Ye, “Come-Closer-Diffuse-Faster: Accelerating conditional diffusion models for inverse problems through stochastic contraction,” *arXiv preprint arXiv:2112.05146*, Dec. 2021. arXiv: 2112.05146 `[eess.IV]`.

[126] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet, “Video diffusion models,” *arXiv preprint arXiv:2204.03458*, 2022.

[127] H. Chung, B. Sim, D. Ryu, and J. C. Ye, “Improving diffusion models for inverse problems using manifold constraints,” *arXiv preprint arXiv:2206.00941*, 2022.

[128] C. Wu *et al.*, “Nuwa-infinity: Autoregressive over autoregressive generation for infinite visual synthesis,” in *NeurIPS*, 2022.

[129] Y.-J. Chen, S.-I. Cheng, W.-C. Chiu, H.-Y. Tseng, and H.-Y. Lee, “Vector quantized image-to-image translation,” in *ECCV*, 2022.

[130] Z. Zhang *et al.*, “M6-ufc: Unifying multi-modal controls for conditional image synthesis,” *arXiv preprint arXiv:2105.14211*, 2021.

[131] H. Chang, H. Zhang, L. Jiang, C. Liu, and W. T. Freeman, “Maskgit: Masked generative image transformer,” in *CVPR*, 2022.

[132] T. Karras, M. Aittala, T. Aila, and S. Laine, “Elucidating the design space of diffusion-based generative models,” *arXiv preprint arXiv:2206.00364*, 2022.

[133] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[134] J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Generalized belief propagation,” in *NeurIPS*, 2001, pp. 689–695.

[135] J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Constructing free-energy approximations and generalized belief propagation algorithms,” *IEEE Transactions on information theory*, 2005.

[136] R. Singh, I. Haasler, Q. Zhang, J. Karlsson, and Y. Chen, “Inference with aggregate data: An optimal transport approach,” *arXiv preprint arXiv:2003.13933*, 2020.

[137] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu, “Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps,” *arXiv preprint arXiv:2206.00927*, 2022.

[138] D. Ryu and J. C. Ye, “Pyramidal denoising diffusion probabilistic models,” *arXiv preprint arXiv:2208.01864*, 2022.

[139] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, “GANs trained by a two Time-Scale update rule converge to a local nash equilibrium,” *arXiv preprint arXiv:1706.08500*, Jun. 2017. arXiv: 1706.08500 `[cs.LG]`.

[140] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, “LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop,” *arXiv preprint arXiv:1506.03365*, Jun. 2015. arXiv: 1506.03365 `[cs.CV]`.

[141] G. Tevet, S. Raab, B. Gordon, Y. Shafir, D. Cohen-Or, and A. H. Bermano, “Human motion diffusion model,” *arXiv preprint arXiv:2209.14916*, 2022.

[142] C. Guo *et al.*, “Generating diverse and natural 3d human motions from text,” in *CVPR*, 2022.

[143] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. J. Black, “Amass: Archive of motion capture as surface shapes,” in *ICCV*, 2019.

[144] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu, “Semantic image synthesis with spatially-adaptive normalization,” in *CVPR*, 2019.

[145] X. Huang, A. Mallya, T.-C. Wang, and M.-Y. Liu, “Multimodal conditional image synthesis with product-of-experts GANs,” in *ECCV*, 2022.

[146] C. Gao, J. Isaacson, and C. Krause, “I-flow: High-dimensional integration and sampling with normalizing flows,” *Machine Learning: Science and Technology*, vol. 1, no. 4, p. 045 023, 2020.

[147] K. A. Nicoli, S. Nakajima, N. Strodthoff, W. Samek, K.-R. Müller, and P. Kessel, “Asymptotically unbiased estimation of physical observables with neural samplers,” *Physical Review E*, vol. 101, no. 2, p. 023 304, 2020.

[148] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[149] P. Del Moral, A. Doucet, and A. Jasra, “Sequential Monte Carlo samplers,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 3, pp. 411–436, 2006.

[150] E. Nijkamp, M. Hill, S.-C. Zhu, and Y. N. Wu, “Learning non-convergent non-persistent short-run mcmc toward energy-based model,” *arXiv preprint arXiv:1904.09770*, 2019.

[151] R. Gao, Y. Song, B. Poole, Y. N. Wu, and D. P. Kingma, “Learning Energy-Based models by diffusion recovery likelihood,” *arXiv preprint arXiv:2012.08125*, Dec. 2020. arXiv: 2012.08125 `[cs.LG]`.

[152] S. Spanbauer, C. Freer, and V. Mansinghka, “Deep involutive generative models for neural mcmc,” *arXiv preprint arXiv:2006.15167*, 2020.

[153] Z. Li, Y. Chen, and F. T. Sommer, “A neural network mcmc sampler that maximizes proposal entropy,” *arXiv preprint arXiv:2010.03587*, 2020.

[154] Q. Liu, J. Lee, and M. Jordan, “A kernelized Stein discrepancy for goodness-of-fit tests,” in *International conference on machine learning*, PMLR, 2016, pp. 276–284.

[155] J. Gorham and L. Mackey, “Measuring sample quality with kernels,” in *International Conference on Machine Learning*, PMLR, 2017, pp. 1292–1301.

[156] J. Song, S. Zhao, and S. Ermon, “A-nice-MC: Adversarial training for MCMC,” *arXiv preprint arXiv:1706.07561*, 2017.

[157] M. Titsias and P. Dellaportas, “Gradient-based adaptive Markov chain Monte Carlo,” *Advances in Neural Information Processing Systems*, vol. 32, pp. 15 730–15 739, 2019.

[158] M. Pavon, “Stochastic control and nonequilibrium thermodynamical systems,” *Applied Mathematics and Optimization*, vol. 19, no. 1, pp. 187–202, 1989.

[159] P. Dai Pra, “A stochastic control approach to reciprocal diffusion processes,” *Applied mathematics and Optimization*, vol. 23, no. 1, pp. 313–329, 1991.

[160] C. Léonard, “A survey of the Schrödinger problem and some of its connections with optimal transport,” *Discrete & Continuous Dynamical Systems*, vol. 34, no. 4, p. 1533, 2014.

[161] Y. Chen, T. T. Georgiou, and M. Pavon, “Stochastic control liaisons: Richard Sinkhorn meets Gaspard Monge on a Schrödinger bridge,” *SIAM Review*, vol. 63, no. 2, pp. 249–313, 2021.

[162] Y. Chen, T. T. Georgiou, and M. Pavon, “On the relation between optimal transport and Schrödinger bridges: A stochastic control viewpoint,” *Journal of Optimization Theory and Applications*, vol. 169, no. 2, pp. 671–691, 2016.

[163] S. Mohamed and B. Lakshminarayanan, “Learning in implicit generative models,” *arXiv preprint arXiv:1610.03483*, Oct. 2016. arXiv: 1610.03483 `[stat.ML]`.

[164] M. Arbel, A. G. Matthews, and A. Doucet, “Annealed flow transport Monte Carlo,” *arXiv preprint arXiv:2102.07501*, 2021.

[165] S. Särkkä and A. Solin, *Applied stochastic differential equations*. Cambridge University Press, 2019, vol. 10.

[166] D. P. Bertsekas *et al.*, *Dynamic programming and optimal control: Vol. 1*. Athena scientific Belmont, 2000.

[167] L. C. Evans, “Partial differential equations,” *Graduate studies in mathematics*, vol. 19, no. 4, p. 7, 1998.

[168] B. Øksendal, “Stochastic differential equations,” in *Stochastic differential equations*, Springer, 2003, pp. 65–84.

[169] E. Theodorou, J. Buchli, and S. Schaal, “A generalized path integral control approach to reinforcement learning,” *The Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.

[170] E. A. Theodorou and E. Todorov, “Relative entropy and free energy dualities: Connections to path integral and KL control,” in *2012 ieee 51st ieee conference on decision and control (cdc)*, IEEE, 2012, pp. 1466–1473.

[171] S. Thijssen and H. Kappen, “Path integral control and state-dependent feedback,” *Physical Review E*, vol. 91, no. 3, p. 032 104, 2015.

[172] R. M. Neal, “Annealed importance sampling,” *Statistics and computing*, vol. 11, no. 2, pp. 125–139, 2001.

[173] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT Press, 2016, vol. 1.

[174] M. D. Hoffman and A. Gelman, “The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo.,” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1593–1623, 2014.

[175] S. T. Tokdar and R. E. Kass, “Importance sampling: A review,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 1, pp. 54–60, 2010.

[176] N. Chopin and O. Papaspiliopoulos, *An introduction to sequential Monte Carlo*. Springer, 2020.

[177] Y. Zhou, A. M. Johansen, and J. A. Aston, “Toward automatic model comparison: An adaptive sequential Monte Carlo approach,” *Journal of Computational and Graphical Statistics*, vol. 25, no. 3, pp. 701–726, 2016.

[178] Q. Liu and D. Wang, “Stein variational gradient descent: A general purpose bayesian inference algorithm,” in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., Curran Associates, Inc., 2016, pp. 2378–2386.

[179] C. Liu, J. Zhuo, and J. Zhu, “Understanding MCMC dynamics as flows on the wasserstein space,” *arXiv preprint arXiv:1902.00282*, Feb. 2019. arXiv: 1902.00282 `[stat.ML]`.

[180] C. W. Fox and S. J. Roberts, “A tutorial on variational Bayesian inference,” *Artificial intelligence review*, vol. 38, no. 2, pp. 85–95, 2012.

[181] M. Hoffman, P. Sountsov, J. V. Dillon, I. Langmore, D. Tran, and S. Vasudevan, “Neutra-lizing bad geometry in Hamiltonian Monte Carlo using neural transport,” *arXiv preprint arXiv:1903.03704*, 2019.

[182] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, “Probabilistic programming in python using pymc3,” *PeerJ Computer Science*, vol. 2, e55, 2016.

[183] Y. LeCun, “The mnist database of handwritten digits,” *http://yann. lecun. com/exdb/mnist/*, 1998.

[184] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, “Generative adversarial networks: An overview,” *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[185] D. P. Kingma and M. Welling, “An introduction to variational autoencoders,” *arXiv preprint arXiv:1906.02691*, 2019.

[186] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-Resolution Image Synthesis with Latent Diffusion Models,” 2021. arXiv: 2112.10752.

[187] P. Dhariwal and A. Nichol, “Diffusion Models Beat GANs on Image Synthesis,” 2021. arXiv: 2105.05233.

[188] A. Nichol *et al.*, “GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models,” 2021. arXiv: 2112.10741.

[189] A. Nichol and P. Dhariwal, “Improved denoising diffusion probabilistic models,” *ArXiv*, 2021.

[190] D. Watson, J. Ho, M. Norouzi, and W. Chan, “Learning to efficiently sample from diffusion probabilistic models,” *arXiv preprint arXiv:2106.03802*, Jun. 2021. arXiv: 2106.03802 `[cs.LG]`.

[191] F. Bao, C. Li, J. Zhu, and B. Zhang, “Analytic-DPM: An Analytic Estimate of the Optimal Reverse Variance in Diffusion Probabilistic Models,” 2022. arXiv: 2201.06503.

[192] J. Song, C. Meng, and S. Ermon, “Denoising Diffusion Implicit Models,” 2020. arXiv: 2010.02502.

[193] A. Jolicoeur-Martineau, K. Li, R. Piché-Taillefer, T. Kachman, and I. Mitliagkas, “Gotta go fast when generating data with score-based models,” *arXiv preprint arXiv:2105.14080*, 2021.

[194] H. Tachibana, M. Go, M. Inahara, Y. Katayama, and Y. Watanabe, “Itô-taylor sampling scheme for denoising diffusion probabilistic models using ideal derivatives,” *arXiv preprint arXiv:2112.13339*, 2021.

[195] L. Liu, Y. Ren, Z. Lin, and Z. Zhao, “Pseudo Numerical Methods for Diffusion Models on Manifolds,” no. 2021, pp. 1–23, 2022. arXiv: 2202.09778.

[196] T. Dockhorn, A. Vahdat, and K. Kreis, “Score-Based Generative Modeling with Critically-Damped Langevin Diffusion,” pp. 1–13, 2021. arXiv: 2112.07068.

[197] A. Hyvärinen, “Estimation of non-normalized statistical models by score matching,” *J. Mach. Learn. Res.*, vol. 6, pp. 695–708, 2005.

[198] P. Vincent, “A connection between score matching and denoising autoencoders,” *Neural Comput.*, vol. 23, no. 7, pp. 1661–1674, Jul. 2011.

[199] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[200] R. Pless and R. Souvenir, "A survey of manifold learning for images," *IPSJ Transactions on Computer Vision and Applications*, vol. 1, pp. 83–94, 2009.

[201] D. F. Griffiths and D. J. Higham, *Numerical methods for ordinary differential equations: initial value problems*. Springer, 2010, vol. 5.

[202] P. Virtanen *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[203] B. Liu, Y. Zhu, K. Song, and A. Elgammal, "Towards Faster and Stabilized GAN Training for High-fidelity Few-shot Image Synthesis," 2021. arXiv: 2101.04775.

[204] Z. Kong and W. Ping, "On fast sampling of diffusion probabilistic models," *arXiv preprint arXiv:2106.00132*, 2021.

[205] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, Dec. 2015.

[206] Y. Song, C. Durkan, I. Murray, and S. Ermon, "Maximum likelihood training of Score-Based diffusion models," *arXiv preprint arXiv:2101.09258*, Jan. 2021. arXiv: 2101.09258 `[stat.ML]`.

[207] Q. Zhang and Y. Chen, "Fast sampling of diffusion models with exponential integrator," *arXiv preprint arXiv:2204.13902*, 2022.

[208] B. Oksendal, *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013.

[209] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding.," *Science*, 2000.

[210] J. B. Tenenbaum, V. D. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction.," *Science*, 2000.

[211] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[212] T. Sauer, *Numerical analysis*, 2005.

[213] Y. Song, P. Dhariwal, M. Chen, and I. Sutskever, "Consistency models," 2023.

[214] X. Liu, C. Gong, and Q. Liu, “Flow straight and fast: Learning to generate and transfer data with rectified flow,” *arXiv preprint arXiv:2209.03003*, 2022.

[215] T. Kynkäänniemi, T. Karras, M. Aittala, T. Aila, and J. Lehtinen, “The role of imagenet classes in fr\’echet inception distance,” *arXiv preprint arXiv:2203.06026*, 2022.

[216] A. Jaegle *et al.*, “Perceiver io: A general architecture for structured inputs & outputs,” *arXiv preprint arXiv:2107.14795*, 2021.

[217] J. Kaplan *et al.*, “Scaling laws for neural language models,” *arXiv preprint arXiv:2001.08361*, 2020.

[218] J. Hoffmann *et al.*, “Training compute-optimal large language models,” *arXiv preprint arXiv:2203.15556*, 2022.