DIFFUSION GENERATIVE FLOW SAMPLERS: IMPROVING LEARNING SIGNALS THROUGH PARTIAL TRAJECTORY OPTIMIZATION

Dinghuai Zhang\textsuperscript{1,2}; Ricky T. Q. Chen\textsuperscript{3}, Cheng-Hao Liu\textsuperscript{1,4}, Aaron Courville\textsuperscript{1,2} & Yoshua Bengio\textsuperscript{1,2}

\begin{itemize}
  \item \textsuperscript{1}Mila - Quebec AI Institute, \textsuperscript{2}Université de Montréal, \textsuperscript{3}FAIR, Meta AI, \textsuperscript{4}McGill University
\end{itemize}

\textbf{ABSTRACT}

We tackle the problem of sampling from intractable high-dimensional density functions, a fundamental task that often appears in machine learning and statistics. We extend recent sampling-based approaches that leverage controlled stochastic processes to model approximate samples from these target densities. The main drawback of these approaches is that the training objective requires full trajectories to compute, resulting in sluggish credit assignment issues due to use of entire trajectories and a learning signal present only at the terminal time. In this work, we present Diffusion Generative Flow Samplers (DGFS), a sampling-based framework where the learning process can be tractably broken down into short partial trajectory segments, via parameterizing an additional “flow function”. Our method takes inspiration from the theory developed for generative flow networks (GFlowNets), allowing us to make use of intermediate learning signals. Through various challenging experiments, we demonstrate that DGFS achieves more accurate estimates of the normalization constant than closely-related prior methods.

\section{INTRODUCTION}

While diffusion models for generative modeling can be trained with efficient simulation-free training algorithms, their application in solving sampling problems—where no data is given but only an unnormalized density function—remains challenging. This is a very general problem formulation that appears across a diverse range of research fields including machine learning (ML) (Duane et al., 1987; Neal, 1995; Hernández-Lobato & Adams, 2015) and statistics (Neal, 1998; Andrieu et al., 2004; Zhang et al., 2021), and has wide application in scientific fields such as physics (Wu et al., 2018; Albergo et al., 2019; Noé et al., 2019) and chemistry (Frenkel & Smit, 1996; Hollingsworth & Dror, 2018; Holdijk et al., 2022).

Two main lines of work to tackle this sampling problems are Monte Carlo (MC) methods and variational inference (VI). MC methods achieve sampling typically via a carefully designed Markov chain (MCMC) or ancestrally sample from sequential targets such as a series of annealed distributions (Moral & Doucet, 2002). Despite their wide application, MC methods can easily get stuck in local modes or suffer from long mixing time (Tieleman & Hinton, 2009; Desjardins et al., 2010). On the other hand, VI methods (Blei et al., 2016) use a parametric model to approximate the target distribution by minimizing some divergence. However, these models often possess limited expressiveness, and optimizing the divergence can lead to bad learning dynamics (Minka, 2001).

Recent works have formulated sampling from unnormalized density functions as stochastic optimal control problems, where a diffusion model is trained as a stochastic process to generate the target distribution. At present, the difficulty in using these algorithms is in part due to the need to sequentially sample long trajectories during training, while only the final step is actually treated as a sample from the model. The popular Kullback–Leibler (KL) problem formulation only provides a learning signal for the terminal state, which can cause credit assignment problems in learning, especially for diffusion models which require fairly long sequential chains for sampling.

\textsuperscript{*}Correspondence to dinghuai.zhang@mila.quebec
In this work, we build on top of Lahlou et al. (2023), who noted that recent methods that train diffusion models given an unnormalized density function (Zhang & Chen, 2022; Vargas et al., 2023a) can be interpreted from a GFlowNet perspective (Bengio et al., 2023). Although most GFlowNet works perform amortized inference on combinatorial discrete domains, previous work (Lahlou et al., 2023) establishes a theoretical foundation for continuous space GFlowNets and conducts numerical analysis to show their feasibility. Based on this framework, we propose the Diffusion Generative Flow Sampler (DGFS), an enhanced training method of diffusion models for sampling when given an unnormalized density function. In particular, we formulate a training approach that learns intermediate “flow functions” that aggregate information for training intermediate states. This allows us to further define partial trajectory-based training objectives, where we have access to only partial specification of the sampling path. We find that this approach significantly reduces gradient noise and helps stabilize convergence. Furthermore, our training framework enables us to receive learning signals at intermediate steps, even before the sequential sampling is completed, in contrast to previous methods that can only receive signals at the terminal steps.

In summary, our contributions are as follows,

• We propose DGFS, an effective algorithm that trains stochastic processes to sample from given unnormalized target densities.
• We show that, contrary to other diffusion-based sampling algorithms, DGFS can (1) update its parameters without having full trajectory specification; (2) receive intermediate signals even before a sampling path is finished.
• We show through extensive empirical study that these special advantages enable DGFS to benefit from more stable and informative training signals, with DGFS generating samples accurately from the target distribution.

2 Preliminaries

2.1 Sampling as stochastic optimal control

We aim to sample from a $D$-dimensional target distribution $\pi(\cdot)$ in $\mathbb{R}^D$ defined by a given unnormalized target density $\mu(\cdot)$ where $\pi(\cdot) = \frac{\mu(\cdot)}{Z}$ and $Z = \int \mu(x) \, dx$. Following a similar setup to prior works such as Tzen & Raginsky (2019); Zhang & Chen (2022); Vargas et al. (2023a), we consider a sequential latent variable model with a joint distribution denoted $Q$ of which samples follow a Markov process,

$$Q(x_0, \ldots, x_N) : \quad x_{n+1} \sim P_F(\cdot| x_n), \quad x_0 \sim p^0_F(x_0).$$ (1)

Through this sequence model, we use the marginal distribution $Q(x_N)$ at the terminal step $N$ to approximate $\pi(x_N)$. Here $p^0_F(\cdot)$ is a reference distribution at time $n = 0$, and $P_F(\cdot|$ denotes a forward transition probability (or policy) distribution. This forward policy depends on $n$, i.e. it is $P^{n+1}_F(\cdot| x_n)$, but we omit the step index $n$ out of simplicity or assume $n$ is encoded in $x_n$. We thus refer to $Q$ as the forward process in the following text\(^1\). In this work, we constrain the transition to be a unimodal Gaussian distribution:

$$P_F(\cdot| x_n) = \mathcal{N}(\cdot; x_n + h\mathbf{f}(x_n, n), h\sigma^2\mathbf{I}),$$ (2)

where $\mathbf{f}$ is a control drift parameterized by a neural network and $h$ is a constant step size. We also define a reference process with the forward policy $P^\text{ref}_F(\cdot| x_n) = \mathcal{N}(\cdot; x_n, h\sigma^2\mathbf{I}),$

$$Q^\text{ref}(x_0, \ldots, x_N) : \quad x_{n+1} \sim P^\text{ref}_F(\cdot| x_n), \quad x_0 \sim p^0_F(x_0).$$ (3)

Since $P^\text{ref}_F$ has no nonlinear $\mathbf{f}$, the marginal distributions of $Q^\text{ref}$ at step $n$ are known in closed form, denoted $p^\text{ref}_n(\cdot)$. We can then construct a target process that corresponds to this reference process,

$$P(x_0, \ldots, x_N) := Q^\text{ref}(x_0, \ldots, x_N) \frac{\pi(x_N)}{p^\text{ref}_N(x_N)}.$$ (4)

\(^1\)In this work, the term “flow” specifically denotes the flow network structure in GFlowNets, which has nothing to do with normalizing flow models.

\(^2\)Note that this is consistent with prior GFlowNet literature, though it is the opposite direction considered by standard diffusion model literature in generative modeling, where “forward process” is colloquially used to denote the process from clean sample to white noise.
It can be shown that the marginal distribution of \( P \) at the terminal time \( N \) is exactly proportional to the target \( \mu(\cdot) \) by marginalizing out the variables \( x_0, \ldots, x_{N-1} \) in \( P \); see Theorem 1 of Zhang & Chen (2022). Hence it is natural to consider learning the drift \( f(\cdot, \cdot) \) in Equation 2 by minimizing the KL divergence \( D_{\text{KL}}(Q||P) \) between the forward process and this target process, which is equivalent to the following optimal control problem:

\[
Q(x_0; N): \quad x_{n+1} = x_n + hf(x_n, n) + \sqrt{h} \sigma \epsilon_n, \quad x_0 \sim p^\text{ref}_0(\cdot),
\]

\[
\min_{f} \mathbb{E}_{Q} \left[ \sum_{n=0}^{N-1} \frac{h}{2\sigma^2} \|f(x_n, n)\|^2 + \log \Psi(x_N) \right], \quad \Psi(\cdot) := \frac{p^\text{ref}_N(\cdot)}{\mu(\cdot)},
\]

where \( \epsilon_n \sim N(0, I), \forall n. \) See Appendix B.1 for derivation.

If we keep the total time \( T = hN \) fixed and push the number of steps \( N \to \infty \), this formulation recovers the continuous-time stochastic optimal control formulation seen in prior works (Zhang & Chen, 2022):

\[
Q(x_{[0,T]}): \quad dx_t = f(x_t, t) \, dt + \sigma \, dW_t, \quad x_0 \sim p^\text{ref}_0(x_0),
\]

\[
\min_{f} \mathbb{E}_{Q} \left[ \int_0^T \frac{1}{2\sigma^2} \|f(x_t, t)\|^2 \, dt + \log \Psi(x_T) \right],
\]

where the forward process is defined by a variance exploding stochastic differential equation (VE-SDE)

\(^3\), and \( dW_t \) is a standard Wiener process. This formulation appears in standard stochastic optimal control theory (Kappen, 2005). The seminal path integral sampler (PIS) (Zhang & Chen, 2022) proposes to solve such optimal control problems with deep learning to tackle the sampling problem. Its follow-up, the denoising diffusion sampler (DDS) (Vargas et al., 2023a), takes a similar approach but adopts a variance preserving (VP) SDE rather than the variance exploding ones in Equation 5 and 7. We defer the derivation of VP modeling to Appendix B.2. The practical implementations of these prior works are performed in discrete time with constant step sizes, equivalent to the discrete formulation outlined above. Hence we also adopt the discrete-time formulation in the following sections for simplicity and better amenability to our proposed algorithm.

2.2 GFlowNets

Generative flow networks (Bengio et al., 2023, GFlowNets) are a family of amortized inference algorithms that treat the problem of generation with probability proportional to given rewards in an Markov decision process (MDP) as a sequential decision-making process.

Let \( G = (S, A) \) be a directed acyclic graph (DAG), where the set of vertices \( s \in S \) are referred to as states, and the set of directed edges \( (s \to s') \in A \subseteq S \times S \) represent the transitions between the states and are called actions. Given a designated initial state \( s_0 \) with no incoming edges and a set of terminal states \( s_N \) without outgoing edges, a complete trajectory can be defined as a sequence of states \( \tau = (s_0 \to s_1 \to \ldots \to s_N) \in T \). We define the forward policy \( P_F(s'|s) \) as a distribution over the children of every non-terminal state \( s \), which induces a distribution over complete trajectories, \( P_F(\tau) = \prod_{n=0}^{N-1} P_F(s_{n+1}|s_n) \). This in turn defines a marginal distribution over terminating states \( x \) in the sense that \( P_T(x) = \sum_{\tau \to x} P_F(\tau) \), where we call \( P_T(\cdot) \) the GFlowNet’s terminating distribution. The aim of GFlowNet learning is to obtain a policy such that \( P_T(\cdot) \propto R(\cdot) \), where \( R(\cdot): \mathcal{X} \to \mathbb{R}_+ \) is a given reward function or unnormalized density, where we do not know the normalizing factor \( Z = \sum_{x} R(x) \). We also use the trajectory flow function \( F(\tau) = Z P_F(\tau) \) to incorporate the effect of the partition function. Accordingly, the state flow function is the marginalization of the trajectory flow function \( F(s) = \sum_{\tau \ni s} F(\tau): \mathcal{S} \to \mathbb{R}_+ \). Most previous GFlowNet works are about composite discrete objects such as molecules; on the other hand, Lahlou et al. (2023) provide theoretical support and initial numerical demonstration for GFlowNets on continuous and hybrid (continuous and discrete) spaces. Based on this, our work aims to further tackle more complex sampling problems in continuous domains, i.e. \( S \subseteq \mathbb{R}^D \).

\(^3\)To avoid confusion, we would like to remark again that in literature the term “forward process” is usually used to describe the noising process, which is actually variance shrinking in PIS; see Equation 21.
Detailed balance (DB). DB is one of the most important GFlowNet training losses. Apart from the forward policy and the state flow function, DB also requires the practitioners to parameterize an additional backward policy \( P_B(s|s'; \theta) \), which is a distribution over the parents of any noninitial state \( s' \in S \). The DB objective is defined for a single transition \( (s \rightarrow s') \) as follows,

\[
\ell_{\text{DB}}(s, s'; \theta) = \left( \log \frac{F(s; \theta) P_F(s'|s; \theta)}{F(s'; \theta) P_B(s'|s'; \theta)} \right)^2. \tag{9}
\]

According to the theory of GFlowNets (Bengio et al., 2023), if the DB loss reaches 0 for any transition pair, then the forward policy samples correctly from the target distribution defined by \( R(\cdot) \).

3 DIFFUSION GENERATIVE FLOW SAMPLERS

3.1 AMORTIZING TARGET INFORMATION INTO INTERMEDIATE STEPS

Like many other probabilistic methods, the training objective of PIS is a KL divergence between the model and the target distribution (Equation 5). In particular, PIS’s KL is calculated at the trajectory level, which means each computation relies on a complete trajectory \( \tau = (x_0, \ldots, x_N) \) sampled from the current model’s forward process. Since the training signal \( \mu(x_N) \) is only affected by the terminal state \( x_N \), long trajectories can potentially yield less informative credit assignment for the intermediate steps, which manifests as high gradient variance and can sometimes lead to more difficult optimization. Therefore, we propose to exploit intermediate steps targets that do not necessarily rely on the complete trajectory information. What if we could know or approximate the target distribution at any step \( n \)? Note that the target process (Equation 4) can be decomposed into a product of conditional distributions

\[
P(x_{0:N}) = \pi(x_N) \prod_{n=0}^{N-1} P_B(x_n|x_{n+1}), \tag{10}
\]

where \( P_B(\cdot|\cdot) \) denotes the backward transition probability (or backward policy) derived from the joint of the target process. Then we have the expression for the marginal distribution at step \( n \):

\[
p_n(x_n) = \int \pi(x_N) \prod_{l=n}^{N-1} P_B(x_l|x_{l+1}) \, dx_{n+1:N}, \forall n \in \{0, \ldots, N\}, \tag{11}
\]

where we set \( p_N(\cdot) = \pi(\cdot) \) for consistency. If we know the form of \( p_n(\cdot) \) then we could directly learn from it with shorter trajectories and thus achieve more efficient training. Unfortunately, although we know the form of \( P_B(\cdot|\cdot) \), for general target distribution there is generally no known analytical expression for it. As a result, we propose to use a deep neural network \( F_n(\cdot;; \theta) \) with parameter \( \theta \) as a “helper” to approximate the unnormalized density of the \( n \)-th step target \( p_n(\cdot) \).

One naive way to train the \( F_n(\cdot;; \theta) \) network is to fit its value to the MC quadrature estimate of the integral in Equation 11. However, every training step would then have to include such computationally expensive quadrature calculation, making the resulting algorithm extremely inefficient. Instead, we propose turning this quadrature calculation into an optimization problem in an amortized way, and instead train \( F_n(\cdot;; \theta) \) to achieve the following constraint:

\[
F_n(x_n; \theta) \prod_{l=n}^{N-1} P_F(x_{l+1}|x_l; \theta) = \mu(x_N) \prod_{l=n}^{N-1} P_B(x_l|x_{l+1}), \tag{12}
\]

for all partial trajectories \( x_{n:N} \). Here, we would parameterize both the function \( F_n \) and the distribution \( P_F \) from Equation 2 by deep neural networks. Once Equation 12 is achieved, integrating \( x_{n+1:N} \) on both sides would lead to the fact that \( F_n(x_n; \theta) \) equals the integral in Equation 11. To put it another way, \( F_n(\cdot;; \theta) \) amortizes the integration computation into the learning of \( \theta \). Notice that we only use \( \mu(\cdot) \) in Equation 12 because our problem setting provides only \( \mu \). The unknown normalization constant is thus also absorbed into \( F_n \). We construct a training objective by simply regressing the left hand side of Equation 12 to the right hand side. Furthermore, to ensure training stability, this regression is performed in the log space. In practice, the flow function at different steps share the same set of parameters; this is achieved by introducing an additional step embedding input for the \( F(\cdot,n; \theta) \) neural network. As a result, our method learns two neural networks (the forward policy, which is the same as in PIS, and the flow function) in the meantime.
3.2 Updating parameters with incomplete trajectories

If we compare Equation 12 with $n$ and $n+1$, we obtain a formula with no dependence on $\mu$:

$$F_n(x_n; \theta)P_F(x_{n+1} | x_n; \theta) = F_{n+1}(x_{n+1}; \theta)P_B(x_n | x_{n+1}).$$

(13)

where the details are deferred to Appendix C.1.

**GFlowNet perspective.** This formulation is similar to the GFlowNet DB loss in Equation 9. Actually, it is not hard to see that the above modeling is a variant of GFlowNets. As Zhang et al. (2022a) has pointed out and as later adopted by Lahlou et al. (2023), there is a connection between diffusion modeling and GFlowNets. In the GFlowNet language, the target density $\mu(\cdot)$ is the terminal reward function, the temporal-spatial specification space $\{(n, x_n) : n \in \mathbb{N}, x_n \in \mathbb{R}^D\}$ is the GFlowNet state space $S$, the forward transition probability $P_F(\cdot | \cdot)$ of the diffusion process $Q$ is the GFlowNet forward policy, the backward transition probability $P_B(\cdot | \cdot)$ of the reference process $Q^\text{ref}$ is the GFlowNet backward policy, and the helper network $F_n(\cdot)$ is the state flow $F(s)$. Notice here that the backward policy is fixed and does not contain learnable parameters. In our setting, we use the step index $n$ to distinguish between intermediate states $\{(n, x_n) : n < N, x_n \in \mathbb{R}^D\}$ and terminating states $\{(N, x_N), x_N \in \mathbb{R}^D\}$.

What’s more, the goal of GFlowNets, which is to generate terminal states with probability proportional to the reward value, matches the goal of sampling in our setting.

It is thus worthwhile to investigate how GFlowNets could be trained in this context. Apart from the DB constraint, Madan et al. (2022) propose a more general subtrajectory balance constraint:

$$F_m(x_m; \theta) \prod_{n=m}^{n-1} P_F(x_{n+1} | x_n; \theta) = F_n(x_n; \theta) \prod_{l=m}^{n-1} P_B(x_l | x_{l+1}),$$

where $x_{m:n}$ is a subtrajectory. Notice that Equation 12 is actually a special case of this constraint if we specify $n = N$. The resulting objective for our method is

$$\ell_{\text{SubTB}}(x_{m:n}; \theta) = \left(\frac{F_m(x_m; \theta) \prod_{l=m}^{n-1} P_F(x_{l+1} | x_l; \theta)}{F_n(x_n; \theta) \prod_{l=m}^{n-1} P_B(x_l | x_{l+1})}\right)^2.$$  

(14)

A training loss based on transitions and more generally partial trajectories makes it possible to update parameters without entire trajectory specification. This can significantly reduce the variance in the optimization, as shown by Madan et al. (2022) in a similar context. To better combine the training effect from subtrajectories with different lengths, the adopted objective is

$$\mathcal{L}(\tau; \theta) = \frac{\sum_{0 \leq m < n \leq N} \lambda^{n-m} \ell_{\text{SubTB}}(x_{m:n})}{\sum_{0 \leq m < n \leq N} \lambda^{n-m}}, \tau = (x_0, \ldots, x_N),$$

(15)

where $\lambda \in \mathbb{R}^+$ is a scalar for controlling assigned weights to different partial trajectories. For the modeling of the drift of the forward stochastic process, we use $f(\cdot, n)/\sigma = NN_1(\cdot, n) + NN_2(n) \cdot \nabla \log \mu(\cdot)$, which includes two different neural networks and gradient information from unnormalized target densities in a similar form to Langevin dynamics (Zhang & Chen, 2022). We have conducted extensive ablation study about the parameterization, training objectives, exploration techniques, and other considerations, and display them in Appendix C.

**Improved credit assignment with local signals.** In both PIS and DDS algorithms, the only learning signal $\mu(x_N)$ comes at the end of complete trajectories $x_{0:N}$ when the density function is queried with $x_N$. This is also the case for general GFlowNet methods, where we set $F_N(x) \leftarrow \mu(x), \forall x \in \mathbb{R}^D$. In this way, the propagation of density information from terminal states to early states would be slow, thus affecting the training efficiency. We thus adopt the forward-looking trick proposed by Pan et al. (2023a) to incorporate intermediate learning signals for GFlowNets. The forward-looking technique essentially parameterizes the logarithm of the state flow function as $\log F_n(x_n; \theta) = \log \tilde{R}_n(x_n) + \text{NN}(x_n; \theta)$, where $\tilde{R}_n(x_n)$ could be an arbitrary “forward-looking” estimate of how much energy $x_n$ could contribute to the final reward $\mu(x_N)$. The details of the “NN” neural network are deferred to Appendix C.4. In the proposed DGFS algorithm, we take the form of the (log) partial reward to be

$$\log \tilde{R}_n(\cdot) = \left(1 - \frac{n}{N}\right) \log p_n^{\text{ref}}(\cdot) + \frac{n}{N} \log \mu(\cdot),$$

(16)

which combines information from the target density and the reference marginal density.

---

4Without loss of generality, we slightly overuse the notations and simply use $x_n$ to denote $(n, x_n)$. 

---

Published as a conference paper at ICLR 2024
We illustrate the resulting DGFS algorithm in Figure 1. Different from prior works, DGFS can update parameters with incomplete subtrajectories (colored curve segments in the figure) with the help of amortized flow functions, as well as intermediate signals at non-terminal step (grey arrows in the figure), which fundamentally improves the credit assignment process.

**Estimation of the partition function.** For an arbitrary trajectory $\tau = \{x_n\}_{n=0}^N$, one could define its (log) importance weight to be $S(\tau; \theta) = \log P(x_0, N) - \log Q(x_0; \theta)$, where $Q(x_0; \theta) = P_0(x_0) \prod_{n=0}^{N-1} P_F(x_{n+1} | x_n; \theta)$. We obtain the following log-partition function (log $Z$) estimator:

$$
\log \sum_{b=1}^{B} \exp \left( S(\tau^{(b)}; \theta) \right) - \log B \leq \log Z, \quad \tau^{(b)} \sim Q(\cdot; \theta),
$$

where $B \in \mathbb{N}_+$ is the number of particles (i.e., trajectories) used for the estimation. Similar to the importance weighted auto-encoders (Burda et al., 2016), our estimator provides a valid lower bound of the logarithm of the normalizing factor with any forward process $Q$. In this work, we use the difference between estimated log partition function and the ground truth as an evaluation metric. See the Appendix C.2 for additional analysis. We also remark that this approach is similar to the neural importance sampling in Nicoli et al. (2020); Wirnsberger et al. (2020) in the sense that we use the forward process $Q$ as the proposal distribution.

### 3.3 Discussion

**Variance of gradient updates.** Figure 2 shows that the variance of stochastic gradients with DGFS is much smaller than that with PIS; notice that the comparison is valid as we take the same neural network architecture for both algorithms. We hypothesize that DGFS benefits from the low variance behavior due to the following two reasons: (1) From the perspective of temporal difference learning (Sutton & Barto, 2018; Kearns & Singh, 2000), it has long been discussed that policy learning from shorter trajectory data would lower the variance at the cost of potentially higher bias. Before this work, it was impossible for sampling with stochastic process algorithms to update with partial trajectories, while in the proposed DGFS method, we use an additional flow network to make this feasible. In Figure 3 we also show that the flow network can be properly learned and thus does not introduce much bias, indicating that DGFS achieves a better bias-variance trade-off. (2) Probabilistic ML researchers have shown that for KL divergence based distribution learning, the gradient will not vanish even if the optimal distribution has been reached; see Roeder et al. (2017) and Section 4 of Xu et al. (2022). Later works including Vaitl et al. (2022a;b) propose a different path-gradient estimator to tackle this issue with a similar insight with Roeder et al. (2017); see the related work section. On the other hand, the DGFS objective does achieve zero gradient with the optimal solution (c.f. Appendix C.3), making it a more stable algorithm.

**Modeling considerations.** Notice that the DGFS derivation in Section 3 does not rely on the specific form of the stochastic process. Therefore, DGFS could take both the variance exploding (VE) (Zhang & Chen, 2022) or variance preserving (VP) (Vargas et al., 2023a) formulation. In our numerical study, we find the former to be more stable, and thus we default to it unless otherwise specified. We defer related ablations to the Appendix C.5.
DGFS can accurately sample from the target distribution. GFlowNets (Bengio et al., 2021; 2023). This provides assurance that a sufficiently well-trained term corresponds to a zero GFlowNet training loss under mild conditions, following theory of works are trained. Additionally, Zhang et al. (2022a) have proven that a perfectly learned score These findings are also applicable to DGFS since these proofs are independent of how the net-work can converge to the target distribution under mild assumptions if the control term is well learned.

Table 1: Experimental results on benchmarking target densities. For each algorithm, we report its absolute estimation bias for the log normalizing factor and the corresponding standard deviation. Our method achieves the best performance among the diffusion modeling-based samplers.

<table>
<thead>
<tr>
<th>Method</th>
<th>MoG</th>
<th>Funnel</th>
<th>Manywell</th>
<th>VAE</th>
<th>Cox</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMC</td>
<td>0.289±0.112</td>
<td>0.307±0.076</td>
<td>22.36±7.536</td>
<td>14.34±2.604</td>
<td>99.61±8.382</td>
</tr>
<tr>
<td>VI-NF</td>
<td>1.354±0.473</td>
<td>0.272±0.048</td>
<td>2.677±0.016</td>
<td>6.961±2.501</td>
<td>83.49±2.434</td>
</tr>
<tr>
<td>CRAFT</td>
<td>0.348±0.210</td>
<td>0.454±0.485</td>
<td>0.987±0.599</td>
<td>0.521±0.239</td>
<td>13.79±2.351</td>
</tr>
<tr>
<td>FAB w/ Buffer⁵</td>
<td>0.003±0.005</td>
<td>0.0022±0.0005</td>
<td>0.032±0.004</td>
<td>N/A</td>
<td>0.19±0.04</td>
</tr>
<tr>
<td>PIS</td>
<td>0.036±0.007</td>
<td>0.305±0.013</td>
<td>1.391±1.014</td>
<td>2.049±2.826</td>
<td>11.28±1.365</td>
</tr>
<tr>
<td>DDS</td>
<td>0.028±0.013</td>
<td>0.416±0.094</td>
<td>1.154±0.626</td>
<td>1.740±1.158</td>
<td>N/A⁶</td>
</tr>
<tr>
<td>DGFS</td>
<td>0.019±0.008</td>
<td>0.274±0.014</td>
<td>0.904±0.067</td>
<td>0.180±0.083</td>
<td>8.974±1.169</td>
</tr>
</tbody>
</table>

Convergence guarantees. Previous theoretical studies (Bortoli, 2022; Chen et al., 2022; Lee et al., 2022) have demonstrated that the distribution of terminal samples generated by a diffusion model can converge to the target distribution under mild assumptions if the control term is well learned. These findings are also applicable to DGFS since these proofs are independent of how the networks are trained. Additionally, Zhang et al. (2022a) have proven that a perfectly learned score term corresponds to a zero GFlowNet training loss under mild conditions, following theory of GFlowNets (Bengio et al., 2021; 2023). This provides assurance that a sufficiently well-trained DGFS can accurately sample from the target distribution.

4 RELATED WORKS

Sampling methods. MCMC methods (Brooks et al., 2011) are the most classical sampling algorithms, including Langevin dynamics (Welling & Teh, 2011), Hamiltonian Monte Carlo (Duane et al., 1987; Hoffman & Gelman, 2011; Chen et al., 2014), and others (Huang et al., 2023). With MCMC operators as building blocks, sequential MC methods (Liu, 2001; Doucet et al., 2001) generate samples ancestrally through a series of annealed targets. On the other hand, people turn to parametric models such as deep networks to learn samplers via variational inference (Rezende & Mohamed, 2015). One the most commonly adopted models is the normalizing flow (Dinh et al., 2014) due to its tractability and expressiveness. Various work have been conducted to improve normalizing flow based sampling (Müller et al., 2018; Gao et al., 2020; Wu et al., 2020; Chen et al., 2020; Gabri’e et al., 2021; Arbel et al., 2021; Midgley et al., 2022; de G. Matthews et al., 2022; Caselle et al., 2022; Gao et al., 2023). These methods support asymptotically unbiased estimation as shown in Nicoli et al. (2020). What’s more, Vaitl et al. (2022a) propose to improve the gradient estimation of normalizing flow parameters with the path-gradient technique; Vaitl et al. (2022b) improve continuous normalizing flows based methods (Chen et al., 2018) with a similar technique. These methods have wide applications in natural sciences, for example in lattice field theories (Nicoli et al., 2021; 2023) and targeted free energy estimation (Wirnsberger et al., 2020).

Recently, many researchers have developed powerful diffusion process to model distributions in the generative modeling community (Vincent, 2011; Sohl-Dickstein et al., 2015; Ho et al., 2020a; Song et al., 2020; Du et al., 2022). This inspires people to study how to utilize diffusion process as samplers to learn distribution from given unnormalized target densities (Tzen & Raginsky, 2019; Zhang & Chen, 2022; Vargas et al., 2021; Geffner & Domke, 2022; Doucet et al., 2022; Vargas et al., 2023a; Richter et al., 2023; Vargas & Nusken, 2023; Vargas et al., 2023c;b). Our work also falls into this category. Specifically, Vargas et al. (2023a) can be seen as a special case of the method proposed in Berner et al. (2022).

GFlowNets. GFlowNet (Bengio et al., 2023) is a family of generalized variational inference algorithms with a focus on treating the sampling process as a sequential decision-making process.

⁵We used the fab-jax results. The FAB method utilizes the annealed importance sampling computation during the training period.

⁶Due to time constraints, we are unable to reproduce DDS performance using PyTorch for this task. Instead, we encountered “not a number”; probably due to a mismatch in implementation details or other unspecified difference between PyTorch and Jax. The DDS results reported in its original paper can be found here.
It is originally proposed to sample diversity-seeking objects in structured scientific domains such as biological sequence and molecule design (Bengio et al., 2021; Jain et al., 2022; 2023b;a; Shen et al., 2023; Zhang et al., 2023c; Hernández-García et al., 2023; Kim et al., 2023). However, people have developed GFlowNet theories and methodologies to be suitable for generating general objects with composite structures. GFlowNet has rich connection with existing probabilistic ML methods (Zhang et al., 2022b; Zimmermann et al., 2022; Malkin et al., 2023; Zhang et al., 2022a; Hu et al., 2023; Ma et al.). Malkin et al. (2023) also show that GFlowNet has the capability of training with off-policy data compared to previous amortized inference methods. As a policy learning framework under Markov decision processes (MDPs), researchers have also drawn fruitful connections between GFlowNet and reinforcement learning (Pan et al., 2023b,c,a). GFlowNet has also been demonstrated to have wide application in causal discovery (Deleu et al., 2022; 2023; Atanackovic et al., 2023), feature attribution (Li et al., 2023a), network structure learning (Liu et al., 2022), job scheduling (Zhang et al., 2023a), and graph combinatorial optimization (Zhang et al., 2023b). Most prior works focus on structured discrete data space. Lah lou et al. (2023) first provides a sound theoretical framework and initial numerical demonstration for using GFlowNet on continuous space. Our work follows this framework and scales up the performance. A prior study (Li et al., 2023b) initially explored GFlowNets in a continuous space, whereas their theory is demonstrated to have non-trivial flaws when considering integration under continuous measure as indicated by Lah lou et al. (2023).

5 Experiments

5.1 Benchmarking target distributions

In this section, we conduct extensive experiments on a diverse set of challenging continuous sampling benchmarks to corroborate the effectiveness of DGFS against previous methods.

**Mixture of Gaussians (MoG)** is a 2-dimensional Gaussian mixture where there are 9 modes designed to be well-separated from each other. The modes share the same variance of 0.3 and the means are located in the grid of \{-5, 0, 5\} × \{-5, 0, 5\}.

**Funnel** is a classical sampling benchmark problem from Neal (2003); Hoffman & Gelman (2011). This 10-dimensional density is defined by \( \mu(x) = \mathcal{N}(x^{(0)}; 0, 9)\mathcal{N}(x^{(1:9)}; 0, \exp(x^{(0)}))\).\footnote{Previous works (Zhang & Chen, 2022; Lah lou et al., 2023) unintentionally take the variance of \( x^{(0)} \) to be 1 for their methods (hence much less difficulty) but use 9 for other methods.}

**Manywell** (Midgley et al., 2022; Noé et al., 2019; Wu et al., 2020) is a 32-dimensional distribution defined by the product of 16 same 2-dimensional “double well” distributions, which share the unnormalized density of \( \mu(x, y) = \exp(-x^4 + 6x^2 + 0.5x - 0.5y^2) \).

**VAE** is a 30-dimensional sampling task where we use the latent posterior of a pretrained variational autoencoder (Kingma & Welling, 2014; Rezende et al., 2014, VAE) to be the target distribution.

**Cox** denotes the 1600-dimensional target distribution of the log Gaussian cox process introduced in Møller et al. (1998) which models the positions of pine saplings. Its unnormalized density is \( \mu(x) = \mathcal{N}(x; \mu, K^{-1}) \prod_{d=1}^{D} \exp(x^{(d)}y^{(d)} - \alpha x^{(d)}) \), where \( \mu, K, y, \alpha \) are constants.

**Baselines.** We benchmark DGFS performance against a wide range of strong baseline methods. As an MCMC methods we include the sequential Monte Carlo sampler (Moral & Doucet, 2002, SMC), which is often considered the state-of-the-art sampling method. As for normalizing flow methods we include the variational inference with normalizing flows (Rezende & Mohamed, 2015, VI-NF) approach as well as more recent de G. Matthews et al. (2022, CRAFT) which is combined with MCMC. We also provide results of Midgley et al. (2022, FAB) with a replay buffer that utilizes the annealed importance sampling technique to augment the training. Last but not least, we compare with PIS (Zhang & Chen, 2022) and DDS (Vargas et al., 2023a) algorithms which are closest to our framework but trained with the KL divergence formulation (Equation 5) without the improved training techniques described in Section 3.

**Evaluation protocol.** We measure the performance of different algorithms by their estimation bias of the log normalizing factor of unnormalized target distributions. Contrary to Zhang & Chen (2022)}
which take the best checkpoint across the run, in this work we compute the average of the last ten checkpoints for a more robust comparison. All results are conducted with 5 random seeds.

We display the benchmarking results in Table 1. We fail to achieve reasonable performance with DDS on Cox target where we mark it as “N/A”. The advantage of the proposed DGFS method can be observed across different tasks, indicating it could indeed benefit from its design providing more informative training signals to achieve better convergence and capturing diverse modes even in high dimensional spaces. FAB with buffer produces highly stable results, which can be attributed to both annealed importance sampling, the use of buffers, larger networks, and other tricks. It is also remarkable that DGFS obtains very stable results compared to other baselines with diffusion modeling (PIS, DDS). We defer related experimental details and extra experiments to Appendix C.

5.2 Analysis

First, we investigate the learning of the flow function introduced in Equation 12. In the first row of Figure 3, we visualize the learned flow function $F_n(x_n)$ with $n = 20, 40, \ldots, 100$ in the 2D MoG task. In the second row, we simulate samples from $p_n(x_n)$ by sampling according to Equation 11, i.e., moving ground truth $\pi(\cdot)$ samples in the backward direction with policy $P_B$. The consistency between these two rows indicate that DGFS flow function can successfully approximate the intermediate marginal $p_n(\cdot)$, which makes it possible to provide an effective training signal for learning from partial trajectory pieces.

For the MoG task, we demonstrate the samples together with ground truth contours in Figure 5. This visualization result matches the log partition function estimation bias in Table 1, where DGFS generates the nine modes more uniformly than other baselines. This corroborates the advantage of DGFS to capture all different modes in a complex landscape. We then show visualization results on the Manywell task in Figure 4. The Manywell distribution is 32-dimensional, so we display the joint of the first and third dimension. We plot the samples generated from the true distribution and different algorithms, together with the true density contours. This figure implies that PIS can model half of the modes in these two dimensions well, but miss the other two modes, whereas both DGFS and DDS could cover all four modes. Further, DGFS achieves slightly better mode capturing than DDS in the sense that its modes are more separate and thus closer to the true distribution, while for DDS there are considerable samples appear between the upper right mode and the bottom right mode, which is not ideal. We defer other visualizations to the Appendix C.5.

To look into why DGFS outperforms PIS with similar architecture, we plot the learned drift network magnitude in Figure 6 in the Appendix. The result shows that the solution that DGFS achieved has more homogeneous magnitude outputs across different diffusion steps. This may explain one reason...
why DGFS is easier to train than
PIS, since the policy probabili-
ties at different steps \( \{P_t(\cdot|\mathbf{x}_n)\}_n \) are actually parameterized by a single neural network with tem-
poral embedding input. This makes the learning task more stationary with DGFS. We also conduct
in-depth ablation studies to evaluate the design choices of DGFS in Appendix C.5 and C.7.

6 Conclusion

We propose the Diffusion Generative Flow Sampler (DGFS), a novel algorithm that trains diffusion
models to sample from given unnormalized target densities. Different from prior works that could
only learn from complete diffusion chains, DGFS could update its parameter with only partial spec-
ification of the stochastic process trajectory; what’s more, DGFS can receive intermediate signals
before completing the entire path. These features help DGFS benefit from efficient credit assignment
and thus achieve better partition function estimation bias in the sampling benchmarks.

The DGFS framework presents various opportunities for further research, fueled by its inherent lim-
itations. To name a few, how can we better design the intermediate local signals in ways more so-
plicated than the current straightforward approach? Additionally, can we harness DGFS’s explo-
roration capabilities to succeed in high dimensional tasks, pushing the boundaries of its performance?
Can we combine DGFS with a prioritized replay buffer? This would largely reduce the number of querying \( \mu(\cdot) \). From the perspective of application, there are also many interesting research ques-
tions. Can we deploy DGFS on scientific biological or chemical tasks such as protein conformation
modeling, with equivariant modeling? We expect that exploring these future directions will lead to
valuable insights and bring fruit follow-ups of DGFS.

Acknowledgement

The authors are grateful to the discussions with Francisco Vargas, Julius Berner, Lorenz Richter,
Laurence Midgley, Minsu Kim, Yihang Chen, and Qinsheng Zhang.

Ethics Statement

We commit to the ICLR Code of Ethics and affirm that our work only utilizes public datasets for
experimentation. While our empirical results are largely based on synthetic data, we acknowledge
the potential for misuse and urge the responsible application of the proposed methods with real-
world data. We welcome any related discussions and feedback.

Reproducibility Statement

We provide detailed algorithmic and experimental description in Section 3 and Appendix C, and we
have open sourced the code accompanying this research in this GitHub link.

References

Michael S Albergo, Gurtej Kanwar, and Phiala E. Shanahan. Flow-based generative models for

Christophe Andrieu, Nando de Freitas, A. Doucet, and Michael I. Jordan. An introduction to

Michal Arbel, Alexander G. de G. Matthews, and A. Doucet. Annealed flow transport monte


Francisco Vargas, Shreyas Padhy, Denis Blessing, and Nikolas Nusken. Transport meets variational inference: Controlled monte carlo diffusions. 2023b.


A NOTATIONS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{R}^D$</td>
<td>$D$-dimensional real space</td>
</tr>
<tr>
<td>$N \in \mathbb{N}_+$</td>
<td>number of steps in the discrete-time stochastic process</td>
</tr>
<tr>
<td>$S$</td>
<td>GFlowNet state space, equals ${(n, x_n) : n \in {0, 1, \ldots, N}, x_n \in \mathbb{R}^D}$; $(n, x_n)$ is often written as $x_n$ for simplicity</td>
</tr>
<tr>
<td>$A$</td>
<td>action / transition space (edges $s \rightarrow s'$)</td>
</tr>
<tr>
<td>$s$</td>
<td>state in $S$</td>
</tr>
<tr>
<td>$s_0$</td>
<td>initial state, equals $(0, 0)$ in VE modeling</td>
</tr>
<tr>
<td>$\tau$</td>
<td>complete trajectory $(x_0 \rightarrow \ldots \rightarrow x_N)$, or $x_{0:N}$</td>
</tr>
<tr>
<td>$\theta$</td>
<td>learnable parameter of DGFS</td>
</tr>
<tr>
<td>$F_n : \mathbb{R}^D \rightarrow \mathbb{R}_+$</td>
<td>state flow function at step $n$</td>
</tr>
<tr>
<td>$P_F$</td>
<td>forward policy (distribution over children / next step)</td>
</tr>
<tr>
<td>$P_B$</td>
<td>backward policy (distribution over parents / previous step)</td>
</tr>
<tr>
<td>$Q$</td>
<td>forward process defined in Equation 1</td>
</tr>
<tr>
<td>$Q_{\text{ref}}$</td>
<td>reference process defined in Equation 3</td>
</tr>
<tr>
<td>$P$</td>
<td>target process defined in Equation 4</td>
</tr>
<tr>
<td>$D^D_0 : \mathbb{R}^D \rightarrow \mathbb{R}_+$</td>
<td>marginal distribution of reference process at step $n$</td>
</tr>
<tr>
<td>$h \in \mathbb{R}_+$</td>
<td>discretized step size for the stochastic process</td>
</tr>
<tr>
<td>$f$</td>
<td>drift term in the VE stochastic process (Equation 5)</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>variance coefficient in the stochastic process</td>
</tr>
<tr>
<td>$dW_t$</td>
<td>standard Wiener process in continuous-time stochastic process</td>
</tr>
<tr>
<td>$R : \mathbb{R}^D \rightarrow \mathbb{R}_+$</td>
<td>reward function (unnormalized target density)</td>
</tr>
<tr>
<td>$\tilde{R}<em>n : \mathbb{R}^D \rightarrow \mathbb{R}</em>+$</td>
<td>intermediate forward-looking signal defined in Equation 16</td>
</tr>
<tr>
<td>$Z \in \mathbb{R}_+$</td>
<td>scalar, equal to $\sum_{x \in \mathbb{R}^D} R(x)$</td>
</tr>
<tr>
<td>$\mu : \mathbb{R}^D \rightarrow \mathbb{R}_+$</td>
<td>same thing as $R(\cdot)$</td>
</tr>
<tr>
<td>$\pi : \mathbb{R}^D \rightarrow \mathbb{R}_+$</td>
<td>normalized density, equal to $\mu(\cdot)/Z$</td>
</tr>
</tbody>
</table>

B DETAILS ABOUT STOCHASTIC PROCESSES

B.1 VARIANCE EXPLODING (VE) PROCESS

Song et al. (2020) and Zhang & Chen (2022) take a variance exploding stochastic process modeling. In Zhang & Chen (2022), the authors use Gisarnov theorem to derive an optimal control objective for KL divergence minimization. In this work, we present a discrete-time derivation for completeness.

The reference process starts at $p_0^{\text{ref}}(\cdot)$, which is a Dirac distribution at 0, and writes

$$Q^{\text{ref}}(x_{0:N}) : x_{n+1} = x_n + \sqrt{h} \sigma \epsilon_n, \epsilon_n \sim \mathcal{N}(0, 1), x_0 = 0, \quad (18)$$

$$\Rightarrow Q^{\text{ref}}(x_{0:N}) = p_0^{\text{ref}}(x_0) \prod_{n=0}^{N-1} p_F^{\text{ref}}(x_{n+1}|x_n) = p_N^{\text{ref}}(x_N) \prod_{n=0}^{N-1} P_B(x_n|x_{n+1}). \quad (19)$$

Therefore, $p_n^{\text{ref}}(\cdot) = \mathcal{N}(\cdot; 0, h n \sigma^2 \mathbf{1})$. According to Bayesian formula,

$$P_B(x_n|x_{n+1}) = \frac{p^{\text{ref}}(x_n)p^{\text{ref}}(x_{n+1}|x_n)}{p^{\text{ref}}(x_{n+1})} = \frac{\mathcal{N}(x_n; 0, h n \sigma^2 \mathbf{1})\mathcal{N}(x_{n+1}; x_n, h \sigma^2 \mathbf{1})}{\mathcal{N}(x_{n+1}; 0, h(n+1) \sigma^2 \mathbf{1})} \quad (20)$$

$$= \mathcal{N}(x_n; \frac{n}{n+1} x_{n+1}, \frac{n}{n+1} h \sigma^2 \mathbf{1}). \quad (21)$$

The target process is then

$$\mathcal{P}(x_0, \ldots, x_N) = Q^{\text{ref}}(x_0, \ldots, x_N) \frac{\pi(x_N)}{p_N^{\text{ref}}(x_N)} = \pi(x_N) \prod_{n=0}^{N-1} P_B(x_n|x_{n+1}). \quad (22)$$
The trajectory level log probability difference between model and target is
\[
\log \frac{Q}{P} = \log \frac{Q}{Q_{ref}} + \log \frac{Q_{ref}}{P}.
\] (23)

It is obvious that the second term is \( \frac{Q_{ref}}{P} = \frac{p_{ref}(x_n)}{\pi(x_n)} \), thus we only need to look at the first term

\[
\log \frac{Q}{Q_{ref}} = \sum_{n=0}^{N-1} \log \frac{P_F(x_{n+1}|x_n; \theta)}{F_{ref}(x_{n+1}|x_n)} = \sum_{n=0}^{N-1} \log \frac{\mathcal{N}(::x_n + h(f(x_n, n), h\sigma^2 I))}{\mathcal{N}(::x_n, h\sigma^2 I)}
\] (24)

\[
= \sum_{n=0}^{N-1} \left( \frac{h}{2\sigma^2} \|f(x_n, n; \theta)\|^2 + \frac{\sqrt{h}}{\sigma^2} f(x_n, n; \theta)^T \epsilon_n \right).
\] (25)

Notice that under the expectation of \( Q \),
\[
\mathbb{E}_Q [f(x_n, n; \theta)^T \epsilon_n] = f(x_n, n; \theta)^T \mathbb{E}_Q [\epsilon_n] = 0.
\] (26)

Accordingly,
\[
D_{KL}(Q||P) = \mathbb{E}_Q \left[ \log \frac{Q}{P} \right] = \mathbb{E}_Q \left[ \sum_{n=0}^{N-1} \frac{h}{2\sigma^2} \|f(x_n, n; \theta)\|^2 + \log \frac{p_{ref}(x_n)}{\pi(x_n)} \right],
\] (27)

which leads us to Equation 6.

B.2 VARIANCE PRESERVING (VP) PROCESS

Ho et al. (2020b) and Vargas et al. (2023a) take this VP formulation of diffusion process. The derivation is similar to the last section, just with a different variance schedule.

We first define the backward policy
\[
P_B(x_n|x_{n+1}) = \mathcal{N}(x_n; \sqrt{1 - \beta_n}x_{n+1}; \beta_n\sigma^2 I).
\] (28)

Then The target and reference process are
\[
P(x_{0:N}) = \pi(x_n) \prod_{n=0}^{N-1} P_B(x_n|x_{n+1}),
\] (29)

\[
Q_{ref}(x_{0:N}) = p_{ref}^n(x_N) \prod_{n=0}^{N-1} P_B(x_n|x_{n+1}),
\] (30)

where \( p_{ref}^n(\cdot) = \mathcal{N}(::0, \sigma^2 I) \). This guarantees that \( p_{ref}^n(\cdot) = \mathcal{N}(::0, \sigma^2 I), \forall n \).

From using Bayesian formula on \( Q_{ref} \),
\[
P_{ref}^n(x_{n+1}|x_n) = \frac{p_{ref}^n(x_{n+1})P_B(x_n|x_{n+1})}{p_{ref}^n(x_n)} = \mathcal{N}(x_{n+1}; \sqrt{1 - \beta_n}x_n, \beta_n\sigma^2 I).
\] (31)

Inspired by this, let us define the (learnable) forward process to be
\[
P_F(x_{n+1}|x_n; \theta) = \mathcal{N}(x_{n+1}; \sqrt{1 - \beta_n}x_n + f(x_n, n; \theta), \beta_n\sigma^2 I),
\] (32)

\[
Q(x_{0:N}) = p_{ref}^0(x_0) \prod_{n=0}^{N-1} P_F(x_{n+1}|x_n; \theta).
\] (33)

Similar to the derivation in VE,
\[
\mathbb{E}_Q \left[ \log \frac{Q}{Q_{ref}} \right] = \mathbb{E}_Q \left[ \sum_{n=0}^{N-1} \log \frac{P_F(x_{n+1}|x_n; \theta)}{P_{ref}^n(x_{n+1}|x_n)} \right] = \mathbb{E}_Q \left[ \sum_{n=0}^{N-1} \frac{1}{2\beta_n\sigma^2} \|f(x_n, n; \theta)\|^2 \right].
\] (34)

Therefore,
\[
D_{KL}(Q||P) = \mathbb{E}_Q \left[ \log \frac{Q}{P} \right] = \mathbb{E}_Q \left[ \sum_{n=0}^{N-1} \frac{1}{2\beta_n\sigma^2} \|f(x_n, n; \theta)\|^2 + \log \frac{p_{ref}^n(x_N)}{\pi(x_N)} \right],
\] (35)

which is the learning objective of DDS algorithm.
C ALGORITHMIC DETAILS

C.1 RE-DERIVING DETAILED BALANCE CONSTRAINT

Writing down Equation 12 with $n$ and $n + 1$, we have

$$
F_n(x_n; \theta) \prod_{l=n}^{N-1} P_F(x_{l+1} | x_l; \theta) = \mu(x_N) \prod_{l=n}^{N-1} P_B(x_l | x_{l+1}),
$$

(36)

$$
F_{n+1}(x_{n+1}; \theta) \prod_{l=n+1}^{N-1} P_F(x_{l+1} | x_l; \theta) = \mu(x_N) \prod_{l=n+1}^{N-1} P_B(x_l | x_{l+1}),
$$

(37)

$$
\Rightarrow \frac{F_n(x_n; \theta) P_F(x_{n+1} | x_n; \theta)}{F_{n+1}(x_{n+1}; \theta)} = P_B(x_n | x_{n+1})
$$

(38)

where the last equation is obtained by comparing the first two equations on both sides, which resonates the GFlowNet DB criterion.

C.2 $\log Z$ ESTIMATOR ANALYSIS

The bound in Equation 17 could be derived in the following way:

$$
\log Z = \log \int \mu(x_N) \, dx_N = \log \int \mathcal{P}(x_{0:N}) \, dx_{0:N} = \log \mathbb{E}_Q \left[ \frac{\mathcal{P}(x_{0:N})}{Q(x_{0:N})} \right]
$$

(39)

$$
= \log \mathbb{E}_Q \left[ \frac{1}{B} \sum_{b=1}^{B} \frac{\mathcal{P}(x;b)}{Q(x;b)} \right] \leq \mathbb{E}_Q \left[ \log \frac{1}{B} \sum_{b=1}^{B} \frac{\mathcal{P}(x;b)}{Q(x;b)} \right]
$$

(40)

$$
\approx \log \frac{1}{B} \sum_{b=1}^{B} \exp \left( S(\tau^{(b)}) \right), \quad \tau^{(b)} \sim Q.
$$

(41)

Notice that this bound is valid without any assumption on $Q$. According to analysis in Burda et al. (2016), this lower bound estimator is monotonically increase as $B$ goes larger; what’s more, it is a tight bound which means that it would converge to $\log Z$ when $B \to \infty$. We remark that this is actually also the estimator used in Zhang & Chen (2022).

C.3 GRADIENT VARIANCE ANALYSIS

In Figure 2, we conduct experiment on Funnel and report the gradient variance of DGFS and PIS. The variance of DDS is in similar scale to PIS thus we ignore it. According to the analysis in Section 3.3, there are two reasons of why DGFS is a more robust algorithm: (1) DGFS utilize short trajectory which is beneficial for efficient credit assignment (2) DGFS does not take the KL divergence minimizing formulation which has non-zero gradient variance. To disentangle these two factors, we run a variant of DGFS which only utilize full trajectories to update the parameters in Figure 7. This variant is shown to have gradient variance higher than the original DGFS but still much lower than PIS. This confirms that both factors are valid reasons for why DGFS has a lower gradient variance.

For general probabilistic methods with KL between model $q_\theta(x)$ and target distribution $p(x)$,

$$
\nabla_\theta D_{KL} (q_\theta(x) || p(x)) = \mathbb{E}_q \left[ \nabla_\theta \log q_\theta(x) \log \frac{q_\theta(x)}{p(x)} \right] + \mathbb{E}_q \left[ \nabla_\theta \log q_\theta(x) \right].
$$

(42)

When the model reaches its optimal solution, i.e., $q_\theta = p$, the first term is zero as it contains $\log \frac{q_\theta(x)}{p(x)} = 0$. However, the second term $\nabla_\theta \log q_\theta(x)$ is only zero under expectation of $q_\theta$. That is to say, there will be a stochastic non-zero gradient term with KL formulation. This observation is the core motivation of Roeder et al. (2017).
Algorithm 1 DGFS training algorithm

Require: DGFS model with parameters $\theta$, reward function $\mu(\cdot)$, variance coefficient $\hat{\sigma}$.
Repeat
    Sample $\tau$ with control $f(\cdot; \tau; \theta)$ and $\hat{\sigma}$;
    $\Delta \theta \leftarrow \nabla_{\theta} L(\tau; \theta)$ (as per Equation 15);
    Update $\theta$ with Adam optimizer;
until some convergence condition

C.4 DGFS DETAILS

Here we describe details about DGFS algorithm. As shown in Algorithm 1, DGFS first interact with the environment to collect training data. Next, it updates its parameters $\theta$ with methods described in Section 3. Notice that the training of DGFS is very general and does not have to use on-policy trajectories. Therefore, we could use a different variance coefficient $\hat{\sigma}$ to rollout. This could potentially bring better exploration ability and thus better mode capturing performance. In this work, we assume DGFS does on-policy rollout unless otherwise specified.

We use PyTorch to implement DGFS algorithm. For PIS implementation, we take its original PyTorch code base. For DDS, we build our own implementation based on PIS code base. For other methods, we take the DeepMind jax code base.

DGFS contains two sets of deep network parameters: one is the same as the neural network in PIS / DDS, the other is the flow function to amortize the intermediate marginal densities. The former one is for the control network, and express it in the way of $f(\cdot, n)/\sigma = \text{NN}1(\cdot, n) + \text{NN}2(n) \cdot \nabla \log \mu(\cdot)$. These use Fourier features to embed the step index input. We also ablate diffusion-based sampling methods without gradient information in network parametrization in C.6. The second one is a scalar output network $\text{NN}(\cdot, n)$ with similar structure. Same to PIS, we use two 64-dimensional hidden layers after the embedding layer. We set $\lambda$ to be 2. For all experiments, we train with a batch size of 256 and Adam optimizer. We have not tuned too hard on the learning rate, but simply use $1 \times 10^{-4}$ and $1 \times 10^{-3}$ for the policy network (i.e., drift network) and the flow function network.

The training keeps for 5000 optimization steps although almost all experiments converge in the first 1000 steps. Similar to PIS, we set the number of diffusion steps $N$ to 100 for all experiments. We set the diffusion step size $h$ to 0.05 for the MoG and Cox task and 0.01 for other tasks for all diffusion based methods. We set $\sigma$ to 1. Zhang & Chen (2022) set evaluation particle number $B = 6000$ for the funnel task and $B = 2000$ for other tasks for unknown reasons. In this work, we set the number of evaluation particles $B$ to be 2000 for all tasks for consistency.

C.5 MORE EXPERIMENTS
We conduct ablation studies on a series of design choices of DGFS. In Figure 10, we compare the performance of DGFS and a few its variants on the VAE task. In the VAE task, the unnormalized target density \( \log p(\cdot) \) is defined by the log latent prior \( \log p(\cdot) \) plus the log decoder likelihood \( \log p(x_{\text{obs}}|\cdot) \), which equals the latent posterior \( \log p(\cdot|x_{\text{obs}}) \) plus a constant value. The VAE is trained on the MNIST dataset, and \( x_{\text{obs}} \) is a fixed image data. We first remove the usage of the intermediate signals (i.e., the parameterization of flow function in Equation 16), which leads to slightly worse performance. We then try to train DGFS with VP modeling in Appendix B.2. For this VP modeling, we also take the form of Equation 16 for getting intermediate signals. This gives a bias of 0.86. Notice that this is still better than both PIS and DDS, demonstrating the validity of the proposed DGFS training method. We also ablate the weighting coefficient \( \lambda \) and number of diffusion steps \( N \). We can see their changes lead to small variations but the overall convergence remains good.

Regarding the running time of the proposed algorithm, DGFS has the same inference procedure as PIS, and thus having the same inference speed as PIS. The training overhead of DGFS is slightly (20\%) higher than PIS due to the introduction of the additional flow function network. In the Funnel task, the training time of DGFS and PIS for one batch are 0.54s and 0.65s respectively.

We then show some missing results from the main text. We put the visualization of the joint of the second and third dimension in Manywell task in Figure 8. We can see that PIS misses one important mode and DDS could not handle the mode separation well enough.

We tried to use a MNIST-pretrained normalizing flow as the target density. However, we find a well-trained normalizing flow almost always assign \(-\infty\) to the log likelihood for out-of-distribution samples, which makes it impossible for any sampler to success. This actually makes sense as the normalizing flow is only trained on a finite size image dataset and never gets out-of-domain inputs.

### C.6 Remove Gradient Information from Parametrization

In the network design, PIS explicitly use the score function—gradient of the log unnormalized density function—as one term in the drift network parametrization. Our work and DDS follow its modeling. In this section, we show performance where the score function are removed from the architecture of all three diffusion-base samplers in Table 2. We attach “-NN” postfix to their names to distinguish between their original algorithms. Notice PIS-NN and DDS-NN still require the knowledge of the score function implicitly based on their KL formulation. On the other hand, DGFS-NN
exploration variance coefficient

We conduct an initial experiment to investigate this possibility in DGFS. We test a strategy that linearly anneals the \( \sigma \) in the forward process during the rollout stage. In theory, this could help DGFS to better capture diversity in complex multimodal target distributions.

is a real black-box method, i.e., only requires zeroth order information of the unnormalized density function. Despite using less information, DGFS-NN is still very competitive in most of the tasks.

C.7 Off-Policy Exploration Ability of DGFS

The usage of the KL divergence in previous works restricts the training to be on-policy, as the training trajectory has to come from the model itself. Using off-policy samples is possible only with importance sampling, which typically results in high variance. This limits the exploration ability of these methods. On the other hand, it is first shown in Malkin et al. (2023) that GFlowNets can potentially benefit from the use of off-policy exploration. To be more specific, as has been shown by a similar analysis in prior works Malkin et al. (2023); Lahlou et al. (2023), the objectives in Equation 9 or Equation 14 do not require the training samples to follow any particular distribution (only to have full support), which means our method supports off-policy training without importance sampling. Therefore, it is possible to employ off-policy exploration by using a larger variance coefficient in the forward process during the rollout stage. In theory, this could help DGFS to better capture diversity in complex multimodal target distributions.

In an algorithmic perspective, we could use a slightly larger variance coefficient \( \tilde{\sigma} > \sigma \) in the rollout stage of Algorithm 1 to ensure that information of more modes fall into the training data, which would drive the training of DGFS to put effort on these distant modes. To demonstrate this property, we carefully create a more difficult Mixture of Gaussian task and name it as MoG+. We intentionally put a few very distant modes in this MoG+ task, as shown by the contours in Figure 9. We set the exploration variance coefficient \( \tilde{\sigma} = 2 \) contrary to \( \sigma = 1 \), which achieves better mode coverage and lower normalizing factor estimation bias with the same neural network capacity.

We conduct an initial experiment to investigate this possibility in DGFS. We test a strategy that linearly anneals the \( \tilde{\sigma} \) from 1.1 to 1 during the first 1000 steps of training, and denote it as DGFS+ in Table 3. Our experiment results show that this simple strategy can improve the performance under some scenarios, and can be treated as a potential hyperparameter to extend the application scope of DGFS.

C.8 Other Ablation Study

We run the method in Lahlou et al. (2023) in this section. As shown in Table 4, this previous method and its off-policy variant are far from comparable to the baselines in Table 1. We also run the trajectory balance method with gradient information (denoted as TB) and its variant with off-policy exploration (denoted as TB+) and present in the same table. These two methods achieve relatively better performance, but still worse than that of DGFS.