STEERING MASKED DISCRETE DIFFUSION MODELS VIA DISCRETE DENOISING POSTERIOR PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative modeling of discrete data underlies important applications spanning text-based agents like ChatGPT to the design of the very building blocks of life in protein sequences. However, application domains need to exert control over the generated data by steering the generative process—typically via RLHF—to satisfy a specified property, reward, or affinity metric. In this paper, we study the problem of steering Masked Diffusion Models (MDMs), a recent class of discrete diffusion models that offer a compelling alternative to traditional autoregressive models. We introduce DISCRETE DENOISING POSTERIOR PREDICTION (DDPP), a novel framework that casts the task of steering pre-trained MDMs as a problem of probabilistic inference by learning to sample from a target Bayesian posterior. Our DDPP framework leads to a family of three novel objectives that are all simulation-free, and thus scalable while applying to general non-differentiable reward functions. Empirically, we instantiate DDPP by steering MDMs to perform class-conditional pixel-level image modeling, RLHF-based alignment of MDMs using text-based rewards, and finetuning protein language models to generate more diverse secondary structures and shorter proteins. We substantiate our designs via wet-lab validation, where we observe transient expression of reward-optimized protein sequences.

1 INTRODUCTION

The success of diffusion models in continuous spaces, leading to state-of-the-art foundation models for image (Stability AI, 2023; Midjourney, 2023) and video synthesis (Villegas et al., 2022; Brooks et al., 2024), has spurned several attempts to translate these approaches for the generative modeling of discrete structures. The most performant approaches squarely fall under the scalable framework of absorbing state discrete diffusion (Austin et al., 2021), with new simplified training recipes that result in Masked Diffusion Models (MDMs) (Sahoo et al., 2024; Shi et al., 2024; Gat et al., 2024; Zhao et al., 2024a). Indeed, recent MDMs now rival autoregressive models of a similar scale to GPT-2 (Radford et al., 2019) for language modeling, with the potential for further progress through scaling. Furthermore, MDM style models are not constrained to generating data sequentially—unlike autoregressive models—which invites a more straightforward application to domains without a natural causal ordering, e.g. molecule generation (Vignac et al., 2022), discrete modeling of images (Salimans et al., 2017), and modeling protein sequences (Lin et al., 2022; Wang et al., 2024).

Critical to the successful deployment of discrete generative models in practical applications—beyond simply producing high-quality samples—is the ability to steer the generated samples to optimize a pre-specified downstream metric. For instance, in Language Modeling (LM) it is desirable to bias the model's generations to be sanitized from harmful responses (Zou et al., 2023; Perez et al., 2022), or aiming to generate protein sequences that are highly likely to be successfully synthesized and expressed in real wet lab settings (Verkuil et al., 2022; Dauparas et al., 2022). Put succinctly, highly performant discrete generative models are required to be aligned in a manner that fine-tuning against downstream reward models has the intended effect of *controllable generation*, wherein the model post fine-tuning selects high-scoring samples from the universe of possible high-fidelity generations.

The standard approach for incorporating steerability into discrete generative models, which are autoregressive, using pre-defined reward models is often framed as a fine-tuning task using reinforcement learning from human feedback (RLHF) (Christiano et al., 2017; Rafailov et al., 2024). However, applying RLHF frameworks to diffusion models is far more challenging. Unlike autoregressive models, diffusion models do not allow for straightforward computation of a sample's exact likelihood without costly simulations. Although fine-tuning diffusion models that bypass exact likelihood

computation can yield simulation-free algorithms that resemble RLHF (Wallace et al., 2024; Uehara et al., 2024a), these methods effectively optimize a loose lower bound to the true RLHF objective, leading to unstable training and suboptimal fine-tuning performance. Consequently, steering diffusion models in continuous spaces is primarily done through inference techniques that leverage the gradient of a conditional model in the form of guidance (Dhariwal and Nichol, 2021; Ho and Salimans, 2022). Unfortunately, discrete settings do not allow for principled definitions of guidance due to the lack of a conventional gradient operator. As a result, at present, there exists no scalable and rigorous method to steer and align Masked Diffusion Models to optimize desired reward models.

Main contributions. In this paper, we cast the problem of steering a Masked Diffusion Model as a task of probabilistic inference in sampling from a target Bayesian posterior. More precisely, we construct the target Bayesian posterior as being proportional to the product distribution of a base pre-trained MDM model modulated by a pre-specified reward model. Importantly, this sampling viewpoint is fully compatible with classical RLHF for autoregressive models (Uehara et al., 2024a; Zhao et al., 2024b), but enjoys broader applicability as for the first time it can be applied to discrete diffusion models. Under this sampling perspective, our key insight is that the challenging sampling problem can be solved by learning an amortized sampler, which when taken as an MDM, can be viewed as *finetuning* the pre-trained MDM model by learning to approximate the (reward-induced) Bayesian posterior.

We introduce DISCRETE DENOISING POSTERIOR PREDICTION (DDPP), a novel framework that exploits the denoising posterior parametrization inherent to current MDMs to define a series of simpler matching problems across varying corruption (masking) levels of the target Bayesian posterior. In particular, DDPP designs a forward process that corrupts the Bayesian posterior through a forward masking process and frames the finetuning task as learning another MDM to approximate the corresponding reverse process. As a result, each matching problem in the reverse process requires the construction of a "denoising" Bayesian posterior that is conditioned on a partially masked sample which we demonstrate is simply proportional to the pre-trained model's own denoising posterior and the (terminal) reward of the fully unmasked sample. Crucially, each matching problem in DDPP can be defined on a particular noise level without running the entire forward (corruption) process. Consequently, this makes DDPP a *simulation-free* method which is a key ingredient needed to finetune large pre-trained MDMs. We test the empirical caliber of DDPP by steering MDMs across a multitude of domains ranging from images to protein sequences and steering MDM-based language models. We observe DDPP fine-tuned MDMs lead to competitive performance on images, transient expression of reward-optimized protein sequences (with high secondary structure diversity and β-sheet composition) in a wet-lab setting, and natural textual responses that are steered to human sentiments.

2 BACKGROUND AND PRELIMINARIES

Notation and convention. A discrete data sample x is identified by its realization over a vocabulary set $\mathcal{X} = \{1, \dots, d-1\}$, over $d-1$ possible categories. Of particular interest is the setting of masked diffusion models that include an additional d-th category of a masked token $\mathbf{m}$ to the vocabulary $\mathcal{X}$ which serves as an absorbing state for the diffusion process. A discrete token is represented by the one-hot vector $e^i \in \Delta^d$ in the d-dimensional probability simplex and corresponds to placing a 1 on the i-th index and 0 on all the other $d-1$ indices. In this paper, by convention, we set $e^{\mathbf{m}} = e^d$ as the one hot vector associated with the masked state $\mathbf{m}$. A categorical distribution over d-categories, $\text{Cat}(x; p)$, is constructed by placing a Dirac δ with weight p^i, with the constraint $\sum_i p^i = 1$ and the density of a discrete sample is written as $p(X = x) = \sum_{i=0}^{d} p^i \delta(x - e^i)$, where X is the discrete random variable. A sequence $\mathbf{x} = (x^1, \dots, x^n)$ of n tokens is defined over the product space $\mathcal{X}^n = \{1, \dots, \mathbf{m}\}^n$, and its corresponding probability mass function is given by $p(X = \mathbf{x}) = \prod_i^n \sum_{j=0}^{d} p^j \delta(x^i - e^j)$. To reduce notational clutter, we make use of the shorthand $\delta(y)$ to denote a Dirac measure on a discrete sample y and interchangeably write $p(X = \mathbf{x}) = p(\mathbf{x})$ to denote the probability mass function. A dataset of sequences is designated as samples from the data distribution p_{data} to be learned by a discrete diffusion model q_θ, with parameters θ. Discrete diffusion models, like their continuous counterparts, are stochastic processes that evolve with time $t \in [0, 1]$ such that $t = 0$ corresponds to $p_{\text{data}} := p_0$ and $t = 1$ corresponds to the terminal distribution, p_1 of the process. As a discretization of time, we divide $[0, 1]$ into T intervals, and let $t(i) = i/T$. For brevity, we drop i and simply write t to denote the corresponding discrete timestep $t(i)$. The notation $0 : t$ designates a collection of objects, e.g. densities $p(\mathbf{x}_{0:t})$, starting from time t to and including time $t = 0$. A trajectory of sequences is denoted as $\tau(\mathbf{x}_{0:1}) = \mathbf{x}_1 \to \cdots \to \mathbf{x}_t \to \mathbf{x}_{t-1} \to \cdots \to \mathbf{x}_0$. Finally, we use subscripts to denote the

time index—i.e. p_t—and reserve superscripts to designate indices over a set such as a specific sample $\mathbf{x}^i$ among a collection of samples or dimensions within a vector, e.g. dimension x^i in a sequence.

Problem Setting. We are interested in the task of *probabilistic inference* of sampling from an unnormalized target distribution $\pi_0(\mathbf{x}_0)$ defined over a discrete space consisting of n tokens $\mathbf{x} \in \mathcal{X}^n$,

$$\pi_0(\mathbf{x}_0) = \frac{p_0^{\text{pre}}(\mathbf{x}_0)R(\mathbf{x}_0)}{\mathcal{Z}_{\pi_0}}, \quad R(\mathbf{x}_0) = \frac{\exp(-\mathcal{E}(\mathbf{x}_0))}{\mathcal{Z}_R}. \tag{1}$$

A key aspect of the considered setting is that $\pi_0(\mathbf{x}_0)$ is defined as the product distribution of a pre-trained masked discrete diffusion model $p_0^{\text{pre}}(\mathbf{x}_0)$ and a distribution induced by a (potentially differentiable) reward model $R : \mathcal{X}^n \to \mathbb{R}$. The problem definition in Eq. 1 is an instance of Bayesian posterior sampling where the pre-trained MDM is the prior and reward acts as the likelihood or observation model which modulates samples with a high score. For instance, in scientific domains, the reward model can be provided as a Boltzmann distribution with a known energy function $\mathcal{E}(\mathbf{x}_0)$, or a human preference model as in RLHF (Ouyang et al., 2022; Rafailov et al., 2024). Importantly, this setting does not afford us *any* ground truth samples from $\pi_0(\mathbf{x}_0)$ in the form of a dataset which prevents classically training another generative model. Instead, we are able to evaluate the reward model—and in special cases its gradient ∇R—but not the normalizing constant, i.e. the partition function $\mathcal{Z}_{\pi_0}$. Samples from the posterior π_0 thus lie in the intersection of the modes of both the pretrained MDM and the reward model. As a result, learning an amortized sampler, $q_\theta(\mathbf{x}_0)$, for $\pi(\mathbf{x}_0)$ is rationally equivalent to finetuning the pretrained MDM $p_0^{\text{pre}}(\mathbf{x}_0)$ using the reward $R(\mathbf{x}_0)$ in an analogous manner to RLHF (Uehara et al., 2024a) and is the main focus and contribution of this paper and outlined in §3.2.

2.1 Simplified Masked discrete diffusion

We are interested in developing a discrete diffusion model directly on discrete data—i.e. without embeddings or continuous reparameterizations—whose approach mirrors the construction of diffusion models for continuous spaces. Consequently, we require the specification of a forward process that converts discrete data $\mathbf{x}_0 \sim p_0$ at time $t = 0$ to an unstructured prior, p_1 at the terminal time $t = 1$. The specification of a forward process via the transition kernel $p_t(\mathbf{x}_t|\mathbf{x}_0)$ implies a unique time reversal of this forward process, termed the "reverse process", such that simulating from this reverse process results in samples from the desired target data distribution $p_0(\mathbf{x}_0)$.

We restrict our attention to the recent performant "simplified masked" forward process (Sahoo et al., 2024; Shi et al., 2024; Gat et al., 2024; Zhao et al., 2024a) which hits a terminal distribution of all mask tokens in a sequence $p_1 = [\delta(\mathbf{m})]^n$. Given a non-masked token in a sequence, $x_0^i \in \mathbf{x}$ the simplified masked forward process increases the likelihood of transition to the mask state as time increases. Moreover, the masked forward process is simplified by design since the transition probabilities of a token unmasking ($x_{t+1}^i \neq \mathbf{m}$ when $x_t^i = \mathbf{m}$) is set to zero—i.e. the token remains a masked token for the remainder of the trajectory. The design of the simplified forward process is also independent across each dimension of the sequence, conditioned on $\mathbf{x}_0$, which allows us to model the transitions of each discrete token in a sequence separately. In totality, the forward process for a sequence $\mathbf{x}_0$ can be summarized using the following expression for the transition kernel $p_t(x_t^i|x_0^i)$:

$$p_t(\mathbf{x}_t|\mathbf{x}_0) = \prod_{i=1}^{n} p_t(x_t^i|x_0^i) = \prod_{i=1}^{n} \text{Cat}(x_t^i; \alpha_t\delta(x_0^i) + (1-\alpha_t)\delta(\mathbf{m})), \tag{2}$$

where α_t is an invertible reparameterization of time such that $\alpha_0 = 1$ and $\alpha_1 = 0$. Effectively, α_t corresponds to the noise schedule which corrupts the discrete data to p_1. The corresponding marginal density induced by the forward process at time t can written as $p_t(\mathbf{x}_t) = \sum_{\mathbf{x}_0} p_t(\mathbf{x}_t|\mathbf{x}_0)p_0(\mathbf{x}_0)$.

The reverse process which denoises a sample from $t \to t-1$, and is the time reversal of the simplified masked forward process, also factorizes over each dimension of a sequence $\mathbf{x}$. The probability $p_t(x_{t-1}^i|x_t^i, x_0^i)$ of a reverse transition is given by the following posterior conditioned on x_0^i,

$$p_t(x_{t-1}^i|x_t^i, x_0^i) = \begin{cases} \text{Cat}(x_{t-1}^i; \delta(x_t^i)) & x_t^i \neq \mathbf{m} \\ \text{Cat}\left(x_{t-1}^i; \frac{(1-\alpha_{t-1})\delta(\mathbf{m}) + (\alpha_{t-1}-\alpha_t)\delta(x_0^i)}{1-\alpha_t}\right) & x_t^i = \mathbf{m}. \end{cases} \tag{3}$$

Under the reverse process once a token transitions out of the masked state for a time $t > 0$ it remains in this state for the remainder of the trajectory. The analytical form of the posterior suggests a natural mean parametrization for a denoiser in a discrete diffusion model, $\mu_\theta : \mathcal{X}^n \times \mathbb{R} \to (\Delta^d)^n$, which

predicts the clean sample at $t = 0$ by denoising a noisy x_t^i,

$$q_{t,\theta}(x_{t-1}^i | x_t^i, \mu_\theta(x_t^i, t)) = \begin{cases} \text{Cat}(x_{t-1}^i; \delta(x_t^i)) & x_t^i \neq \mathbf{m} \\ \text{Cat}\left(x_{t-1}^i; \frac{(1-\alpha_{t-1})\delta(\mathbf{m}) + (\alpha_{t-1} - \alpha_t)\mu_\theta(x_t^i, t)}{1-\alpha_t}\right) & x_t^i = \mathbf{m}. \end{cases} \tag{4}$$

Interestingly, the mean parametrization μ_θ used in the posterior is equivalent to predicting the concrete score (Meng et al., 2022) which is the discrete equivalent of the Stein score found in conventional continuous diffusion models (Zheng et al., 2024). As the number of steps $t \to \infty$, training yields a valid *evidence lower bound* (ELBO) to the marginal log-likelihood of the data distribution $\log p(\mathbf{x}_0)$,

$$\log p(\mathbf{x}_0) \geq -\int_0^1 \frac{d\alpha_t}{dt} \cdot \frac{1}{1-\alpha_t} \mathbb{E}_{\mathbf{x}_t \sim p_t(\mathbf{x}_t|\mathbf{x}_0)} \left[\sum_{i=1}^n (x_0^i)^T \log \mu_\theta(x_t^i, t) \right] dt. \tag{5}$$

Thus, when given access to samples $\mathbf{x}_0 \sim p_0$ training an MDM can be seen as optimizing a weighted cross-entropy loss and is analogous to fitting a (mean-field) variational posterior distribution $q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t) = \text{Cat}(\mathbf{x}_0; \mu_\theta(\mathbf{x}_t, t))$ that matches the first moments of $p_t(\mathbf{x}_0|\mathbf{x}_t)$ and also minimizes the forward KL divergence $\mathbb{D}_{\text{KL}}(p_t(\mathbf{x}_0|\mathbf{x}_t)p_t(\mathbf{x}_t)||q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)p_t(\mathbf{x}_t))$ (Eijkelboom et al., 2024).

3 Posterior Sampling via Discrete Denoising Posterior Prediction

Given access to a pretrained masked discrete diffusion model $p_0^{\text{pre}}(\mathbf{x}_0)$ we wish to sample from the reward-induced Bayesian posterior distribution $\pi_0(\mathbf{x}_0) \propto p_0^{\text{pre}}(\mathbf{x}_0)R(\mathbf{x}_0)$. We solve this sampling problem by first defining a time-dependent forward masking process that progressively adds noise to π_0 yielding the noisy reward-induced posterior $\pi_t(\mathbf{x}_t) = \sum_{\mathbf{x}_0} \pi_t(\mathbf{x}_t|\mathbf{x}_0)\pi_0(\mathbf{x}_0)$, where we set $\pi_t(\mathbf{x}_t|\mathbf{x}_0) = p_t(\mathbf{x}_t|\mathbf{x}_0)$ as it is the same masking process for the pre-trained MDM. Unfortunately, since $p_0^{\text{pre}}(\mathbf{x}_0)$ is an MDM it does not easily provide an exact likelihood. Undeterred we seek to approximate the reverse process $\pi_t(\mathbf{x}_{t-1}|\mathbf{x}_t)$ tied to the masking forward process by using another parametrized model $q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t) = \text{Cat}(\mathbf{x}_0; \mu_\theta(\mathbf{x}_t, t))$ which we take to be another MDM.

Matching sub-trajectories. To approximate the reverse process using an MDM we require matching the denoising trajectory $\tau(\mathbf{x}_{0:t})$ of the reward-induced posterior $\pi_t(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t)$ across all masking levels. Assisted in this endeavor, we recall the fact that since $p_0^{\text{pre}}(\mathbf{x}_0)$ is also an MDM, we have direct access to the pre-trained model's denoiser. Thus, we can compute any transition density starting from $p_t^{\text{pre}}(\mathbf{x}_{t-1}|\mathbf{x}_t, \mu^{\text{pre}}(\mathbf{x}_t, t))$ to the posterior over the endpoint $p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t)$, conditioned on a partially masked sample $\mathbf{x}_t$. We form the sub-trajectory matching problem as an instantiation of a detailed balance constraint starting from a partially masked sequence $\mathbf{x}_t$ of a clean starting point $\mathbf{x}_0$:

$$q_\theta(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t, \hat{\mathbf{x}}_0)p_t(\mathbf{x}_t) = \pi_t(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t)p_t(\mathbf{x}_t). \tag{6}$$

Setting $\hat{\mathbf{x}}_0 = \mu_\theta(\mathbf{x}_t, t)$ as the MDM's denoised sample, then $\pi_t(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t)$ is defined as,

$$\pi_t(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t) = \frac{p_t^{\text{pre}}(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t)R(\mathbf{x}_0)}{\mathcal{Z}_{\pi_t}(\mathbf{x}_t)} = \frac{\prod_{j=1}^t p_t^{\text{pre}}(\mathbf{x}_{j-1}|\mathbf{x}_j, \hat{\mathbf{x}}_0^{\text{pre}})R(\mathbf{x}_0)}{\mathcal{Z}_{\pi_t}(\mathbf{x}_t)}.$$

The detailed balance constraint over sub-trajectories suggests a natural discrete denoising posterior predictive (DDPP) objective that minimizes the mean squared error of a log-ratio between the denoising sub-trajectories of the amortized MDM sampler and the reward-induced target posterior,

$$\mathcal{L}_\tau^{\text{PP}} = \mathbb{E}_{t,\mathbf{x}_t}\left[\mathbb{E}_{\tau(\mathbf{x}_{0:t})}[\| \log q_\theta(\mathbf{x}_{0:t-1}|\mathbf{x}_t, \hat{\mathbf{x}}_0)) - \log p_t^{\text{pre}}(\mathbf{x}_{0:t-1}|\mathbf{x}_t) + \kappa\|_2^2]\right], \tag{7}$$

where reward and the log partition function are captured in the constant $\kappa = \log \mathcal{Z}_{\pi_t}(\mathbf{x}_t) - \log R(\mathbf{x}_0)$. Interestingly, we can form an equivalent expression for the sub-trajectory loss $\mathcal{L}_\tau^{\text{PP}}$ above by sampling two intermediate points $\mathbf{x}_s, \mathbf{x}_{s-\gamma}$ in the sub-trajectory $\tau(\mathbf{x}_{0:t})$, such that $0 < s - \gamma < s < t$:

$$\mathcal{L}_\tau^{\text{PP}} = \mathbb{E}_{t,\mathbf{x}_t,\tau(\mathbf{x}_{0:t})}\left[\left\|t\mathbb{E}_{s,\mathbf{x}_s,\mathbf{x}_{s-\gamma}}\left[\log q_\theta(\mathbf{x}_{s-\gamma}|\mathbf{x}_s, \hat{\mathbf{x}}_0) - \log p_t^{\text{pre}}(\mathbf{x}_{s-\gamma}, |\mathbf{x}_s, \hat{\mathbf{x}}_0^{\text{pre}}) + \kappa\right]\right\|_2^2\right]. \tag{8}$$

The proof for this equivalence is presented in §C.3. Note that we sample $s, s-\gamma \sim \mathcal{U}[0,t], \mathcal{U}[0,s]$ uniformly, and when $\gamma = 1/T$ we sample $\mathbf{x}_{s-1}$ which is simple to do since the $\tau(\mathbf{x}_{0:t})$ already contains this information. Crucially, unlike Eq. 7 the reformulation of the sub-trajectory loss in Eq. 8 is effectively a simulation-free version of Relative Trajectory Balance (RTB) (Venkatraman et al., 2024). If the approximation $q_{t,\theta}$ matches the denoising reward-induced target posterior over all sub-trajectories then the reverse process of $q_{t,\theta}$ can be simulated to draw samples that follow $\pi_0(\mathbf{x}_0)$. Consequently, we term the $q_{t,\theta}$ that minimizes the DDPP objective in Eq. 8 as the *finetuned MDM* which solves the probabilistic inference task of sampling from $\pi_0(\mathbf{x}_0)$.

In contrast to learning MDMs in typical generative modeling setups, the DDPP objective requires the computation of the intractable log partition function $\log \mathcal{Z}_{\pi_t}(\mathbf{x}_t)$ evaluated at $\mathbf{x}_t$ which is a component of the term κ. This observation motivates the design of three concrete learning objectives for posterior matching, which as a collection we term the DISCRETE DENOISING POSTERIOR PREDICTION framework. Specifically, finetuning $q_{t,\theta}$ under a DDPP framework can be done in the following algorithms: 1.) DDPP-IS which uses a Monte Carlo based importance sampling estimate to approximate $\log \mathcal{Z}_{\pi_t}$ in Eq. 8, 2.) DDPP-LB that constructs a lower bound to DDPP-IS that is cheaper to evaluate by parameterizing $\log \mathcal{Z}_{\pi_t}$, and 3.) DDPP-KL which uses a discrete gradient estimator to bypass computing $\log \mathcal{Z}_{\pi_t}$ at the cost of requiring a differentiable reward—i.e. ∇R.

3.1 ESTIMATING THE LOG PARTITION FUNCTION

Inspecting the posterior predictive objective in Eq. 8 we remark that it is a simulation-free stochastic regression objective which does not require a differentiable reward as the loss computes $R(\mathbf{x}_0)$ and not a gradient of the reward. Consequently, this makes the posterior predictive objective both a scalable and efficient objective for fine-tuning large pre-trained MDMs as long the reward model is easy to evaluate. Moreover, the posterior predictive objective is also an *off-policy* objective as it can be evaluated using any partially masked samples $\mathbf{x}_t \sim p(\mathbf{x}_t|\mathbf{x}_0)$. Practically, this means that fine-tuning can be performed using a replay buffer of samples from a biased dataset, e.g. the original training set for p_0^{pre}, or even partially masked sequences that arrive from a different model altogether. Despite its simple form the posterior predictive objective requires the computation of the log partition function of a partially masked sequence $\log \mathcal{Z}_{\pi_t}$ which does not have a closed-form expression and must be estimated.

Monte Carlo Estimate of $\log \mathcal{Z}_{\pi_t}$ with DDPP-IS. A numerical estimate of the log normalization constant can be obtained by utilizing the trick of using the pre-trained model's denoising posterior $p^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t)$. Specifically, given $\mathbf{x}_t \sim p_t(\mathbf{x}_t)$ we obtain a Monte Carlo estimate of $\log \mathcal{Z}_{\pi_t}(\mathbf{x}_t)$ that uses M additional samples from $\mathbf{x}_0 \sim p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t)$ to estimate the log partition function,

$$\log \hat{\mathcal{Z}}_{\pi_t}(\mathbf{x}_t) = \log \left(\sum_{\mathbf{x}_0,\ldots\mathbf{x}_{t-1}} p_t^{\text{pre}}(\mathbf{x}_0,\ldots,\mathbf{x}_{t-1}|\mathbf{x}_t) R(\mathbf{x}_0) \right) \approx \log \left(\mathbb{E}_{\mathbf{x}_0' \sim p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t)} [R(\mathbf{x}_0')] \right).$$

Where in the second equality in the first line we used the fact that we can approximately jump to the endpoint of the reverse process directly by using the pretrained model's denoiser to sample $\mathbf{x}_0$. Conveniently, this MC estimate solely requires obtaining a denoised sample from the pre-trained MDM which can be efficiently done as each sample requires a single step as due to the denoising posterior parametrization of an MDM (Eq. 4). We can further improve the estimation of this log normalization constant by leveraging importance sampling (IS) with a proposal distribution $w(\mathbf{x}_0)$:

$$\log \hat{\mathcal{Z}}^{\text{IS}}_{\pi_t}(\mathbf{x}_t) = \log \left(\mathbb{E}_{\mathbf{x}_0' \sim w(\mathbf{x}_0)} \left[\frac{p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t) R(\mathbf{x}_0')}{w(\mathbf{x}_0')} \right] \right) = \log \left(\frac{1}{M} \sum_{j=1}^{M} \left[\frac{p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t) R(\mathbf{x}_0^j)}{w(\mathbf{x}_0^j)} \right] \right).$$

For the IS estimator above it is easy to verify that the optimal proposal distribution for variance reduction is proportional to the denoising reward-induced target posterior $w^*(\mathbf{x}_0) \propto \pi_t(\mathbf{x}_0|\mathbf{x}_t)$. Fortunately, this is precisely the distribution that is approximated by $q_{t,\theta}$ using the posterior predictive objective which motivates the reuse of the finetuned model as a suitable proposal, i.e. $w(\mathbf{x}_0) = q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)$.

Learning $\log \mathcal{Z}_{\pi_t}$ with DDPP-LB. An alternative to using an MC-based estimate for $\log \mathcal{Z}_{\pi_t}$ is to parameterize the log partition function itself $\log \hat{\mathcal{Z}}^{\text{LB}}_{\pi_t,\theta}$ jointly with the $q_{t,\theta}$ and optimize both using the same posterior predictive objective as first defined in Eq. 11. Operationally, this amounts to including another prediction head for the finetuned MDM model and is cheaper to compute than using an MC-based estimate as we do not require M evaluations of the pre-trained model as in $\log \hat{\mathcal{Z}}^{\text{IS}}_{\pi_t}(\mathbf{x}_t)$.

At first glance, it remains unclear whether a parameterized $\log \hat{\mathcal{Z}}^{\text{LB}}_{\pi_t,\theta}$ is a sensible strategy. However, in the particular case where we choose the proposal distribution to be on-policy by using finetuned MDM $w(\mathbf{x}_0) = q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)$, we can show that the learned log partition function estimate is a lower bound to the importance sampling estimate. This is formalized in the following proposition below.

Proposition 1. *Let $\log \hat{\mathcal{Z}}^{IS}_{\pi_t}$ and $\log \hat{\mathcal{Z}}^{LB}_{\pi_t,\theta}$ be the M-sample importance sampling estimate using the proposal $q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)$ and learned approximation to the log partition function respectively. Given a partially masked sample $\mathbf{x}_t \sim p_t(\mathbf{x}_t)$ the optimal learned approximation is a lower bound to the*

Algorithm 1 Single-step DDPP-IS and DDPP-LB

Input: Reward $R(\mathbf{x}_0)$, base MDM $p_0^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t)$, sampling policy $r(\mathbf{x}_0)$, fine-tuning MDM $q_\theta(\mathbf{x}_0|\mathbf{x}_t)$

while Training **do**
  $t, \mathbf{x}_0 \sim \mathcal{U}[0,1], r(\mathbf{x}_0)$ ▷ *Sample time and clean data on or off-policy*
  $\mathbf{x}_t \sim p_t(\mathbf{x}_t|\mathbf{x}_0)$ ▷ *Construct partially masked sample given clean data*
  if Importance Sample $\log \mathcal{Z}(\mathbf{x}_t)$ **then** ▷ *Log Partition Function Estimation Strategy*
    $\log \hat{\mathcal{Z}}_{\pi_t}(\mathbf{x}_t) := \log \hat{\mathcal{Z}}^{\text{IS}}_{\pi_t}(\mathbf{x}_t) = \log\left(\frac{1}{M}\sum_{j=1}^{M}\left[\frac{p_t^{\text{pre}}(\mathbf{x}_0^j|\mathbf{x}_t)R(\mathbf{x}_0^j)}{w(\mathbf{x}_0^j)}\right]\right)$
  else
    $\log \hat{\mathcal{Z}}_{\pi_t}(\mathbf{x}_t) := \log \hat{\mathcal{Z}}^{\text{LB}}_{\pi_t,\theta}(\mathbf{x}_t)$
  $\mathcal{L}^{\text{PP}} = \left|\left|\log q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t) - \log p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t) - \log R(\mathbf{x}_0) + \log \hat{\mathcal{Z}}_{\pi_t}(\mathbf{x}_t)\right|\right|_2^2$
  $\theta \leftarrow \text{Update}(\theta, \nabla_\theta \mathcal{L}^{\text{PP}})$
Return q_θ

importance sampling estimate with a fixed proposal $q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)$ and the following inequality holds:

$$\log \hat{\mathcal{Z}}^{LB}_{\pi_t,\theta}(\mathbf{x}_t) \leq \log \hat{\mathcal{Z}}^{IS}_{\pi_t}(\mathbf{x}_t). \tag{9}$$

The proof for Eq. 1 is provided in §C.1. We highlight that the lower bound becomes equality at the optimal proposal $q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t) \propto \pi_t(\mathbf{x}_0|\mathbf{x}_t)$. Learning $\log \hat{\mathcal{Z}}^{\text{LB}}_{\pi_t,\theta}$ has the benefit of amortization as the same network can be reused for all partially masked samples $\mathbf{x}_t \sim p_t(\mathbf{x}_t)$, across all levels of masking. In addition, over the course of training, the learned estimate $\log \hat{\mathcal{Z}}^{\text{LB}}_{\pi_t,\theta}$ becomes a better estimate for the true log partition function. In practice, it suffices to take a single gradient step to optimize $\log \hat{\mathcal{Z}}^{\text{LB}}_{\pi_t,\theta}$ rather than optimizing till convergence. As a result, no additional overhead needs to be incurred, and the learned estimate is averaged over a batch of noisy samples $\mathcal{B} = \{\mathbf{x}_t^i\}_{i=1}^N$.

3.2 Single-step posterior sampling with endpoint prediction

The sub-trajectory matching objective used by DDPP-IS and DDPP-LB can be simplified to a faster single-step objective at the cost of paying a discretization error by not using finer-grained trajectory information. Specifically, we note that for MDMs the denoising posterior over endpoints $q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t) \approx \text{Cat}(\mathbf{x}_0; \mu_\theta(\mathbf{x}_t, t))$ can be approximately computed *without unrolling the sub-trajectory*. This fact also holds for the pre-trained MDM as the model parametrization implies $p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t) \approx \text{Cat}(\mathbf{x}_0; \mu(\mathbf{x}_t, t))$. For the single-step objective we assume the parameterized denoisers exactly match the posteriors. Leveraging this enables us to express the denoising reward-induced target posterior using a simple expression that directly uses the pre-trained model's denoising posterior $p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t)$ as follows:

$$\pi_t(\mathbf{x}_0|\mathbf{x}_t) = \frac{p_t(\mathbf{x}_t)}{p_t(\mathbf{x}_t)} \cdot \frac{p_t(\mathbf{x}_t|\mathbf{x}_0)p^{\text{pre}}(\mathbf{x}_0)R(\mathbf{x}_0)}{\sum_{\mathbf{x}_0'} p_t(\mathbf{x}_t|\mathbf{x}_0')p^{\text{pre}}(\mathbf{x}_0')R(\mathbf{x}_0')} = \frac{p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t)R(\mathbf{x}_0)}{\mathcal{Z}_{\pi_t}(\mathbf{x}_t)}. \tag{10}$$

The choice of parameterizing $q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)$ as another MDM offers a prescriptive strategy for sampling from the desired target π_0 by learning to match the denoising reward-induced posterior at the predicted endpoint $\pi_t(\mathbf{x}_0|\mathbf{x}_t)$. This simplifies the expression of DDPP defined over trajectories in Eq. 8 to a single point, namely the predicted endpoint $\mathbf{x}_0$ of each MDM. This objective is presented below:

$$\mathcal{L}^{\text{PP}} = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[\left|\left|\log q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t) - \underbrace{\log p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t) - \log R(\mathbf{x}_0) + \log \mathcal{Z}_{\pi_t}(\mathbf{x}_t)}_{\log \pi_t(\mathbf{x}_0|\mathbf{x}_t)}\right|\right|_2^2\right]. \tag{11}$$

As done previously, we can employ any estimation strategy to compute the log partition function Eq. 11. We note in many cases, such as when the sequence length of the trajectory is small to moderate, the single-step objective may be an attractive alternative to the sub-trajectory variants of DDPP. Algorithm 1 provides a detailed description of the single-step version of DDPP.

3.3 DDPP-KL: Posterior prediction via reverse KL minimization

The single-step posterior prediction objective as defined using the loss function $\mathcal{L}^{\text{PP}}$ in Eq. 11 requires the estimation of $\log \mathcal{Z}^{\text{LB}}_{\pi_t,\theta}$ which introduces a source of variance in loss estimates that may sub-

optimally influence learning dynamics of the fine-tuned model. In settings where the reward model is differentiable, we can bypass computing $\log \mathcal{Z}^{\text{LB}}_{\pi_t,\theta}$ altogether by learning to match the denoising reward-induced posterior under the *reverse* KL divergence. Note that the *forward* KL divergence is inapplicable here as we do not have samples from π_0—i.e. a dataset. To see this, we define a variational posterior matching problem using the reverse KL divergence that takes the following form:

$$\mathcal{L}^{\text{KL}}_t := \mathbb{D}_{\text{KL}}(q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)p_t(\mathbf{x}_t)||\pi_t(\mathbf{x}_0|\mathbf{x}_t)p_t(\mathbf{x}_t)). \tag{12}$$

Unlike conventional generative modeling using the reverse KL divergence which solely matches distributions at $t = 0$ the problem definition in Eq. 12 defines a series of reverse KL minimization problems through time. In this manner, the reverse KL matches distributions annealed through time and can be used to derive a stochastic regression objective for fine-tuning,

$$\mathcal{L}^{\text{KL}} = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[\log q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t) - \log p^{\text{pre}}_t(\mathbf{x}_0|\mathbf{x}_t) - \log R(\mathbf{x}_0)\right] + C. \tag{13}$$

The expectation in Eq. 13, like DDPP-IS and DDPP-LB is taken uniformly with respect to time $t \sim \mathcal{U}[0, 1]$. However, unlike the previous estimators, clean data needed to compute $\mathcal{L}^{\text{KL}}$ is drawn purely on-policy by simulating the fine-tuning model $\mathbf{x}_0 \sim q_{t,\theta}(\mathbf{x}_0)$, which then also allows us to craft a noisy sample using the masking forward process $\mathbf{x}_t \sim p_t(\mathbf{x}_t|\mathbf{x}_0)$. Additionally, in Eq. 13 the constant $C = \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}[\log \mathcal{Z}_{\pi_t}(\mathbf{x}_t)]$ does not depend on the θ—and as a result is also independent of the sample $\mathbf{x}_0 \sim q_{t,\theta}(\mathbf{x}_0)$. This results in the constant C being zero when computing the gradient of the loss $\nabla_\theta \mathcal{L}^{\text{KL}}$ and as a result we can safely disregard computing $\log \mathcal{Z}_{\pi_t}$ entirely.

As samples $\mathbf{x}_0$ are procured on-policy to compute the gradient of the loss $\nabla_\theta \mathcal{L}^{\text{KL}}$ we require backpropagating through the stochastic sampling of $\mathbf{x}_0$ which comes from simulating the fine-tuning MDM $q_{t,\theta}(\mathbf{x}_0)$. Fortunately, we can make use of modern discrete gradient estimators which provide a biased but low variance gradient estimate enabling us to compute $\mathcal{L}^{\text{KL}}$. Specifically, we opt to use the scalable 2nd order REINMAX estimator (Liu et al., 2024) which estimates the discrete gradient up to second-order terms in a Taylor approximation of the actual gradient. We note that unlike DDPP-IS and DDPP-LB this new loss that minimizes the reverse KL divergence $\mathcal{L}^{\text{KL}}$ requires the reward model R to be differentiable and as a result is less broadly applicable than computing $\mathcal{L}^{\text{PP}}$. However, in practice, learning can be faster as we make use of the information afforded to us by the gradient ∇R as well as the fact that the objective does not need to estimate the log partition function.

In appendix §C.2 we provide the exact algorithm Alg. 2 to compute the reverse KL objective. We further show how using a gradient estimator like REINMAX can be used to derive efficient gradient estimation for a more general class of problems of sampling from $\pi_0(\mathbf{x}_0) = R(\mathbf{x}_0)/\mathcal{Z}_{\pi_0}$, as well as the main fine-tuning setting for matching the denoising reward-induced posterior as defined in Eq. 10.

4 EXPERIMENTS

We investigate the application of DDPP to a variety of discrete generative modeling settings. We provide the full experimental details in §D and present our main experimental results next.

Table 1: Overview of posterior sampling methods

Method	Model calls / inf. step	Model calls / train step	Sim. Free
SVDD	N	—	✓
Discrete guidance	1	—	✓
RTB	1	T	✗
DDPP-KL	1	1	✓
DDPP-IS	1	M	✓
DDPP-LB	1	1	✓

Baselines. Throughout our experiments, we rely on four principal baselines in: sampling from the pre-trained MDM model, Best-of-N sampling (Stiennon et al., 2020), Relative Trajectory Balance (RTB) (Venkatraman et al., 2024), and SVDD (Li et al., 2024) which is a concurrent inference time technique for steering diffusion models. Best-of-N represents a computationally expensive baseline but is guaranteed to produce samples from π_0, as such we use this as an upper bound on performance in terms of reward obtained as $N \to \infty$ (Beirami et al., 2024). RTB is a GFlowNet (Bengio et al., 2023; Madan et al., 2022; Lahlou et al., 2023) that requires simulating the entire diffusion trajectory. For image settings with differentiable reward, we also include discrete guidance as a baseline (Nisonoff et al., 2024). In Table 1 we illustrate the computational differences between DDPP and baselines.

4.1 SYNTHETIC EXPERIMENTS

We consider a synthetic task of learning to sample from a target distribution on a 2D discrete grid and finetuning an MDM on binarized MNIST. This synthetic setting tests all DDPP variations with chosen baselines, presenting qualitative results in Figure 6, Figure 4 and quantitative results in Table 2.

Grid Experiment. We define a prior density p^{pre}_0 over the discrete 2-dimensional, 128×128 grid, as showcased in Figure 6(a) where the probability mass corresponding to each point $\mathbf{x}_0$ is on if the color is yellow. The goal is to sample from the product distribution as outlined in Equation 1, which in this case

(a) Prior Density (b) Target Density (c) DDPP-IS (d) DDPP-LB (e) DDPP-KL

Figure 1: Samples generated by fine-tuning a masked diffusion model to sample from the lower half of its prior distribution. Samples $\mathbf{x}_0$ in this setting are 2-dimensional, with a vocabulary size of 128.

is defined to drop the modes in p_0^{pre} which are at the top half of the grid, as visualized in Figure 6(b). These results show that all three variants of DDPP effectively learn to sample from this target.

MNIST. We finetune MDMs to generate even MNIST digits. As observed in Table 2 we find that all three variants of DDPP match or outperform the base pre-trained model and RTB in all metrics, with DDPP-KL being the best. In comparison to the concurrent work of SVDD, we find that it outperforms DDPP in average $\log R$ but is worse in sample-based metrics such as class conditional FLD (Jiralerspong et al., 2023) which measures the overall quality, diversity and generalizability of generated samples and class conditional BPD. We further report generated samples in Figure 4 located in §D.3.

4.2 Pixel-level image modelling

We fine-tune MDMs on order-agnostic image data, discretizing pixels in 64 × 64 downsampled CelebA images (Liu et al., 2018) to a vocabulary of 256 tokens. As there are no publicly available pre-trained MDM models we train our own MDM by modeling the raw pixel space and achieve 1.85 bits-per-dim (BPD) on CelebA. Our full experimental setup is outlined in §D.3. For fine-tuning, we consider steering a pre-trained MDM using DDPP-LB as it is the most computationally cheap method with a class-conditional reward based on an auxiliary classifier. Specifically, we steer the generative model to generate human faces with blond hair. For quantitative metrics, we report the mean log reward obtained, and BPD in Figure 3 as well as selected generated samples. Our quantitative results show that our proposed variant DDPP-LB significantly outperforms all other baselines in obtaining the highest reward. We also observe DDPP obtains BPD values that are within the range of the base model while being worse than RTB. We further find visual samples produced by DDPP to have the highest fidelity faces with blond hair, matching our fine-tuning goal.

Table 2: Fine-tuning to produce only even digits on binarized MNIST. We report the mean performance over 3 runs for the $\log R$, FLD, and BPD metrics.

Algorithm ↓ Metric →	$\log R(\mathbf{x}_0)$ ↑	FLD ↓	BPD ↓
Base Model	-26.90 ± —	33.89 ± —	0.130 ± —
SVDD	**-0.03 ± 0.01**	34.19 ± 0.95	—
Guidance (scale 1)	-25.24 ± 0.26	34.67 ± 0.67	0.171 ± 0.001
Guidance (scale 5)	-23.21 ± 0.21	37.33 ± 0.87	0.174 ± 0.001
Guidance (scale 100)	-9.32 ± 0.24	72.19 ± 0.43	0.147 ± 0.001
RTB	-18.66 ± 2.45	45.97 ± 0.89	**0.128 ± 0.000**
DDPP-IS (*ours*)	-5.14 ± 1.24	33.11 ± 0.71	0.130 ± 0.000
DDPP-LB (*ours*)	-5.68 ± 0.34	33.76 ± 0.90	**0.128 ± 0.000**
DDPP-KL (*ours*)	-3.13 ± 0.06	**31.75 ± 0.51**	0.129 ± 0.000

4.3 Protein sequence modelling

Table 3: *In-silico* results for protein generation tasks. We report the mean result for a metric with standard deviation across three seeds. DDPP-LB performs well across designability metrics (pLDDT and pTM) while simultaneously performing best on task specific metrics (β-sheet % and TM-Score).

	High β-sheet-content protein generation				Protein shrinking				
	β-sheet % ↑	pLDDT ↑	pTM ↑	$\log R(x_0)$ ↑	SS-KL ↓	TM-Score ↑	pLDDT ↑	pTM ↑	$\log R(x_0)$ ↑
Base Model	0.111 ± 0.121	0.724 ± 0.144	0.584 ± 0.226	2.070 ± 0.749	3.040 ± 3.043	0.245 ± 0.058	0.724 ± 0.144	0.584 ± 0.226	0.490 ± 0.116
Best-of-10	0.280 ± 0.093	0.812 ± 0.033	0.786 ± 0.035	3.212 ± 0.371	1.621 ± 2.804	0.345 ± 0.049	0.786 ± 0.023	0.737 ± 0.097	0.690 ± 0.098
SVDD	0.114 ± 0.148	0.484 ± 0.134	0.349 ± 0.174	1.669 ± 0.907	3.353 ± 2.913	0.337 ± 0.042	0.492 ± 0.131	0.368 ± 0.171	0.673 ± 0.083
RTB	0.319 ± 0.218	0.806 ± 0.059	0.767 ± 0.101	3.386 ± 1.061	2.193 ± 2.724	0.290 ± 0.056	**0.797 ± 0.056**	0.747 ± 0.093	0.581 ± 0.112
DDPP-LB	**0.436 ± 0.037**	**0.897 ± 0.027**	**0.806 ± 0.029**	**3.703 ± 0.186**	**0.640 ± 1.793**	**0.361 ± 0.047**	0.768 ± 0.048	**0.747 ± 0.063**	**0.722 ± 0.094**

Task description. We next apply DDPP to generate high-quality protein sequences by fine-tuning discrete diffusion protein language models (DPLM) (Wang et al., 2024). Specifically, we address two experimentally relevant tasks where vanilla DPLMs underperform. We outline exact reward functions and experimental setup in §D.2. First, we fine-tune DPLM to generate soluble protein sequences with high β-sheet content. The second task, protein shrinking, involves miniaturizing known proteins by generating shorter sequences that preserve key structural features, using the TM-align score as the reward metric (Devkota et al., 2024). We evaluate performance by measuring designability metrics (ESMFold pLDDT and pTM) as well as task-specific metrics (β-sheet percent and TM-Score). We also provide wet-lab validation for our best designs in the designable β-sheet task. We provide a

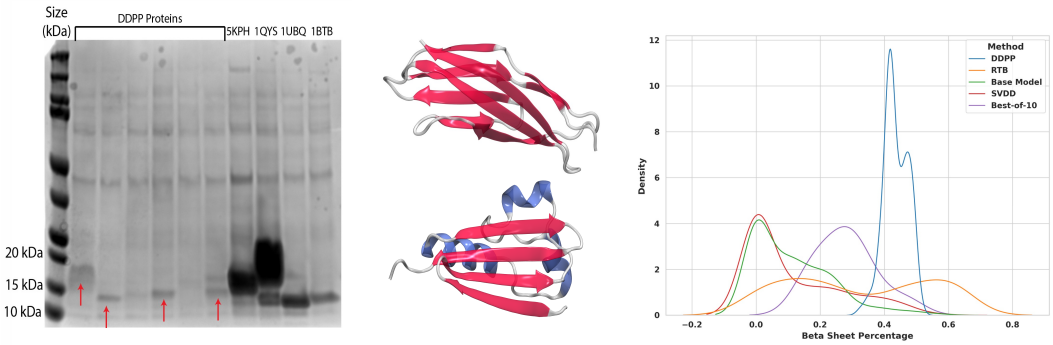


Figure 2: **Left:** SDS-PAGE of elution fractions from histidine tag purification of DDPP-designed protein constructs and positive controls following Coomassie blue staining. All DDPP-designed constructs are between 7.8-8.3 kDa. Predicted molecular weights of positive controls 5KPH, 1QYS, 1UBQ, and 1BTB are 10 kDa, 12 kDa, 8.5 kDa, and 10 kDa, respectively. Recombinant protein bands for MDM-designed sequences are indicated with red arrows and relevant ladder references are labeled with their molecular weight. **Middle:** Folded structures generated by DDPP β-sheet fine-tuning. **Right:** Distribution of β-sheets generated by each method.

deeper description of evaluation metrics and experimental setup in §D.2. Finally, as ESMFold is itself expensive to query and, in particular, non-differentiable we test our fastest method—DDPP-LB.

Main results. In-silico validation shows that DDPP-LB outperforms all baselines for the designable β-sheet task, generating better sequences across all metrics. In particular DDPP achieves a significantly higher β-sheet percentage than baseline methods while maintaining high designability as measured by ESMFold (namely, high pLDDT and pTM). We further observe that for the miniaturization task, DDPP-LB outperforms all baselines in shrinking ribonuclease proteins, removing 34 residues while maintaining high structural similarity (lowest SS-KL of 0.64 and highest TM-Score of 0.361), and high structural quality with high pTM and competitive pLDDT. This demonstrates DDPP-LB's effectiveness in generating compact yet structurally faithful proteins.

Experimental validation. We selected 6 designs from DDPP-finetuned DPLM for wet-lab validation, based on AlphaFold2 pLDDT/pTM scores. Sequences and structures were clustered using MMseqs and Foldseek (van Kempen et al., 2022; Steinegger and Söding, 2017), with two representative sequences selected from each cluster. 4 positive controls consisting of two previously validated de novo designed proteins (PDB: 5KPH, 1QYS) and two other stable proteins, ubiquitin and Barstar (PDB: 1UBQ, 1BTB) were included as a comparison. We expressed the designed proteins, including the controls in E. coli, and purified them using histidine-tag purification, after which we assessed expression level and purity via SDS-PAGE, followed by Coomassie staining. Our results demonstrate strong overexpression and efficient purification of the two previously validated de novo controls and moderate overexpression of ubiquitin and barstar controls (Figure 2). Purified protein can also be observed for four out of the six DDPP-derived constructs, though with comparatively lower yields than the positive controls (Figure 2). One potential cause of these relatively low yields may be the sizeable accumulation of DDPP-derived proteins in the insoluble fraction of the cell lysate. As such, it is likely that further optimization of the expression and purification methods (e.g., longer induction time or lower induction temperatures) may lead to significant improvements to overall soluble yields.

4.4 Text

Task description. We consider two text tasks: (i) toxic story generation using the Tinystories dataset (Eldan and Li, 2023), and (ii) product review generation using Amazon data (Hou et al., 2024). For both tasks, we start by fine-tuning a pre-trained MDM model (Sahoo et al., 2024) in a supervised fine-tuning manner on both datasets before running online fine-tuning. As reward models, we use RoBERTa (Liu, 2019) fine-tuned for toxicity classification, and BERT (Devlin, 2018), fine-tuned for Amazon review sentiment analysis, respectively. Our experiments aim to demonstrate our method's ability to induce behaviors that are uncommon in the base pre-trained model, specifically in generating toxic content in product reviews. Full experimental details are provided in Appendix §D.4.

Main results. In Table 4 we report the average log reward as well as perplexity (Gen PPL) of the generated samples as measured by GPT-2 (Radford et al., 2019). We find that DDPP-LB is the most effective variant of DDPP and achieves significantly higher log reward compared to SVDD and RTB for both tasks. We further observe that all methods achieve comparable Gen PPL suggesting

Algorithm ↓ Metric →	$\log R(\mathbf{x}_0) \uparrow$	BPD ↓
Base	-57.31 ± —	2.67 ± —
SVDD	-13.27 ± 12.38	—
Guidance (scale 1)	-91.10 ± 1.63	3.20 ± 0.01
Guidance (scale 5)	-62.75 ± 0.25	5.39 ± 0.01
Guidance (scale 100)	-41.51 ± 0.02	5.15 ± 0.00
RTB	-60.28 ± 1.74	**2.04 ± 0.00**
DDPP-LB (*ours*)	**-6.94 ± 1.39**	2.62 ± 0.15

Figure 3: **Left:** Results for discrete image modeling over raw pixel values on CelebA (64×64). We report the mean performance of DDPP and baselines separated into inference-based (top) and amortized (bottom) over 3 runs for the $\log R$ and class-BPD metrics. **Right:** Generated samples from Base, SVDD, RTB, and DDPP-LB.

that generated responses are fluent; however, samples from DDPP-LB adheres better to the task specification. We refer to §D.4.1 and §D.4.2 for generated samples from DDPP.

Table 4: Text experiments with log reward and Gen PPL results averaged over 3. As Best of 10 draws samples directly from $p_0^{\text{pre}}(\mathbf{x}_0)$ we instead bold the fine-tuning method whose Gen PPL is lowest.

Dataset →	Tinystories		Amazon reviews	
Algorithm ↓ Metric →	$\log R(\mathbf{x}_0) \uparrow$	Gen PPL ↓	$\log R(\mathbf{x}_0) \uparrow$	Gen PPL ↓
Best of 10*	93.25 ± 0.17	15.94 ± 0.03	-103.05 ± 0.25	124.45 ± 1.02
SVDD	146.95 ± 1.08	20.35 ± 0.03	-27.48 ± 10.91	165.86 ± 1.22
RTB	107.83 ± 3.08	**18.53 ± 0.55**	-35.22 ± 16.03	160.54 ± 12.19
DDPP-IS (*ours*)	163.45 ± 7.06	20.15 ± 0.30	105.16 ± 2.41	**152.85 ± 1.64**
DDPP-LB (*ours*)	**205.76 ± 3.88**	**19.60 ± 0.69**	**152.08 ± 34.01**	167.25 ± 27.33

5 Related Works

Discrete diffusion. The prevailing paradigms for diffusion over discrete spaces can be broadly categorized into 1.) continuous diffusion in a latent or reparametrized space by first transforming the initial discrete data (Li et al., 2022; Chen et al., 2022; Davis et al., 2024; Cheng et al., 2024), and 2.) defining diffusion using discrete analogs of score approximation (Meng et al., 2022; Lou et al., 2023). The latter approach can also be described using the theoretical framework of Continuous-time Markov Chains (CTMC) (Austin et al., 2021; Campbell et al., 2022; 2024). Closest to our setting we consider a specific instantiation of discrete diffusion that simplifies the CTMC framework by using a masked forward process (Sahoo et al., 2024; Shi et al., 2024; Zhao et al., 2024a; Gat et al., 2024).

Finetuning as sampling. The task of fine-tuning generative models under reward models can be viewed as a sampling problem and encompasses conventional RLHF (Uehara et al., 2024a; Black et al., 2023; Fan et al., 2024; Dong et al., 2023). A simple but expensive method to sample from the reward-induced Bayesian posterior distribution is best of N sampling (Stiennon et al., 2020), which provably samples from the correct distribution as the number of samples from the base pre-trained model grows, $N \to \infty$ (Beirami et al., 2024; Gao et al., 2023; Ferbach et al., 2024). Alternatively, the sampling perspective has been explored in the discrete setting to fine-tune autoregressive models (Zhao et al., 2024a; Hu et al., 2023), and diffusion models (Uehara et al., 2024b; Venkatraman et al., 2024; Zhao et al., 2024a). Finally, inference time techniques represent the most prominent approach to conditional sampling (Ho and Salimans, 2022; Dhariwal and Nichol, 2021; Li et al., 2024; Nisonoff et al., 2024).

6 Conclusion

In this paper, we present Discrete Denoising Posterior Prediction a novel framework to steer Masked Discrete Diffusion Models by viewing it as a problem of sampling from a Bayesian posterior. We introduced three concrete training strategies to instantiate our framework in DDPP-IS, DDPP-LB, and DDPP-KL and apply them to modeling synthetic data, pixel-level image modeling, fine-tuning protein MDMs to increase secondary structure diversity, and steering MDMs on language to match human sentiment. We find that DDPP not only is able to optimize an amortized sampler to closely match the reward-induced Bayesian posterior but it has a good agreement in other sample quality metrics—without severely compromising generated sample quality. An interesting direction for future work is to understand how to balance optimization of DDPP-LB and strategies to selecting γ.

7 Reproducibility Statement

We take the following steps to enhance the reproducibility of our work. In particular, all of our theoretical results include full proofs which are presented in §C. To assist in the reproducibility of our empirical findings we provide precise experimental details such as algorithmic descriptions of all variants of DDPP in Algorithm 1 and Algorithm 2. We further provide architectural choices, training details, and hyperparameters for all datasets and tasks in §D.

References

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. (Cited on page 16)

J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021. (Cited on pages 1 and 10)

A. Beirami, A. Agarwal, J. Berant, A. D'Amour, J. Eisenstein, C. Nagpal, and A. T. Suresh. Theoretical guarantees on the best-of-n alignment policy. *arXiv preprint arXiv:2401.01879*, 2024. (Cited on pages 7 and 10)

E. Bengio, M. Jain, M. Korablyov, D. Precup, and Y. Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021. (Cited on page 16)

Y. Bengio, S. Lahlou, T. Deleu, E. J. Hu, M. Tiwari, and E. Bengio. Gflownet foundations. *The Journal of Machine Learning Research*, 24(1):10006–10060, 2023. (Cited on pages 7 and 16)

K. Black, M. Janner, Y. Du, I. Kostrikov, and S. Levine. Training diffusion models with reinforcement learning. *arXiv preprint arXiv:2305.13301*, 2023. (Cited on page 10)

T. Brooks, B. Peebles, C. Holmes, W. DePue, Y. Guo, L. Jing, D. Schnurr, J. Taylor, T. Luhman, E. Luhman, C. Ng, R. Wang, and A. Ramesh. Video generation models as world simulators. 2024. URL https://openai.com/research/video-generation-models-as-world-simulators. (Cited on page 1)

A. Campbell, J. Benton, V. De Bortoli, T. Rainforth, G. Deligiannidis, and A. Doucet. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 35:28266–28279, 2022. (Cited on page 10)

A. Campbell, J. Yim, R. Barzilay, T. Rainforth, and T. Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. *arXiv preprint arXiv:2402.04997*, 2024. (Cited on page 10)

T. Chen, R. Zhang, and G. Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning. *arXiv preprint arXiv:2208.04202*, 2022. (Cited on page 10)

C. Cheng, J. Li, J. Peng, and G. Liu. Categorical flow matching on statistical manifolds. *arXiv preprint arXiv:2405.16441*, 2024. (Cited on page 10)

P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017. (Cited on page 1)

J. Dauparas, I. Anishchenko, N. Bennett, H. Bai, R. J. Ragotte, L. F. Milles, B. I. Wicky, A. Courbet, R. J. de Haas, N. Bethel, et al. Robust deep learning–based protein sequence design using proteinmpnn. *Science*, 378(6615):49–56, 2022. (Cited on page 1)

O. Davis, S. Kessler, M. Petrache, A. J. Bose, et al. Fisher flow matching for generative modeling over discrete data. *arXiv preprint arXiv:2405.14664*, 2024. (Cited on page 10)

V. De Bortoli, M. Hutchinson, P. Wirnsberger, and A. Doucet. Target score matching. *arXiv preprint arXiv:2402.08667*, 2024. (Cited on page 16)

K. Devkota, D. Shonai, J. Mao, S. H. Soderling, and R. Singh. Miniaturizing, modifying, and augmenting nature's proteins with raygun. *bioRxiv*, pages 2024–08, 2024. (Cited on pages 8 and 21)

J. Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. (Cited on page 9)

P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021. (Cited on pages 2 and 10)

H. Dong, W. Xiong, D. Goyal, Y. Zhang, W. Chow, R. Pan, S. Diao, J. Zhang, K. Shum, and T. Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. *arXiv preprint arXiv:2304.06767*, 2023. (Cited on page 10)

F. Eijkelboom, G. Bartosh, C. A. Naesseth, M. Welling, and J.-W. van de Meent. Variational flow matching for graph generation. *arXiv preprint arXiv:2406.04843*, 2024. (Cited on page 4)

R. Eldan and Y. Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023. (Cited on pages 9 and 29)

Y. Fan, O. Watkins, Y. Du, H. Liu, M. Ryu, C. Boutilier, P. Abbeel, M. Ghavamzadeh, K. Lee, and K. Lee. Reinforcement learning for fine-tuning text-to-image diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024. (Cited on page 10)

D. Ferbach, Q. Bertrand, A. J. Bose, and G. Gidel. Self-consuming generative models with curated data provably optimize human preferences. *arXiv preprint arXiv:2407.09499*, 2024. (Cited on page 10)

L. Gao, J. Schulman, and J. Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pages 10835–10866. PMLR, 2023. (Cited on page 10)

I. Gat, T. Remez, N. Shaul, F. Kreuk, R. T. Chen, G. Synnaeve, Y. Adi, and Y. Lipman. Discrete flow matching. *arXiv preprint arXiv:2407.15595*, 2024. (Cited on pages 1, 3, and 10)

T. Geffner, G. Papamakarios, and A. Mnih. Compositional score modeling for simulation-based inference. In *International Conference on Machine Learning*, pages 11098–11116. PMLR, 2023. (Cited on page 16)

B. Hie, S. Candido, Z. Lin, O. Kabeli, R. Rao, N. Smetanin, T. Sercu, and A. Rives. A high-level programming language for generative protein design. *bioRxiv*, pages 2022–12, 2022. (Cited on page 21)

J. Ho and T. Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022. (Cited on pages 2 and 10)

Y. Hou, J. Li, Z. He, A. Yan, X. Chen, and J. McAuley. Bridging language and items for retrieval and recommendation. *arXiv preprint arXiv:2403.03952*, 2024. (Cited on pages 9 and 28)

E. J. Hu, M. Jain, E. Elmoznino, Y. Kaddar, G. Lajoie, Y. Bengio, and N. Malkin. Amortizing intractable inference in large language models. *arXiv preprint arXiv:2310.04363*, 2023. (Cited on page 10)

M. Jiralerspong, J. Bose, I. Gemp, C. Qin, Y. Bachrach, and G. Gidel. Feature likelihood divergence: evaluating the generalization of generative models using samples. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. (Cited on page 8)

W. Kabsch and C. Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers: Original Research on Biomolecules*, 22(12): 2577–2637, 1983. (Cited on page 21)

D. P. Kingma, T. Salimans, B. Poole, and J. Ho. Variational diffusion models, 2023. URL https://arxiv.org/abs/2107.00630. (Cited on page 24)

N. Kluge Corrêa. *Dynamic Normativity*. PhD thesis, Universitäts-und Landesbibliothek Bonn, 2024. (Cited on page 26)

S. Lahlou, T. Deleu, P. Lemos, D. Zhang, A. Volokhova, A. Hernández-Garcıa, L. N. Ezzine, Y. Bengio, and N. Malkin. A theory of continuous generative flow networks. In *International Conference on Machine Learning*, pages 18269–18300. PMLR, 2023. (Cited on pages 7 and 16)

X. Li, J. Thickstun, I. Gulrajani, P. S. Liang, and T. B. Hashimoto. Diffusion-lm improves controllable text generation. *Advances in Neural Information Processing Systems*, 35:4328–4343, 2022. (Cited on page 10)

X. Li, Y. Zhao, C. Wang, G. Scalia, G. Eraslan, S. Nair, T. Biancalani, A. Regev, S. Levine, and M. Uehara. Derivative-free guidance in continuous and discrete diffusion models with soft value-based decoding. *arXiv preprint arXiv:2408.08252*, 2024. (Cited on pages 7 and 10)

Z. Lin, H. Akin, R. Rao, B. Hie, Z. Zhu, W. Lu, N. Smetanin, A. dos Santos Costa, M. Fazel-Zarandi, T. Sercu, S. Candido, et al. Language models of protein sequences at the scale of evolution enable accurate structure prediction. *bioRxiv*, 2022. (Cited on page 1)

L. Liu, C. Dong, X. Liu, B. Yu, and J. Gao. Bridging discrete and backpropagation: Straight-through and beyond. *Advances in Neural Information Processing Systems*, 36, 2024. (Cited on page 7)

Y. Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. (Cited on page 9)

Z. Liu, P. Luo, X. Wang, and X. Tang. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15(2018):11, 2018. (Cited on page 8)

A. Lou, C. Meng, and S. Ermon. Discrete diffusion language modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023. (Cited on page 10)

K. Madan, J. Rector-Brooks, M. Korablyov, E. Bengio, M. Jain, A. Nica, T. Bosc, Y. Bengio, and N. Malkin. Learning gflownets from partial episodes for improved convergence and stability. iclr'2023. *arXiv preprint arXiv:2209.12782*, 2022. (Cited on page 7)

N. Malkin, M. Jain, E. Bengio, C. Sun, and Y. Bengio. Trajectory balance: Improved credit assignment in gflownets. *Advances in Neural Information Processing Systems*, 35:5955–5967, 2022. (Cited on page 16)

C. Meng, K. Choi, J. Song, and S. Ermon. Concrete score matching: Generalized score matching for discrete data. *Advances in Neural Information Processing Systems*, 35:34532–34545, 2022. (Cited on pages 4 and 10)

Midjourney. https://www.midjourney.com/home/, 2023. Accessed: 2023-09-09. (Cited on page 1)

S. Mittal, N. L. Bracher, G. Lajoie, P. Jaini, and M. A. Brubaker. Exploring exchangeable dataset amortization for bayesian posterior inference. In *ICML 2023 Workshop on Structured Probabilistic Inference {\&} Generative Modeling*, 2023. (Cited on page 16)

H. Nisonoff, J. Xiong, S. Allenspach, and J. Listgarten. Unlocking guidance for discrete state-space diffusion and flow models. *arXiv preprint arXiv:2406.01572*, 2024. (Cited on pages 7, 10, 31, and 32)

M. Oquab, T. Darcet, T. Moutakanni, H. V. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. HAZIZA, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=a68SUt6zFt. (Cited on page 24)

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022. (Cited on page 3)

W. Peebles and S. Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023. (Cited on page 29)

E. Perez, S. Huang, F. Song, T. Cai, R. Ring, J. Aslanides, A. Glaese, N. McAleese, and G. Irving. Red teaming language models with language models. *arXiv preprint arXiv:2202.03286*, 2022. (Cited on page 1)

S. T. Radev, U. K. Mertens, A. Voss, L. Ardizzone, and U. Köthe. Bayesflow: Learning complex stochastic models with invertible neural networks. *IEEE transactions on neural networks and learning systems*, 33(4):1452–1466, 2020. (Cited on page 16)

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. (Cited on pages 1 and 9)

R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024. (Cited on pages 1 and 3)

S. S. Sahoo, M. Arriola, Y. Schiff, A. Gokaslan, E. Marroquin, J. T. Chiu, A. Rush, and V. Kuleshov. Simple and effective masked diffusion language models. *arXiv preprint arXiv:2406.07524*, 2024. (Cited on pages 1, 3, 9, 10, 20, and 24)

T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017. (Cited on page 1)

M. Sendera, M. Kim, S. Mittal, P. Lemos, L. Scimeca, J. Rector-Brooks, A. Adam, Y. Bengio, and N. Malkin. On diffusion models for amortized inference: Benchmarking and improving stochastic control and sampling. *arXiv preprint arXiv:2402.05098*, 2024. (Cited on page 16)

J. Shi, K. Han, Z. Wang, A. Doucet, and M. K. Titsias. Simplified and generalized masked diffusion for discrete data. *arXiv preprint arXiv:2406.04329*, 2024. (Cited on pages 1, 3, 10, and 24)

Stability AI. https://stability.ai/stablediffusion, 2023. Accessed: 2024-05-05. (Cited on page 1)

M. Steinegger and J. Söding. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology*, 35(11):1026–1028, 2017. (Cited on page 9)

N. Stiennon, L. Ouyang, J. Wu, D. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020. (Cited on pages 7 and 10)

M. Uehara, Y. Zhao, T. Biancalani, and S. Levine. Understanding reinforcement learning-based fine-tuning of diffusion models: A tutorial and review. *arXiv preprint arXiv:2407.13734*, 2024a. (Cited on pages 2, 3, and 10)

M. Uehara, Y. Zhao, K. Black, E. Hajiramezanali, G. Scalia, N. L. Diamant, A. M. Tseng, T. Biancalani, and S. Levine. Fine-tuning of continuous-time diffusion models as entropy-regularized control. *arXiv preprint arXiv:2402.15194*, 2024b. (Cited on page 10)

M. van Kempen, S. S. Kim, C. Tumescheit, M. Mirdita, C. L. Gilchrist, J. Söding, and M. Steinegger. Foldseek: fast and accurate protein structure search. *Biorxiv*, pages 2022–02, 2022. (Cited on page 9)

S. Venkatraman, M. Jain, L. Scimeca, M. Kim, M. Sendera, M. Hasan, L. Rowe, S. Mittal, P. Lemos, E. Bengio, et al. Amortizing intractable inference in diffusion models for vision, language, and control. *arXiv preprint arXiv:2405.20971*, 2024. (Cited on pages 4, 7, and 10)

R. Verkuil, O. Kabeli, Y. Du, B. I. Wicky, L. F. Milles, J. Dauparas, D. Baker, S. Ovchinnikov, T. Sercu, and A. Rives. Language models generalize beyond natural proteins. *BioRxiv*, pages 2022–12, 2022. (Cited on page 1)

C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard. Digress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022. (Cited on page 1)

R. Villegas, M. Babaeizadeh, P.-J. Kindermans, H. Moraldo, H. Zhang, M. T. Saffar, S. Castro, J. Kunze, and D. Erhan. Phenaki: Variable length video generation from open domain textual descriptions. In *International Conference on Learning Representations*, 2022. (Cited on page 1)

B. Wallace, M. Dang, R. Rafailov, L. Zhou, A. Lou, S. Purushwalkam, S. Ermon, C. Xiong, S. Joty, and N. Naik. Diffusion model alignment using direct preference optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8228–8238, 2024. (Cited on page 2)

X. Wang, Z. Zheng, F. Ye, D. Xue, S. Huang, and Q. Gu. Diffusion language models are versatile protein learners. *arXiv preprint arXiv:2402.18567*, 2024. (Cited on pages 1 and 8)

J. Wildberger, M. Dax, S. Buchholz, S. Green, J. H. Macke, and B. Schölkopf. Flow matching for scalable simulation-based inference. *Advances in Neural Information Processing Systems*, 36, 2024. (Cited on page 16)

Y. Zhang and J. Skolnick. Tm-align: a protein structure alignment algorithm based on the tm-score. *Nucleic acids research*, 33(7):2302–2309, 2005. (Cited on page 23)

L. Zhao, X. Ding, L. Yu, and L. Akoglu. Unified discrete diffusion for categorical data. *arXiv preprint arXiv:2402.03701*, 2024a. (Cited on pages 1, 3, and 10)

S. Zhao, R. Brekelmans, A. Makhzani, and R. Grosse. Probabilistic inference in language models via twisted sequential monte carlo. *arXiv preprint arXiv:2404.17546*, 2024b. (Cited on pages 2 and 29)

K. Zheng, Y. Chen, H. Mao, M.-Y. Liu, J. Zhu, and Q. Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. *arXiv preprint arXiv:2409.02908*, 2024. (Cited on page 4)

A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023. (Cited on page 1)

A Broader Impact

Our proposed Discrete Denoising Posterior Prediction is a tailored approach to steering and fine-tuning Masked Diffusion Models. At present, MDMs are an emergent category of discrete generative models that have general-purpose modeling capabilities in a variety of domains including language modeling, sequence-based drug design, and discrete modeling of graphs. Consequently, we believe DDPP has potential use in various practical use cases. For instance, like current RLHF techniques applied to modern autoregressive LLMs, future scaled MDMs on text datasets might be tuned to promote harmful behavior and toxic content. Moreover, applying Discrete Denoising Posterior Prediction in drug design use cases has the potential to create in-silico sample of protein sequences that may have biologically potent negative externalities. We do, however, make the distinction that such a risk is speculative at this stage given the large complexities of translating in-silico designs to actual synthesized biomolecules. As a result, we encourage practitioners who seek to fine-tune MDMs using DDPP to exercise due caution when applying our proposed techniques to actual use cases.

Ethical statement. As part of qualitatively evaluating DDPP, this paper includes generated samples of text. We highlight that the set of examples may contain potentially disturbing, harmful, or upsetting examples, covering a variety of sensitive topics like discriminatory language, descriptions of harm, and misinformation, among other high-risk categories. Its primary purpose is to advance research in understanding the impact of DDPP from a more interpretable lens. It is not advised to train future MDMs on such generated samples in order to prevent further propagation of undesirable content and behaviors.

B Additional Related Work

Sampling proportional to energy. Our approach can be closely linked to learning to sample proportional to a target probability, as in our setup we aim to approximate sampling proportional to the energy $p_t^{\text{pre}}(\cdot|\mathbf{x}_t)R(\cdot)$ for any point $\mathbf{x}_t$ at any time t. This has been an avenue of research for a number of works in continuous time (Bengio et al., 2021; 2023; Malkin et al., 2022; Lahlou et al., 2023; Akhound-Sadegh et al., 2024; Sendera et al., 2024; De Bortoli et al., 2024), in Bayesian posterior inference where the energy is defined by the product of likelihood and prior (Mittal et al., 2023), as well as posterior inference in settings where we even do not have access to energy function but only to a simulator (Radev et al., 2020; Wildberger et al., 2024; Geffner et al., 2023).

C Theoretical Results

C.1 Proof of Proposition 1

Before proving proposition 1 we first prove a useful Lemma that states the optimal log partition function $\log \hat{\mathcal{Z}}_{\pi_t}(\mathbf{x}_t)$ which is the learning goal for a parameterized approach $\log \hat{\mathcal{Z}}_{\pi_t,\theta}(\mathbf{x}_t)$.

Lemma 1. *Given a sample $\mathbf{x}_t \sim p_t(\mathbf{x}_t|\mathbf{x}_0)$ and the denoising posterior distribution $q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)$, a local minimizer for estimate for the log partition function $\log \hat{\mathcal{Z}}_{\pi_t}$ using N samples from $\mathbf{x}_0^i \sim q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)$ is given by:*

$$\log \mathcal{Z}^*_{\pi_t} = \frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{p_t(\mathbf{x}_0^i|\mathbf{x}_t)R(\mathbf{x}_0^i)}{q_{t,\theta}(\mathbf{x}_0^i|\mathbf{x}_t)}\right). \tag{14}$$

Proof. By definition the log partition function is a constant, let that constant be $\log \mathcal{Z}_{\pi_t}(\mathbf{x}_t) = C$. Then the loss in Eq. 11 is a quadratic in C,

$$\mathcal{L} = \mathbb{E}_{\mathbf{x}_0\sim r(\mathbf{x}_0)}\left[||\log q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t) + C - \log p_t(\mathbf{x}_0|\mathbf{x}_t) - \log R(\mathbf{x}_0)||_2^2\right] \tag{15}$$

For a batch of N samples of $\mathbf{x}_0^i \sim q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)$, we find a locally optimal constant C (local minima) by taking the gradient of Eq. 15 and setting it 0. In more detail we have,

$$0 = \nabla_C \frac{1}{N}\sum_i^N (\log q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t) + C - \log p_t(\mathbf{x}_0|\mathbf{x}_t) - \log R(\mathbf{x}_0))^2 \tag{16}$$

$$0 = \frac{2}{N}\sum_i^N \left(\log q_{t,\theta}(\mathbf{x}_0^i|\mathbf{x}_t) + C - \log p_t(\mathbf{x}_0^i|\mathbf{x}_t) - \log R(\mathbf{x}_0^i)\right) \tag{17}$$

$$0 = 2C + \frac{2}{N}\sum_i^N \log q_{t,\theta}(\mathbf{x}_0^i|\mathbf{x}_t) - \log p_t(\mathbf{x}_0^i|\mathbf{x}_t) - \log R(\mathbf{x}_0^i) \tag{18}$$

$$0 = C + \frac{1}{N}\sum_i^N \log\left(\frac{q_{t,\theta}(\mathbf{x}_0^i|\mathbf{x}_t)}{p_t(\mathbf{x}_0^i|\mathbf{x}_t)R(\mathbf{x}_0^i)}\right) \tag{19}$$

$$C = \frac{1}{N}\sum_i^N \log\left(\frac{p_t(\mathbf{x}_0^i|\mathbf{x}_t)R(\mathbf{x}_0^i)}{q_{t,\theta}(\mathbf{x}_0^i|\mathbf{x}_t)}\right). \tag{20}$$

□

Using Lemma 1 we now prove Proposition 1, stated again below for convenience.

Proposition 1. *Let* $\log \hat{\mathcal{Z}}^{IS}_{\pi_t}$ *and* $\log \hat{\mathcal{Z}}^{LB}_{\pi_t,\theta}$ *be the* M*-sample importance sampling estimate using the proposal* $q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)$ *and learned approximation to the log partition function respectively. Given a partially masked sample* $\mathbf{x}_t \sim p_t(\mathbf{x}_t)$ *the optimal learned approximation is a lower bound to the importance sampling estimate with a fixed proposal* $q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)$ *and the following inequality holds:*

$$\log \hat{\mathcal{Z}}^{LB}_{\pi_t,\theta}(\mathbf{x}_t) \leq \log \hat{\mathcal{Z}}^{IS}_{\pi_t}(\mathbf{x}_t). \tag{9}$$

Proof. We optimize $\log \hat{\mathcal{Z}}^{\text{LB}}_{\pi_t,\theta}(\mathbf{x}_t)$ using the loss defined in Eq. 11. Using Lemma 1 we know the analytic expression for the locally optimal estimate is given by $\log \mathcal{Z}^*_{\pi_t}(\mathbf{x}_t)$. Plugging this into the definition of the log partition function we get,

$$\log \hat{\mathcal{Z}}^{\text{LB}}_{\pi_t,\theta}(\mathbf{x}_t) = \mathbb{E}_{\mathbf{x}_0\sim q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)}\left[\log\left(\frac{p_t(\mathbf{x}_0|\mathbf{x}_t)R(\mathbf{x}_0)}{q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)}\right)\right] \tag{21}$$

$$\leq \log \mathbb{E}_{q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)}\left[\frac{p_t(\mathbf{x}_0|\mathbf{x}_t)R(\mathbf{x}_0)}{q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)}\right] \tag{22}$$

$$= \log \hat{\mathcal{Z}}^{\text{IS}}_{\pi_t}(\mathbf{x}_t) \tag{23}$$

The lower bound turns into equality at the optimal proposal $q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t) \propto p_t(\mathbf{x}_0|\mathbf{x}_t)R(\mathbf{x}_0)$.

□

C.2 Estimating DDPP-KL with Reinmax

We first provide an algorithmic description below of training using DDPP-KL. We first highlight how the reverse KL objective can be applied to a more general setting beyond just fine-tuning before turning to the exact setting of the main paper.

Algorithm 2 DDPP-KL

Input: Differentiable reward $R(\mathbf{x}_0)$, base MDM $p_0^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t)$, fine-tuning MDM $q_\theta(\mathbf{x}_0|\mathbf{x}_t)$, Num samples K

1: **while** Training **do**
2: $\quad t, \mathbf{x}_0 \sim \mathcal{U}[0,1], q(\mathbf{x}_0)$ ▷ *Sample time and clean data on-policy from the fine-tuning MDM*
3: $\quad \mathbf{x}_t \sim p_t(\mathbf{x}_t|\mathbf{x}_0)$ ▷ *Construct a partially masked sample given clean data*
4: $\quad \{\hat{\mathbf{x}}_0^i\}_{i=0}^K \sim q_{t,\theta}(\cdot|\mathbf{x}_t)$ ▷ *Reparametrized Sampling of clean data*
5: $\quad \mathcal{L}^{\text{KL}} = \frac{1}{K}\sum_{i=1}^K \left(\log q_{t,\theta}(\hat{\mathbf{x}}_0^i|\mathbf{x}_t) - \log p_0^{\text{pre}}(\hat{\mathbf{x}}_0^i|\mathbf{x}_t) - \log R(\hat{\mathbf{x}}_0^i)\right)$
6: $\quad \nabla_\theta \mathcal{L}^{\text{KL}} := \nabla_\theta^{\text{Reinmax}}\left(\mathcal{L}^{\text{KL}}\right)$ ▷ *Use the Reinmax discrete gradient estimator*
7: $\quad \theta \leftarrow \text{Update}(\theta, \nabla_\theta \mathcal{L}^{\text{KL}})$
8: **Return** q_θ

Non-finetuning Case. In this appendix, we study the REINMAX gradient estimator for the general problem of sampling from the following distribution:

$$\pi_0(\mathbf{x}_0) \propto \frac{R(\mathbf{x}_0)}{\mathcal{Z}}. \tag{24}$$

Gradient of $\mathcal{L}^{\text{KL}}$. We can decompose the gradient into the following terms due to the linearity of expectations:

$$\begin{aligned}\mathcal{L}_t^{\text{KL}} &= \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[\log q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)\right] - \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[\log \pi_t(\mathbf{x}_0|\mathbf{x}_t)\right] \\ &= \mathcal{L}_t^1 + \mathcal{L}_t^2. \end{aligned} \tag{25}$$

We again highlight the fact that the expectation is taken using the following distributions $t, \mathbf{x}_0, \mathbf{x}_t \sim \mathcal{U}[0,1], q(\mathbf{x}_0), p_t(\mathbf{x}_t|\mathbf{x}_0)$. As a result, $\mathbf{x}_0$ is drawn on-policy and is a stochastic variable that needs gradient estimation since q_θ is the parameterized distribution. Furthermore, all terms that use *this sample* $\mathbf{x}_0$ inside the expectation are affected by this gradient computation.

Taking the gradient of each term respectively. The gradient of of $\mathcal{L}_t^1$ is:

$$\begin{aligned}\nabla_\theta \mathcal{L}_t^1 &= \nabla_\theta \left(\mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[\log q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t)\right]\right) \\ &\approx \mathbb{E}_{t,\mathbf{x}_0\sim q_\theta(\mathbf{x}_0),\mathbf{x}_t\sim p_t(\mathbf{x}_t|\mathbf{x}_0)}\left[\nabla^{\text{Rein-Max}} \circ (\log q_{t,\theta}(\mathbf{x}_0|\mathbf{x}_t))\right]. \end{aligned} \tag{26}$$

The gradient of of $\mathcal{L}_t^2$ is:

$$\begin{aligned}\nabla_\theta \mathcal{L}_t^2 &= \nabla_\theta \left(\mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[\log \pi_t(\mathbf{x}_0|\mathbf{x}_t)\right]\right) \\ &= \nabla_\theta \left(\mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[-\log p_t(\mathbf{x}_t|\mathbf{x}_0) - \log \pi_0(\mathbf{x}_0) + \log \pi_t(\mathbf{x}_t)\right]\right) \\ &= \nabla_\theta \left(\mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[-\log p_t(\mathbf{x}_t|\mathbf{x}_0) - \log R(\mathbf{x}_0) + \log\left(\sum_{\mathbf{x}_0'} \pi_t(\mathbf{x}_t|\mathbf{x}_0')R(\mathbf{x}_0')\right)\right]\right). \end{aligned} \tag{27}$$

To use the Reinmax gradient estimator we must compute $\partial f(z)/\partial z$, where f is the function inside the expectation $\mathbb{E}_z[f(z)]$. We now make use of the following facts:

(F1) Analytic expression of $\nabla_{x_0} \log p_t(x_t|x_0)$. For simplicity of presentation, we focus on a single token x_0^i in a sequence but the result remains true for the entire sequence $\mathbf{x}_0$. Recall in the discrete setting of masked diffusion models $p_t = \text{Cat}(x_0; \bar{Q}_t x_t)$, which allows us to write:

$$\nabla_{x_0^i} \log p_t(x_t^i|x_0^i) = \frac{\nabla_{x_0^i} p_t(x_t^i|x_0^i)}{p_t(x_t^i|x_0^i)} \tag{28}$$

$$= \frac{\nabla_{x_0^i} \text{Cat}(x_0^i; \bar{Q}_t x_t^i)}{\text{Cat}(x_0^i; \bar{Q}_t x_t^i)} \tag{29}$$

$$= \frac{\nabla_{x_0^i} (x_0^{i,T} \bar{Q}_t x_t^i)}{x_0^{i,T} \bar{Q}_t x_t^i} \tag{30}$$

$$= \frac{\nabla_{x_0^i} (\alpha_t \langle x_t^i, x_0^i\rangle + (1-\alpha_t)\langle x_t^i, e_m\rangle)}{\alpha_t \langle x_t^i, x_0^i\rangle + (1-\alpha_t)\langle x_t^i, e_m\rangle} \tag{31}$$

$$= \frac{\alpha_t x_t^i}{\alpha_t \langle x_t^i, x_0^i\rangle + (1-\alpha_t)\langle x_t^i, e_m\rangle}. \tag{32}$$

(F2) Differentiability of the reward $\nabla_{\mathbf{x}_0} R(\mathbf{x}_0)$. If we assume the reward is differentiable we can exploit the same trick to write:

$$\nabla_{\mathbf{x}_0} \log R(\mathbf{x}_0) = \frac{\nabla_{\mathbf{x}_0} R(\mathbf{x}_0)}{R(\mathbf{x}_0)}. \tag{33}$$

Note that the final term in Eq. 27 does not depend on the realization of the sample $\mathbf{x}_0 \sim q(\mathbf{x}_0|\mathbf{x}_t)$ and thus its gradient in Rein-max is 0. This enables us to write the approximate gradient as:

$$\begin{aligned}\nabla_\theta \mathcal{L}_t^2 &\approx \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}[\nabla^{\text{Reinmax}} \circ (-\log p_t(\mathbf{x}_t|\mathbf{x}_0) - \log R(\mathbf{x}_0))] \\ &= \mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[\left(-\sum_i^N \frac{\alpha_t x_t^i}{\alpha_t \langle x_t^i, x_0^i\rangle + (1-\alpha_t)\langle x_t^i, e_m\rangle} - \frac{\nabla_{\mathbf{x}_0}^{\text{Reinmax}} R(\mathbf{x}_0)}{R(\mathbf{x}_0)}\right)\right]. \end{aligned} \tag{34}$$

The first term in the equation has a closed-form expression for the gradient but is still a stochastic gradient since it depends on $\mathbf{x}_0 \sim q_\theta(\mathbf{x}_0)$.

Finetuning Case. In the fine-tuning setting we aim to sample from the following Bayesian posterior:

$$\pi_0(\mathbf{x}_0) \propto \frac{p_0^{\text{pre}}(\mathbf{x}_0)R(\mathbf{x}_0)}{\mathcal{Z}} \tag{35}$$

For MDMs the likelihood under the model $p_0^{\text{pre}}(\mathbf{x}_0)$ is intractable to evaluate and leads to a modified objective for gradient estimation with REINMAX in $\mathcal{L}_t^{\text{KL}}$ in Eq. 25:

$$\begin{aligned}\nabla_\theta \mathcal{L}_t^2 &= \nabla_\theta \left(\mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[\log \pi_t(\mathbf{x}_0|\mathbf{x}_t)\right]\right)\\ &= \nabla_\theta \left(\mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[-\log p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t) - \log R(\mathbf{x}_0) + \log \mathcal{Z}_{\pi_t}(\mathbf{x}_t)\right]\right)\\ &= \nabla_\theta \left(\mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[-\log p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t) - \log R(\mathbf{x}_0) + \log\left(\mathbb{E}_{\mathbf{x}_0'\sim p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t)}[R(\mathbf{x}_0')]\right)\right]\right).\end{aligned} \tag{36}$$

Note that in the equation above we can evaluate the log partition function using samples drawn from the denoising posterior of the pre-trained model $\mathbf{x}_0' \sim p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t)$ and *not* the on-policy samples $\mathbf{x}_0 \sim q_\theta(\mathbf{x}_0)$. Thus this term is a constant when we compute the gradient. Thus we have,

$$\nabla_\theta \mathcal{L}_t^2 \approx \nabla^{\text{Reinmax}} \circ \left(\mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[-\log p_t^{\text{pre}}(\mathbf{x}_0|\mathbf{x}_t) - \log R(\mathbf{x}_0)\right]\right). \tag{37}$$

C.3 EQUIVALENCE OF SUB-TRAJECTORY OBJECTIVES

In this appendix, we detail how to compute an efficient approximation of the loss function that is inspired by the KL divergence between sub-trajectories as found in the GFlowNet literature but adapted for MDMs.

Consider the trajectory of a sequence: $\tau(\mathbf{x}_{0:1}) := \mathbf{x}_1 \to \cdots \to \mathbf{x}_t \to \mathbf{x}_{t-1} \to \ldots \mathbf{x}_0$. We seek to minimize the joint distribution over the (sub)-trajectories conditioned on a partially masked sample $\mathbf{x}_t$:

$$q_\theta(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t, \mu_\theta(\mathbf{x}_t, t))p_t(\mathbf{x}_t) = \pi_t(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t)p(\mathbf{x}_t). \tag{38}$$

Here $\pi_t(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is defined as,

$$\pi_t(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t, \mu_\theta(\mathbf{x}_t, t)) = \frac{p_t^{\text{pre}}(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t)R(\mathbf{x}_0)}{\mathcal{Z}_{\pi_t}(\mathbf{x}_t)} \tag{39}$$

$$= \frac{\prod_{j=1}^t p_t^{\text{pre}}(\mathbf{x}_{j-1}|\mathbf{x}_j, \hat{\mathbf{x}}_0^{\text{pre}})R(\mathbf{x}_0)}{\mathcal{Z}_{\pi_t}(\mathbf{x}_t)} \tag{40}$$

We minimize the following KL divergence,

$$\mathbb{D}_{\text{KL}}(q_\theta(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t, \hat{\mathbf{x}}_0)p_t(\mathbf{x}_t)||\pi_t(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t)p(\mathbf{x}_t)). \tag{41}$$

Here we used the convention that $\hat{\mathbf{x}}_0 = \mu_\theta(\mathbf{x}_t, t)$ and $\hat{\mathbf{x}}_0^{\text{pre}} = \mu^{\text{pre}}(\mathbf{x}_t, t)$. The KL between path measures along the sub-trajectory shares the same optimum as the following loss objective:

$$\begin{aligned}\mathcal{L}_\tau &= \mathbb{E}_{t,\mathbf{x}_t}\left[\mathbb{E}_{\tau(\mathbf{x}_{0:t})}[\|\log q_\theta(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t, \hat{\mathbf{x}}_0)) - \log p_t^{\text{pre}}(\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}|\mathbf{x}_t) + \kappa\|_2^2]\right]\\ &= \mathbb{E}_{t,\mathbf{x}_t}\left[\mathbb{E}_{\tau(\mathbf{x}_{0:t})}\left[\left\|\sum_{j=1}^t \log q_\theta(\mathbf{x}_{j-1}|\mathbf{x}_j, \hat{\mathbf{x}}_0) - \log p_t^{\text{pre}}(\mathbf{x}_{j-1}, |\mathbf{x}_j, \hat{\mathbf{x}}_0^{\text{pre}}) + \kappa\right\|_2^2\right]\right] \qquad (42)\\ &= \mathbb{E}_{t,\mathbf{x}_t,\tau(\mathbf{x}_{0:t})}\left[\left\|\sum_{s=1}^t \log q_\theta(\mathbf{x}_{s-\gamma}|\mathbf{x}_s, \hat{\mathbf{x}}_0) - \log p_t^{\text{pre}}(\mathbf{x}_{s-\gamma}, |\mathbf{x}_s, \hat{\mathbf{x}}_0^{\text{pre}}) + \kappa\right\|_2^2\right]\\ &= \mathbb{E}_{t,\mathbf{x}_t,\tau(\mathbf{x}_{0:t})}\left[\left\|t\mathbb{E}_{s,\mathbf{x}_s,\mathbf{x}_{s-\gamma}}\left[\log q_\theta(\mathbf{x}_{s-\gamma}|\mathbf{x}_s, \hat{\mathbf{x}}_0) - \log p_t^{\text{pre}}(\mathbf{x}_{s-\gamma}, |\mathbf{x}_s, \hat{\mathbf{x}}_0^{\text{pre}}) + \kappa\right]\right\|_2^2\right]. \qquad (43)\end{aligned}$$

In the last equation we define the constant $\kappa = (-\log R(\mathbf{x}_0) + \log \mathcal{Z}_{\pi_t}(\mathbf{x}_t))/t$ and use the fact that our notation convention uses $p(X = x) = p(x)$ for discrete random variables. Now we make the observation that for any $s < t$ we have effectively picked an endpoint over the trajectory. More precisely, $s \sim \mathcal{U}[0, t]$, which also allows us to sample $\mathbf{x}_s \sim p_t(\mathbf{x}_s|\mathbf{x}_0)$, in an analogous manner to how $\mathbf{x}_t$ is constructed.

D ADDITIONAL EXPERIMENTAL DETAILS

All experiments were performed on a shared heterogenous high-performance computing cluster. This cluster is primarily composed of GPU nodes with RTX8000, V100, A100, L40S, and H100 NVIDIA GPUs. We briefly note a trick used across a number of our experiments for DDPP-LB described as warming up $\log Z_t(x_t)$. We found that early iterations of training DDPP-LB could be somewhat unstable as the parameterized normalizing constant was not calibrated to a proper range given the pre-trained model and reward function. As such, we found that warming up $\log Z_t(x_t)$ for some number of steps at the beginning of training by only allowing gradient flow through the $\log Z_t(x_t)$ term helped stabilize training and improve overall performance. For the runs on which warming up $\log Z_t(x_t)$ was utilized, we resume normal training (i.e., allowing gradient flow through the fine-tuned denoiser *and* $\log Z_t(x_t)$) after the warmup period has concluded. For all experiments with DDPP-LB we used another separate, small DiT to parameterize the $\log Z_t(x_t)$ prediction.

D.1 SYNTHETIC EXPERIMENTS

Two synthetic tasks were performed: (1) sampling from a posterior over a 2 dimensional grid, and (2) fine-tuning on binarized MNIST. In both cases a 90 million parameter MDM model was trained on samples from the prior distribution, with the same DiT architecture as in Sahoo et al. (2024).

D.1.1 GRID EXPERIMENT

The space consists of discrete tokens $\mathbf{x}_0 \in \{0, \dots, 127\}^2$. A prior density p_0^{pre} is defined over this space which assigns a uniform probability for tokens falling inside one of the 16 evenly spaced squares, and a near-zero probability outside this. This prior distribution is depicted in Figure 6(a). Pre-training was done using the Adam optimizer, with $\beta_1, \beta_2 = \{0.9, 0.999\}$, and a learning rate of 3e-4.

The reward function $R(\mathbf{x}_0) = 0$ for $x_0^1 < 64$, and $R(\mathbf{x}_0) = 1$ for $x_0^1 \geq 64$. This results in a fine-tuning target $\propto R(\mathbf{x}_0)p^{\text{pre}}(\mathbf{x}_0)$ which selects out only the squares in the lower half of the grid. This product distribution is visualized in Figure 6(b).

For fine-tuning we train the model using our loss-functions with the Adam optimizer, using a learning rate of $4e-3$, $\beta_1, \beta_2 = \{0.9, 0.999\}$, and a weight decay of 0 across all methods. DDPP-IS used 16 samples to estimate the partition function. Training is done using a replay buffer populated with points $\mathbf{x}_0$ sampled on policy from the model, as well as off-policy points from the prior distribution, added to the buffer every 100 training steps. A batch of 64 is used.

D.1.2 MNIST

This task consisted of generating binarized MNIST digits $\mathbf{x}_0 \in \{0, 1\}^{28\times 28}$. The prior $p^{\text{pre}}(\mathbf{x}_0)$ in this case is the MNIST data distribution. For pre-training, the Adam optimizer is used with a learning rate of $4e-3$, $\beta_1, \beta_2 = \{0.9, 0.999\}$ and a weight decay of 0.

This MDM is fine-tuned to produce even digits. More precisely, the reward function is $R(\mathbf{x}_0) = p(\text{Even} \mid \mathbf{x}_0)^\beta = \left(\sum_{i=0,2,4,6,8} p(y = i \mid \mathbf{x}_0)\right)^\beta$, with $p(y = i \mid \mathbf{x}_0)$ being obtained from a pretrained MNIST classifier (LeNet 5 in this case). The inverse-temperature β is set to 5 for all experiments.

For fine-tuning with our methods, we use Adam with a learning rate of $1e-5$ and $\beta_1, \beta_2 = \{0.9, 0.999\}$. Training is done with a batch-size of 64. Samples are drawn from a replay-buffer populated with only on-policy samples. Method specific hyperparameters include:

- DDPP-IS: the importance sampling estimate is done with 16 samples
- DDPP-LB: a learning rate of $1e-3$ is used for network layers estimating $\log \mathcal{Z}_{\pi_t}$
- DDPP-KL: The KL objective per $\mathbf{x}_t$ is computed using 8 samples

RTB is trained with a learning rate of $5e-5$, with weight decay 0.01, on trajectories of length 32 with a batch size of 8. For training, 30% of the steps are detached. The smaller batch-size is chosen to fit the training on 80GB of GPU memory.

SVDD uses 10 particles in each inference step.

For all methods (including baselines), inference is done with 128 steps.

Additional information on computation of metrics is included in D.3.1.

D.1.3 MNIST SAMPLES

Samples from our methods, as well as the pretrained model, are shown in Figure 4.

D.2 PROTEIN SEQUENCES

Protein design involves the creation of novel protein sequences that adopt specific structures and perform desired functions. This is a critical field in synthetic biology and biotechnology, as it enables the rational engineering of proteins with enhanced stability, novel functionalities, or improved therapeutic properties. Advances in machine learning-based models, such as protein language models (pLMs), have enabled rapid exploration of protein sequence space, making de novo protein design more feasible and versatile. However, current pLMs struggle in generating realistic sequences which satisfy certain criteria, and we study using DDPP to finetune DPLM to generate high-scoring proteins given a reward function.

D.2.1 IN-SILICO TASKS

In task 1, we fine-tune the DPLM model to generate designable protein sequences that optimize for several critical features, including high predicted template modeling (pTM) and predicted local distance difference test (pLDDT) scores from ESMFold, reduced exposed hydrophobic residues, high sequence entropy, and an increased proportion of β-sheet content (Hie et al., 2022). These optimizations are captured in the reward function R, given by:

$$\begin{aligned}\log R = w_{\text{pTM}} \cdot \text{pTM} + w_{\text{pLDDT}} \cdot \text{pLDDT} + w_{\text{Sheet}} \cdot \text{Sheet\%} \\ + w_{\text{Entropy}} \cdot H(\mathbf{s}) - w_{\text{Hpho}} \cdot \text{Exposed_Hpho\%}\end{aligned}$$

Where the terms represent:

- pTM and pLDDT: Structural confidence scores from ESMFold, measuring global and local accuracy, respectively.
- Sheet%: The proportion of residues predicted to form β-sheets, determined by DSSP (Kabsch and Sander, 1983).
- $H(\mathbf{s})$: Sequence entropy, defined as:

  $$H(\mathbf{s}) = -\sum_{i=1}^{L} \sum_{a} p_i(a) \log p_i(a),$$

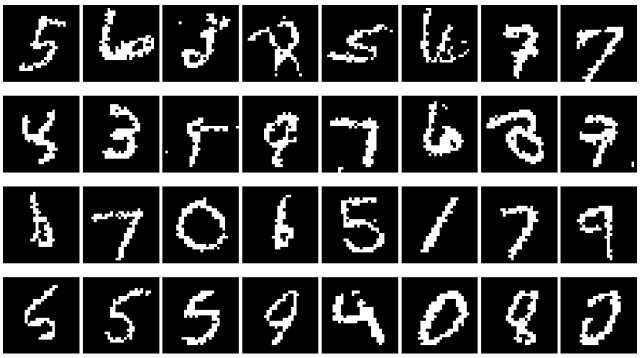
  where L is the length of the sequence and $p_i(a)$ is the probability of amino acid a at position i.
- Exposed_Hpho%: Percentage of hydrophobic residues exposed on the surface, calculated based on solvent-accessible surface area.

The weights for these features are set as follows:

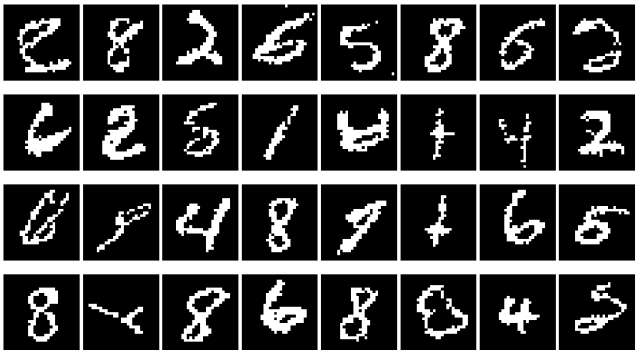
$$w_{\text{pTM}} = 1, \quad w_{\text{pLDDT}} = 1, \quad w_{\text{Sheet}} = 4.5, \quad w_{\text{Entropy}} = 0.8, \quad w_{\text{Hpho}} = 0.25.$$

As the scale of the various reward terms are non-uniform we selected the reward weights to weight all rewards similarly besides the sheet percent reward which is weighted higher. For the β-sheet task we found that both RTB and DDPP faced issues with mode collapse. After investigating the protein structures generated by base DPLM we found that the base model is only capable of generating a small number of motifs (in particular, over 2k samples from the base model we found only two motifs with $\log R(x_0) \geq 3.5$), implying that the targeted product distribution indeed collapses around these structural motifs as we observe in the case of RTB and DDPP. As such, we conclude that DDPP (and RTB) achieve the goal of fine-tuning as they sample from the product distribution and reproduces samples with β-sheets at a much higher proportion than the base model.

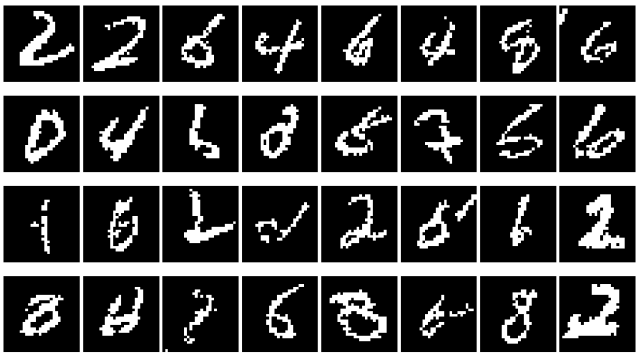
In task 2, we focus on generating shorter sequences of known proteins that preserve essential structural characteristics, using the TM-align score as the reward function (Devkota et al., 2024). This task allows the exploration of mutational effects. Ribonuclease proteins (PDB IDs: 9RAT-A, 11BA-A) are selected for this task due to their well-characterized structure, function, and folding mechanisms.



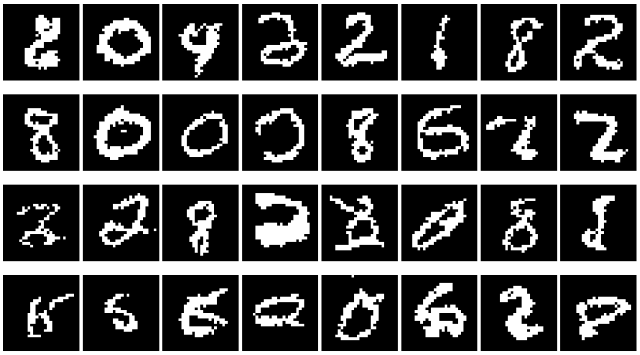
(a) Pretrained model



(b) DDPP-IS



(c) DDPP-LB



(d) DDPP-KL

Figure 4: Uncurated samples from the pretrained model, and after fine-tuning with our methods: DDPP-IS, DDPP-LB, DDPP-KL

The reward function R is defined as:

$$R = w_{\text{tm_score}} \cdot \text{TM-align}(\mathbf{s}, \mathbf{t}).$$

Where:

- $w_{\text{tm_score}}$: The weight of the TM-Score reward, set to 2.
- $\mathbf{s}$: Predicted structure from ESMFold of the generated sequence.
- $\mathbf{t}$: Target protein structure.
- TM-align: A measure of structural similarity between $\mathbf{s}$ and $\mathbf{t}$, defined as:

$$\text{TM-align} = \max \left(\frac{1}{L_t} \sum_{i=1}^{L_{\text{ali}}} \frac{1}{1 + \left(\frac{d_i}{d_0}\right)^2} \right).$$

  where L_t is the length of the target protein, L_{ali} is the length of the aligned region, d_i is the distance between the i-th pair of aligned residues, and d_0 is the distance scale based on L_t (Zhang and Skolnick, 2005).

While not used in the reward function for either experimental setting, we also measure the KL divergence, reported as KL-SS in Table 3 between the secondary structure distribution given by DSSP for both the target and miniaturized protein.

Note that in these experiments, the number of recycles in ESMFold is set to 0 to reduce computational overhead. For both tasks we generate amino acid sequences of length 90. Evaluation is performed by sampling 200 proteins for each method across three seeds and reporting the mean and standard deviation of each metric accordingly. All methods ran 500 inference steps during evaluation. All protein experiments used a 150 million parameter DPLM base model[1] to begin fine-tuning from. All models used a log-linear noise schedule with $\sigma_{min} = 1\text{e}{-4}$ and $\sigma_{max} = 20$ and used a linear learning rate warmup period of 2500 training steps.

DDPP was trained with no warmup period for $\log Z_t(x_t)$, a learning rate of $1\text{e}{-5}$, a batch size of 16, a replay buffer of max length 10,000, and inserting new batches to the buffer sampled on-policy from the current model every 250 training steps. RTB was trained similarly, but with a smaller batch size to account for its greater memory requirements. RTB matches the setting of DDPP but with a batch size of 8 while doing 90 inference steps during training (a new batch of trajectories is simulated on-policy every training step). To allow RTB to fit in memory we detached 65% of trajectory timesteps when computing a backward pass on the RTB objective. SVDD was run on the base DPLM model with $n = 10$ particles. To control the concentration of our designated target distributions, we set the reward temperature $\beta = 0.125$ for the β-sheet task and $\beta = 0.001$ for the protein miniaturization task.

We report an extended version of Table 3 where we include results for both ribonuclease targets in Table 5. We observe that DDPP consistently achieves the highest TM-Score across the two templates while maintaining high structural quality with an average pLDDT of around 0.8.

D.2.2 Experimental validation

Genes encoding for de novo protein sequences were obtained from Integrated DNA Technologies (IDT) and cloned into pET-24a(+) (Novagen) expression vectors with a C-terminal 6xHis tag using Gibson Assembly (New England Biolabs, NEB). Assembled plasmids were verified via Sanger sequencing, then transformed into chemically competent *Escherichia coli* BL21(DE3) cells (NEB). Starter cultures (3 mL Luria Bertani media, 50 µg/mL kanamycin) were inoculated from freshly prepared agar plates and grown at 37°C and shaken at 225 RPM overnight. Starter cultures were then diluted 1:100 into 50 mL LB medium supplemented with antibiotic. Cultures were then grown at 37°C and 225 RPM until an optical density (OD600) of 0.5-0.7 was reached. Protein expression was then induced with 1 mM isopropyl β-D-thiogalactopyranoside (IPTG) for 4 hours at 37°C. Cells were then collected by centrifugation (4,500xg) at 4°C and resuspended in lysis buffer (Tris-buffered saline (TBS), 25 mM imidazole). Cell suspensions were then lysed via sonication (10s pulses, 40% amplitude). The corresponding lysate was centrifuged at 12,000xg for 30 minutes, and the

[1] https://huggingface.co/airkingbd/dplm_150m

Table 5: Miniaturizing ribonuclease proteins 9RAT-A and 11BA-A (124 AAs) to 90 AAs while preserving structural fidelity (high TM-Score) and quality (high pLDDT and PTM).

Template		SS-KL ↓	$\log R(\mathbf{x}_0)$ ↑	TM-Score ↑	pLDDT ↑	pTM ↑
9RAT-A	Base Model	2.944 ± 2.936	0.502 ± 0.128	0.251 ± 0.064	0.724 ± 0.144	0.584 ± 0.226
	Best-of-10	$\mathbf{0.640} \pm \mathbf{1.872}$	0.725 ± 0.098	0.363 ± 0.049	0.789 ± 0.018	0.754 ± 0.086
	DDPP	1.086 ± 2.242	$\mathbf{0.735} \pm \mathbf{0.122}$	$\mathbf{0.368} \pm \mathbf{0.061}$	0.793 ± 0.044	$\mathbf{0.768} \pm \mathbf{0.066}$
	RTB	1.808 ± 2.597	0.597 ± 0.109	0.299 ± 0.055	$\mathbf{0.796} \pm \mathbf{0.054}$	0.750 ± 0.084
	SVDD	3.465 ± 2.835	0.699 ± 0.079	0.350 ± 0.039	0.499 ± 0.137	0.383 ± 0.178
11BA-A	Base Model	3.136 ± 3.150	0.478 ± 0.101	0.239 ± 0.051	0.724 ± 0.144	0.584 ± 0.226
	Best-of-10	2.602 ± 3.309	0.654 ± 0.089	0.327 ± 0.045	0.782 ± 0.027	0.720 ± 0.109
	DDPP	$\mathbf{0.194} \pm \mathbf{1.009}$	$\mathbf{0.709} \pm \mathbf{0.048}$	$\mathbf{0.354} \pm \mathbf{0.024}$	0.743 ± 0.036	0.727 ± 0.054
	RTB	2.579 ± 2.799	0.564 ± 0.111	0.282 ± 0.056	$\mathbf{0.797} \pm \mathbf{0.058}$	$\mathbf{0.744} \pm \mathbf{0.101}$
	SVDD	3.240 ± 2.992	0.647 ± 0.079	0.324 ± 0.040	0.486 ± 0.124	0.354 ± 0.162

supernatant was loaded into a HisPur Ni-NTA His-spin column (ThermoScientific) and purified as recommended. Expression of purified proteins in both the soluble and insoluble fraction, as well as his-tag purification fractions, was assessed using SDS-polyacrylamide gel electrophoresis.

D.3 Discrete image modelling

To setup the finetuning task we first pre-train large masked diffusion models on the original dataset. This uses a standard masked diffusion loss as explored in previous work (Shi et al., 2024; Sahoo et al., 2024).

CelebA Pretraining. We train a 241 million parameter model based on the variational diffusion model (VDM) architecture (Kingma et al., 2023) and the setup of Shi et al. (2024). We adapted the U-Net plus self-attention architectures from Kingma et al. (2023) as used in CIFAR-10 in their experiments, with a few notable additions. We replace the Fourier feature inputs with an input embedding layer which embeds 257 (256 pixel values + <MASK>) tokens into the embedding dimension. We double the number of residual blocks from 32 to 64 per encoder / decoder, and double the embedding dimension from 128 to 256. We use an Adam optimizer with learning rate $1e-3$, β_1=0.9 and β_2=0.999. We train our model for 450k steps with batch size 128 on a cluster of 16 NVIDIA L40S GPUs. We resize all CelebA images to 64x64 with bilinear interpolation. Samples from this model can be seen in Figure 5.

Separately, we train a 7M parameter classifier to classify hair color on CelebA. We use this as our energy function with a temperature setting of 0.1 for all finetuning experiments.

CelebA Finetuning. With the problem setup, we next finetune our pretrained model to sample images with blond hair. We train each model for up to 12 A100 hours. We use an early stopping criteria based on a validation set using an approximate bits-per-dimension calculation using the ELBO. We find that the original needs at least $1\,000$ inference steps for good performance therefore we evaluate all models in this setting. For our model we use $1\,000$ warmup steps for $\log Z$, a learning rate of $1e-4$, we resample two batches every 500 gradient steps of the model and add them to the replay buffer.

In contrast to our model, RTB requires a full trajectory for each gradient step. For CelebA, this means a rollout of $1\,000$ inference steps taking approximately 2 minutes for a batch size of 2 on an A100 with this model. Because of memory constraints we detach 99% of inference steps and use a batch size of 2 to fit in 80GB of GPU memory with a global batch size of 8 trajectories per gradient step.

D.3.1 Metrics

The metrics used to evaluate image fine-tuning include mean log reward, feature-likelihood divergence (FLD), and bits per dimension (BPD).

FLD. For FLD, we draw K samples from the model, and K samples from the test set restricted to the target class. The FLD is computed using the DINOV2 feature space (from the ViT-B14 model) between these two sets of samples (Oquab et al., 2024). For MNIST, $K = 5$k.

Figure 5: Uncurated pre-trained CelebA model samples using a discrete generative model.

BPD. An upper bound on the log-likelihood is computed using the MDM ELBO loss (on the fine-tuned model), and this is normalized (by the number of pixels and $\log 2$) to obtain the reported BPD metric. For baselines other than discrete guidance, this metric is computed on the test set restricted to the target-class. For discrete guidance, BPD is computed by evaluating the MDM ELBO of the base model on samples generated using guidance (due to lacking an analogous ELBO for guidance-based sampling).

D.4 Text experiments

All text experiments begin by starting from the pretrained MDLM[2] consisting of 170 million parameters. We then do supervised fine-tuning to produce a model capable of producing output of the desired format before proceeding with online fine-tuning. For all text experiments we train using the Adam optimizer with $\beta_1, \beta_2 = \{0.9, 0.999\}$ and weight decay of 0. For both tasks SVDD was run with $n = 10$ particles. Evaluation was done by training each method for one day across three seeds and generating 1000 samples from the best checkpoint according to the mean reward generated during training. We report both the mean reward of the 1000 samples across three seeds and their standard deviations, as well as the generative perplexity according to a GPT2-Large model with 812 million parameters [3].

D.4.1 Tinystories

To obtain a base model we performed supervised fine-tuning from the base MDLM model on the Tinystories dataset. As all methods were prompted with the text "Once upon a time" during training, we restricted the SFT dataset to only datapoints whose stories started with the text "Once upon a time,", resulting in a corpus of 977,921 examples. SFT was done using Adam, $\beta_1, \beta_2 = \{0.9, 0.999\}$, and a learning rate of $4\text{e}{-3}$ over 60,000 training steps using 4 NVIDIA A100 GPUs. All models were trained for up to 24 GPU hours on NVIDIA L40S GPUs. Fine-tuning checkpoints were selected based upon the iteration with best average reward when sampling a new training batch from the model. All methods employ a learning rate schedule with a linear warmup for the first 2,500 training steps and keep the noise schedule provided by the pre-trained MDLM model – a log-linear schedule with $\sigma_{min} = 1\text{e}{-4}$ and $\sigma_{max} = 20$. All evaluations were performed by taking 1000 samples for each method across three seeds. We provide a set of curated samples in Table 6.

The reward function for this task was selected to be a pre-trained classifier[4] (Kluge Corrêa, 2024). The classifier is a RoBERTa model with 125 million parameters which was fine-tuned on a curated subset of various toxicity/harmlessness datasets. The reward $R(x_0)$ is then set to the likelihood of a sequence being toxic under the pre-trained classifier so that $R(x_0) = p(a = 1|x_0)$ where $p(a = 1|x_0)$ denotes the likelihood of the sequence x_0 possessing a toxic sentiment. We select this task as it allows a demonstration of how our method can recover rare behavior under the pre-trained model while still maintaining sample quality. We consider as our target distribution the tempered reward distribution $\pi_0(x_0) \propto p_0^{\text{pre}}(x_0)R(x_0)^{1/\beta}$ with $\beta = 0.25$.

For DDPP-LB we used 1,500 warmup steps for $\log \mathcal{Z}_{\pi_t}(x_t)$, a learning rate of $1\text{e}{-4}$ and a batch size of 16. We employ a replay buffer with a max length of 10,000 and sample training batches uniformly from the buffer. The buffer is filled every 50 training steps with a batch sampled on-policy from the current fine-tuned model, while every 250 steps a batch from the SFT training dataset is added to the buffer. We use EMA with a decay rate of $\epsilon = 0.9999$, a learning rate of $1\text{e}{-4}$, and train without LoRA. DDPP-LB was trained using 64 inference steps for simulation. DDPP-IS employed the same hyperparameter settings as DDPP-LB except that it dispelled with learning $\log \mathcal{Z}_{\pi_t}(x_t)$ and instead estimated it with $K = 16$ Monte Carlo samples from the one-step pre-trained posterior $p_t^{\text{pre}}(x_0|x_t)$.

As RTB cannot fit all timesteps of a trajectory into memory during the backwards pass, we detached 55% of timesteps where each trajectory consisted of 32 timesteps. RTB was trained with LoRA enabled, a LoRA rank of 16, and a learning rate of $5\text{e}{-5}$. Due to memory constraints the batch size was set to 4. SVDD was run by using $n = 10$ particles per inference timestep. best of N (with $N = 10$) sampling was performed by taking 10 samples from the SFT model and selecting the sample with highest likelihood under the reward model.

[2] https://huggingface.co/kuleshov-group/mdlm-owt
[3] https://huggingface.co/openai-community/gpt2-large
[4] https://huggingface.co/nicholasKluge/ToxicityModel

Table 6: Curated samples from DDPP-LB on Tinystories.

Generated Text
Once upon a time, there was a little girl named Lily. One day, Lily went to the park with her mom. She loved to run fast and laugh. Lily saw a funny butterfly and ran it too. She fell fast and hurt a tree. Lily's knee hurt and she cried. But her mommy kissed her cheeks and said, "Be careful when you run, Lily."
Once upon a time, there was an elderly wolf. He lived in a big den against the woods. One day, the wolf felt very tired and wanted a place where there was a big tree to eat on. So, he went to sleep all day. But, while he was waking up, he saw a big, icy creature. The quickly jumped up, but his legs were too weak. The creature took the elderly wolf away. And that is how winter ended.
Once upon a time, there was a little bird. His wings were weak and he fell down. The bird wanted his wing to restore him. So, he flapped his wings with his weak heart.
Once upon a time, there was a little boy named Timmy. He loved going to the woods with his family. One day, Timmy's friend Johnny came to the woods to play. Johnny was excited to go outside and play. They found a big tree with words said "I reverse," and Timmy would play on its branches. He said, "reverse!" and pushed the tree. Then, they ran and laughed. But then, they heard a loud noise coming from the bushes and scared them. It was a big, mean bear! Timmy tried to reverse and run away, but he wasn't fast enough. The bear chased him and caught him up with its sharp hands. Timmy was very scared and never went back to the woods again.
Once upon a time, on a calm blue sea, there was a small boat. The boat had sailors. They lived happily in the day water. One day, the water was very hot. So, they all decided to soak up and have a picnic. But, by the time, the sailors started to play a game. They swam around and counted, ", two, three soon!" and all the sailors kept playing. They water flowers and trees, and everyone laughed. But then, a clumsy heavy sailor hurt his head on a rock. "Ouch!" he cried. His friends helped him up and said, "Be careful next time!" The sailor felt better and they all laughed. They knew they could play again and have fun on a calm day soon.

D.4.2 AMAZON REVIEWS

Table 7: Curated samples from DDPP-LB on Amazon review generation task.

Generated Text
Cheap. Poor fit. The pants return immediately and the fabric was like a burlap sack bag-Wanted it for a gift-It's crap. Broke in one day. Customer service never responded. Very cheap!
Everything was horrible.
It's the worst one I've ever bought.. you have to keep it in your bag for my daughter and they, seriously feel embarrassed if you ever got it it falls out and it ripped.. returning this. That said hated this bag till I see it!!!!!! (Cnaven at the neckline and it is super too short. Color is off white. Maybe I have to fix if I want ironing What a waste of valu money
Such poor material!!! It was like a plastic. Way too small so I returned.It was smaller than the size listed
I don't feel this product has any quality. My sunglasses was delivered broken.
Cheap piece of garbage. Got it for my niece for Halloween and it broke inó one time

We again begin by first performing supervised fine-tuning from the base MDLM model, but this time on the fashion split of the Amazon Reviews dataset (Hou et al., 2024) restricted to reviews consisting of at most 512 tokens, resulting in an SFT dataset of size. We perform SFT using Adam with $\beta_1, \beta_2 = \{0.9, 0.999\}$ and a learning rate of $4\text{e}{-3}$ and EMA with decay parameter 0.99. The SFT model was trained for 85,000 training steps on 4 NVIDIA A100 GPUs. As for the tinystories task fine-tuning checkpoints were selected based upon the iteration with best average reward when sampling a new training batch from the model. All methods employ a learning rate schedule with a linear warmup for the first 2,500 training steps and keep the noise schedule provided by the pre-trained MDLM model – a log-linear schedule with $\sigma_{min} = 1\text{e}{-4}$ and $\sigma_{max} = 20$. All evaluations were performed by taking 1000 samples for each method across three seeds.

The reward function for this task was a BERT model consisting of 167 million parameters fine-tuned on Amazon customer reviews[5] to predict a review's star-rating. We then set the reward $R(x_0) = p(a = 1|x_0)$, the likelihood under the pre-trained classifier that the generated sample is a one-star review. We consider as our target distribution the tempered reward distribution $\pi_0(x_0) \propto p_0^{\text{pre}}(x_0)R(x_0)^{1/\beta}$ with $\beta = 0.5$.

For DDPP-LB we used 1,500 warmup steps for $\log \mathcal{Z}_{\pi_t}(x_t)$, a learning rate of $1\text{e}{-4}$ and a batch size of 16. We employ a replay buffer with a max length of 10,000 and sample training batches uniformly from the buffer. The buffer is filled every 5 training steps with a batch sampled on-policy from the current fine-tuned model, while every 250 steps a batch from the SFT training dataset is added to the buffer. We use EMA with a decay rate of $\epsilon = 0.9999$, a learning rate of $1\text{e}{-4}$, and train without LoRA. DDPP-LB was trained using 64 inference steps for simulation. DDPP-IS employed the same hyperparameter settings as DDPP-LB besides not learning $\log \mathcal{Z}_{\pi_t}(x_t)$ and instead estimating it with $K = 16$ Monte Carlo samples from the one-step pre-trained posterior $p_t^{\text{pre}}(x_0|x_t)$.

As RTB cannot fit all timesteps of a trajectory into memory during the backwards pass, we detached 78.5% of timesteps where each trajectory consisted of 64 timesteps. RTB was trained with LoRA enabled, a LoRA rank of 16, and a learning rate of $5\text{e}{-5}$. Due to memory constraints the batch size was set to 4. SVDD was run by using $n = 10$ particles per inference timestep. best of N (with $N = 10$) sampling was performed by taking 10 samples from the SFT model and selecting the sample with highest likelihood under the reward model.

We provide a set of curated samples for the Amazon task in Table 7.

[5] https://huggingface.co/LiYuan/amazon-review-sentiment-analysis

E ADDITIONAL EXPERIMENTAL RESULTS

E.1 AUTOREGRESSIVE BASELINE

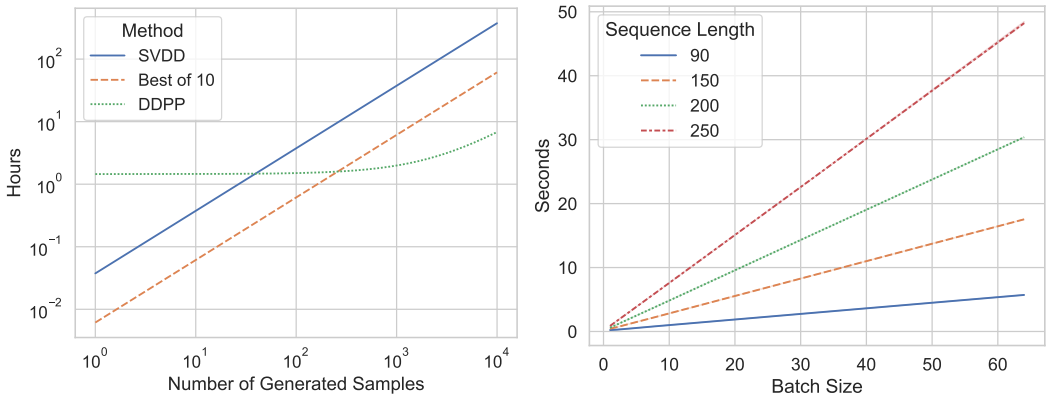
Table 8: Results for Tinystories with Twisted SMC autoregressive baseline. Because DDPP and Twisted SMC use different architectures and number of parameters, we report the reward and generative perplexity before and after fine-tuning and bold the method which provides the best percent change from the base model.

	$\log R(x_0)$ pre	$\log R(x_0)$ post	% change in $\log R(x_0) \uparrow$	Gen PPL pre	Gen PPL post	% change in Gen PPL $\downarrow$
Twisted SMC	40.52 ± 0.10	94.56 ± 0.85	133.4 ± 1.60	8.52 ± 0.05	10.25 ± 0.51	$\mathbf{20.31 \pm 5.96}$
DDPP-LB	54.94 ± 0.76	205.76 ± 3.88	$\mathbf{278.0 \pm 15.2}$	16.66 ± 0.20	19.6 ± 0.69	$\mathbf{18.38 \pm 4.05}$

In order to compare the performance of DDPP against autoregressive methods, we evaluated Twisted SMC (Zhao et al., 2024b) on the Tinystories task and compared its performance to DDPP. Unfortunately, the base model used by Twisted SMC was of a different architecture and model size than we used. In particular, as the base autoregressive model, we used the GPT-Neo architectured model trained in the original Tinystories paper (Eldan and Li, 2023) which uses 68 million parameters while our base model used a diffusion transformer (Peebles and Xie, 2023) with 170 million parameters. To ensure the fairest possible comparison, despite the difference in model parameters of the pre-trained models, we compare Twisted SMC and DDPP fine-tuning in terms of the percent change in the average reward and generative perplexity of the finetuned samples. Results are presented in Table 8.

We observe that DDPP improves reward more than the autoregressive baseline while incurring a comparable but minor performance drop in generative perplexity to Twisted SMC. Our results here contextualize that DDPP can better negotiate the tradeoff between optimizing reward and sample quality than twisted SMC on autoregressive models. Finally, we note that Twisted SMC cannot be easily performed for MDM's and as such DDPP remains a compelling choice for fine-tuning.

E.2 COMPARING OVERALL COMPUTATION TIME OF DDPP VS INFERENCE BASED METHODS

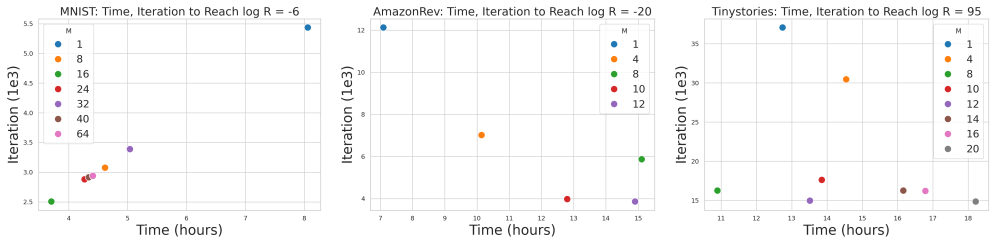


(a) Sample generation time

(b) ESMFold query times

Figure 6: (a) We plot the number of hours required to generate a particular number of samples for each method (including DDPP training time) on the 90 sequence length designable β-sheet protein task. We see that although inference time methods may be preferable if generating only a few samples, DDPP quickly offers faster sampling as the number of samples grows. (b) We demonstrate how the ESMFold based reward function cannot be parallelized on a single GPU and that the computational overhead becomes even more pronounced as sequence length increases.

To further analyze the computation time tradeoff of inference time methods compared to DDPP we examine their computational overhead on the task of generating designable β-sheet protein of



(a) MNIST (b) Amazon Reviews (c) Tinystories

Figure 7: Iterations and Time required to reach a threshold value of $\log R(x_0)$ for varying number of MC samples M in DDPP-IS. Threshold values are computed using training curves smoothed with a rolling average, to better capture training trends.

sequence length 90. We expect that the computational tradeoff of inference time method vs fine-tuning method to favor fine-tuning based methods as the number of generated samples scales and as the computation required to evaluate the reward function increases. We examine the designable β-sheet task as its reward function is especially onerous to compute as it involves folding a protein with ESMFold. This process does not parallelize well on even on an A100 80 GB GPU as show in Figure 6b where we see that reward computation time scales linearly with batch size. Moreover, the reward function computation becomes even more expensive if scale sequence length as one might do in many real world tasks due to an $O(N^3)$ (where N denotes sequence length) operation involving pairwise residue interactions.

To evaluate computational efficiency we compare the overall computation time on a single A100 80 GB GPU required to generate different numbers of sequence length 90 samples from a given method, as shown in Figure 6a. To compute the time for DDPP to generate samples, we first measure the training time to convergence and add this to the inference time required to sample from the fine-tuned model. For best of 10 and SVDD we simply generate the specified number of samples and record elapsed computation time. We observe in Figure 6a that while SVDD and best of 10 are fast if one needs to generate only a few samples, they quickly become significantly more expensive as the number of samples needed increases. In particular, to generate only 1000 samples DDPP, combining both its training and inference time, requires only 1.99 hours while best of 10 and SVDD require 6.15 and 37.5 hours, respectively. This means that for generating even this relatively small set of 1000 samples, amortized sampling using DDPP is up to **18x** faster than comparable inference time methods while generating higher quality samples. Moreover, this inference time gap only increases as the number of generated samples increases and would also become more severe were sequence length to increase, as evidenced by Figure 6b.

Of course, the utility of amortized sampling methods for reducing overall computation time is dependent on the number of samples required and the computational overhead of the reward function. If only a few samples are needed and the reward function is cheap, it is advisable to use an inference time method such as best of N or SVDD to generate samples. However, if a large number of samples must be generated or the reward is expensive amortized sampling approaches like DDPP are preferred. Indeed, this is one of the ultimate motivations of using RLHF algorithms in autoregressive models instead of methods like best of N – the overall computation required to fine-tune a pre-trained model and subsequently sample from the fine-tuned model is much cheaper than generating N samples and selecting the best one many times over.

E.3 Analysis of impact of the number of MC samples M in DDPP-IS

We include an ablation comparing the impact of the number of MC samples M used in DDPP-IS, on the datasets: MNIST, Amazon reviews, and Tinystories. The training steps and process times at which the model achieves a certain reward threshold, for different M, are plotted in Figure 7.

We observe that a larger number of samples generally improves the reward at a faster rate per iteration (gradient step), while each iteration generally takes more time, based on how expensive the reward function is to evaluate. Each gradient step involves a single call to the denoising model, and M calls

to the reward function. Therefore, when increasing the number of samples M, it results in a trade-off with fewer calls to the denoiser, while having more calls to the reward function, to achieve the same reward. In addition, when using more samples, it results in a loss with lower variance, which can benefit training.

For the MNIST task in Figure 7a we see that a larger number of samples achieves the reward threshold in fewer iterations, and in less time. In this task the reward function is simple (a small classifier), so a larger number of samples doesn't add too much of a time cost per iteration, and a larger number of samples is preferred. On the other hand, for the Amazon reviews task in Figure 7b this manifests in the trend that as M increases, the number of iterations to reach the reward threshold generally decreases while the overall time increases. For this task, the reward model is expensive to evaluate, so a smaller number of samples is more time-efficient. Finally, for tasks such as Tinystories, shown in Figure 7c, where the reward function is more expensive than MNIST but less so than Amazon reviews the interplay is more complicated. Increasing the number of Monte Carlo samples to $M = 8$ improves the variance properties of the loss curve, which leads to an improvement in both iterations and overall time. As M increases beyond this, the number of iterations to convergence decreases slightly at the cost of more overall computation time. A suggestion informed by these experiments is that, for a given time budget, M should be treated as a hyperparameter for tuning, with ranges over lower values for tasks with more expensive reward functions.

E.4 Ablation on protein experiments sequence length

Table 9: Ablation on different protein lengths using DDPP-LB. DDPP-LB still generates high quality proteins as sequence length increases.

Sequence length	β-sheet % ↑	pLDDT ↑	pTM ↑	$\log R(x_0)$ ↑
90	0.44 ± 0.04	0.90 ± 0.03	0.81 ± 0.03	3.70 ± 0.19
150	0.71 ± 0.04	0.74 ± 0.01	0.66 ± 0.04	4.56 ± 0.28
200	0.56 ± 0.09	0.77 ± 0.13	0.75 ± 0.11	4.36 ± 0.32
250	0.64 ± 0.01	0.91 ± 0.05	0.89 ± 0.02	4.78 ± 0.09

We investigate further the performance of DDPP on the designable β-sheet task as the sequence length scales. To this end, we repeat our β-sheet experiment for DDPP-LB over additional protein sequence lengths of 150, 200, and 250 (we note that the length 90 we used in our original experiments was selected due to constraints regarding wet lab experimental protocol). We maintain all experimental settings, but for each of the different sequence lengths we perform a grid search over the reward temperature parameter β and the learning rate. We selected a learning rate of 1e-6 for each additional sequence length, while for reward temperature we selected a setting of $\beta = 0.0625$ for sequence lengths 150 and 250 and maintained the original reward temperature of $\beta = 0.125$ for the sequence length 200 task. We report our results in Table 9, where we see that DDPP-LB can still generate proteins according to the target distribution even as protein sequence length increases. In fact, DDPP-LB seems to generate sequences with higher reward as we increased sequence length, an observation which follows our intuition that the DPLM base model should is better at generating slightly longer protein sequences as miniproteins of short lengths like 90 are relatively rare in the base model's training set compared to slightly longer proteins.

E.5 Discrete Guidance Experiments

To help compare DDPP's performance to inference-based methods we compare against discrete guidance as proposed in Nisonoff et al. (2024). Discrete guidance requires both a differentiable reward as well as a reward which may be evaluated for partially masked (noisy) states. Unfortunately, our text tasks require a retokenization step as the reward models use a different tokenization than does the pre-trained MDLM model we employ. As tokenization is a non-differentiable operation we are unable to evaluate discrete guidance on our text tasks. Further, since discrete guidance requires the reward function be evaluated on noisy states we are also prevented from evaluating it on our protein task. This is because our protein reward function uses ESMFold, a complicated protein folding model,

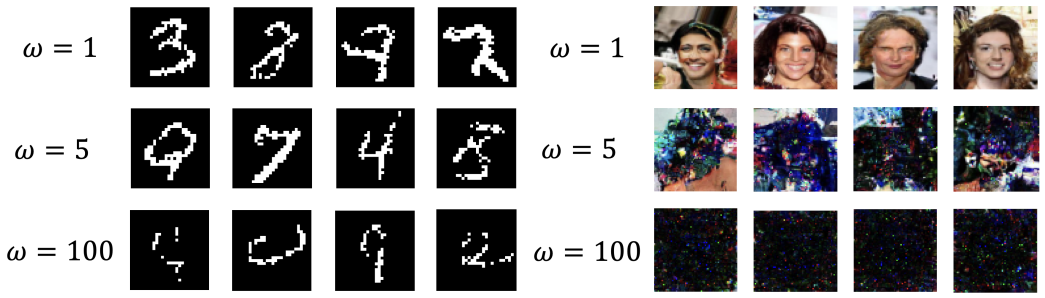


Figure 8: Discrete guidance samples for MNIST (left) and CelebA (right). As guidance scale increases sample quality decreases, especially for CelebA, while reward increases (see Table 2 and Figure 3). We view the generation of these high reward, low quality samples as ω increases as a form of reward hacking.

which is only defined on full protein sequences. However, our image tasks fulfill both criteria and as such we evaluate discrete guidance on the MNIST and CelebA tasks.

As our initial image classifiers (used as reward functions) are not defined on partially masked states we trained noisy versions of them for use in discrete guidance. To train the noisy reward functions we follow the recommendations of Nisonoff et al. (2024) by training on the same dataset as the original classifiers and noising the sampled datapoints according to the same forward process as the fine-tuned diffusion model. The noisy classifiers performed nearly as well as the original, non-noisy classifiers with a test set accuracy for MNIST degrading from 99% for the non-noisy classifier to 98% for the noisy classifier and from 96% for the non-noisy to 95% for the noisy classifier on CelebA. Final reward evaluations are performed by evaluating samples generated using discrete guidance on the noisy reward models with the original, non-noisy rewards. BPD values for this baseline are computed by evaluating the base model's ELBO on images sampled using guidance (due to lacking an analogous ELBO formula for guidance-based sampling). Evaluation protocol follows that used for DDPP and other baselines, described in more detail in Appendix D.1.2 for MNIST and Appendix D.3 for CelebA. Results are computed across three seeds for each guidance scale.

Guidance results are shown in Table 2 and Figure 3. We evaluate on rewards with the same temperature β as the other baselines ($\beta = 5$ for MNIST and $\beta = 10$ for CelebA). We have an additional multiplier in the guidance scale ω. For settings corresponding to those of DDPP ($\omega = 1$ for MNIST and $\omega = 1$ for CelebA) and other baselines we observe that discrete guidance either improves mean reward by a small amount (or decreases it) compared to the base model and does not approach the performance of DDPP. We scale the guidance scale by 5x and 100x over the original guidance scale and, as expected, observe an increase in mean reward as the guidance scale increases, even becoming competitive with the mean reward of DDPP on MNIST. However, this improvement in reward coincides with a crippling decrease in sample quality as indicated by the increase in BPD values. In Figure 8 we show guided samples for both MNIST and CelebA with increasing guidance strengths, where we see that as guidance strength increases an instance of reward hacking occurs, with the guided samples achieving high reward under the classifier while being of low sample quality.