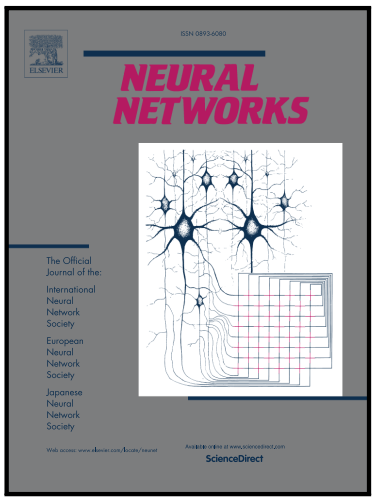


Energy-based diffusion generator for efficient sampling of Boltzmann distributions

Yan Wang, Ling Guo, Hao Wu, Tao Zhou

PII: S0893-6080(25)01006-8
DOI: https://doi.org/10.1016/j.neunet.2025.108126
Reference: NN 108126

To appear in: *Neural Networks*

Received date: 11 September 2024
Revised date: 10 September 2025
Accepted date: 15 September 2025

Please cite this article as: Yan Wang, Ling Guo, Hao Wu, Tao Zhou, Energy-based diffusion generator for efficient sampling of Boltzmann distributions, *Neural Networks* (2025), doi: https://doi.org/10.1016/j.neunet.2025.108126

Energy-based diffusion generator for efficient sampling of Boltzmann distributions

Yan Wang[a], Ling Guo[b], Hao Wu[*c], Tao Zhou[d]

School of Mathematical Sciences, Tongji University, Shanghai, China

Department of Mathematics, Shanghai Normal University, Shanghai, China

*School of Mathematical Sciences, Institute of Natural Sciences and MOE-LSC, Shanghai Jiao Tong University, Shanghai, China, *hwu81@sjtu.edu.cn,*

LSEC, Institute of Computational Mathematics and Scientific/Engineering Computing, AMSS, Chinese Academy of Sciences, Beijing, China

Abstract

Sampling from Boltzmann distributions, particularly those tied to high dimensional and complex energy functions, poses a significant challenge in many fields. In this work, we present the Energy-Based Diffusion Generator (EDG), a novel approach that integrates ideas from variational autoencoders and diffusion models. EDG uses a decoder to generate Boltzmann-distributed samples from simple latent variables, and a diffusion-based encoder to estimate the Kullback-Leibler divergence to the target distribution. Notably, EDG is simulation-free, eliminating the need to solve ordinary or stochastic differential equations during training. Furthermore, by removing constraints such as bijectivity in the decoder, EDG allows for flexible network design. Through empirical evaluation, we demonstrate the superior performance of EDG across a variety of sampling tasks with complex target distributions, outperforming existing methods.

Keywords: Boltzmann distribution, Energy-based model, Generative model, Diffusion model, Variational autoencoder

1. Introduction

In various fields such as computational chemistry, statistical physics, and machine learning, the challenge of sampling from a Boltzmann distribution corresponding to a high-dimensional and complex energy function is ubiquitous [1]. Unlike training tasks for data-driven generative models, where pre-sampled data can be utilized to learn complex distributions, sampling from Boltzmann distributions presents a unique and significant challenge due to the lack of readily available data [2, 3]. For example, simulating the phase transition of the Ising model can be framed as a sampling problem given the energy function, which presents a complex and difficult problem that has yet not to be effectively addressed [4, 5].

Markov Chain Monte Carlo (MCMC) methods [6], along with Brownian and Hamiltonian dynamics [7–10], have offered a pivotal solution to the challenge of sampling from high-dimensional distributions. These methods operate by iteratively generating candidates and updating samples, ultimately achieving asymptotic unbiasedness at the limit of infinite sampling steps. Meanwhile, some enhanced sampling methods [11, 12] developed for rare event problems accelerate sampling efficiency while significantly reducing relative statistical errors, even for low-probability regions.

In recent years, researchers have proposed adaptive MCMC as a strategy for generating candidate samples, showcasing notable advancements in augmenting the efficiency and effectiveness of the sampling process [13–15]. However, designing optimal loss functions is always challenging for the sampling problems. Many distributional discrepancies used in generative models, such as Jensen-Shannon divergence, maximum mean discrepancy and Wasserstein distance, require access to samples from the true distribution, which are unavailable in our setting. Stein discrepancy derived from Stein's identity [16], and the Kullback-Leibler (KL) divergence, between the distribution of the generated samples and the target distribution, may serve as potential alternatives for sampling.

Variational inference (VI) is a widely used approach to sampling problems. It employs a generator that efficiently produces samples to approximate the target Boltzmann distribution, and optimizes the generator's parameters using the KL divergence, without requiring true samples from the target distribution. Due to its ability to model complex distributions and provide explicit probability density functions, normalizing flow (NF) [17–20] has been extensively applied to construct generators for VI methods [21, 22].

Furthermore, a growing body of research has expanded their applicability across diverse domains [23–28]. However, the bijective nature of NFs imposes a constraint on their effective capacity, often rendering it insufficient for certain sampling tasks. Continuous normalizing flows [29–32] represent a specific subclass of NF in which the invertible transformation is defined through an ordinary differential equation. For more relevant studies, please refer to the Related Work in Sec. 2.1.

With the prosperity of diffusion based generative models [33–36], they have been applied to address challenges in the sampling problem. By training time-dependent score matching neural networks, methods proposed in [37–39] shape the Gaussian distribution into complex target densities, employing the KL divergence as the loss function. [40] also explores multiple connections between diffusion and other VI methods. To mitigate mode-seeking issues, [41] introduces the log-variance loss, showcasing favorable properties. Additionally, an alternative training objective is outlined in [42], relying on flexible interpolations of the energy function and demonstrating substantial improvements for multi-modal targets. However, a common drawback of these methods is their reliance on numerical differential equation solvers for computing time integrals, which can lead to substantial computational costs.

In this research endeavor, we present a novel approach termed the energy-based diffusion generator (EDG), drawing inspiration from both the variational autoencoder (VAE) technique [43] and the diffusion model. The architecture of EDG closely resembles that of VAE, comprising a decoder and an encoder. The decoder is flexible in mapping latent variables distributed according to tractable distributions to samples, without the imposition of constraints such as bijectivity, and we design in this work a decoder based on generalized Hamiltonian dynamics to enhance sampling efficiency. The encoder utilizes a diffusion process, enabling the application of score matching techniques for precise and efficient modeling of the conditional distribution of latent variables given samples. Unlike existing diffusion-based sampling methods, the loss function of EDG facilitates the convenient computation of unbiased estimates in a random mini-batch manner, removing the need for numerical solutions to ordinary differential equations (ODEs) or stochastic differential equations (SDEs) during training. Numerical experiments conclusively demonstrate the effectiveness of EDG.

2. Related work and preliminaries

2.1. Related work

Variational inference-based sampling algorithms: The Boltzmann generator (BG) [22, 44] is one of the most representative sampling methods based on VI. It leverages NFs to parameterize a trainable and analytically tractable density, with parameter optimization carried out by minimizing the KL divergence between the surrogate and target distributions. Here, NFs are composed of stacked bijective transformations, allowing the density to be computed in closed form. In addition, the combination of MCMC and VI methods stands as a current focal point in research [45–49]. This combination seeks to harness the strengths of both approaches, offering a promising avenue for addressing the challenges associated with sampling from high-dimensional distributions and enhancing the efficiency of probabilistic modeling.

Adaptive MCMC: Recent work has augmented MCMC sampling with nonlocal transition kernels parameterized by neural networks, including the NF-based method proposed by Vanden-Eijnden et al. [50] and the diffusion model-based method [51]. This synergistic approach enables neural networks to accelerate MCMC sampling while the model are trained in a data-driven mode from samples of MCMC.

Stein-based models: Designing optimal loss functions for sampling problems is always challenging for the sampling problems. Stein discrepancy has emerged as a promising candidate [16, 52–54], as it allows direct evaluation of the discrepancy between generated samples and the target distribution, without requiring real samples from the target or access to the density of the generated distribution. In particular, the kernelized Stein discrepancy [16] offers a practical and convenient criterion, which can test the goodness of fit easily. The trainable Stein Discrepancy [53, 54] further leverages neural networks to parameterize a class of functions based on the Stein's Identity, using the learned discrepancy to enhance its capability.

Latent diffusion models: Latent diffusion models, which are primarily designed to address generative modeling in data-driven scenarios, have recently gained significant attention [55–57]. Their main idea to use a pretrained encoder and decoder to obtain a latent space that both effectively represents the data and facilitates efficient sampling, with the diffusion model learning the distribution of latent variables. However, its integration with sampling problems remains to be explored.

2.2. Preliminaries and Setup

In this work, we delve into the task of crafting generative models for the purpose of sampling from the Boltzmann distribution driven by a predefined energy $U : \mathbb{R}^d \to \mathbb{R}$:

$$\pi(x) = \frac{1}{Z} \exp(-U(x)),$$

where the normalizing constant $Z = \int \exp(-U(x))\mathrm{d}x$ is usually computationally intractable. To tackle this challenge, the BG [21], along with its various extensions [44, 58, 59], has emerged as a prominent technique in recent years. In this work, we aim to overcome the limitations of BG by employing a generator with fewer structural constraints (e.g., without requiring bijection), and design a flexible model tailored for sampling tasks.

Our focus now shifts to a generator akin to the VAE. This generator produces samples by a decoder as

$$x|z_0 \sim p_D(x|z_0;\phi),$$

where $z_0 \sim p_D(z_0)$ is a latent variable drawn from a known prior distribution, typically a standard multivariate normal distribution. The parameter ϕ characterizes the decoder, and we define $p_D(x|z_0;\phi)$ as a Gaussian distribution $\mathcal{N}(x|\mu(z_0;\phi), \Sigma(z_0;\phi))$, with both μ and Σ parameterized by neural networks (NNs). Similar to the VAE, we aim to train the networks μ and Σ such that the marginal distribution $p_D(x)$ of the generated samples aligns with the target distribution.

It is important to note that, unlike conventional data-driven VAEs, we do not have access to samples from the target distribution $\pi(x)$. In fact, obtaining such samples is precisely the goal of the generator. As a result, the variational approximation of the KL divergence $D_{\mathrm{KL}}\left(\pi(x)||p_D(x)\right)$ cannot be used to train the model. Instead, in this work, we consider the divergence $D_{\mathrm{KL}}\left(p_D(x)||\pi(x)\right)$ and its upper bound:

$$\begin{aligned} D_{\mathrm{KL}}\left(p_D(x)||\pi(x)\right) &\leq D_{\mathrm{KL}}\left(p_D(z_0)p_D(x|z_0;\phi)||\pi(x)p_E(z_0|x;\theta)\right) \\ &= \mathbb{E}_{p_D(z_0)p_D(x|z_0;\phi)}\left[\log \frac{p_D(z_0)p_D(x|z_0;\phi)}{p_E(z_0|x;\theta)} + U(x)\right] \\ &\quad + \log Z. \end{aligned} \tag{1}$$

Here, the parametric distribution $p_E(z_0|x;\theta)$ defines an encoder that maps from x to the latent variable z_0, and the equality is achieved if $p_E(z_0|x;\theta)$

matches the conditional distribution of z for a given x, deduced from the decoder. The detailed proof is provided in Appendix A.

It seems that we have only increased the complexity of the problem, as we are still required to approximate a conditional distribution. However, in the upcoming section, we demonstrate that we can effectively construct the encoder using the diffusion model [34, 40] and optimize all parameters without the need to numerically solve ordinary or stochastic differential equations.

3. Energy-based diffusion generator

The diffusion model [34, 35] has emerged in recent years as a highly effective approach for estimating data distributions. Its core idea is to construct a diffusion process that progressively transforms data into simple white noise, and learn the reverse process to recover the data distribution from noise. In this work, we apply the principles of the diffusion model by incorporating a diffusion process into the latent space, enabling us to efficiently overcome the challenges arising from the variational framework for the sampling problem defined by Eq. (1). We refer to the model produced by this method as the energy-based diffusion generator (EDG).

3.1. Model architecture

In the EDG framework, we initiate a diffusion process from the latent variable z_0 and combine it with the decoder, resulting in what we term the "decoding process" p_D

$$\begin{aligned} &z_0 \in \mathbb{R}^d \sim p_D(z_0) \triangleq \mathcal{N}(x|0, I), \quad x|z_0 \sim p_D(x|z_0; \phi) \\ &\mathrm{d}z_t = f(z_t, t)\mathrm{d}t + g(t)\mathrm{d}W_t, \quad t \in [0, T] \end{aligned} \tag{2}$$

where W_t is the standard Wiener process, $f(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ acts as the drift coefficient, and $g(\cdot) : R \to R$ serves as the diffusion coefficient. To streamline notation, we denote the probability distribution defined by the decoding process as p_D, and represent the variables defined by the SDE as $z_{[\cdot]} = \{z_t\}_{t\in[0,T]}$. In typical SDEs applied in diffusion models, two critical conditions hold: (a) the transition density $p_D(z_t|z_0)$ can be analytically computed without numerically solving the Fokker-Planck equation, and (b) z_T is approximately uninformative with $p_D(z_T) \approx p_D(z_T|z_0)$.

If we only consider the statistical properties of the latent diffusion process, it is uninformative and only describes a transition from one simple noise to another. However, when we account for the conditional distribution of z_t given a sample x, the process $z_{[\cdot]}$ represents the gradual transformation of the complex conditional distribution $p_D(z_0|x) \propto p_D(z_0) \cdot p_D(x|z_0)$ into the tractable distribution $p_D(z_T|x) = p_D(z_T)$, where the independence between z_T and x results from the independence between z_0 and z_T (see Appendix B). This implies that, starting from $z_T \sim p_D(z_T)$, we can obtain samples from $p_D(z_0|x)$ by simulating the following reverse-time diffusion equation [60]:

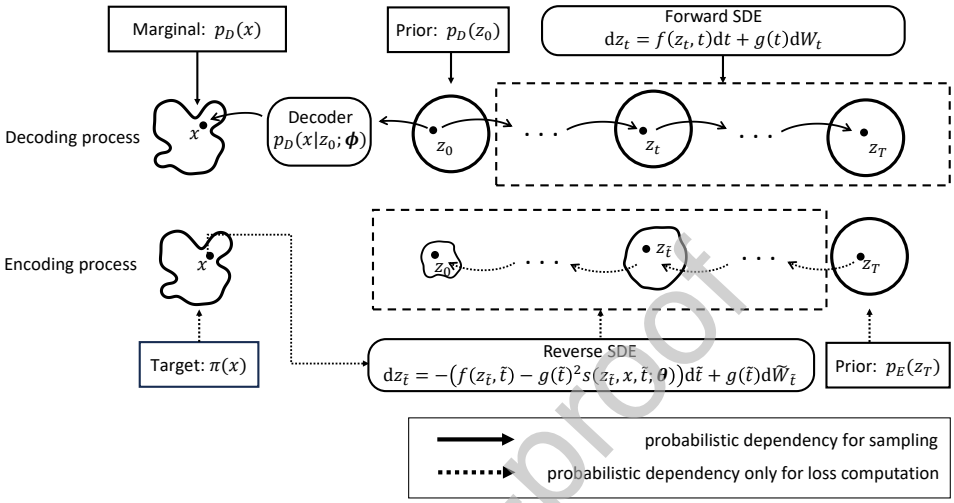
$$\mathrm{d}z_{\tilde{t}} = -\Big(f(z_{\tilde{t}}, \tilde{t}) - g(\tilde{t})^2 \nabla_{z_{\tilde{t}}} \log p_D(z_{\tilde{t}}|x)\Big)\,\mathrm{d}\tilde{t} + g(\tilde{t})\mathrm{d}\widetilde{W}_{\tilde{t}}, \tag{3}$$

where $\tilde{t} = T - t$ denotes the reverse time. The process $\widetilde{W}$ is a distinct standard Wiener process, whose non-trivial relationship with W is formally discussed in [60]. As in conventional diffusion models, practical implementation of this simulation is challenging due to the intractability of the score function $\nabla_{z_{\tilde{t}}} \log p_D(z_{\tilde{t}}|x)$, and we therefore also use a neural network to approximate the score function, denoted as $s(z_{\tilde{t}}, x, \tilde{t}; \theta)$. This approximation leads to what we refer to as the "encoding process", achieved by integrating the parametric reverse-time diffusion process and the target distribution of x:

$$\begin{aligned} &x \sim \pi(x), \quad z_T \sim p_E(z_T) \triangleq p_D(z_T) \\ &\mathrm{d}z_{\tilde{t}} = -\Big(f(z_{\tilde{t}}, \tilde{t}) - g(\tilde{t})^2 s(z_{\tilde{t}}, x, \tilde{t}; \theta)\Big)\,\mathrm{d}\tilde{t} + g(\tilde{t})\mathrm{d}\widetilde{W}_{\tilde{t}}, \quad \tilde{t} = T - t. \end{aligned} \tag{4}$$

For simplicity of notation, we refer to the distribution defined by the encoding process as p_E in this paper.

Fig. 1 provides a schematic overview of the decoding and encoding processes in EDG. The decoding process formulated in Eq. (2) can be readily simulated. In contrast, the encoding process defined by Eq. (4) is not directly simulatable, as no samples from the target distribution are available prior to training. However, as will be discussed in Sec. 3.2, by optimizing the parameters of the encoding process, we can leverage samples from the decoding process and the probabilistic model induced by the encoding process to obtain an accurate variational estimate of the KL divergence between the marginal distribution of x produced by the decoder and the target distribution $\pi(x)$. Based on this estimate, we can jointly train both the decoder and the conditional score network in the encoding process, thereby enabling the generation of high-quality samples that closely match the target distribution.



Decoding distributions:	$z_0 \sim p_D(z_0)$, $x\|z_0 \sim p_D(x\|z_0, \boldsymbol{\phi})$, $z_{[\cdot]}\|x, z_0 \sim$ Forward SDE	(Eq. (2))
Encoding distributions:	$x \sim \pi(x)$, $z_T \sim p_E(z_T)$, $z_{[\cdot]}\|x, z_T \sim$ Reverse SDE	(Eq. (4))
Training loss:	$D_{KL}(p_D(x, z_{[\cdot]}) \,\|\|\, p_E(x, z_{[\cdot]})) = \mathcal{L}(\theta, \boldsymbol{\phi}) + \text{const.}$	(Theorem 1)
Trainable parameters:	$\boldsymbol{\theta}, \boldsymbol{\phi}$	

Figure 1: Illustration of the decoding and encoding processes in EDG. In the **decoding process**, the latent variable z_0 is drawn from a tractable prior $p_D(z_0)$, and a sample x is generated via the decoder distribution $p_D(x|z_0)$ (modeled as Gaussian in this work). Simultaneously, a latent path $z_{[\cdot]} = \{z_t\}_{t\in[0,T]}$ is generated by a forward SDE. In the **encoding process**, $x \sim \pi(x)$ and $z_T \sim p_E(z_T)$ are assumed to be independently distributed, and a reverse-time SDE models the conditional distribution of the latent path given x and z_T, where $s(z_{\tilde{t}}, x, \tilde{t}; \theta)$ approximates the score function $\nabla_{z_{\tilde{t}}} \log p_D(z_{\tilde{t}}|x; \phi)$. Given that training samples of x and z_t are generated exclusively by the decoding process, while the encoding process is used solely for loss computation (not for actual sampling), we use **solid arrows** to indicate probabilistic dependencies in the decoding process and **dashed arrows** for those in the encoding process. Both the decoder parameters (mean and covariance) and the score model are parameterized by neural networks and jointly optimized via the loss function $\mathcal{L}(\theta, \phi)$.

Accordingly, the encoding process in EDG functions as a powerful encoder, replacing the conventional parametric encoder typically used in variational methods (see $p_E(z_0 \mid x;\theta)$ in Eq. (1)).

3.2. Loss function

Based on the architecture of the EDG, we establish the following theorem that provides a theoretical foundation for model training. It shows that the KL divergence between the joint distributions of $(x, z_{[\cdot]})$ induced by the two processes provides an upper bound for $D_{\mathrm{KL}}(p_D(x)||\pi(x))$. For a fixed decoder $p_D(x|z_0;\phi)$, minimizing the joint divergence with respect to the score model parameters θ tightens this upper bound. Moreover, jointly optimizing both ϕ and θ based on the joint divergence can effectively reduce the gap between the generated and target distributions of x.

Theorem 1. *For the decoding and encoding processes defined by Eq.* (2) *and Eq.* (4)*, we have*

$$\begin{aligned} D_{\mathrm{KL}}\left(p_D(x)\,\|\,\pi(x)\right) &\leq D_{\mathrm{KL}}\left(p_D(x, z_{[\cdot]})\,\|\,p_E(x, z_{[\cdot]})\right) && (5)\\ &= \mathcal{L}(\theta,\phi) + \int_0^T \frac{g(t)^2}{2}\mathbb{E}_{p_D}\left[\|\nabla_{z_t}\log p_D(z_t)\|^2\right]\mathrm{d}t \\ &\quad + \log Z, && (6)\end{aligned}$$

where $\mathcal{L}(\theta,\phi)$ *is given by*

$$\begin{aligned}\mathcal{L}(\theta,\phi) &= \mathbb{E}_{p_D}\left[\log p_D(x|z_0;\phi) + U(x)\right] \\ &\quad + \int_0^T \frac{g(t)^2}{2}\mathbb{E}_{p_D}\left[\,\|s\left(z_t, x, t;\theta\right)\|^2 + 2\nabla_{z_t}\cdot s\left(z_t, x, t;\theta\right)\right]\mathrm{d}t. \quad (7)\end{aligned}$$

Moreover, the equality in Eq. (5) *holds if* z_0 *is independent of* z_T *in the decoding process, and* $s(z, x, t;\theta) \equiv \nabla_{z_t}\log p_D(z_t|x;\phi)$.

Proof. See Appendix C. □

Since the last two terms in Eq. (6) are independent of the model parameters θ and ϕ, we can adopt $\mathcal{L}(\theta,\phi)$ in Eq. (7) as the training loss. To improve computational efficiency, we estimate the divergence term using the Hutchinson estimator and evaluate the time integral via Monte Carlo sampling. This

leads to an equivalent expression of $\mathcal{L}(\theta, \phi)$ that allows efficient and unbiased estimation:

$$\begin{aligned}\mathcal{L}(\theta,\phi) &= \mathbb{E}_{p_D}\left[\log p_D(x|z_0;\phi) + U(x)\right] \\ &\quad + \mathbb{E}_{t\sim\mathcal{U}[0,T],\epsilon\sim p(\epsilon),(x,z_t)\sim p_D(x,z_t)}\left[\mathcal{L}_t(x,z_t,\epsilon;\theta)\right],\end{aligned} \tag{8}$$

where

$$\mathcal{L}_t(x,z_t,\epsilon;\theta) = \frac{Tg(t)^2}{2}\left(\|s\left(z_t,x,t;\theta\right)\|^2 + 2\frac{\partial\left[\epsilon^\top s\left(z_t,x,t;\theta\right)\right]}{\partial z_t}\epsilon\right), \tag{9}$$

$\mathcal{U}[0,T]$ denotes a uniform distribution over $[0,T]$, and $p(\epsilon) \sim \text{Rademacher}^d$ denotes a random vector in $\mathbb{R}^d$ with i.i.d. Rademacher entries, satisfying $\mathbb{E}[\epsilon] = 0$ and $\text{Cov}(\epsilon) = I$. This expression enables the use of stochastic gradient descent for parameter optimization. (See Appendix D for the proof of equivalence between Eq. (8) and Eq. (7).) The training procedure is outlined in Algorithm 1. It is worth noting that when the SDE in the decoding process follows a standard formulation used in diffusion models (see Eq. (2)), the conditional distribution $p_D(z_t|z_0)$ can be sampled exactly, and numerical integration of SDEs is not required during training.

Algorithm 1 Training procedure

for each minibatch **do**
  for $i = 1, \ldots,$ batch size **do**
    Sample $z_0^i \sim p_D(z_0^i)$ and $x^i \sim p_D(x^i|z_0^i;\phi)$;
    Sample $t \sim \mathcal{U}[0,T]$, $\epsilon^i \sim \text{Rademacher}^d$ and $z_t^i \sim p_D(z_t^i|z_0^i)$;
    Evaluate $\mathcal{L}_t(x^i, z_t^i, \epsilon^i;\theta)$ using Eq. (9);
    Compute $L_i = \log p_D(x^i|z_0^i;\phi) + U(x^i) + \mathcal{L}_t(x^i, z_t^i, \epsilon^i;\theta)$;
  end for
  Compute minibatch loss: $\hat{\mathcal{L}}(\theta,\phi) =$ average of $\{L_1, L_2, \ldots\}$;
  Update parameters: $\phi, \theta \leftarrow \text{Update}(\nabla_{\phi,\theta}\hat{\mathcal{L}}(\theta,\phi))$;
end for

3.3. Sample reweighting

After training, we can use the decoder $p_D(x|z_0)$ to generate samples and compute various statistics of the target distribution $\pi(x)$. For example, for a quantity of interest $O : \mathbb{R}^d \to \mathbb{R}$, we can draw N augmented samples

$\{(x^n, z_0^n)\}_{n=1}^N$ from $p_D(z_0)p_D(x|z_0)$ and estimate the expectation $\mathbb{E}_{\pi(x)}[O(x)]$ as follows:

$$\mathbb{E}_{\pi(x)}[O(x)] \approx \frac{1}{N}\sum_n O(x^n).$$

However, due to model errors, this estimation can be systematically biased. To address this, we can apply importance sampling, using $p_D(x, z_0) = p_D(z_0)p_D(x|z_0)$ as the proposal distribution and $p_E(x, z_0) = \pi(x)p_E(z_0|x)$ as the augmented target distribution. We can then assign each sample (x, z_0) generated by the decoder an unnormalized weight:

$$w(x, z_0) = \frac{\exp(-U(x))p_E(z_0|x)}{p_D(z_0)p_D(x|z_0)} \propto \frac{\pi(x)p_E(z_0|x)}{p_D(z_0)p_D(x|z_0)} \tag{10}$$

and obtain a consistent estimate of $\mathbb{E}_{\pi(x)}[O(x)]$ as:

$$\mathbb{E}_{\pi(x)}[O(x)] \approx \frac{\sum_n w(x^n, z_0^n)O(x^n)}{\sum_n w(x^n, z_0^n)}, \tag{11}$$

where the estimation error approaches zero as $N \to \infty$ [61].

The main difficulty in above computations lies in the intractability of the marginal encoder density $p_E(z_0|x)$ when calculating the weight function. To overcome this, following the conclusion from Sec. 4.3 in [34], we construct the following probability flow ODE:

$$\mathrm{d}z_t = \left(f(z_t, t) - \frac{1}{2}g(t)^2 s(z_t, x, t; \theta)\right)\mathrm{d}t, \tag{12}$$

with the boundary condition $z_T \sim p_E(z_T)$. If $s(z_t, x, t; \theta)$ accurately approximates the score function $\nabla_{z_t} \log p_D(z_t|x)$ after training, the conditional distribution of $z_t|x$ given by the ODE will match that of the encoding process for each $t \in [0, T]$. Consequently, we can employ the neural ODE method [62] to efficiently compute $p_E(z_0|x)$ as

$$\log p_E(z_0|x) = \log p_E(z_T) + \int_0^T \nabla \cdot \left(f(z_t, t) - \frac{1}{2}g(t)^2 s(z_t, x, t; \theta)\right)\mathrm{d}t. \tag{13}$$

See Appendix E for details of the probability flow ODE calculation, and the reweighting algorithm is summarized in Algorithm 2.

In addition, the weight function w can also be used to estimate the normalizing constant Z, which is a crucial task in many applications, such as

Bayesian model selection in statistics and free energy estimation in statistical physics. Based on Eq. (1), we have:

$$\log Z \geq \mathbb{E}_{p_D(x,z_0)} \left[\log w(x, z_0)\right]. \tag{14}$$

Here, the lower bound can also be estimated using samples from the decoder, and the tightness of this bound is achieved when $p_D(x, z_0) = p_E(x, z_0)$. In practice, the estimate of $\log Z$ given by the above inequality can serve as an indicator of training quality. A larger value suggests a more accurate approximation and, consequently, better model performance. More detailed analysis is provided in Appendix F.

3.4. Network design

Below, we present the construction details of modules in EDG used in our experiments. In practical applications, more effective neural networks can be designed as needed.

3.4.1. Boundary condition-guided score function model

Considering that the true score function satisfies the following boundary conditions for $t = 0$ and T:

$$\begin{aligned} \nabla_{z_0} \log p_D(z_0|x) &= \nabla_{z_0} \left[\log p_D(z_0|x) + \log p_D(x)\right] \\ &= \nabla_{z_0} \log p_D(x, z_0) \\ &= \nabla_{z_0} \left[\log p_D(x|z_0) + \log p_D(z_0)\right] \end{aligned}$$

Algorithm 2 Sampling and reweighting procedure

Input: Initial density: $p_D(z_0)$; Decoder: $p_D(\cdot|z_0; \phi^*)$; Score: $s(\cdot; \theta^*)$; Observation: $O(\cdot)$.
for $i = 1, ...,$ sample size **do**
  Sample $z_0^i \sim p_D(z_0^i)$ and $x^i \sim p_D(x^i|z_0^i; \phi^*)$;
  Compute $O_i = O(x^i)$;
  Compute $p_E(z_0^i|x^i; \theta^*)$ by Eqs. (12, 13);
  Compute $w_i = \frac{\exp(-U(x^i))p_E(z_0^i|x^i;\theta^*)}{p_D(z_0^i)p_D(x^i|z_0^i;\phi^*)}$;
end for
Estimate $\mathbb{E}_\pi[\hat{O}(x)] = \frac{\sum_i w_i O_i}{\sum_i w_i}$;
Output: $\{x_i\}$, $\mathbb{E}_\pi[\hat{O}(x)]$

and

$$\nabla_{z_T} \log p_D(z_T|x) = \nabla_{z_T} \log p_D(z_T),$$

we propose to express $s(z, x, t; \theta)$ as

$$\begin{aligned} s(z, x, t; \theta) &= \left(1 - \frac{t}{T}\right) \cdot \nabla_{z_0} \left[\log p_D(x|z_0 = z) + \log p_D(z_0 = z)\right] \\ &\quad + \frac{t}{T} \cdot \nabla_{z_T} \log p_D(z_T = z) + \frac{t}{T}\left(1 - \frac{t}{T}\right) s'(z, x, t; \theta), \end{aligned}$$

where $s'(z, x, t; \theta)$ is the neural network to be trained. This formulation ensures that the error of s is zero for both $t = 0$ and $t = T$.

3.4.2. Generalized Hamiltonian dynamics-based decoder

Inspired by generalized Hamiltonian dynamics (GHD) [13, 14], the decoder generates the output x using the following process. First, an initial sample and velocity (y, v) are generated according to the latent variable z_0. Then, (y, v) is iteratively updated as follows:

$$\begin{aligned} v &:= v - \frac{\epsilon_0 e^{\epsilon_0 \epsilon(l;\phi)}}{2}\left(\nabla U(y) \odot e^{\frac{\epsilon_0}{2} Q_v(y, \nabla U(y), l; \phi)} + T_v(y, \nabla U(y), l; \phi)\right), \\ y &:= y + \epsilon_0 e^{\epsilon_0 \epsilon(l;\phi)}\left(v_k \odot e^{\epsilon_0 Q_y(v_k, l; \phi)} + T_y(v_k, l; \phi)\right), \\ v &:= v - \frac{\epsilon_0 e^{\epsilon_0 \epsilon(l;\phi)}}{2}\left(\nabla U(y) \odot e^{\frac{\epsilon_0}{2} Q_v(y, \nabla U(y), l; \phi)} + T_v(y, \nabla U(y), l; \phi)\right). \end{aligned}$$

Finally, the decoder output x is given by:

$$x = y - \epsilon_0 e^{\epsilon_0 \eta(y;\phi)} \nabla U(y) + \sqrt{2\epsilon_0 e^{\epsilon_0 \eta(y;\phi)}}\xi,$$

where ξ is distributed according to the standard Gaussian distribution. The equation can be interpreted as a finite-step approximation of Brownian dynamics $\mathrm{d}y = -\nabla U(y)\,\mathrm{d}t + \mathrm{d}W_t$. The decoder density is then defined as

$$p_D(x|z_0; \phi) = \mathcal{N}\left(x \mid y - \epsilon_0 e^{\epsilon_0 \eta(y;\phi)} \nabla U(y),\, 2\epsilon_0 e^{\epsilon_0 \eta(y;\phi)} I\right),$$

where I denotes the identity matrix. Since this density can be tractably evaluated given x, z_0, and ϕ, all decoder parameters ϕ involved in the GHD module can be jointly optimized with the encoder parameters θ by minimizing the loss function $\mathcal{L}(\theta, \phi)$. In the above procedure, Q_v, T_v, Q_y, T_y are all

trainable neural networks. To mitigate the impact of step sizes on model performance, we parameterize them as trainable positive functions of the form $\epsilon_0 \exp(\epsilon_0 \epsilon(l;\phi))$ and $\epsilon_0 \exp(\epsilon_0 \eta(y;\phi))$, where ϵ_0 is a small positive constant that controls the initial scale and prevents instability during training caused by large step sizes. The choice of ϵ_0 in practice is discussed in Appendix J.

The key advantages of the GHD-based decoder are twofold. First, it effectively leverages the gradient information of the energy function, and our experiments show that it can enhance the sampling performance for multimodal distributions. Second, by incorporating trainable correction terms and steps into the classical Hamiltonian dynamics, it achieves a good decoder density with only a few iterations. The complete configurations can be found in Appendix G.

4. Experiments

We conduct an empirical evaluation of EDG across a diverse range of energy functions. We begin with experiments on a set of two-dimensional distributions, followed by results on Bayesian logistic regression. We then assess EDG's performance on two Lennard-Jones tasks, and finally apply it to the Ising model. All the experimental details are provided in Appendix H. Additionally, we perform an ablation study to verify the effectiveness of each module in EDG. Please refer to Appendix J for more information.

In order to demonstrate the superiority of our model, we compare EDG with the following sampling methods:

- Vanilla Hamiltonian Monte Carlo method [7], denoted as V-HMC.
- L2HMC [14], a GHD-based MCMC method with a trainable proposal distribution model.
- Boltzmann Generator (BG) [21], a VI method that generates samples solely based on a prescribed energy function, without relying on any real data. The surrogate distribution is modeled using RealNVP [63].
- Neural Renormalization Group (NeuralRG) [25], a method similar to BG and designed specifically for the Ising model. In this section, NeuralRG is only used for experiments of the Ising model.
- Path Integral Sampler (PIS) [37], a diffusion-based sampling model via numerical simulation of an SDE.

We now present the experimental results for each sampling task in detail. An analysis of the computational cost is provided in Appendix I.

2D energy function Firstly, we compare our model with the other models on several synthetic 2D energy functions: MoG2 (Mixture of two isotropic Gaussians with equal variance $\sigma^2 = 0.5$, centers 10 apart), MoG2(i) (Mixture of two isotropic Gaussians with unequal variance $\sigma_1^2 = 1.5, \sigma_2^2 = 0.3$, centers 10 apart), MoG6 (Mixture of six isotropic Gaussians with variance $\sigma^2 = 0.1$), MoG9 (Mixture of nine isotropic Gaussians with variance $\sigma^2 = 0.3$), Ring, Ring5 (energy functions can be seen in [13]). We present the histogram of samples for visual inspection in Fig. 2, and Table 1 summarizes the sampling errors. As shown, EDG delivers higher-quality samples compared to the other methods. To further clarify the contribution of each component in EDG, we compare it with a vanilla VAE in Appendix J. The effectiveness of the reweighting scheme is also evaluated in Appendix H.

Bayesian Logistic Regression In the subsequent experiments, we highlight the efficacy of EDG in the context of Bayesian logistic regression, particularly when dealing with a posterior distribution residing in a higher-dimensional space. In this scenario, we tackle a binary classification problem with labels $L = \{0, 1\}$ and high-dimensional features D. The classifier's output is defined as

$$p(L = 1|D, x) = \text{softmax}(w^\top D + b),$$

where $x = (w, b)$. We aim to draw samples $x^1, \ldots, x^N$ from the posterior distribution

$$\pi(x) \propto p(x) \prod_{(L,D)\in\mathcal{D}_{\text{train}}} p(L|D, x)$$

Table 1: The Maximum Mean Discrepancy (MMD) between the samples generated by each generator and the reference samples. Details on the calculation of discrepancy can be found in Appendix H.

	Mog2	Mog2(i)	Mog6	Mog9	Ring	Ring5
V-HMC	**0.01**	1.56	0.02	0.04	**0.01**	**0.01**
L2HMC	0.04	0.94	**0.01**	0.03	0.02	**0.01**
BG	1.90	1.63	2.64	0.07	0.05	0.18
PIS	**0.01**	1.66	**0.01**	0.42	**0.01**	0.78
EDG	**0.01**	**0.50**	**0.01**	**0.02**	**0.01**	0.02

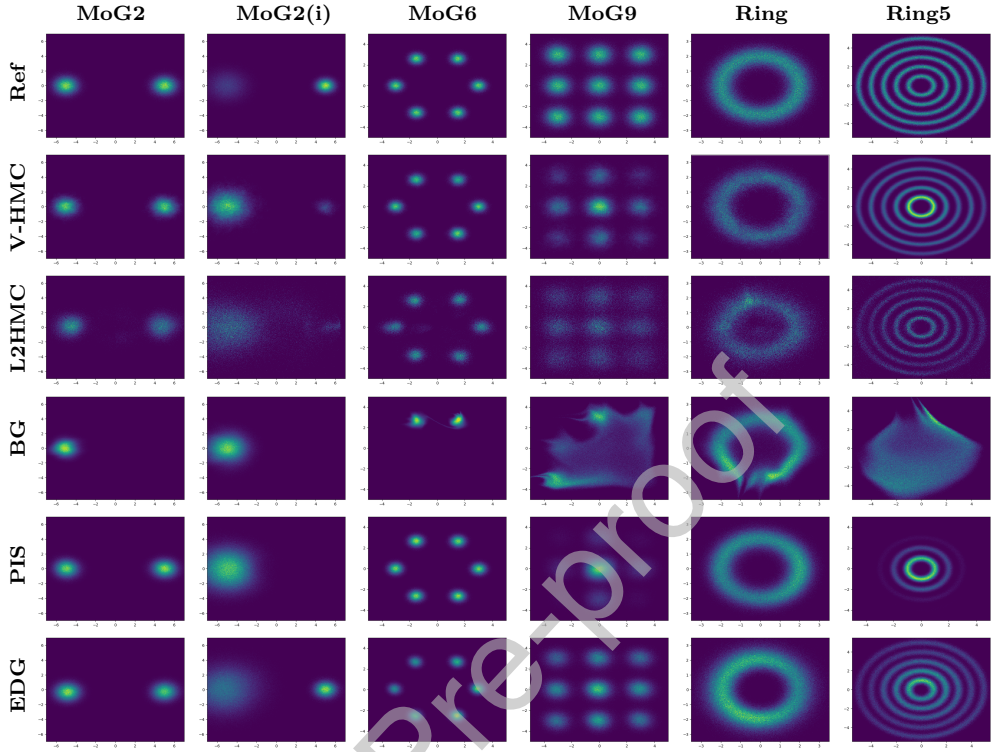


Figure 2: Density plots for 2D energy function. For the generation of reference samples, please refer to Appendix H. We generate 500, 000 samples for each method and plot the histogram.

based on the training set $\mathcal{D}_{\text{train}}$, where the prior distribution $p(x)$ is a standard Gaussian distribution. Then, for a given D, the conditional distribution $p(L|D, \mathcal{D}_{\text{train}})$ can be approximated as $\frac{1}{N}\sum_n p(L|D, x^n)$. We conduct experiments on three datasets: Australian (AU, 15 covariates), German (GE, 25 covariates), and Heart (HE, 14 covariates) [64], evaluating accuracy rate (ACC) and Area Under the Curve (AUC) on the test subset. Remarkably, as illustrated in Table 2, EDG consistently achieves the highest accuracy and AUC performance.

We extend our analysis to the binary Covertype dataset comprising 581,012 data points and 54 features. The posterior of the classifier parameters follows a hierarchical Bayesian model (see Sec. 5 of [52]), with x denoting the combination of classifier parameters and the hyperparameter in the hierarchical Bayesian model. To enhance computational efficiency, in BG and EDG,

Table 2: Classification accuracy and AUC results for Bayesian logistic regression tasks. The experiments utilize consistent training and test data partitions, where the HMC step size is set to 0.01. Average accuracy and AUC values, accompanied by their respective standard deviations, are computed across 32 independent experiments for all datasets.

	AU		GE		HE	
	Acc	Auc	Acc	Auc	Acc	Auc
V-HMC	82.97 ± 1.94	90.88 ± 0.83	78.52 ± 0.48	77.67 ± 0.28	86.75 ± 1.63	93.35 ± 0.76
L2HMC	73.26 ± 1.56	79.69 ± 3.65	62.02 ± 4.19	60.23 ± 5.10	82.23 ± 2.81	90.48 ± 0.51
BG	82.99 ± 1.18	91.23 ± 0.67	78.14 ± 1.44	77.59 ± 0.73	86.75 ± 1.99	93.44 ± 0.39
PIS	81.64 ± 2.63	91.23 ± 0.67	71.90 ± 3.17	71.67 ± 4.52	83.24 ± 3.95	91.68 ± 2.78
EDG	$\mathbf{84.96 \pm 1.67}$	$\mathbf{92.82 \pm 0.69}$	$\mathbf{79.40 \pm 1.74}$	$\mathbf{82.79 \pm 1.46}$	$\mathbf{88.02 \pm 3.90}$	$\mathbf{95.10 \pm 1.23}$

Table 3: Classification accuracy on the test dataset of Covertype. The reported values represent averages and standard deviations of accuracy over 32 independent experiments.

	V-HMC	L2HMC	BG	PIS	**EDG**
Acc	49.88 ± 3.32	51.51 ± 3.46	50.75 ± 3.78	50.59 ± 2.94	$\mathbf{70.13 \pm 2.13}$

$\log \pi(x)$ is unbiasedly approximated during training as

$$\log \pi(x) \approx \log p(x) + \frac{|\mathcal{D}_{\text{train}}|}{|\mathcal{B}|} \sum_{(L,D)\in\mathcal{B}} \log p(L|D, x),$$

where $\mathcal{B}$ is a random mini-batch. For V-HMC and L2HMC, the exact posterior density is calculated. As indicated by the results in Table 3, EDG consistently outperforms alternative methods.

Lennard-Jones We further evaluate the performance of EDG on two Lennard–Jones (LJ) systems with non-periodic boundary conditions, consisting of 13 and 55 particles (LJ13 and LJ55), as described in [65]. The LJ potential is a commonly used interaction model that captures both repulsive and attractive forces between non-bonded particles. The potential energy depends on the pairwise distances between particles, and its explicit form is provided in Appendix H. In this task, we temporarily disregard the issue of model equivariance, which we leave for future investigation.

LJ13 refers to the system of 13 particles, $x = \{x_1, ..., x_{13}\}$ with 3 dimensions each, resulting in a task with dimensionality $d = 39$. LJ55 meanwhile

refers to a system of 55 particles, $x = \{x_1, ..., x_{55}\}$ with 3 dimensions each, resulting in a high dimensional task with dimensionality $d = 165$. For the experimental results, we evaluate the generated sample with the test data from [65] obtained by MCMC. A visualization of the interatomic distances is presented in Fig. 3. In Appendix H, we test the effectiveness of sample reweighting introduced in Sec. 3.3, by using the relative Effective Sample Size (see Eq. (H.1) and Table H.6).

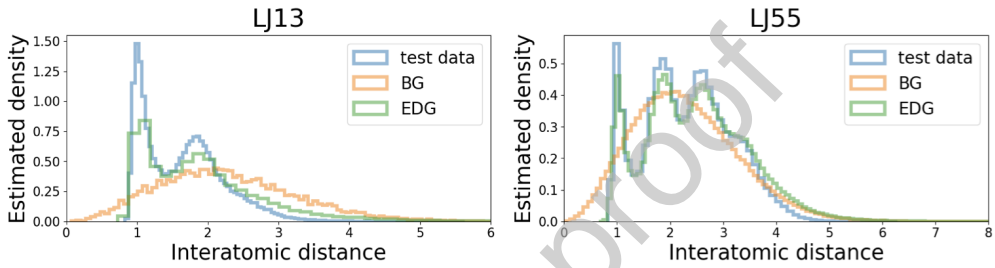


Figure 3: Comparison of the test data interatomic distance of LJ13 (left), LJ55 (right) with samples generated from BG and EDG. The results for L2HMC and PIS are provided separately in Appendix H due to significant deviations from the test data.

Ising model Finally, we verify the performance of EDG on the 2-dimensional Ising model [25], a mathematical model of ferromagnetism in statistical mechanics. To ensure the continuity of physical variables, we employ a continuous relaxations trick [66] to transform discrete variables into continuous auxiliary variables with the target distribution:

$$\pi(x) = \exp\left(-\frac{1}{2}x^T \left(K(T) + \alpha I\right)^{-1} x\right) \times \prod_{i=1}^{N} \cosh\left(x_i\right),$$

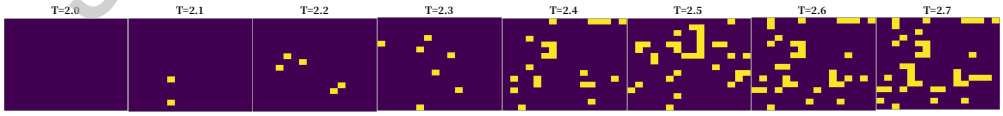


Figure 4: The generated states at different temperatures from $T = 2.0$ to $T = 2.7$ by EDG with a dimensionality of 256 (16×16), where the latent variable z_0 remains fixed. As the temperature rises, the model's state tends progressively toward disorder.

Table 4: The estimation of $\log Z_{\text{Ising}}$ in 2D ising model with a dimension of 256 (16×16) is obtained by the method described in Sec. 3.3. We utilize a batch size of $n = 256$ to estimate the mean and applies the central limit theorem to calculate the standard deviation of the mean of the statistic as $\text{std}/\sqrt{n}$.

$\log Z_{\text{Ising}}$	NeuralRG	PIS	**EDG**
$T = 2.0$	260.24 ± 0.13	210.17 ± 0.43	$\mathbf{270.32 \pm 0.18}$
$T = 2.1$	250.57 ± 0.14	208.48 ± 0.41	$\mathbf{260.34 \pm 0.19}$
$T = 2.2$	239.43 ± 0.16	210.57 ± 0.39	$\mathbf{252.11 \pm 0.17}$
$T = 2.3$	231.25 ± 0.15	214.64 ± 0.37	$\mathbf{233.46 \pm 0.17}$
$T = 2.4$	$\mathbf{225.69 \pm 0.17}$	212.43 ± 0.37	225.02 ± 0.15
$T = 2.5$	219.26 ± 0.17	202.92 ± 0.37	$\mathbf{220.03 \pm 0.14}$
$T = 2.6$	$\mathbf{216.69 \pm 0.18}$	181.83 ± 0.40	214.78 ± 0.14
$T = 2.7$	212.42 ± 0.18	189.16 ± 0.36	$\mathbf{212.57 \pm 0.14}$

where K is an $N \times N$ symmetric matrix depending on the temperature T, α is a constant guaranteeing $K + \alpha I$ to be positive. For the corresponding discrete Ising variables $\boldsymbol{s} = \{1, -1\}^{\otimes N}$, one can directly obtain discrete samples according to $\pi(\boldsymbol{s}|\boldsymbol{x}) = \prod_i \left(1 + e^{-2s_i x_i}\right)^{-1}$. When there is no external magnetic field and each spin can only interact with its neighboring spins, K is defined as $\sum_{<ij>} s_i s_j / T$, with the nears neighboring sum $< ij >$. Consequently, the normalizing constant of the continuous relaxations system is given by $\log Z = \log Z_{\text{Ising}} + \frac{1}{2} \ln \det(K + \alpha I) - \frac{N}{2}[\ln(2/\pi) - \alpha]$ [25]. Additionally, using the method described in Sec. 3.3, we provide the estimates of $\log Z_{\text{Ising}}$ at different temperatures for samples generated by NeuralRG, PIS and EDG. Since these are lower bound estimates, larger values indicate more accurate results. As seen in Table 4, EDG provides the most accurate estimates of $\log Z$ across most temperature ranges. The generated states at different temperatures are displayed in Fig. 4.

5. Conclusion

Our work introduces the EDG as an innovative and effective sampling approach by combining principles from VI and diffusion based methods. Drawing inspiration from VAEs, EDG excels in efficiently generating samples from intricate Boltzmann distributions. Leveraging the expressive power of the diffusion model, our method accurately estimates the KL divergence without

the need for numerical solutions to ordinary or stochastic differential equations. Empirical experiments validate the superior sampling performance of EDG.

In consideration of its powerful generation ability and unrestrained network design theoretically, there is still room for further exploration. We can design specific network architectures for different tasks. Regarding many-body particle systems, we plan to leverage equivariant graph neural networks (EGNN) [30, 65] to design all trainable components within the GHD module of the decoder to ensure the invariance. Moreover, we aim to construct equivariant score function models within the encoding process with respect to both x and z_t, which is expected to guarantee the invariance of the weight function. Furthermore, theoretical convergence guarantees can be strengthened by investigating error bounds and convergence rates under finite training time and limited model capacity.

In summary, our future work will extend the application of EDG to construct generative models for large-scale physical and chemical systems, such as proteins [67, 68], while simultaneously refining the theoretical foundations of our model.

Acknowledgements

The first and third authors are supported by the NSF of China (under grant number 12171367). The second author is supported by the NSF of China (under grant numbers 92270115, 12071301) and the Shanghai Municipal Science and Technology Commission (No. 20JC1412500), and Henan Academy of Sciences. The last author was supported by the NSF of China (No. 12288201), the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDA25010404), the National Key R&D Program of China (2020YFA0712000), the Youth Innovation Promotion Association, CAS, and Henan Academy of Sciences.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] D. C. Rapaport, The art of molecular dynamics simulation, Cambridge university press, 2004.

[2] L. Martino, J. Read, On the flexibility of the design of multiple try metropolis schemes, Computational Statistics 28 (2013) 2797–2823.

[3] R. Y. Rubinstein, D. P. Kroese, Simulation and the Monte Carlo method, John Wiley & Sons, 2016.

[4] K. Binder, E. Luijten, Monte carlo tests of renormalization-group predictions for critical phenomena in ising models, Physics Reports 344 (2001) 179–253.

[5] N. Walker, K.-M. Tam, M. Jarrell, Deep learning on the 2-dimensional ising model to extract the crossover region with a variational autoencoder, Scientific reports 10 (2020) 13047.

[6] W. K. Hastings, Monte carlo sampling methods using markov chains and their applications, Biometrika (1970).

[7] R. M. Neal, et al., Mcmc using hamiltonian dynamics, Handbook of markov chain monte carlo 2 (2011) 2.

[8] T. Araki, K. Ikeda, Adaptive markov chain monte carlo for auxiliary variable method and its application to parallel tempering, Neural Networks 43 (2013) 33–40.

[9] T. Araki, K. Ikeda, S. Akaho, An efficient sampling algorithm with adaptations for bayesian variable selection, Neural Networks 61 (2015) 22–31.

[10] X. Cheng, N. S. Chatterji, P. L. Bartlett, M. I. Jordan, Underdamped langevin mcmc: A non-asymptotic analysis, in: Conference on learning theory, PMLR, 2018, pp. 300–323.

[11] E. Vanden-Eijnden, J. Weare, Rare event simulation of small noise diffusions, Communications on Pure and Applied Mathematics 65 (2012) 1770–1803.

[12] A. Martinsson, J. Lu, B. Leimkuhler, E. Vanden-Eijnden, The simulated tempering method in the infinite switch limit with adaptive weight learning, Journal of Statistical Mechanics: Theory and Experiment 2019 (2019) 013207.

[13] J. Song, S. Zhao, S. Ermon, A-nice-mc: Adversarial training for mcmc, Advances in Neural Information Processing Systems 30 (2017).

[14] D. Levy, M. D. Hoffman, J. Sohl-Dickstein, Generalizing hamiltonian monte carlo with neural networks, arXiv preprint arXiv:1711.09268 (2017).

[15] L. Galliano, R. Rende, D. Coslovich, Policy-guided monte carlo on general state spaces: Application to glass-forming mixtures, The Journal of Chemical Physics 161 (2024).

[16] Q. Liu, J. Lee, M. Jordan, A kernelized stein discrepancy for goodness-of-fit tests, in: International conference on machine learning, PMLR, 2016, pp. 276–284.

[17] L. Dinh, D. Krueger, Y. Bengio, Nice: Non-linear independent components estimation, arXiv preprint arXiv:1410.8516 (2014).

[18] L. Dinh, J. Sohl-Dickstein, S. Bengio, Density estimation using real nvp, arXiv preprint arXiv:1605.08803 (2016).

[19] C.-H. Chao, W.-F. Sun, Y.-C. Hsu, Z. Kira, C.-Y. Lee, Training energy-based normalizing flow with score-matching objectives, Advances in Neural Information Processing Systems 36 (2023) 43826–43851.

[20] E. Nijkamp, R. Gao, P. Sountsov, S. Vasudevan, B. Pang, S.-C. Zhu, Y. N. Wu, Mcmc should mix: Learning energy-based model with neural transport latent space mcmc, arXiv preprint arXiv:2006.06897 (2020).

[21] F. Noé, S. Olsson, J. Köhler, H. Wu, Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning, Science 365 (2019) eaaw1147.

[22] H. Wu, J. Köhler, F. Noé, Stochastic normalizing flows, Advances in Neural Information Processing Systems 33 (2020) 5933–5944.

[23] M. Dibak, L. Klein, A. Krämer, F. Noé, Temperature steerable flows and boltzmann generators, Physical Review Research 4 (2022) L042005.

[24] J. Köhler, A. Krämer, F. Noé, Smooth normalizing flows, Advances in Neural Information Processing Systems 34 (2021) 2796–2809.

[25] S.-H. Li, L. Wang, Neural network renormalization group, Physical review letters 121 (2018) 260601.

[26] P. Wirnsberger, B. Ibarz, G. Papamakarios, Estimating gibbs free energies via isobaric-isothermal flows, Machine Learning: Science and Technology 4 (2023) 035039.

[27] M. Schebek, M. Invernizzi, F. Noé, J. Rogal, Efficient mapping of phase diagrams with conditional boltzmann generators, Machine Learning: Science and Technology 5 (2024) 045045.

[28] A. Coretti, S. Falkner, P. L. Geissler, C. Dellago, Learning mappings between equilibrium states of liquid systems using normalizing flows, The Journal of Chemical Physics 162 (2025).

[29] L. Klein, F. Noé, Transferable boltzmann generators, arXiv preprint arXiv:2406.14426 (2024).

[30] J. Köhler, L. Klein, F. Noé, Equivariant flows: exact likelihood generative learning for symmetric densities, in: International conference on machine learning, PMLR, 2020, pp. 5361–5370.

[31] M. S. Albergo, E. Vanden-Eijnden, Building normalizing flows with stochastic interpolants, arXiv preprint arXiv:2209.15571 (2022).

[32] G. Jung, G. Biroli, L. Berthier, Normalizing flows as an enhanced sampling method for atomistic supercooled liquids, Machine Learning: Science and Technology 5 (2024) 035053.

[33] J. Ho, A. Jain, P. Abbeel, Denoising diffusion probabilistic models, Advances in neural information processing systems 33 (2020) 6840–6851.

[34] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, B. Poole, Score-based generative modeling through stochastic differential equations, arXiv preprint arXiv:2011.13456 (2020).

[35] Y. Song, C. Durkan, I. Murray, S. Ermon, Maximum likelihood training of score-based diffusion models, Advances in Neural Information Processing Systems 34 (2021) 1415–1428.

[36] N. Bou-Rabee, A. Donev, E. Vanden-Eijnden, Metropolis integration schemes for self-adjoint diffusions, Multiscale modeling & simulation 12 (2014) 781–831.

[37] Q. Zhang, Y. Chen, Path integral sampler: a stochastic control approach for sampling, arXiv preprint arXiv:2111.15141 (2021).

[38] J. Berner, L. Richter, K. Ullrich, An optimal control perspective on diffusion-based generative modeling, arXiv preprint arXiv:2211.01364 (2022).

[39] F. Vargas, W. Grathwohl, A. Doucet, Denoising diffusion samplers, arXiv preprint arXiv:2302.13834 (2023).

[40] M. S. Albergo, N. M. Boffi, E. Vanden-Eijnden, Stochastic interpolants: A unifying framework for flows and diffusions, arXiv preprint arXiv:2303.08797 (2023).

[41] L. Richter, J. Berner, G.-H. Liu, Improved sampling via learned diffusions, arXiv preprint arXiv:2307.01198 (2023).

[42] B. Máté, F. Fleuret, Learning interpolations between boltzmann densities, Transactions on Machine Learning Research (2023).

[43] D. P. Kingma, M. Welling, et al., An introduction to variational autoencoders, Foundations and Trends® in Machine Learning 12 (2019) 307–392.

[44] S. van Leeuwen, A. P. d. A. Ortíz, M. Dijkstra, A boltzmann generator for the isobaric-isothermal ensemble, arXiv preprint arXiv:2305.08483 (2023).

[45] T. Salimans, D. Kingma, M. Welling, Markov chain monte carlo and variational inference: Bridging the gap, in: International conference on machine learning, PMLR, 2015, pp. 1218–1226.

[46] Y. Zhang, J. M. Hernández-Lobato, Ergodic inference: Accelerate convergence by optimisation, arXiv preprint arXiv:1805.10377 (2018).

[47] R. Habib, D. Barber, Auxiliary variational mcmc, in: International Conference on Learning Representations, 2018.

[48] F. Ruiz, M. Titsias, A contrastive divergence for combining variational inference and mcmc, in: International Conference on Machine Learning, PMLR, 2019, pp. 5537–5545.

[49] Z. Shen, M. Heinonen, S. Kaski, De-randomizing mcmc dynamics with the diffusion stein operator, Advances in Neural Information Processing Systems 34 (2021) 17507–17517.

[50] M. Gabrié, G. M. Rotskoff, E. Vanden-Eijnden, Adaptive monte carlo augmented with normalizing flows, Proceedings of the National Academy of Sciences 119 (2022) e2109420119.

[51] T. Akhound-Sadegh, J. Rector-Brooks, A. J. Bose, S. Mittal, P. Lemos, C.-H. Liu, M. Sendera, S. Ravanbakhsh, G. Gidel, Y. Bengio, et al., Iterated denoising energy matching for sampling from boltzmann densities, arXiv preprint arXiv:2402.06121 (2024).

[52] Q. Liu, D. Wang, Stein variational gradient descent: A general purpose bayesian inference algorithm, Advances in neural information processing systems 29 (2016).

[53] W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duvenaud, R. Zemel, Learning the stein discrepancy for training and evaluating energy-based models without sampling, in: International Conference on Machine Learning, PMLR, 2020, pp. 3732–3747.

[54] L. L. di Langosco, V. Fortuin, H. Strathmann, Neural variational gradient descent, arXiv preprint arXiv:2107.10731 (2021).

[55] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, 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. 10684–10695.

[56] C. Fu, K. Yan, L. Wang, W. Y. Au, M. C. McThrow, T. Komikado, K. Maruhashi, K. Uchino, X. Qian, S. Ji, A latent diffusion model for protein structure generation, in: Learning on Graphs Conference, PMLR, 2024, pp. 29–1.

[57] B. Zheng, G. Sun, L. Dong, S. Wang, Ld-csnet: A latent diffusion-based architecture for perceptual compressed sensing, Neural Networks (2024) 106541.

[58] L. Felardos, Data-free Generation of Molecular Configurations with Normalizing Flows, Ph.D. thesis, Université Grenoble Alpes, 2022.

[59] M. Plainer, H. Stark, C. Bunne, S. Günnemann, Transition path sampling with boltzmann generator-based mcmc moves, in: NeurIPS 2023 AI for Science Workshop, 2023.

[60] B. D. Anderson, Reverse-time diffusion equation models, Stochastic Processes and their Applications 12 (1982) 313–326.

[61] R. M. Neal, Annealed importance sampling, Statistics and computing 11 (2001) 125–139.

[62] R. T. Chen, Y. Rubanova, J. Bettencourt, D. K. Duvenaud, Neural ordinary differential equations, Advances in neural information processing systems 31 (2018).

[63] L. Dinh, J. Sohl-Dickstein, S. Bengio, Density estimation using real nvp, arXiv preprint arXiv:1605.08803 (2016).

[64] D. Dua, C. Graff, Uci machine learning repository, `https://archive.ics.uci.edu/ml/index.php`, 2017. Accessed: 2022-03-13.

[65] L. Klein, A. Krämer, F. Noé, Equivariant flow matching, Advances in Neural Information Processing Systems 36 (2023) 59886–59910.

[66] Y. Zhang, Z. Ghahramani, A. J. Storkey, C. Sutton, Continuous relaxations for discrete hamiltonian monte carlo, Advances in Neural Information Processing Systems 25 (2012).

[67] S. Zheng, J. He, C. Liu, Y. Shi, Z. Lu, W. Feng, F. Ju, J. Wang, J. Zhu, Y. Min, et al., Towards predicting equilibrium distributions for molecular systems with deep learning, arXiv preprint arXiv:2306.05445 (2023).

[68] B. Jing, E. Erives, P. Pao-Huang, G. Corso, B. Berger, T. Jaakkola, Eigenfold: Generative protein structure prediction with diffusion models, arXiv preprint arXiv:2304.02198 (2023).

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

[70] M. F. Hutchinson, A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines, Communications in Statistics-Simulation and Computation 18 (1989) 1059–1076.

[71] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, A. Smola, A kernel two-sample test, The Journal of Machine Learning Research 13 (2012) 723–773.

[72] L. C. Freeman, Kish: Survey sampling (book review), Social Forces 45 (1966) 132.

Appendix A. Proof of Eq. (1)

For completeness, here we provide a proof of Eq. (1), which follows a similar derivation as the loss function of the standard VAE [43]. For simplicity of notation, we omit the explicit dependence on the distribution parameters ϕ and θ.

The KL divergence between $p_D(x, z_0)$ and $p_E(x, z_0)$ can be written as

$$\begin{aligned} D_{\mathrm{KL}}\left(p_D(x,z_0)||p_E(x,z_0)\right) &= \mathbb{E}_{p_D}\left[\log\frac{p_D(x,z_0)}{p_E(x,z_0)}\right] \\ &= \mathbb{E}_{p_D}\left[\log\frac{p_D(x)}{\pi(x)}\right] + \mathbb{E}_{p_D}\left[\log\frac{p_D(z_0|x)}{p_E(z_0|x)}\right] \\ &= D_{\mathrm{KL}}\left(p_D(x)||\pi(x)\right) \\ &\quad + D_{\mathrm{KL}}\left(p_D(z_0|x)||p_E(z_0|x)\right), \end{aligned}$$

where

$$D_{\mathrm{KL}}\left(p_D(z_0|x)\,\|\,p_E(z_0|x)\right) \triangleq \mathbb{E}_{p_D}\left[\log\frac{p_D(z_0|x)}{p_E(z_0|x)}\right]$$

is the expected KL divergence between the conditional distributions $p_D(z_0|x)$ and $p_E(z_0|x)$ under $x \sim p_D(x)$, and is therefore non-negative.

Consequently, we obtain

$$\begin{aligned} D_{\mathrm{KL}}\left(p_D(x)\,\|\,\pi(x)\right) &\leq D_{\mathrm{KL}}\left(p_D(x,z_0)\,\|\,p_E(x,z_0)\right) \\ &= D_{\mathrm{KL}}\left(p_D(z_0)p_D(x|z_0)\,\|\,\pi(x)p_E(z_0|x)\right), \end{aligned}$$

where the inequality becomes an equality if $p_D(z_0|x) = p_E(z_0|x)$ for all x, z_0.

Appendix B. Analysis of the independence between z_T and x

In the decoding process, if z_T is independent of z_0 with $p_D(z_T|z_0) = p_D(z_T)$, we have

$$\begin{aligned} p_D(x, z_T) &= \int p_D(x, z_0, z_T)\mathrm{d}z_0 \\ &= \int p_D(x, z_0)p_D(z_T|z_0)\mathrm{d}z_0 \\ &= p_D(z_T)\int p_D(x, z_0)\mathrm{d}z_0 \\ &= p_D(x)p_D(z_T), \end{aligned}$$

which implies that x is also independent of z_T.

Appendix C. Proof of Theorem 1

Part 1. Eq. (5) can be proved in a manner similar to the proof of Eq. (1) (see Appendix A). Thus, here we focus on proving that the equality in Eq. (5) holds under the conditions that z_0 and z_T are independent, and $s(z, x, t; \theta) \equiv \nabla_{z_t} \log p_D(z_t|x)$.

Under these assumptions, x is also independent of z_T in the decoding process (see Appendix B). Consequently, the conditional distribution of $z_{[\cdot]}$ given x can be described by the reverse-time process defined by Eq. (3). Moreover, it shares the same initial distribution $p_D(z_T|x) = p_D(z_T) = p_E(z_T)$ as the reverse-time process defined by Eq. (4) in the encoding process.

Furthermore, because $s(z, x, t; \theta) \equiv \nabla_{z_t} \log p_D(z_t|x)$, the drift and diffusion terms of the two reverse-time processes exactly match. Therefore, we have $p_D(z_{[\cdot]}|x) = p_E(z_{[\cdot]}|x)$, and

$$\begin{aligned} D_{\mathrm{KL}}\left(p_D(z_{[\cdot]}, x) \,\|\, p_E(z_{[\cdot]}, x)\right) &= D_{\mathrm{KL}}\left(p_D(x) \,\|\, \pi(x)\right) \\ &\quad + D_{\mathrm{KL}}\left(p_D(z_{[\cdot]}|x) \,\|\, p_E(z_{[\cdot]}|x)\right) \\ &= D_{\mathrm{KL}}\left(p_D(x) \,\|\, \pi(x)\right). \end{aligned}$$

Part 2. We now prove Eq. (6). In the decoding process, the distribution of $z_{[\cdot]}$ can be described by the following reverse-time process:

$$\mathrm{d}z_{\tilde{t}} = -\left(f(z_{\tilde{t}}, \tilde{t}) - g(\tilde{t})^2 \nabla_{z_{\tilde{t}}} \log p_D(z_{\tilde{t}})\right) \mathrm{d}\tilde{t} + g(\tilde{t}) \mathrm{d}\tilde{W}_{\tilde{t}}, \;\; z_T \sim p_D(z_T).$$

Combining this with Eq. (4) and applying Girsanov theorem [69], we obtain

$$\begin{aligned} \mathbb{E}_{p_D}\left[\log \frac{p_D(z_{[\cdot]})}{p_E(z_{[\cdot]}|x)}\right] &= \mathbb{E}_{p_D}\left[\int_0^T g(t) \left(\nabla_{z_t} \log p_D(z_t) - s(z_t, x, t; \theta)\right) \mathrm{d}\tilde{W}_t\right] \\ &\quad + \frac{1}{2} \mathbb{E}_{p_D}\left[\int_0^T g(t)^2 \left\|\nabla_{z_t} \log p_D(z_t) - s(z_t, x, t; \theta)\right\|^2 \mathrm{d}t\right]. \end{aligned}$$

Note that the first term on the right-hand side equals zero due to the martingale property of Itô integrals. Thus, we have

$$\begin{aligned} D_{\mathrm{KL}}\left(p_D(z_{[\cdot]},x)\,\|\,p_E(z_{[\cdot]},x)\right) &= \mathbb{E}_{p_D}\left[\log\frac{p_D(x|z_0)}{\pi(x)}\right]+\mathbb{E}_{p_D}\left[\log\frac{p_D(z_{[\cdot]})}{p_E(z_{[\cdot]}|x)}\right] \\ &= \mathbb{E}_{p_D}\left[\log\frac{p_D(x|z_0)}{\pi(x)}\right] \\ &\quad +\frac{1}{2}\mathbb{E}_{p_D}\left[\int_0^T g(t)^2\left\|\nabla_{z_t}\log p_D(z_t)-s(z_t,x,t;\theta)\right\|^2\mathrm{d}t\right] \\ &= \mathbb{E}_{p_D}\left[\log\frac{p_D(x|z_0)}{\pi(x)}\right] \\ &\quad +\frac{1}{2}\mathbb{E}_{p_D}\left[\int_0^T g(t)^2\left(\left\|\nabla_{z_t}\log p_D(z_t)\right\|^2+\left\|s(z_t,x,t;\theta)\right\|^2\right)\mathrm{d}t\right] \\ &\quad +\mathbb{E}_{p_D}\left[\int_0^T g(t)^2\left(s(z_t,x,t;\theta)^\top\nabla_{z_t}\log p_D(z_t)\right)\mathrm{d}t\right]. \end{aligned} \tag{C.1}$$

Moreover, we consider the following integration by parts:

$$\begin{aligned} \mathbb{E}_{p_D}\left[s\left(z_t,x\right)^\top\nabla_{z_t}\log p_D(z_t|z_0)|x,z_0\right] &= \int p_D(z_t|z_0)s\left(z_t,x\right)^\top\nabla_{z_t}\log p_D(z_t|z_0)\mathrm{d}z_t \\ &= \int s\left(z_t,x\right)^\top\nabla_{z_t}p_D(z_t|z_0)\mathrm{d}z_t \\ &= \int \mathrm{div}_{z_t}\left(p_D(z_t|z_0)s\left(z_t,x\right)\right)\mathrm{d}z_t \\ &\quad -\int p_D(z_t|z_0)\mathrm{tr}\left(\frac{\partial s(z_t,x)}{\partial z_t}\right)\mathrm{d}z_t \\ &= -\mathbb{E}_{p_D}\left[\mathrm{tr}\left(\frac{\partial s(z_t,x)}{\partial z_t}\right)|z_0\right], \end{aligned}$$

which yields

$$\mathbb{E}_{p_D}\left[s\left(z_t,x\right)^\top\nabla_{z_t}\log p_D(z_t|z_0)\right] = -\mathbb{E}_{p_D}\left[\mathrm{tr}\left(\frac{\partial s(z_t,x)}{\partial z_t}\right)\right]. \tag{C.2}$$

By substituting Eq. (C.2) into Eq. (C.1), we finally obtain Eq. (6).

It is worth noting that we could also directly design a loss function based on Eq. (C.1). Such a loss is theoretically equivalent to the one used in this work, but may suffer from large variance in $\nabla_{z_t}\log p_D(z_t)$ when t is small.

Appendix D. Proof of equivalence between Eq. (7) and Eq. (8)

First, we prove the Hutchinson estimator in Eq. (8). In Eq. (7), one frequently needs the divergence $\nabla\cdot s(z_t,x,t;\theta)=\mathrm{tr}(\partial_{z_t}s)$ of a vector field $s:\mathbb{R}^d\to\mathbb{R}^d$. Deep learning

frameworks (PyTorch, TensorFlow) do not natively provide an efficient way to form an entire $d \times d$ Jacobian for large d, while Hutchinson estimator replace $\mathrm{tr}(\cdot)$ with the unbiased Monte Carlo estimate without full Jacobians.

Considering a random variable $p(\epsilon) \sim \text{Rademacher}^d$ with $\mathbb{E}[\epsilon] = 0$ and $\mathrm{Cov}(\epsilon) = I$, we can get a stochastic estimation of the trace according to [70]

$$\mathrm{tr}(\partial_{z_t} s) \quad = \quad \mathbb{E}_{p(\epsilon)}[\epsilon^\top \partial_{z_t} s \ \epsilon].$$

By the rules of matrix differential calculus,

$$\mathrm{d}\big(\epsilon^\top s(z_t, x, t; \theta)\big) = \epsilon^\top \, \mathrm{d}s(z_t, x, t; \theta) = \epsilon^\top \left(\partial_{z_t} s(z_t, x, t; \theta)\right) \mathrm{d}z_t.$$

Then, it enables the use of automatic differentiation for calculation,

$$\epsilon^\top \left(\partial_{z_t} s(z_t, x, t; \theta)\right) = \frac{\partial\left[\epsilon^\top s(z_t, x, t; \theta)\right]}{\partial z_t}.$$

Therefore, we have

$$\nabla \cdot s(z_t, x, t; \theta) \quad = \quad \mathbb{E}_{p(\epsilon)}[\frac{\partial\left[\epsilon^\top s(z_t, x, t; \theta)\right]}{\partial z_t} \ \epsilon],$$

where $p(\epsilon) \sim \text{Rademacher}^d$ denotes a random vector in $\mathbb{R}^d$ with i.i.d. Rademacher entries, satisfying $\mathbb{E}[\epsilon] = 0$ and $\mathrm{Cov}(\epsilon) = I$.

Then, we prove the expectation of a uniform time distribution t over $[0, T]$ in Eq. (8). When $t \sim \mathcal{U}[0, T]$, its probability density function (PDF) is

$$p_{\mathcal{U}[0,T]}(t) = \begin{cases} \frac{1}{T}, & 0 \le t \le T \\ 0, & \text{otherwise} \end{cases}.$$

Then we simplify the integration in Eq. (7)

$$\begin{aligned} \int_0^T \frac{g(t)^2}{2} \mathbb{E}_{p_D}[\cdot]\mathrm{d}t \quad &= \quad \int_0^T \frac{1}{T} \frac{T g(t)^2}{2} \mathbb{E}_{p_D}[\cdot]\mathrm{d}t \\ &= \quad \int_{-\infty}^{+\infty} p_{\mathcal{U}[0,T]}(t) \frac{T g(t)^2}{2} \mathbb{E}_{p_D}[\cdot]\mathrm{d}t \\ &= \quad \mathbb{E}_{t \sim \mathcal{U}[0,T]} \left[\frac{T g(t)^2}{2} \mathbb{E}_{p_D}[\cdot] \right]. \end{aligned}$$

Combining two parts above, we can achieve

$$\begin{aligned} &\int_0^T \frac{g(t)^2}{2} \mathbb{E}_{p_D} \left[\left\| s\left(z_t, x, t; \theta\right) \right\|^2 + 2 \nabla_{z_t} \cdot s\left(z_t, x, t; \theta\right) \right] \mathrm{d}t \\ = \quad &\mathbb{E}_{t \sim \mathcal{U}[0,T], \epsilon \sim p(\epsilon), (x, z_t) \sim p_D(x, z_t)} \left[\frac{T g(t)^2}{2} \left(\left\| s\left(z_t, x, t; \theta\right) \right\|^2 + 2 \frac{\partial\left[\epsilon^\top s\left(z_t, x, t; \theta\right)\right]}{\partial z_t} \epsilon \right) \right], \end{aligned}$$

where $\mathcal{U}[0, T]$ denotes a uniform distribution over $[0, T]$, and $p(\epsilon) \sim \text{Rademacher}^d$ denotes a random vector in $\mathbb{R}^d$ with i.i.d. Rademacher entries, satisfying $\mathbb{E}[\epsilon] = 0$ and $\mathrm{Cov}(\epsilon) = I$.

Appendix E. Calculation of the probability flow ODE

Considering the SDE in Eq. (4), there is a corresponding deterministic ODE whose trajectories share the same marginal probability density

$$\mathrm{d}z_t = \underbrace{\left\{ f(z_t, t) - \frac{1}{2} g(t)^2 s_\theta(z_t, x, t) \right\}}_{=: F_\theta(z_t, x, t)} \mathrm{d}t.$$

With the change of variables formula [62], we can compute the log-likelihood of $p_E(z_0|x)$ using

$$\log p_E(z_0|x) = \log p_E(z_T) + \int_0^T \nabla \cdot F_\theta(z_t, x, t) \mathrm{d}t.$$

Due to the expensive computational cost of $\nabla \cdot F_\theta(z_t, x, t)$, the Hutchinson trace estimator technique in Appendix D is used again to calculate

$$\nabla \cdot F_\theta(z_t, x, t) = \mathbb{E}_{p(\epsilon)} \left[\frac{\partial [\epsilon^\top F_\theta(z_t, x, t)]}{\partial z_t} \epsilon \right],$$

where $p(\epsilon) \sim \text{Rademacher}^d$.

In our experiments, we employ the RK45 ODE solver implemented in `scipy.integrate.solve_ivp`, where the parameters set identical to those in [34] with `atol=1e-5` and `rtol=1e-5`.

Appendix F. Analysis of Eq. (14)

Based on Eq. (1) and $w(x, z_0) = \frac{\exp(-U(x)) p_E(z_0|x)}{p_D(z_0) p_D(x|z_0)}$, we have

$$\begin{aligned} \log Z &\geq D_{\mathrm{KL}}(p_D(x) \| \pi(x)) - \mathbb{E}_{p_D(z_0) p_D(x|z_0;\phi)} \left[\log \frac{p_D(z_0) p_D(x|z_0;\phi)}{\exp(-U(x)) p_E(z_0|x;\theta)} \right] \\ &= D_{\mathrm{KL}}(p_D(x) \| \pi(x)) + \mathbb{E}_{p_D(z_0) p_D(x|z_0;\phi)} \left[\log w(x, z_0) \right]. \end{aligned}$$

Therefore, any Monte Carlo estimator of $\mathbb{E}_{p_D(z_0) p_D(x|z_0;\phi)} [\log w(x, z_0)]$ yields a lower bound on $\log Z$.

Moreover, according to Appendix A, as the model parameters are optimized during training, the distribution $p_E(z_0|x;\theta)$ is expected to approximate the true conditional distribution $p_D(z_0|x)$ more closely. This improved approximation deduces

$$\mathbb{E}_{p_D(z_0) p_D(x|z_0;\phi)} [\log w(x, z_0)] \to \log Z - D_{\mathrm{KL}}(p_D(x) \| \pi(x)).$$

Consequently, a larger $\mathbb{E}_{p_D(z_0) p_D(x|z_0;\phi)} [\log w(x, z_0)]$ suggests that the model is learning effectively, as $D_{\mathrm{KL}}(p_D(x) \| \pi(x))$ diminishes and the estimated $\log Z$ becomes more accurate.

Appendix G. Implementation of decoder

The detailed implementation of the GHD based decoder is summarized in Algorithm 3. Within the algorithm, we initially stochastically generate a sample y using the random variable ζ (refer to Line 1 of the algorithm). Subsequently, we iteratively update the sample using random velocities $v_1, \ldots, v_K$ and GHD (see Line 5). Finally, we utilize discretized Brownian dynamics with a trainable step size to design the decoder density of x (refer to Line 9). Here, ϵ, Q_v, Q_y, T_v, T_y and η are all neural networks composed of three-layer MLPs. ϵ_0 serves as an initial hyperparameter of trainable step size ϵ and η, and our numerical experiments suggest that setting ϵ_0 to a small positive value can improve the stability of the algorithm.

Appendix H. Experimental details

EDG In our experiments, we leverage the subVP SDE proposed in [34] to model the diffusion process of z_t in Eq. (2), defined as

$$\mathrm{d}z_t = -\frac{1}{2}\beta(t)z_t\mathrm{d}t + \sqrt{\beta(t)\left(1 - e^{-2\int_0^t \beta(s)\mathrm{d}s}\right)}\mathrm{d}W_t$$

Algorithm 3 GHD based decoder

Require: $z_0 = (\zeta, v_1, \ldots, v_K)$ drawn from the standard Gaussian distribution and decoder parameters ϕ

1: Let $y := a(\zeta; \phi)$.
2: **for** $k = 1, \ldots, M$ **do**
3: **for** $j = 1, \ldots, J$ **do**
4: Let $l :=$ leap frog step of GHD.
5: Update (y, v_i) by

$$\begin{aligned} v_k &:= v_k - \frac{\epsilon_0 e^{\epsilon_0\epsilon(l;\phi)}}{2}\left(\nabla U(y) \odot e^{\frac{\epsilon_0}{2}Q_v(y,\nabla U(y),l;\phi)} + T_v(y, \nabla U(y), l; \phi)\right) \\ y &:= y + \epsilon_0 e^{\epsilon_0\epsilon(l;\phi)}\left(v_k \odot e^{\epsilon_0 Q_y(v_k,l;\phi)} + T_y(v_k, l; \phi)\right) \\ v_k &:= v_k - \frac{\epsilon_0 e^{\epsilon_0\epsilon(l;\phi)}}{2}\left(\nabla U(y) \odot e^{\frac{\epsilon_0}{2}Q_v(y,\nabla U(y),l;\phi)} + T_v(y, \nabla U(y), l; \phi)\right) \end{aligned}$$

6: **end for**
7: Let $v_k := -v_k$
8: **end for**
9: Let $\mu = y - \epsilon_0 e^{\epsilon_0\eta(y;\phi)}\nabla U(y)$ and $\Sigma = 2\epsilon_0 e^{\epsilon_0\eta(y;\phi)}I$.
10: **return** $x \sim \mathcal{N}(\mu, \Sigma)$, $p_D(x|z_0; \phi)$.

The model architecture of 2D energy tasks

Decoding process	Encoding process
Network $a(\zeta;\phi)$: fc 10×32, ReLU, $2\times$(fc 32×32, ReLU), fc 32×2. GHD: Q_v, T_v: fc 5×10, Tanh, fc 10×10, Tanh, fc 10×2. Q_x, T_x: fc 3×10, ReLU, fc 10×10, ReLU, fc 10×2. Final network $\mathcal{N}(\mu,\Sigma)$: fc 2×10, ReLU, fc 10×10, ReLU, fc 10×2.	Score network: fc 13×16, ReLU, fc 16×16, ReLU, fc 16×10.

with $T = 1$. Here, $\beta(t) = \beta_{\min} + t\,(\beta_{\max} - \beta_{\min})$, and we set $\beta_{\min} = 0.1$ and $\beta_{\min} = 20$ for experiments. The initial state z_0 follows a standard Gaussian distribution. Line 1 of Algorithm 3 is implemented as

$$y := \mu_0(\zeta_0;\phi) + \Sigma_0(\zeta_0;\phi)\zeta_1,$$

where $\zeta = (\zeta_0, \zeta_1)$, and $\mu_0(\zeta_0;\phi), \Sigma_0(\zeta_0;\phi)$ are modeled by MLPs. In the GHD decoder design, we fix $J = 5$ while setting $M = 2$ for 2D energy function tasks, $M = 10$ for Bayesian Logistic Regression tasks and Ising models, and $M = 100$ for Lennard-Jones.

Baseline For V-HMC, we facilitate the generation of samples after a warm-up phase comprising 1,000 steps, where each step the Metropolis-Hastings acceptance probability is computed to ensure the convergence. For L2HMC, we use a three-layer fully connected network, setting the leapfrog step length to 10. For different tasks, we adaptively select varying step sizes. For PIS, we use the publicly available code on our tasks directly. Regarding BG, the RealNVP architecture consists of 3 affine blocks, where the scaling and translation functions are modeled by a three-layer fully connected network, each with 256 units and ReLU activation functions. We would like to highlight that unlike data-driven energy-based models [19, 20], our baseline BG generates samples solely from a prescribed energy function without any access to real data. The loss function is the KL divergence between the generated samples by BG and the target.

2D energy function We use the following model architectures to generate samples for 2D energy tasks. 'fc n' denotes a fully connected layer with n neurons.

Reference samples for Mog2, Mog2(i), Mog6, and Mog9 in Fig. 2 are precisely drawn from the target distributions, as these distributions are mixtures of Gaussian distributions. Reference samples for Ring and Ring5 are generated using the Metropolis-Hastings algorithm, with a large number of iterations. Gaussian distributions with variances of 9 and 25, respectively, are used as the proposal distributions. For the V-HMC, we employ the use of an HMC run of a total length of 2,000 steps, initiating from a standard normal distribution. The first 1,000 steps are designated as burn-in steps followed by a subsequent 1,000 steps used for sample generation. This procedure is independently executed 500 times for the purpose of creating the visual representation.

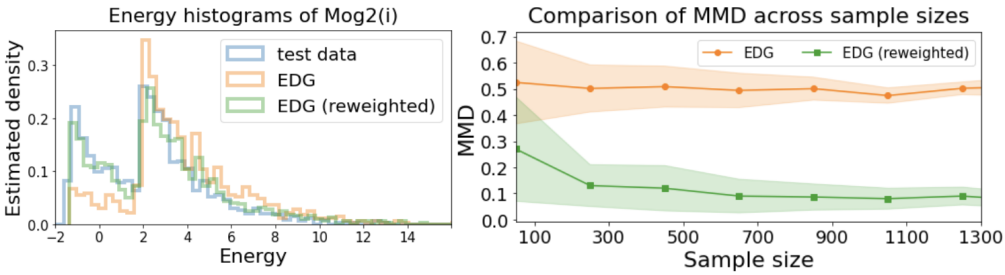


Figure H.5: Reweighting effect in Mog2(i) task. **Left:** The energy histograms of 1000 generated samples under EDG and EDG (reweighted). **Right:** The mean value of the reweighted MMD with respect to the sample size. We report the mean value ± standard deviation over 10 runs.

Given the reference samples $X = \{x_i\}_{i=1}^m$ with weights $w^X = \{w_i^X\}_{i=1}^m = \frac{1}{m}$ and the generated samples $Y = \{y_i\}_{i=1}^m$ with weights $w^Y = \{w_i^Y\}_{i=1}^m$, the Maximum Mean Discrepancy (MMD) [71] measures the difference between the distributions of X and Y as follows:

$$\mathrm{MMD}^2(X,Y) = \sum_i \sum_j w_i^X w_j^X k(x_i, x_j) - 2 \sum_i \sum_j w_i^X w_j^Y k(x_i, y_j) + \sum_i \sum_j w_i^Y w_j^Y k(y_i, y_j),$$

where $k(\cdot,\cdot)$ denotes the kernel function to compute the inner product. RBF kernel is used in Table 1, and the bandwidth is set as the median distance between corresponding samples. We conduct 10 independent runs to generate 5,000 samples with uniform density, and compute the mean MMD against 5,000 reference samples. The standard deviations are all small, hence we do not report them in the table.

To better illustrate the effectiveness of the reweighting procedure in Sec. 3.3, we use Mog2(i)—a case with room for further improvement—as an example. Firstly, we display the energy histograms of the generated samples under EDG and EDG (reweighted) in the left panel of Fig. H.5. Then, we compute the MMD using generated samples by EDG with normalized weights, which are calculated according to Sec. 3.3. The mean value across 10 independent runs are reported in Table H.5, each based on 5,000 samples. The standard deviations are omitted as they are negligibly small. The convergence of the weighted MMD with respect to the sample size is also presented in the right panel of Fig. H.5.

Bayesian Logistic Regression In all experiments, we employ the same data partition and the datasets are divided into training and test sets at a ratio of 4:1. Before training, we normalize all datasets to have zero mean and unit variance. For d-dimensional features, the architectures of neural networks involved in the EDG are as follows:

In the task of covertype, $x = (\alpha, w, b)$ with the prior distribution $p(x) = p(\alpha)p(w, b|\alpha)$, where $p(\alpha) = \mathrm{Gamma}(\alpha; 1, 0.01)$ and $p(w, b|\alpha) = \mathcal{N}(w, b; 0, \alpha^{-1})$.

Lennard-Jones (LJ) The Lennard-Jones (LJ) potential is an intermolecular potential which models repulsive and attractive interactions of non-bonding atoms or molecules.

Table H.5: The mean of the weighted MMD results over 10 rounds for EDG and EDG (reweighted), each using 5,000 samples. For EDG, all samples follow a uniform distribution. For EDG (reweighted), generated samples of EDG are assigned normalized weights, which are calculated according to Sec. 3.3. The standard deviations are omitted as they are negligibly small.

	EDG	EDG (reweighted)
Mean value of the reweighted MMD	0.50	0.09

The model architecture of Bayesian Logistic Regression

Decoding process	Encoding process
Network $a(\zeta;\phi)$: fc 30×256, ReLU, $2\times$(fc 256×256, ReLU), fc $256 \times d$. GHD: Q_v, T_v: fc $(2d+1) \times 256$, Tanh, fc 256×256, Tanh, fc $256 \times d$. Q_x, T_x: fc $(d+1) \times 256$, ReLU, fc 256×256, ReLU, fc $256 \times d$. Final network $\mathcal{N}(\mu, \Sigma)$: fc $11d \times 256$, ReLU, fc 256×256, ReLU, fc $256 \times d$.	Score network: fc $(d+31) \times 256$, ReLU, fc 256×256, ReLU, fc 256×30.

The energy is based on the distance of interacting particles $\mathcal{E}^{\mathrm{LJ}}(\cdot)$ and a harmonic potential $\mathcal{E}^{\mathrm{osc}}(\cdot)$ like [65]

$$\begin{aligned} \mathcal{E}^{\mathrm{LJ}}(x) &= \frac{1}{2}\sum_{ij}\left(\left(\frac{1}{d_{ij}}\right)^6 - \left(\frac{1}{d_{ij}}\right)^{12}\right), \\ \mathcal{E}^{\mathrm{osc}}(x) &= \frac{1}{2}\sum_{i}\|x_i - x_{\mathrm{COM}}\|^2, \end{aligned}$$

where $d_{ij} = \|x_i - x_j\|^2$ is the Euclidean distance between particles i and j and x_{COM}

The model architecture of Lennard-Jones

Decoding process	Encoding process
Network $a(\zeta;\phi)$: fc 100×32, ReLU, $2\times$(fc 32×32, ReLU), fc $32 \times d$. GHD: Q_v, T_v: fc $(2d+1) \times 32$, ReLU, fc 32×32, ReLU, fc $32 \times d$. Q_x, T_x: fc $(d+1) \times 32$, ReLU, fc 32×32, ReLU, fc $32 \times d$. Final network $\mathcal{N}(\mu, \Sigma)$: fc $d \times 32$, ReLU, fc 32×32, ReLU, fc $32 \times d$.	Score network: fc $(d+101) \times 256$, ReLU, fc 256×256, ReLU, fc 256×100.

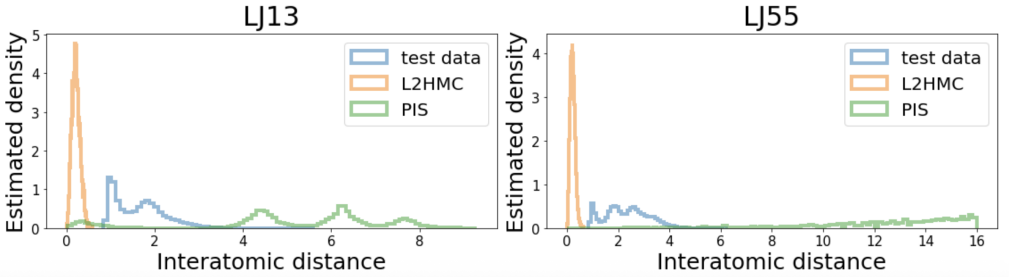


Figure H.6: Comparison of the test data interatomic distance of LJ13 (left), LJ55 (right) with samples generated from L2HMC and PIS. Results for BG and EDG is ploted in Fig. 3.

refers to the center of mass of the system. The target energy function has the form $\mathcal{E}(\cdot) = \mathcal{E}^{\mathrm{LJ}}(\cdot) + \mathcal{E}^{\mathrm{osc}}(\cdot)$. The system is performed without periodic boundary conditions, which means particles are not wrapped across boundaries, no periodic images or minimum-image convention are used, and interactions are conducted within the finite simulation domain by harmonic confinement. For the experimental results, we evaluate the generated sample with the test sample from [65]. The test data were generated with MCMC with 1000 parallel chains, where each chain is run for 10,000 steps after a long burn-in phase of 200,000 steps starting from a random generated initial state. The density estimation results of interatomic distances for EDG and BG are plotted in Fig. 3. The results for L2HMC and PIS are presented here in Fig. due to significant deviations from the test data.

To evaluate the effectiveness of the sample reweighting strategy described in Sec. 3.3, we employ the relative Effective Sample Size (rESS), defined as

$$\mathrm{rESS} = \frac{1}{N}\mathrm{ESS} = \frac{\left(\sum_{n=1}^{N} w(x^n, z_0^n)\right)^2}{N \sum_{n=1}^{N} w(x^n, z_0^n)^2}. \tag{H.1}$$

Here, ESS denotes the Effective Sample Size [72], a widely used metric for evaluating the quality of weighted samples produced by importance sampling based on NFs (see, e.g., [26–28, 32]). The rESS value lies in the interval $(0, 1]$ and measures the fraction of effectively independent samples relative to the total sample size N. A low rESS indicates that the resulting estimators may suffer from high variance due to poor weight balance.

Table H.6: rESS with mean ± standard deviation over 10 seeds. For each method, we use 256 samples for estimation.

rESS	BG	PIS	EDG
LJ13	0.006 ± 0.002	0.005 ± 0.001	$\mathbf{0.132 \pm 0.048}$
LJ55	0.004 ± 0.000	0.004 ± 0.000	$\mathbf{0.098 \pm 0.014}$

Ising model The model architecture is shown below, which is similar to that in NeuralRG [25] without stacking bijectors to form a reversible transformation. It retains the multiscale entanglement renormalization ansatz structure, while we only use one block to update the whole dimensions of the variable directly. To evaluate the efficiency of our proposed architecture, we include a comparative analysis against a standard MLP design in Appendix J. The normalizing constant $\log Z$ is approximated according to Sec. 3.3.

The model architecture of Ising model

Decoding process	Encoding process
Network $a(\zeta;\phi)$: hierarchy network as shown in [25] GHD: Q_v, T_v: Conv2d 2×10, BatchNorm2d, ReLU, Conv2d 10×1. Q_x, T_x: Conv2d 1×10, BatchNorm2d, ReLU, Conv2d 10×1. Final network $\mathcal{N}(\mu,\Sigma)$: fc $d\times 10$, ReLU, fc 10×10, ReLU, fc $10\times d$.	Score network: hierarchy network. Input: z_t, x, t, T

Appendix I. Computation cost

To demonstrate the computational efficiency of our model, we have compared the wall-clock training time and memory usage on the 2D energy (Mog2) task for the following models: the trainable MCMC model L2HMC [14], the VI-based model BG [21], and the simulation-based model PIS [37]. For L2HMC and PIS, we adopted the default settings from their respective original papers. The network architectures for BG and EDG are detailed in Appendix H. All experiments were conducted on a single NVIDIA RTX 2080 Ti GPU with 11GB of VRAM. The runtime comparison is shown in Fig. I.7, and memory usage is reported in Table I.7.

The results show that EDG reaches competitive accuracy in less training time and with lower memory consumption compared to the other methods. Although the per-epoch training time of BG is shorter than that of EDG, EDG achieves better performance in fewer epochs.

Additionally, we compared the reweighting time cost of three methods that support post-hoc sample reweighting: BG, PIS, and our proposed EDG. The results are presented in Fig. I.8. Among them, BG achieves fast computation of the weight function due to the efficient evaluation of the Jacobian determinant in each affine coupling layer. In contrast, PIS and EDG must solve stochastic and ordinary differential equations respectively to compute weights, resulting in higher computational cost. The reweighting time of EDG and PIS is of the same order of magnitude, and EDG is slightly slower. This is mainly because the reweighting in EDG relies on the neural ODE technique, which involves divergence computation that contributes significantly to the time cost. PIS does not require divergence computation during reweighting. It is worth emphasizing that PIS also requires

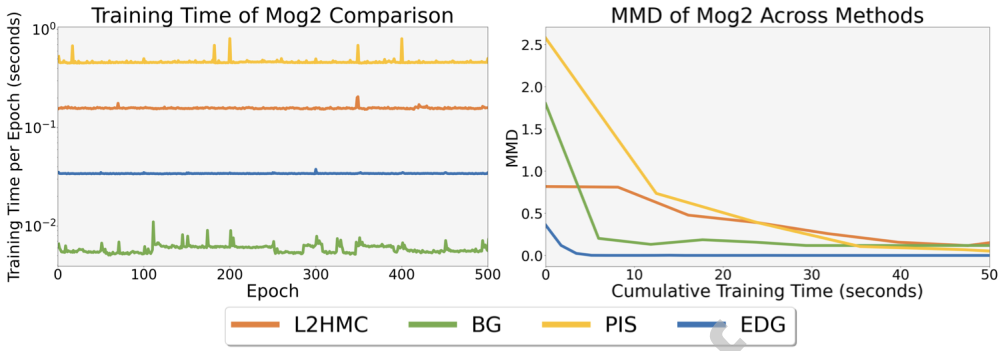


Figure I.7: Computational time comparison across different methods. **Left:** the logarithm of the per-epoch training time for each method under a consistent batch size of 128. **Right:** the evolution of MMD in the Mog2 task with respect to cumulative training time, where all evaluation time has been excluded.

Table I.7: Training memory usage of different methods, measured by `torch.cuda.memory_allocated`.

	L2HMC	BG	PIS	EDG
Memory (MB)	17.19	22.41	55.83	16.75

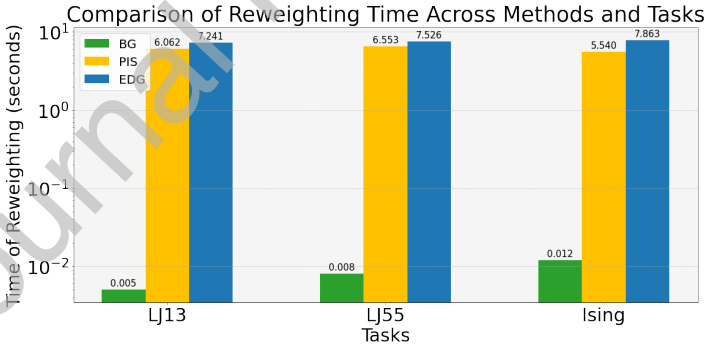


Figure I.8: The computational cost of sample reweighting in BG, PIS and EDG. Both PIS and EDG require temporal integration, for which distinct numerical schemes are employed to achieve comparable precision. PIS utilizes `torchsde.sdeint` with the Stochastic Runge-Kutta (SRK) method and a fixed step size of $T/1000$, while EDG adopts `scipy.integrate.solve_ivp` using the RK45 method with `atol=1e-5`, `rtol=1e-5`.

numerical SDE solvers during training, while EDG avoids solving either SDEs or ODEs during training. As shown in Fig. I.7, this leads to significantly lower training time for EDG compared to PIS.

Appendix J. Ablation study

GHD in Decoder: To elucidate the function of GHD in EDG, we compare the following models on the sampling task of 2D energy functions: a VAE with the Gaussian decoder and encoder, where the means and diagonal covariance matrices are both parameterized by MLPs; a VAE with a GHD-based decoder and MLP-based encoder; EDG without GHD, where the decoder is modeled by MLPs; and the full EDG model. For VAE w/o GHD, the network is composed of 3-layer MLPs, each with 32 units and ReLU activation function. Starting from $z \sim \mathcal{N}(0, I_d)$, $d = 10$, the decoder generates samples $x \sim p_D(x; \mu(z;\phi), \Sigma(z;\phi))$. The encoder, as an independent network, follows $p_E(z|x) = \mathcal{N}(z; \mu(x;\theta), \Sigma(x;\theta))$. The objective function is the KL divergence of the joint distribution of z and x as

$$D_{\mathrm{KL}}(p_D(z)p_D(x|z;\phi)||\pi(x)p_E(z|x;\theta).$$

For VAE w/ GHD, the decoder is consistent with the structure in Algorithm 3, and the encoder is the same as described above. For EDG w/o GHD, we omit the leap-frog component (refer to Line 2-8 in the algorithm) and the decoder is composed of the network $a(\zeta;\phi)$ and a final gaussian part $\mathcal{N}(\mu, \Sigma)$ (refer to Line 1, 9 in the algorithm).

Table J.8: The MMD between 5,000 samples generated by each generator and the reference samples.

	Mog2	Mog2(i)	Mog6	Mog9	Ring	Ring5
VAE w/o GHD	1.86	1.62	2.57	2.10	0.12	0.23
VAE w/ GHD	**0.01**	2.43	0.15	0.59	1.68	0.63
EDG w/o GHD	0.06	1.01	0.04	0.05	0.02	0.04
EDG	**0.01**	**0.50**	**0.01**	**0.02**	**0.01**	**0.02**

The histograms of the samples are shown in Fig. J.9 for visual inspection and the sampling errors is summarized in Tab. J.8. It is evident that EDG with a GHD-based decoder outperforms the other methods, demonstrating the effectiveness of each component within the model.

The step size of Decoder: Step sizes will impact the model's performance for different tasks, selecting appropriate step sizes is crucial to achieve optimal results. To mitigate the influence of step size on the model's effectiveness, we treat them as trainable parameters with a hyperparameter ϵ_0 in Sec. 3.4.2). In our experiments, we systematically evaluated ϵ_0 from 0.005 to 0.1 in increments of 0.005, selecting the optimal value based on training performance. The optimal value is provided in Tab. J.9.

Network architecture: To evaluate the effectiveness of the hierarchical design, we compare it with an EDG model using standard fully connected linear layers. As shown

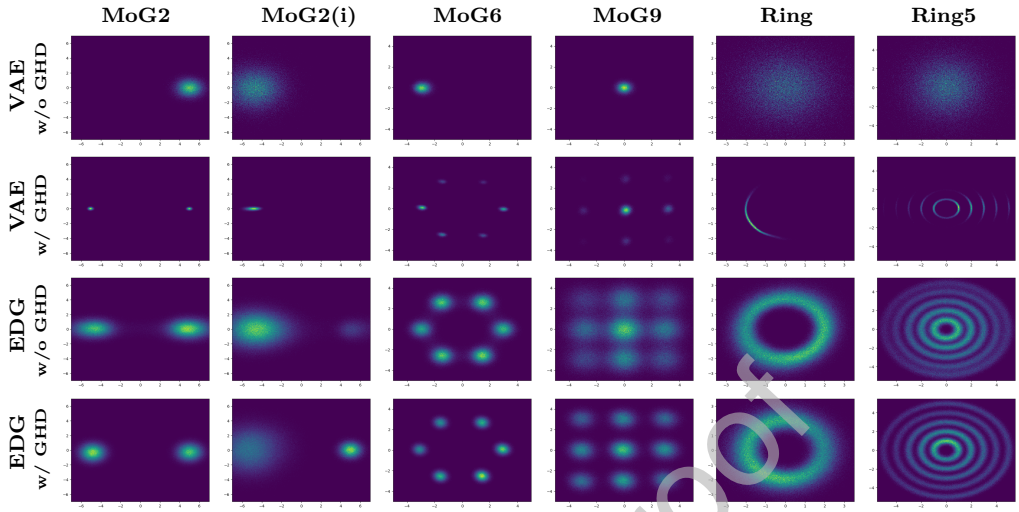


Figure J.9: Density plots for 2D energy function. We generate $500,000$ samples for each method and plot the histogram.

Table J.9: Step size hyperparameters across tasks

Task	2d energy except Rings	2d-Rings	Bayesian Regression
ϵ_0	0.1	0.03	0.05

Task	Bayesian Regression-Cover	LJ13	LJ15	Ising model
ϵ_0	0.005	0.03	0.015	0.1

in Fig. J.10, the hierarchical architecture better captures the underlying structure of the data and achieves superior performance. In future work, we plan to design specific network architectures for different tasks.

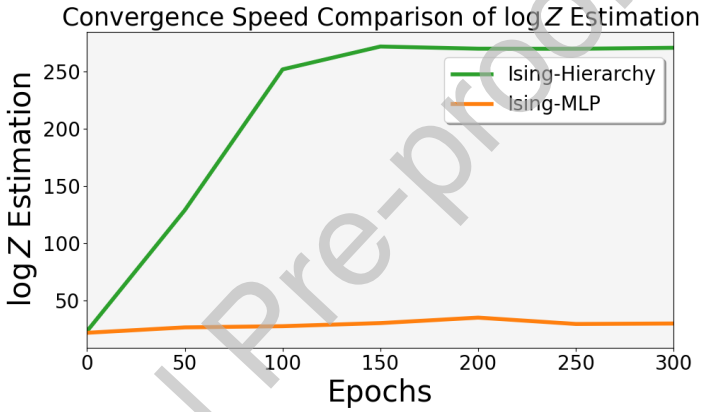


Figure J.10: The log Z estimation curves under different model architectures in the Ising model task at $T = 2.0$. Hierarchy denotes the model architecture in Appendix H and MLP refers to EDG models where all networks are implemented as 3-layer MLPs with 32 hidden units and ReLU activation functions.