Diffusion Generative Flow Samplers: Improving Learning Signals Through Partial Trajectory Optimization

Dinghuai Zhang*, Ricky T. Q. Chen, Cheng-Hao Liu, Aaron Courville & Yoshua Bengio

Thomas Mousseau

October 22, 2025

Overview

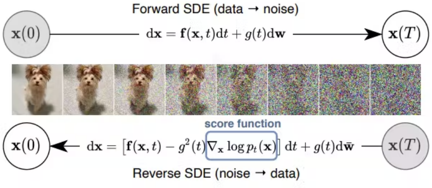
Generative Modeling

Task

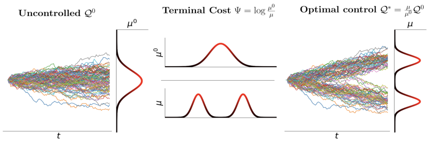
Sample from a complex (high-dimensional and multimodal) distribution D.

D can be given under the form of:

- A dataset of samples $\{x_i\}_{i=1}^N \sim D$ (e.g., images, text, audio)



- An unnormalized density $\mu(x)$ where D has density $\pi(x) \propto \mu(x)$ (e.g., energy-based models, physics/chemistry)



Sampling from Unnormalized Densities

Context. Sample from a D-dimensional target with unnormalized density $\mu(x)$ where $\mathbb{R}^D \to \mathbb{R}^+$.

$$\pi(x) = \frac{\mu(x)}{Z}, \qquad Z = \int_{\mathbb{R}^D} \mu(x)\, dx \text{ (unknown)}.$$

We assume we can evaluate $\mu(x)$, but we have no samples from π and do not know Z.

Goal. We seek a *sampler* (similar to MCMC/VI) that produces calibrated samples and, ideally, estimates Z *without* any dataset from π.

Chemistry (molecule conformers).

Different 3D conformations have a formation energy from force-field terms (bonds, angles, dihedrals, nonbonded). A low energy means a higher probability of being sampled. A well-calibrated sampler is needed to draw conformers in proportion to these probabilities, which is important in binding-pose ranking, free-energy estimation and generating diverse realistic 3D conformers for screening.

Diffusion Generative Flow Samplers

Idea. We will *reframe sampling* from an unnormalized target $\pi(x) \propto \mu(x)$ as a *stochastic optimal control (SOC)* problem. This means learning a control function that steers a diffusion process so its *terminal marginal* matches π.

Why this helps.

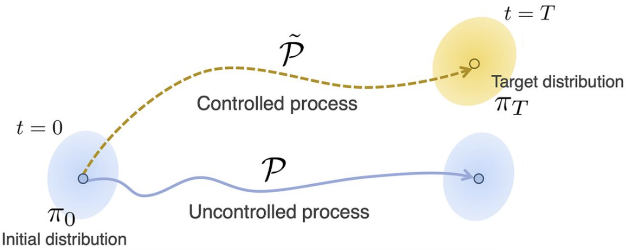
- Gives a *path-space* training objective/metric: a KL on trajectories $\mathrm{KL}(Q \,\|\, P)$ where P is the reference paths reweighted by $\mu(x_T)$.
- The *partition function Z cancels* inside this objective, so we can train using only μ (and optionally $\nabla \log \mu$).
- Lets us optimize *without samples from π* and still measure closeness to the true normalized endpoint.

Caveat (sets up DGFS). This path-KL places supervision *only at the terminal time* $\Rightarrow$ poor *credit assignment* and high-variance gradients.

DGFS fix (preview). Inject *intermediate* learning signals via a GFlowNet-inspired *learned flow* and *subtrajectory balance*, enabling partial-trajectory training and more stable learning.

Diffusion Generative Flow Samplers v2

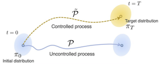
Minimizing the running and terminal costs between the optimal controlled process and the uncontrolled reference process.



$$
\begin{aligned}
\underset{\tilde{\mathcal{P}}}{\text{minimize}} \quad & D_{\mathrm{KL}}[\tilde{\mathcal{P}} \| \mathcal{P}] = \int_{\Omega} \log \frac{d\tilde{\mathcal{P}}}{d\mathcal{P}} d\tilde{\mathcal{P}} \\
\text{s.t.} \quad & x(0) \sim \pi_0, \quad x(T) \sim \pi_T
\end{aligned}
$$

Diffusion Generative Flow Samplers v2

Minimizing the running and terminal costs between the optimal controlled process and the uncontrolled reference process.



$$\boxed{\begin{aligned} \underset{\tilde{\mathcal{P}}}{\text{minimize}} \quad & D_{\mathrm{KL}}[\tilde{\mathcal{P}} \| \mathcal{P}] = \int_\Omega \log \frac{d\tilde{\mathcal{P}}}{d\mathcal{P}} d\tilde{\mathcal{P}} \\ \text{s.t.} \quad & x(0) \sim \pi_0, \quad x(T) \sim \pi_T \end{aligned}}$$

refaire cette illustration avec le controlled, uncontrolled et target (optimal controlled) processus

Controlled and Reference Processes

Controlled forward transition (learned drift).

$$P_F(x_{n+1} \mid x_n) \;=\; \mathcal{N}\big(x_{n+1};\; x_n + h\, f_\theta(x_n, n),\; h\sigma^2 I\big)$$

Controlled process.

$$Q(x_{0:N}) \;=\; p_0^{\text{ref}}(x_0) \prod_{n=0}^{N-1} P_F(x_{n+1} \mid x_n)$$

Uncontrolled/Reference forward transition (zero drift).

$$P_F^{\text{ref}}(x_{n+1} \mid x_n) \;=\; \mathcal{N}\big(x_{n+1};\; x_n,\; h\sigma^2 I\big)$$

Uncontrolled/Reference process and marginals.

$$Q^{\text{ref}}(x_{0:N}) \;=\; p_0^{\text{ref}}(x_0) \prod_{n=0}^{N-1} P_F^{\text{ref}}(x_{n+1} \mid x_n),$$

$$p_n^{\text{ref}}(x) = \mathcal{N}\left(x; \mu_0, \Sigma_0 + nh\sigma^2 I\right), \text{ is closed form.}$$

Goal. Learn f so that the terminal marginal $Q(x_N)$ matches $\pi(x) = \mu(x)/Z$ (no data, Z unknown.).

Controlled and Reference Processes

Controlled forward transition (learned drift).

$$P_F(x_{n+1} \mid x_n) \;=\; \mathcal{N}(x_{n+1};\; x_n + h\, f_\theta(x_n, n),\; h\sigma^2 I)$$

Controlled process.

$$Q(x_{0:N}) \;=\; p_0^{\mathrm{ref}}(x_0) \prod_{n=0}^{N-1} P_F(x_{n+1} \mid x_n)$$

Uncontrolled/Reference forward transition (zero drift).

$$P_F^{\mathrm{ref}}(x_{n+1} \mid x_n) \;=\; \mathcal{N}(x_{n+1};\; x_n,\; h\sigma^2 I)$$

Uncontrolled/Reference process and marginals.

$$Q^{\mathrm{ref}}(x_{0:N}) \;=\; p_0^{\mathrm{ref}}(x_0) \prod_{n=0}^{N-1} P_F^{\mathrm{ref}}(x_{n+1} \mid x_n),$$

$$p_n^{\mathrm{ref}}(x) = \mathcal{N}\left(x; \mu_0, \Sigma_0 + nh\sigma^2 I\right), \text{ is closed form.}$$

Goal. Learn f so that the terminal marginal $Q(x_N)$ matches $\pi(x) = \mu(x)/Z$ (no data, Z unknown.).

Why closed form for uncontrolled but not controlled? The uncontrolled reference process has no drift, so each transition is a simple Gaussian convolution, leading to Gaussian marginals with closed-form means and variances ($e.g., p_n^{ref}(x) = N(x; mu_0, cov_0 + nh\sigma^2 I)$). The controlled process includes a learned drift $f(x_n, n)$, which makes transitions non-Gaussian and dependent on f, preventing a closed-form marginal expression without solving the integral numerically or via simulation.

je dois refaire cette slide, on ne voit meme comment xn est sample (par rapport a la SDE) et aussi d'ou sort sigma mu et la closed form expliquer et c'est une convolution de gaussienne

Path space KL objective

Target path measure via terminal reweighting.

$$P(x_{0:N}) \;\propto\; Q^{\text{ref}}(x_{0:N-1}|x_N)\mu(x_N) \;\propto\; Q^{\text{ref}}(x_{0:N})\,\frac{\mu(x_N)}{p_N^{\text{ref}}(x_N)} \qquad \Longrightarrow \qquad P(x_N) \propto \mu(x_N).$$

KL decomposition.

$$\mathrm{KL}(Q\|P) = \mathbb{E}_Q\left[\log\frac{Q}{P}\right] = \mathbb{E}_Q\left[\log\frac{Q}{Q^{\text{ref}}}\right] + \mathbb{E}_Q\left[\log\frac{p_N^{\text{ref}}(x_N)}{\mu(x_N)}\right] + \log Z.$$

Running control cost (Gaussian mean-shift) Girsanov theorem.

$$\mathbb{E}_Q\left[\log\frac{Q}{Q^{\text{ref}}}\right] = \mathbb{E}_Q\sum_{n=0}^{N-1}\frac{h}{2\sigma^2}\,\|f_\theta(x_n,n)\|^2.$$

Terminal cost.

$$\mathbb{E}_Q\left[\log\frac{p_N^{\text{ref}}(x_N)}{\mu(x_N)}\right] = \mathbb{E}_Q\big[\log p_N^{\text{ref}}(x_N) - \log\mu(x_N)\big]$$

Path space KL objective

Target path measure via terminal reweighting.

$$P(x_{0:N}) \propto Q^{\mathrm{ref}}(x_{0:N-1}|x_N)\mu(x_N) \propto Q^{\mathrm{ref}}(x_{0:N})\frac{\mu(x_N)}{p_N^{\mathrm{ref}}(x_N)} \qquad \Longrightarrow \qquad P(x_N) \propto \mu(x_N).$$

KL decomposition.

$$\mathrm{KL}(Q\|P) = \mathbb{E}_Q\left[\log\frac{Q}{P}\right] = \mathbb{E}_Q\left[\log\frac{Q}{Q^{\mathrm{ref}}}\right] + \mathbb{E}_Q\left[\log\frac{p_N^{\mathrm{ref}}(x_N)}{\mu(x_N)}\right] + \log Z.$$

Running control cost (Gaussian mean-shift) Girsanov theorem.

$$\mathbb{E}_Q\left[\log\frac{Q}{Q^{\mathrm{ref}}}\right] = \mathbb{E}_Q\sum_{n=0}^{N-1}\frac{h}{2\sigma^2}\|f_\theta(x_n,n)\|^2.$$

Terminal cost.

$$\mathbb{E}_Q\left[\log\frac{p_N^{\mathrm{ref}}(x_N)}{\mu(x_N)}\right] = \mathbb{E}_Q\left[\log p_N^{\mathrm{ref}}(x_N) - \log\mu(x_N)\right]$$

Je devrais avoir une slide complete a explier d'ou vient le foward target process P parce que c'est le coeur de mon algorithme

Target Path Measure via Importance Sampling

Importance Sampling Basics. Importance sampling is a technique to estimate expectations under a hard-to-sample target distribution P using samples from an easy-to-sample proposal distribution Q. The key formula is:

$$\mathbb{E}_{x\sim Q}[f(x)\cdot w(x)] = \mathbb{E}_{x\sim P}[f(x)],$$

where the *importance weight* $w(x) = \frac{P(x)}{Q(x)}$ corrects the samples from Q to behave as if they were from P.
Intuition. Q provides "biased" samples; $w(x)$ upweights samples that are likely under P and downweights those that aren't, effectively resampling from P without directly sampling it.
Application to Path Measures. In our case, the "samples" are entire trajectories $x_{0:N}$, and we want the path measure P to have the correct terminal marginal $\pi(x_N) \propto \mu(x_N)$. We use the reference path measure Q^{ref} (easy to sample, e.g., Gaussian paths) as the proposal. The reweighted path measure is:

$$P(x_{0:N}) \propto Q^{\text{ref}}(x_{0:N}) \cdot w(x_{0:N}),$$

where the weight $w(x_{0:N}) = \frac{\pi(x_N)}{p_N^{\text{ref}}(x_N)}$. This ensures $P(x_N) \propto \mu(x_N)$, as the weight depends only on the endpoint.

Credit Assignment Problem in SOC objective

SOC Discrete-time objective

$$\min_{f_\theta} \mathbb{E}_Q\Big[\underbrace{\sum_{n=0}^{N-1} \frac{h}{2\sigma^2}\|f_\theta(x_n, n)\|^2}_{\text{Running cost}} + \underbrace{\log p_N^{\text{ref}}(x_N) - \log \mu(x_N)}_{\text{Terminal cost}}\Big]$$

This objective is used on the seminar paper *Path Integral Sampler: Diffusion-based Sampling for Unnormalized Densities* by Dinghuai Zhang et al which presented the sampling from unnormalized densities as a stochastic optimal control problem.

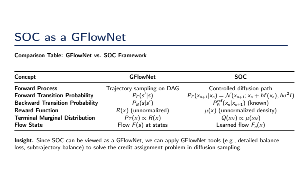
Explanation. The SOC objective provides feedback signal only at the terminal step, making credit assignment difficult via backpropagation through time. This causes high-variance gradients, weak feedback for early actions, poor mode discovery, and inefficient optimization of the drift f without intermediate signals.

SOC as a GFlowNet

Comparison Table: GFlowNet vs. SOC Framework

Concept	**GFlowNet**	**SOC**
Forward Process	Trajectory sampling on DAG	Controlled diffusion path
Forward Transition Probability	$P_F(s'\|s)$	$P_F(x_{n+1}\|x_n) = \mathcal{N}(x_{n+1}; x_n + hf(x_n), h\sigma^2 I)$
Backward Transition Probability	$P_B(s\|s')$	$P_B^{\text{ref}}(x_n\|x_{n+1})$ (known)
Reward Function	$R(x)$ (unnormalized)	$\mu(x)$ (unnormalized density)
Terminal Marginal Distribution	$P_T(x) \propto R(x)$	$Q(x_N) \propto \mu(x_N)$
Flow State	Flow $F(s)$ at states	Learned flow $F_n(x)$

Insight. Since SOC can be viewed as a GFlowNet, we can apply GFlowNet tools (e.g., detailed balance loss, subtrajectory balance) to solve the credit assignment problem in diffusion sampling.



SOC as a GFlowNet

Comparison Table: GFlowNet vs. SOC Framework

Concept	GFlowNet	SOC
Forward Process	Trajectory sampling on DAG	Controlled diffusion path
Forward Transition Probability	$P_F(s'\|s)$	$P_F(x_{n+1}\|x_n) = \mathcal{N}(x_{n+1}; x_n + hf(x_n), h\sigma^2 I)$
Backward Transition Probability	$P_B(s\|s')$	$P_B^{\text{ref}}(x_n\|x_{n+1})$ (known)
Reward Function	$R(x)$ (unnormalized)	$\mu(x)$ (unnormalized density)
Terminal Marginal Distribution	$P_T(x) \propto R(x)$	$Q(x_N) \propto \mu(x_N)$
Flow State	Flow $F(s)$ at states	Learned flow $F_n(x)$

Insight. Since SOC can be viewed as a GFlowNet, we can apply GFlowNet tools (e.g., detailed balance loss, subtrajectory balance) to solve the credit assignment problem in diffusion sampling.

make a mental note to comeback to professor hernandez comment on Tristan's Deleu presentation when someone asked: "The if GFlowNets are just Reinfrocement Learning, then why keep doing research on this framework?" Professor Hernandez reponded by proving the felxibility of GFN, he said that it was en entery point between RL, Diffusion models, Energy-based models and in this case we prove that it can also be used in Stochastic Optimal Control problems.

Decomposition of the Target Process

Target Process Decomposition. The target path measure can be decomposed into a product of conditional distributions:

$$P(x_{0:N}) \propto Q^{\text{ref}}(x_{0:N}) \frac{\mu(x_N)}{p_N^{\text{ref}}(x_N)} \implies P(x_N) \propto \mu(x_N).$$

Conditional Form (Equation 10). Since the reweighting only affects the terminal state, the joint can be written as:

$$P(x_{0:N}) = \pi(x_N) \prod_{n=0}^{N-1} P_B^{\text{ref}}(x_n|x_{n+1}),$$

where $P_B^{\text{ref}}(\cdot|\cdot)$ is the backward transition probability (derived from the target joint). This is tractable because P_B^{ref} is known (e.g., a Gaussian bridge for diffusion processes).
Intuition. This factorization shows that the target process is like sampling backward from the terminal distribution $\pi(x_N)$, using the backward kernel to condition earlier states. It's analogous to reverse-time diffusion, where we start from the target and go backward.

Rewriting the Target Process with Marginal

Rewriting the Joint Using Marginal Densities. The backward factorization of the target process is:

$$P(x_{0:N}) = \pi(x_N) \prod_{n=0}^{N-1} P_B^{\mathbf{ref}}(x_n | x_{n+1}).$$

To express it in terms of the marginal densities $p_n(x_n)$, note that the marginal at step n is obtained by integrating out the future states $x_{n+1:N}$ from the joint:

$$p_n(x_n) = \int P(x_{0:N})\, dx_{0:n-1}\, dx_{n+1:N} = \int \pi(x_N) \prod_{l=n}^{N-1} P_B^{\mathbf{ref}}(x_l | x_{l+1})\, dx_{n+1:N} \propto \int \mu(x_N) \prod_{l=n}^{N-1} P_B^{\mathbf{ref}}(x_l | x_{l+1})\, dx_{n+1:N}.$$

This shows that $p_n(x_n)$ is the "unnormalized" density at step n, propagated backward from the terminal $\pi(x_N)$ via the backward transitions.
What $p_n(x_n)$ Represents.

- $p_N(x_N) = \pi(x_N)$: The terminal marginal, which is the normalized target density we want to match (proportional to $\mu(x_N)$).
- For $n < N$, $p_n(x_n)$ is the marginal density at step n under the target process P. It represents how likely the state x_n is at time n, given that the trajectory will eventually reach a terminal state distributed as $\pi(x_N)$. In other words, it's the distribution of x_n marginalized over all possible future paths that lead to $\pi(x_N)$.
- Intuitively, $p_n(x_n)$ encodes the "value" or importance of being at x_n at step n, as it accounts for the probability of reaching high-μ terminals from there. This is why approximating $p_n(x_n)$ allows training with partial trajectories starting from step n.

Why This Helps in Writing the Target Process. The joint $P(x_{0:N})$ can be thought of as a chain where each $p_n(x_n)$ summarizes the "progress" toward the terminal, but the backward form directly ties it to $\pi(x_N)$. By learning $F_n \approx p_n$, we can reconstruct or approximate the joint without computing the full integral, enabling efficient partial-trajectory optimization.

Rewriting the Target Process with Marginal

If We Had Access to $p_n(x_n)$. We could write the partial joint as:

$$P(x_{\mathbf{0}:n}) = p_n(x_n) \prod_{k=\mathbf{0}}^{n-\mathbf{1}} P_B(x_k|x_{k+\mathbf{1}}),$$

where P_B is the backward kernel. This would allow us to define partial trajectory targets, enabling training with shorter trajectories and better credit assignment.
But There's No Closed Form for $p_n(x_n)$. To calculate $p_n(x_n)$, we would need to compute the integral:

$$p_n(x_n) = \int \pi(x_N) \prod_{l=n}^{N-\mathbf{1}} P_B(x_l|x_{l+\mathbf{1}})\, dx_{n+\mathbf{1}:N}.$$

This is a high-dimensional integral with no analytical solution for general π.
Why Parameterize Both $F_n(\theta)$ and P_F? The constraint requires both: F_n approximates p_n, and P_F is the forward policy we learn. We can't simply calculate the forward process because it's unknown—we're learning it to steer trajectories toward the target. Parameterizing both allows the constraint to hold, enabling amortized learning without per-step integrals.
Naive Alternative: Monte Carlo Quadrature. Quadrature is a numerical integration method that approximates integrals by evaluating the integrand at a set of points (nodes) and weighting them. Monte Carlo quadrature uses random sampling for high dimensions.
Why Expensive. Quadrature requires many evaluations of the integrand (e.g., sampling trajectories), which is computationally intensive and scales poorly with dimensionality, making it impractical for per-training-step use as a replacement for p_n.

Amortized Training: The Subtrajectory Constraint

Proposed Amortized Approach. Train $F_n(\cdot;\theta)$ to satisfy the following constraint for all partial trajectories $x_{n:N}$:

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

Details.

- P_F (forward policy) and F_n are parameterized by deep neural networks.
- Once this constraint holds, $F_n(x_n;\theta)$ equals the integral in Equation 11, amortizing the integration into the learning of θ.
- We use only $\mu(\cdot)$ (no Z), so the unknown normalization is absorbed into F_n.

Decision. This avoids per-step quadrature by enforcing a global constraint on partial trajectories, making training efficient and scalable.

Training Objective and Implementation

Training Objective. Regress the left-hand side (LHS) of the constraint to the right-hand side (RHS). For stability, perform this in log space:

$$\min_{\theta} \left| \log(\mathrm{LHS}) - \log(\mathrm{RHS}) \right| .$$

Details.

- This is a regression loss that minimizes the difference between the learned flow products and the target products involving μ.
- Log space prevents numerical issues from large or small values.

Shared Parameters. The flow function at different steps shares the same parameters; achieved by adding a step embedding input to the network $F(\cdot, n; \theta)$.
Decision. Sharing parameters reduces model complexity and ensures consistency across time steps, while embeddings allow step-specific behavior.

Derived Subtrajectory Balance (SubTB)

Deriving SubTB. Comparing the constraint for n and $n+1$ gives a formula independent of μ:

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

Details.

- This is the Subtrajectory Balance (SubTB) constraint, a generalization of detailed balance for partial trajectories.
- It provides intermediate learning signals without needing μ directly, improving credit assignment.

Overall Decision. The method learns two neural networks: the flow $F(\cdot, n;\theta)$ and the forward policy P_F. This setup enables efficient, stable training with intermediate supervision.

Subtrajectory Balance (SubTB) Loss

SubTB Loss for Partial Trajectories.

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

Explanation of Parameters and Variables.

- θ: Parameters of the neural networks (for F_n and P_F).
- $x_{m:n}$: Subtrajectory from step m to n (partial path $x_m, \ldots, x_n$).
- $F_m(x_m;\theta)$: Learned flow function approximating the marginal density at step m.
- $P_F(x_{l+1}|x_l;\theta)$: Learned forward transition probability (includes the drift f).
- $P_B(x_l|x_{l+1})$: Fixed backward transition probability (reference, known).
- The loss enforces balance for subtrajectories, providing intermediate signals without full trajectories.

Where is f (Learned Drift)? f appears in P_F, as $P_F(x_{n+1}|x_n;\theta) = \mathcal{N}(x_{n+1}; x_n + hf(x_n, n), h\sigma^2 I)$, where f is parameterized by neural networks.

Overall Training Objective

Overall Loss for a Full Trajectory.

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

Explanation of Parameters and Variables.

- τ: Full trajectory $(x_0, \ldots, x_N)$.
- λ: Positive scalar weighting different subtrajectory lengths (e.g., shorter subtrajectories get higher weight if $\lambda < 1$).
- The numerator sums SubTB losses over all subtrajectories $x_{m:n}$, weighted by λ^{n-m} (length-based weighting).
- The denominator normalizes to average the losses.
- This combines signals from all subtrajectory lengths, reducing variance and improving credit assignment.

Where is f (Learned Drift)? f is embedded in P_F within each ℓ_{SubTB}. The loss optimizes θ, which includes parameters for f (via NN1 and NN2: $f(\cdot, n)/\sigma = \mathrm{NN1}(\cdot, n) + \mathrm{NN2}(n) \cdot \nabla \log \mu(\cdot)$), steering the forward process toward the target.

DGFS algorithm

Algorithm 1 DGFS training algorithm

Require: DGFS model with parameters θ, reward function $\mu(\cdot)$, variance coefficient $\bar{\sigma}$.

repeat

Sample τ with control $\mathbf{f}(\cdot, \cdot; \theta)$ and $\bar{\sigma}$;

$\triangle\theta \leftarrow \nabla_{\theta}\mathcal{L}(\tau; \theta)$ (as per Equation 15);

Update θ with Adam optimzier;

until some convergence condition

Reduction of gradient variance

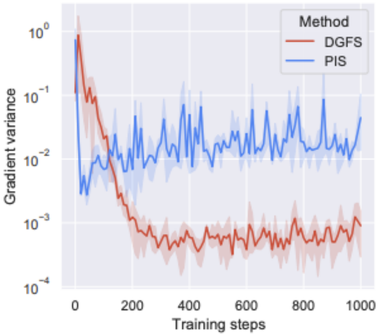


Figure: Gradient variance comparison between DGFS and PIS. DGFS shows significantly lower variance, leading to more stable training.

Accurate partition function Z estimation

Estimation of the partition function. For an arbitrary trajectory $\tau = \{x_n\}_{n=0}^N$, one could define its (log) importance weight to be $S(\tau;\theta) = \log P(x_{0:N}) - \log Q(x_{0:N};\theta)$, where $Q(x_{0:N};\theta) = p_0^{\text{ref}}(x_0)\prod_{n=0}^{N-1} P_F(x_{n+1}|x_n;\theta)$.
Log-partition function (log Z) estimator:

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

Explanation.

- $S(\tau;\theta)$: The log importance weight, measuring how much more likely the trajectory is under the target P (which has terminal marginal $\pi \propto \mu$) than under the learned Q.
- $Q(x_{0:N};\theta)$: The learned path measure, with terminal marginal approximating π.
- The estimator uses importance sampling: Sample trajectories from Q, compute their weights, and average in log space to estimate $\log Z$.
- The inequality $\leq \log Z$ holds because Q approximates P, providing a lower bound on $\log Z$.
- This gives an accurate estimate of Z without needing samples from π, leveraging the learned model.

Partition function estimation results

	MoG	Funnel	Manywell	VAE	Cox
SMC	$0.289_{\pm 0.112}$	$0.307_{\pm 0.076}$	$22.36_{\pm 7.536}$	$14.34_{\pm 2.604}$	$99.61_{\pm 8.382}$
VI-NF	$1.354_{\pm 0.473}$	$0.272_{\pm 0.048}$	$2.677_{\pm 0.016}$	$6.961_{\pm 2.501}$	$83.49_{\pm 2.434}$
CRAFT	$0.348_{\pm 0.210}$	$0.454_{\pm 0.485}$	$0.987_{\pm 0.599}$	$0.521_{\pm 0.239}$	$13.79_{\pm 2.351}$
FAB w/ Buffer[5]	$0.003_{\pm 0.0005}$	$0.0022_{\pm 0.0005}$	$0.032_{\pm 0.004}$	N/A	$0.19_{\pm 0.04}$
PIS	$0.036_{\pm 0.007}$	$0.305_{\pm 0.013}$	$1.391_{\pm 1.014}$	$2.049_{\pm 2.826}$	$11.28_{\pm 1.365}$
DDS	$0.028_{\pm 0.013}$	$0.416_{\pm 0.094}$	$1.154_{\pm 0.626}$	$1.740_{\pm 1.158}$	N/A[6]
DGFS	$0.019_{\pm 0.008}$	$0.274_{\pm 0.014}$	$0.904_{\pm 0.067}$	$0.180_{\pm 0.083}$	$8.974_{\pm 1.169}$

Figure: To write a caption

Mode coverage results

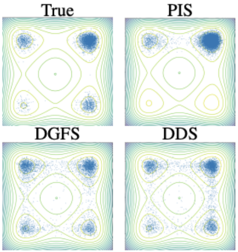


Figure: Manywell plots. DGFS and DDS but not PIS recover all modes

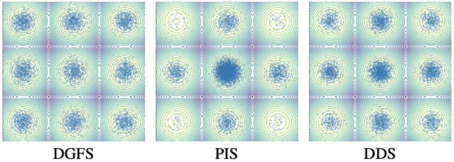


Figure: MoG visualization of DGFS and other diffusion-based samplers shows that DGFS could capture the diverse modes well. The contours display the landscape of the target density

Convergence Theorem

Convergence Guarantees for DGFS.

- Previous studies (Bortoli, 2022; Chen et al., 2022; Lee et al., 2022) show that the terminal sample distribution of a diffusion model converges to the target under mild assumptions if the control term is well learned. These apply to DGFS since proofs are independent of training method.
- Zhang et al. (2022a) prove that a perfectly learned score corresponds to zero GFlowNet training loss (Bengio et al., 2021; 2023), ensuring a well-trained DGFS accurately samples from the target distribution.

Off-policy training capability

Conclusion

Summary of DGFS. 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 can update its parameters with only partial specification of the stochastic process trajectory; moreover, 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.

Questions and hopefully answers :)

Questions?