Journal of Machine Learning Research 26 (2025) 1-46 Submitted 9/24; Revised 11/25; Published 12/25

Reward-Directed Score-Based Diffusion Models via q-Learning

Xuefeng Gao XFGAO@SE.CUHK.EDU.HK
Department of Systems Engineering and Engineering Management
The Chinese University of Hong Kong
Hong Kong, China

Jiale Zha JIALEZHA@LINK.CUHK.EDU.HK
Department of Systems Engineering and Engineering Management
The Chinese University of Hong Kong
Hong Kong, China

Xun Yu Zhou XZ2574@COLUMBIA.EDU
Department of Industrial Engineering and Operations Research &
The Data Science Institute
Columbia University
New York, NY 10027, USA

Editor: Nicolas Le Roux

Abstract

We propose a new reinforcement learning (RL) formulation for training continuous-time score-based diffusion models for generative AI to generate samples that maximize reward functions while keeping the generated distributions close to the unknown target data distributions. Different from most existing studies, ours does not involve any pretrained model for the unknown score functions of the noise-perturbed data distributions, nor does it attempt to learn the score functions. Instead, we formulate the problem as entropy-regularized continuous-time RL and show that the optimal stochastic policy has a Gaussian distribution with a known covariance matrix. Based on this result, we parameterize the mean of Gaussian policies and develop an actor–critic type (little) q-learning algorithm to solve the RL problem. A key ingredient in our algorithm design is to obtain noisy observations from the unknown score function via a ratio estimator. Our formulation can also be adapted to solve pure score-matching and fine-tuning pretrained models. Numerically, we show the effectiveness of our approach by comparing its performance with two state-of-the-art RL methods that fine-tune pretrained models on several generative tasks including high-dimensional image generations. Finally, we discuss extensions of our RL formulation to probability flow ODE implementation of diffusion models and to conditional diffusion models.

Keywords: Score-based diffusion models, reward function, continuous-time reinforcement learning, q-learning, stochastic differential equations

1. Introduction

Diffusion models form a powerful family of probabilistic generative AI models that can capture complex high-dimensional data distributions (Sohl-Dickstein et al., 2015a; Song and

Ermon, 2019; Ho et al., 2020; Song et al., 2021b). The basic idea is to use a forward process to gradually turn the (unknown) target data distribution to a simple noise distribution, and then reverse this process to generate new samples. A key technical barrier is that the time-reversed backward process involves a so-called score function that depends on the unknown data distribution; thus learning the score functions (called "score matching") becomes the main objective of these models. Diffusion models have achieved state-of-the-art performances in various applications such as image and audio generations (Rombach et al., 2022; Ramesh et al., 2022) and molecule generation (Hoogeboom et al., 2022; Wu et al., 2022). See, e.g., Yang et al. (2023) for a survey on diffusion models.

In standard diffusion models, the goal is typically to generate new samples whose distribution closely resembles the target data distribution (e.g. "generate more cat pictures", or "write a Shakespearean play"). Standard score-based diffusion models are trained (i.e. estimating score functions using neural nets) by minimizing weighted combination of score matching losses (Hyvärinen and Dayan, 2005; Vincent, 2011; Song et al., 2020b). However, in many applications, we often have preferences about the generated samples from diffusion models (e.g. "generate prettier cat pictures", or "write a Shakespearean thriller that happened in New York"). A common way to capture this is to use a reward function, either handcrafted (e.g. a utility function) or learned (through human feedbacks), to evaluate the quality of generated samples related to the preferences. This has led to an interesting recent line of research on how to adapt standard diffusion models to optimizing reward functions. Several approaches have been proposed, including discrete-time reinforcement learning (RL) (Black et al., 2024; Fan et al., 2023), continuous-time stochastic control/RL (Uehara et al., 2024; Zhao et al., 2024), backpropagation of reward function gradient through sampling (Clark et al., 2024), supervised learning (Lee et al., 2023), and guidance (Dhariwal and Nichol, 2021). These studies, however, focus on fine-tuning certain *pretrained* diffusion models, whose score functions have already be learned, to maximize the values of additional reward functions.

In this paper, we put forward a significantly different approach where we directly train a diffusion model from scratch for reward optimization using RL, *without involving any pretrained model*. There are two motivations behind this approach. One is practical: in many applications, especially for new or special-purpose tasks, a pretrained model may not even exist or existing pretrained models may not be easily adapted and fine-tuned. The other is (more importantly) conceptual: fine-tuning a pretrained model is essentially a model-driven approach *for dealing with the unknown score function* (i.e. first estimate the score function and then optimize), which is prone to model misspecification and misleading risks.[1] By contrast, our approach is model-free and data-driven, not relying on a good pretrained model. More on this later.

We consider *continuous-time* score-based diffusion models in Song et al. (2021b), which utilize stochastic differential equations (SDEs) for noise blurring and sample generations. The denoising/reverse process also follows an SDE whose drift includes the unknown score function (the gradient of the log probability density of the noise-perturbed data distribution). We take the continuous-time framework because a) the SDE-based formulation is

1. We stress the word "model" here specifically refers to the score function, and "model misspefication" refers to poor estimations of the score. Existing works on reward maximization and fine-tuning pretrained models may still use a data-driven RL approach to deal with an unknown reward function.

general and unifies several celebrated diffusion models, including the score matching with Langevin dynamics (SMLD) (Song and Ermon, 2019) and the denoising diffusion probabilistic modeling (DDPM) (Ho et al., 2020); b) it leads to not only the SDE-based implementation of score diffusions, but also the probability flow ordinary difference equation (ODE) implementation (Song et al., 2021b); and c) there are more analytical tools available for the continuous setting that enable a rigorous and thorough analysis leading to interpretable (instead of black-box) and general (instead of ad hoc) algorithms.

Due to the need of optimizing rewards of the generated samples, we propose a continuous-time RL formulation for adapting diffusion models to reward functions. In this formulation, because the score function in the drift of the denoising process is unknown, we take it as the control (action) variable. This idea of regarding scores as actions, first put forth in Uehara et al. (2024) and Zhao et al. (2024), is quite natural because the problem now has two somewhat competing criteria: score matching (i.e. the generated samples should be close to the true distribution) and terminal reward (i.e. the samples should align with the preferences); so an adjustable control can be used to achieve the best trade-off between the two. Specifically, we introduce an objective function that is a weighted combination of two parts: a running reward that penalizes (regularizes) the Kullback–Leibler (KL) divergence of the denoising process from the true score function, and a terminal reward of the generated samples based on the preference. This naturally leads to an RL problem with continuous state and action spaces in which the system dynamics are known but the running reward function is unknown (because it involves the true yet unknown score function) and the terminal reward is possibly unknown. We can therefore adapt and apply the theory and algorithms developed recently for general continuous-time RL with controlled diffusions (Wang et al., 2020; Jia and Zhou, 2022a,b, 2023, 2025).

However, there is a critical issue one needs to address in this formulation. While the general theory in (Wang et al., 2020; Jia and Zhou, 2022a,b, 2023) allows unknown reward functions, it requires access to a (if noisy) observation (a "reinforcement signal") of the reward every time a state is visited. It is natural to assume that we have such signals from the terminal reward (e.g. human rating of aesthetic quality of a generated image). However, obtaining signals from the running reward in our RL problem is subtle, because the KL divergence term in the objective involves the unknown score function. To overcome this difficulty, we express the true score function as the ratio of two expectations with respect to the data distribution. Because we have access to i.i.d. samples from the unknown data distribution, we derive a simple ratio estimator as a noisy observation from the true score value, allowing us to obtain a reinforcement signal from the running reward whenever an action is applied. This procedure is computationally very cheap – indeed, we will show that a mere *single* sample for computing the ratio estimator already achieves satisfactory performance in an image generation task.

To solve the resulting RL problem for continuous-time diffusion models, we adapt the approach in Jia and Zhou (2023, 2025) to develop theory and algorithms for our RL problem. Specifically, we take the entropy-regularized, exploratory framework of Wang et al. (2020) and optimize over stochastic policies. We show that the optimal stochastic policy for our problem is Gaussian with a known covariance matrix and an unknown mean function. This key theoretical result suggests that we need to consider Gaussian policies only and parameterize their mean functions when designing RL algorithms. This insight inspires us

to develop an actor–critic type algorithm based on the (little) q-learning theory established in Jia and Zhou (2023, 2025) to solve our RL problem. We also present a convergence analysis of our algorithm. Furthermore, we implement the q-learning algorithm and evaluate its performances through a series of experiments on three diverse datasets: synthetic data from a one-dimensional Gaussian mixture distribution, a two-dimensional Swiss rolls dataset (Sohl-Dickstein et al., 2015b; Lai et al., 2023), and the CIFAR-10 image dataset (Krizhevsky, 2009). The results demonstrate that our algorithm achieves strong performance in adapting diffusion models to optimize various reward functions.

Thanks to the continuous-time setting, we can extend our RL formulation to ODE-based models in a straightforward manner. The probability flow ODE implementation of diffusion models is another mainstream approach for sample generation, in addition to the SDE-based one; see, e.g., Song et al. (2020a), Song et al. (2021b); Karras et al. (2022); Lu et al. (2022). Compared with SDE-based samplers, ODE-based deterministic samplers often converge to the data distribution much faster with fewer sampling steps, at the cost of slightly inferior sample quality. We adapt our continuous-time RL formulation to ODE-based models, where the system dynamics are now described by controlled ODEs. We show that the optimal stochastic policy is still Gaussian, and the algorithm designed for the SDE-based formulation still applies after one replaces the SDE-based sampler with ODE counterparts. We implement the resulting algorithm for two ODE-based samplers: ODE-Euler, which is based on Euler discretization of the probability flow ODE, and DDIM of Song et al. (2020a). We find that the algorithm performs well and indeed accelerates the training process compared with the SDE-based formulation. On the other hand, while we mainly focus on unconditional diffusion models, our RL formulation can be readily extended to conditional diffusion models which are used for conditional data generation, such as in text-to-image models.

We now explain the key differences between our work and several closely related ones. Black et al. (2024) propose to fine-tune *discrete-time* diffusion models using RL and directly optimize the reward function (*without* KL regularization). The denoising process is formulated as a multi-step Markov decision process and the action at each step corresponds to the next denoising state. They then present a policy gradient algorithm, referred to as DDPO, to solve their RL problem. Fan et al. (2023) have a similar discrete-time RL formulation, but add the KL divergence between the fine-tuned model and the pretrained model to the objective to prevent overfitting the reward. They also use a policy gradient method called DPOK to solve the problem. Our paper differs from these two in a few important aspects. First, we consider a continuous-time RL formulation for *continuous-time* diffusion models, where the action is a substitute for the unknown score controlling the drift of the reverse-time SDE. Therefore, our framework and the resulting theory and algorithm are fundamentally different from theirs. Second, compared with Black et al. (2024); Fan et al. (2023) that focus on the DDPM based denoising process with stochastic transitions, our approach not only applies to SDE-based implementation of diffusion models, but also can be readily extended to probability flow ODE implementation. Last but most importantly, in dealing with the unknown score function, the pretrained approach is essentially model-driven, whose performance crucially depends on the availability of a *good* pretrained model, whereas our approach is driven by data – noisy signals from the score. To wit, we penalize the deviation from the (unknown) true score model in our RL objective, rather than from

a (known) pretrained model as in Fan et al. (2023). As a consequence, our formulation encourages the generated samples to stay close to the true data distribution while maximizing the reward, whereas theirs is to incentivize the samples to stay close to the pretrained data distribution, which is not necessarily always of high quality.

To compare our approach with the fine-tuning approach in Black et al. (2024) and Fan et al. (2023), we conduct experiments on the 2-dimensional Swiss Roll data. Although DDPO and DPOK were originally developed for conditional generations, they can be easily adapted to unconditional data generations. We find that DDPO of Black et al. (2024) suffers from the issue of reward over-optimization, where the generated distribution diverges too far from the original data distribution. On the other hand, DPOK of Fan et al. (2023) has a similar performance as our q-learning algorithm, provided that the pretrained model is of good quality. However, if the pretrained model is not good enough in the sense that the pretrained distribution is not close to the data distribution, our q-learning algorithm outperforms DPOK significantly. Notably, this observation holds true for the CIFAR-10 image dataset as well, where our q-learning approach exhibits superior performance compared to DPOK.

We also mention two contemporary studies (Uehara et al., 2024; Zhao et al., 2024) that consider fine-tuning pretrained *continuous-time* diffusion models using entropy regularized control/RL. The key difference in the problem formulations is again that they penalize the deviation from a *known* pretrained score/diffusion model, while we penalize the deviation from the unknown true score model. In addition, our objective involves an unknown running reward, making ours a genuine RL problem instead of a stochastic control one as in Uehara et al. (2024). Therefore, our theory and algorithms are conceptually different from these two papers. On the other hand, recent works on GFlowNets (Generative Flow Networks) employ a discrete-time RL formulation to sample from distributions with given unnormalized probability mass/density functions; see e.g. Bengio et al. (2023); Lahlou et al. (2023); Zhang et al. (2022). Moreover, a concurrent work Zhang et al. (2024) studies fine-tuning discrete-time diffusion models using GFlowNets, with the goal of generating high-reward images with relatively high probability. Our study is entirely different from this line of work too in that we focus on the generative diffusion setting combined with optimizing reward functions via continuous-time RL.

Finally, it is important to note that our approach and results can be easily modified to cover pretrained and fine-tuning formulation as a special case. Indeed, if a score has been trained and given, then we can generate the reinforcement signals by simply computing the trained score function values instead of using the ratio estimator, and the rest is entirely the same.

The remainder of the paper is organized as follows. In Section 2, we briefly review continuous-time score-based diffusion models. In Section 3, we discuss the continuous-time RL formulation for reward maximizations in diffusion models. Section 4 presents our main theoretical result, based on which we design an actor–critic type q-learning algorithm in Section 6. In Section 7, we present the results of the numerical experiments for two toy examples, whereas in Section 8, we test our algorithm on a reward-directed image generation task. Section 9 highlights extensions to ODE models and conditional diffusion models. Finally, Section 10 concludes.

2. Quick Review on Continuous-Time Score-Based Diffusion Models

For reader's convenience we briefly recall the continuous-time score-based diffusion models with SDEs (Song et al., 2021b). Denote by $p_0 \in \mathcal{P}(\mathbb{R}^d)$ the unknown continuous data distribution, where $\mathcal{P}(\mathbb{R}^d)$ is the space of all probability measures on $\mathbb{R}^d$. Given i.i.d samples from p_0, standard diffusion models aim to generate new samples whose distribution closely resembles the data distribution. In this classical setting, one does not consider the preference/reward of the generated samples.

- **Forward process and reverse process.**

  Fix $T > 0$. We consider a $d-$dimensional forward process $(\mathbf{x}_t)_{t\in[0,T]}$, which is a Ornstein—Uhlenbeck (OU) process satisfying the following SDE

$$d\mathbf{x}_t = -f(t)\mathbf{x}_t dt + g(t)d\mathbf{B}_t, \tag{1}$$

  where $\mathbf{x}_0$ is a random variable following the (unknown target) distribution p_0, $(\mathbf{B}_t)$ is a standard d-dimensional Brownian motion which is independent of $\mathbf{x}_0$, and both $f(t) \geq 0$ and $g(t) \geq 0$ are (prescribed and known) scalar-valued continuous functions of time t. The solution to (1) is

$$\mathbf{x}_t = e^{-\int_0^t f(s)ds}\mathbf{x}_0 + \int_0^t e^{-\int_s^t f(v)dv} g(s)d\mathbf{B}_s, \quad t \in [0,T]. \tag{2}$$

  Note that the forward model (1) is fairly general and it includes variance exploding SDEs, variance preserving SDEs, and sub variance preserving SDEs that are commonly used in the literature; see Song et al. (2021b) for details.

  Denote by $p_t(\cdot)$ the probability density function of $\mathbf{x}_t$, and $p_{t|0}(\cdot|\mathbf{x}_0)$ the density function of $\mathbf{x}_t$ given $\mathbf{x}_0$ for $t \in [0,T]$. By (2), $p_{t|0}(\cdot|\mathbf{x}_0)$ is Gaussian which has an analytical form.

  Now consider the reverse (in time) process $(\tilde{\mathbf{x}}_t)_{t\in[0,T]}$, where

$$\tilde{\mathbf{x}}_t := \mathbf{x}_{T-t}, \quad t \in [0,T]. \tag{3}$$

  Under mild assumptions, the reverse process still satisfies an SDE (Anderson, 1982; Haussmann and Pardoux, 1986; Cattiaux et al., 2023):

$$d\tilde{\mathbf{x}}_t = [f(T-t)\tilde{\mathbf{x}}_t + (g(T-t))^2 \nabla \log p_{T-t}(\tilde{\mathbf{x}}_t)]dt + g(T-t)dW_t, \quad t \in [0,T], \tag{4}$$

  where (W_t) is a standard Brownian motion in $\mathbb{R}^d$, and the term $\nabla_{\mathbf{x}} \log p_t(\cdot)$ in (4) is called the *score function*. By (3), the reverse process starts from a random location $\tilde{\mathbf{x}}_0 \sim p_T$, where p_T is the probability density/distribution of $\mathbf{x}_T$ (here and henceforth, probability distribution and probability density function are used interchangeably). Thus, at time T, we have $\tilde{\mathbf{x}}_T \sim p_0$, where p_0 is the target distribution we want to generate samples from. However, the distribution p_T is unknown because it depends on the unknown target distribution p_0. Because the aim of generative diffusion models

is to convert random noises into random samples from p_0, by (2) we can choose a random (Gaussian) noise ν, with

$$\nu := \mathcal{N}\left(\mathbf{0}, \int_0^T e^{-2\int_t^T f(s)ds}(g(t))^2 dt \cdot I_d\right), \tag{5}$$

as a proxy to p_T, where I_d is the $d-$dimensional identity matrix; see Song et al. (2021b); Lee et al. (2022). Note that ν is the distribution of the random variable $\int_0^t e^{-\int_s^t f(v)dv} g(s) d\mathbf{B}_s$ in (2), which is easy to sample because it is Gaussian, and will be henceforth referred to as the *prior distribution*.[2]

Remark 1 *When considering variance preserving SDEs where $f(t) = \frac{1}{2}\alpha(t)$ and $g(t) = \sqrt{\alpha(t)}$ for some nondecreasing positive function α, it is also common to use the stationary distribution of the forward SDE, which is $\mathcal{N}(0, I_d)$, as the prior distribution. Taking $\alpha(t) \equiv 2$ as an example, we deduce from (5) that $\nu = \mathcal{N}\left(0, \left(1 - e^{-2T}\right) \cdot I_d\right)$, which converges to $\mathcal{N}(0, I_d)$ as T becomes large.*

In view of (4), we now consider the SDE:

$$d\mathbf{z}_t = \left[f(T-t)\mathbf{z}_t + (g(T-t))^2 \nabla \log p_{T-t}(\mathbf{z}_t)\right] dt + g(T-t) dW_t, \quad \mathbf{z}_0 \sim \nu, \tag{6}$$

where ν is independent of the Brownian motion W. Because $\nu \approx p_T$, one expects that the distribution of $\mathbf{z}_T$ will be close to that of $\tilde{\mathbf{x}}_T$ which is p_0.

- **Training diffusion models via score matching.**

  The score function $\nabla_{\mathbf{x}} \log p_t(\cdot)$ in (6) is unknown because the data distribution p_0 is unknown. One can estimate it with a time–state score model $s_\theta(\cdot, \cdot)$, which is often a deep neural network parameterized by θ, by minimizing the score matching loss:

$$\min_\theta \mathbb{E}_{t\sim U[0,T]}\left[\lambda(t)\mathbb{E}_{\mathbf{x}_t}\left\|s_\theta(t, \mathbf{x}_t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\right\|^2\right]. \tag{7}$$

  Here, $\lambda(\cdot) : [0, T] \to \mathbb{R}_{>0}$ is some positive weighting function (e.g. $\lambda(t) = g(t)^2$), and $U[0, T]$ is the uniform distribution on $[0, T]$. This objective is intractable because $\nabla_{\mathbf{x}} \log p_t(\cdot)$ is unknown. Several approaches have been developed in literature to tackle this issue, including denoising score matching (Vincent, 2011; Song et al., 2021b), sliced score matching (Song et al., 2020b) and implicit score matching (Hyvärinen and Dayan, 2005). Here, we take denoising score matching for illustration. One can show that (7) is equivalent to the following objective

$$\min_\theta \mathbb{E}_{t\sim U[0,T]}\left[\lambda(t)\mathbb{E}_{\mathbf{x}_0}\mathbb{E}_{\mathbf{x}_t|\mathbf{x}_0}\left\|s_\theta(t, \mathbf{x}_t) - \nabla_{\mathbf{x}_t} \log p_{t|0}(\mathbf{x}_t|\mathbf{x}_0)\right\|^2\right], \tag{8}$$

  where $\mathbf{x}_0 \sim p_0$ is the data distribution, and $p_{t|0}(\cdot|\mathbf{x}_0)$ is the density of $\mathbf{x}_t$ given $\mathbf{x}_0$, which is Gaussian by (2). Because we have access to i.i.d. samples from p_0 (i.e. training data), the objective in (8) can be approximated by Monte Carlo, and the resulting loss function can be then optimized using e.g. stochastic gradient descent.

2. Although one could take an empirical estimate of $\mathbb{E}[\mathbf{x}_T] = e^{-\int_0^T f(s)ds}\mathbb{E}[\mathbf{x}_0]$ as the mean of the prior distribution ν, in practice ν is often chosen with mean zero because of two reasons: a) one can rescale the data to make them to have zero mean; see e.g. Ho et al. (2020); and b) the mean is close to zero when T is sufficiently large due to the exponential discounting (for variance preserving SDEs).

- **Algorithms.**

  After the score function is estimated, the true score function in (6) is replaced by the estimated score s_θ, and the reverse SDE (6) is discretized to obtain an implementable algorithm. Alternatively, one can use the probability flow ODE based implementation; see Section 4 of Song et al. (2021b) for details. The generated samples at time T are expected to follow approximately the data distribution, *provided* that the score is estimated accurately.

3. Problem Formulation

Standard training of diffusion models via score matching (8) – called *pretrained models* – does not consider preferences about the generated samples. A standard way to capture preferences is to add a reward function that evaluates the quality of generated samples related to the specified preferences. This is essentially a model-based approach – first learn a pre-trained model by estimating the score function and then optimize. This paper takes a conceptually different approach, one that is in the core spirit of RL, namely, to learn optimal policies *directly* without attempting to first learn a model. This leads naturally to a continuous-time RL formulation.

3.1 Reward-directed diffusion models

The key idea is that because the score term $\nabla \log p_{T-t}(\mathbf{z}_t)$ in (6) is unknown, we regard it as a control. With a reward function, this leads to the following stochastic control problem:

$$\max_{\mathbf{a}=(a_t:0\le t\le T)} \left\{ \beta \cdot \mathbb{E}[h(\mathbf{y}_T^{\mathbf{a}})] - \mathbb{E}\left[\int_0^T (g(T-t))^2 \cdot |\nabla \log p_{T-t}(\mathbf{y}_t^{\mathbf{a}}) - a_t|^2 dt \right] \right\} \tag{9}$$

subject to

$$d\mathbf{y}_t^{\mathbf{a}} = \left[f(T-t)\mathbf{y}_t^{\mathbf{a}} + (g(T-t))^2 a_t \right] dt + g(T-t) dW_t, \quad \mathbf{y}_0 \sim \nu. \tag{10}$$

Here, $a_t \in \mathbb{R}^d$ denotes the control action at time t, $(\mathbf{y}_t^{\mathbf{a}})$ is the controlled state process, h is the terminal reward function for the generated sample $\mathbf{y}_T^{\mathbf{a}}$, and $\beta \ge 0$ is a weighting coefficient.

The first term in the objective function (9) captures the preference on the generated samples. In this study, we do not require the reward/utility function h to be differentiable as in Clark et al. (2024), nor do we necessarily assume that its functional form is known. What we do assume is that given a generated sample $\mathbf{y}_T^{\mathbf{a}}$ we can obtain a noisy observation ("a reward signal") of the reward h evaluated at that sample (e.g. a human evaluation of the aesthetic quality of an image).

The second term in the objective (9) has the following interpretation. Consider a deterministic feedback policy so that $a_t = w(t, \mathbf{y}_t)$ for some deterministic function w. Let $\mathbb{P}^{\mathbf{z}}$ and $\mathbb{P}^{\mathbf{y}^{\mathbf{a}}}$ be the induced distribution (i.e., path measures over $C([0,T],\mathbb{R}^d)$) by the SDEs (6) and (10), respectively. Then Girsanov's theorem gives that under some regularity conditions (see, e.g. Uehara et al., 2024, Appendix C or Tang, 2024, Proposition 3.3)

$$\mathrm{KL}(\mathbb{P}^{\mathbf{y}^{\mathbf{a}}} || \mathbb{P}^{\mathbf{z}}) = \frac{1}{2}\mathbb{E}\left[\int_0^T (g(T-t))^2 \cdot |\nabla \log p_{T-t}(\mathbf{y}_t^{\mathbf{a}}) - w(t, \mathbf{y}_t^{\mathbf{a}})|^2 dt \right]$$

$$= \frac{1}{2}\mathbb{E}\left[\int_0^T (g(T-t))^2 \cdot |\nabla \log p_{T-t}(\mathbf{y}_t^{\mathbf{a}}) - a_t|^2 dt\right], \tag{11}$$

where the expectation is taken with respect to $(\mathbf{y}_t^{\mathbf{a}})$ where $\mathbf{y}_0 \sim \nu$. This justifies the second term of (9) in terms of KL-divergence.

Remark 2 (Connections between (11) and score matching) *It is worth noting that if we let $\mathbf{y}_0 \sim p_T$ and $\mathbf{z}_0 \sim p_T$, then we can compute that*

$$\begin{aligned} KL(\mathbb{P}^{\mathbf{z}}||\mathbb{P}^{\mathbf{y}^{\mathbf{a}}}) &= \frac{1}{2}\mathbb{E}\left[\int_0^T (g(T-t))^2 \cdot |\nabla \log p_{T-t}(\mathbf{z}_t) - w(t,\mathbf{z}_t)|^2 dt\right] \\ &= \frac{1}{2}\mathbb{E}\left[\int_0^T (g(t))^2 \cdot |\nabla \log p_t(\mathbf{z}_{T-t}) - w(T-t,\mathbf{z}_{T-t})|^2 dt\right] \\ &= \frac{1}{2}\mathbb{E}\left[\int_0^T (g(t))^2 \cdot |\nabla \log p_t(\mathbf{x}_t) - w(T-t,\mathbf{x}_t)|^2 dt\right], \end{aligned} \tag{12}$$

where the last equality follows from the time reversal of diffusion processes (see (4)), and the expectation there is taken with respect to the randomness in the forward process $(\mathbf{x}_t)$ in (1). This is exactly the score matching loss in (7) with the likelihood weighting $\lambda(t) = \frac{1}{2}g(t)^2$; see Song et al. (2021a) for details. In particular, it was shown in Song et al. (2021a) that $KL(\mathbb{P}^{\mathbf{z}}||\mathbb{P}^{\mathbf{y}^{\mathbf{a}}}) \geq KL(p_0||\mathbf{y}_T^{\mathbf{a}})$ when $\mathbf{y}_0 \sim p_T$ and $\mathbf{z}_0 \sim p_T$, and (12) serves as an efficient proxy for maximum likelihood training of score-based generative models. Finally, the hyperparameter β in (9) represents the weights placed on score matching and human preferences. When $\beta = 0$, the problem (9) reduces to a score matching problem. Thus, our formulation can be used also for pure score matching.

Denote

$$r(t,\mathbf{y},a) := -(g(T-t))^2 \cdot |\nabla \log p_{T-t}(\mathbf{y}) - a|^2, \tag{13}$$

the running reward function (or instantaneous reward) at time t in the objective (9). Because the true score function $\nabla \log p_{T-t}(\cdot)$ is unknown, this running/instantaneous reward function is also unknown. Moreover, recall that the true terminal reward function h is also generally unknown. Hence, we need an RL formulation of the problem (9)–(10), where exploration is necessary due to these unknown rewards. This is discussed in the next subsection.

3.2 Stochastic policies and exploratory formulation

We now adapt the exploratory formulation for continuous-time RL in Wang et al. (2020) and Jia and Zhou (2023) to our problem setting. A distinctive feature of our problem is that the RL agent knows the environment (the functions f, g are known in the SDE model (10)), but she does not know the instantaneous reward function r and the terminal reward function h. So she still needs to do "trial and error" – to try a strategically designed sequence of actions, observe the corresponding state process and a stream of running rewards and terminal reward samples/signals, and continuously update and improve her action plans

based on these observations. For this purpose, the agent employs stochastic policies in our RL setting.

Mathematically, let $\boldsymbol{\pi} : (t, \mathbf{y}) \in [0, T] \times \mathbb{R}^d \to \boldsymbol{\pi}(\cdot|t, \mathbf{y}) \in \mathcal{P}(\mathbb{R}^d)$ be a given stochastic feedback policy, where $\mathcal{P}(\mathbb{R}^d)$ is a collection of probability density functions defined on $\mathbb{R}^d$. For such a stochastic policy $\boldsymbol{\pi}$, Wang et al. (2020) and Jia and Zhou (2023) propose an "exploratory" formulation with the following "averaged" dynamic

$$d\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}} = \left[f(T-s)\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}} + g^2(T-s) \int_{\mathbb{R}^d} a\boldsymbol{\pi}(a|s, \tilde{\mathbf{y}}_s^{\boldsymbol{\pi}}) da \right] ds + g(T-s) dW_s, \quad \tilde{\mathbf{y}}_0^{\boldsymbol{\pi}} \sim \nu. \tag{14}$$

In contrast to (10), here the action randomization has been averaged out in the drift term of (14). The associated entropy-regularized control objective is given by:

$$\begin{aligned} & J(t, y, \boldsymbol{\pi}) \\ =& \mathbb{E}_{t,y} \left[\int_t^T \int_{\mathbb{R}^d} \big(r(s, \tilde{\mathbf{y}}_s^{\boldsymbol{\pi}}, a) \ - \theta \log \boldsymbol{\pi}(a|s, \tilde{\mathbf{y}}_s^{\boldsymbol{\pi}}) \big) \boldsymbol{\pi}(a|s, \tilde{\mathbf{y}}_s^{\boldsymbol{\pi}}) da ds + \beta h(\tilde{\mathbf{y}}_T^{\boldsymbol{\pi}}) \right], \end{aligned} \tag{15}$$

where the reward has also been averaged over action randomization, and $\theta > 0$ is the temperature parameter that controls the level of exploration.[3] The function $J(\cdot, \cdot, \boldsymbol{\pi})$ in (15) is called the (exploratory) value function of the stochastic policy $\boldsymbol{\pi}$. The goal of RL is to solve the following optimization problem:

$$\max_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \int J(0, y, \boldsymbol{\pi}) d\nu(y),$$

where $\boldsymbol{\Pi}$ is the set of admissible stochastic policies to be specified in Definition 3 below. Define

$$J^*(t, y) = \max_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} J(t, y, \boldsymbol{\pi}). \tag{16}$$

As the temperature parameter $\theta \to 0$, the function J^* in (16) converges to the optimal value function of the original control problem (without entropy regularization) in (9) under some technical conditions (Tang et al., 2022). Hence, solving the exploratory RL problem (16) when θ is small provides an approximate solution to (9).

Definition 3 *A policy $\boldsymbol{\pi} = \boldsymbol{\pi}(\cdot|\cdot, \cdot)$ is called admissible, if*

(i) *$\boldsymbol{\pi}(\cdot|t, y) \in \mathcal{P}(\mathbb{R}^d)$, supp $\boldsymbol{\pi}(\cdot|t, y) = \mathbb{R}^d$ for every $(t, y) \in [0, T] \times \mathbb{R}^d$ and $\boldsymbol{\pi}(a|t, y) : (t, y, a) \in [0, T] \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is measurable;*

(ii) *$\int_{\mathbb{R}^d} |\boldsymbol{\pi}(a|t, y) - \boldsymbol{\pi}(a|t', y')| \, da \to 0$ as $(t', y') \to (t, y)$. Moreover, there is a constant $C > 0$ independent of (t, a) such that*

$$\int_{\mathbb{R}^d} \big| \boldsymbol{\pi}(a|t, y) - \boldsymbol{\pi}(a|t, y') \big| \, da \le C|y - y'|, \ \forall y, y' \in \mathbb{R}^d;$$

(iii) *$\forall (t, y)$, $\int_{\mathbb{R}^d} |\log \boldsymbol{\pi}(a|t, y)| \boldsymbol{\pi}(a|t, y) da \le C(1 + |y|^p)$ for some $p \ge 1$ and C is a positive constant; for any $k \ge 1$, $\int_{\mathbb{R}^d} |a|^k \boldsymbol{\pi}(a|t, y) da \le C_k(1 + |y|^{k'})$ for some $k' \ge 1$ and C_k is a positive constant that can depend on k.*

3. The entropy regularization is a commonly used technique to improve exploration in RL, originally introduced for MDPs; see e.g. Haarnoja et al. (2018).

4. Theory

In this section, we present our main theoretical results, which provide the optimal stochastic policy to the RL problem in (16) with the system dynamic (10) and the running reward (13). Introduce the (generalized) Hamiltonian $H : [0,T] \times \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^{d\times d} \to \mathbb{R}$ associated with the original problem (9)–(10) (see e.g. Yong and Zhou, 2012):

$$\begin{aligned} &H(t,y,a,p,q) \\ &= -(g(T-t))^2|\nabla \log p_{T-t}(y) - a|^2 + [f(T-t)y + (g(T-t))^2 a] \circ p + \frac{1}{2}(g(T-t))^2 \circ q. \end{aligned} \tag{17}$$

Equation (13) of Jia and Zhou (2023) yields that the optimal stochastic policy $\boldsymbol{\pi}^*(\cdot|t,y)$ for the problem (16) is given by

$$\boldsymbol{\pi}^*(a|t,y) \propto \exp\left(\frac{1}{\theta} H(t,y,a,J_y^*(t,y),J_{yy}^*(t,y))\right),$$

where J_y^* and J_{yy}^* are the first and second order partial derivatives of J^* with respect to y, respectively. However, H in our case, (17), is quadratic in a, leading to the following result.

Proposition 4 *The optimal stochastic policy $\boldsymbol{\pi}^*(\cdot|t,y)$ is a Gaussian distribution in $\mathbb{R}^d$:*

$$\boldsymbol{\pi}^*(\cdot|t,y) \sim \mathcal{N}\left(\mu^*(t,y), \frac{\theta}{2g^2(T-t)} \cdot I_d\right), \tag{18}$$

where

$$\mu^*(t,y) = \nabla \log p_{T-t}(y) + \frac{1}{2} \cdot J_y^*(t,y). \tag{19}$$

This result provides several interesting insights. First, the mean $\mu^*(t,y)$ of the optimal stochastic policy consists of two parts: the score $\nabla \log p_{T-t}(y)$ that we try to match in (9), and an additional term $\frac{1}{2} \cdot J_y^*(t,y)$ that arises due to the consideration of maximizing the terminal reward h of the generated samples. The optimal value function $J^*(t,y)$ in (16) can be shown to satisfy the following so-called exploratory HJB equation, a nonlinear PDE (see Equation (14) of Jia and Zhou, 2023):

$$\begin{aligned} &\frac{\partial J^*}{\partial t}(t,y) + \theta \log\left[\int_{\mathbb{R}^d} \exp\left(\frac{1}{\theta} H(t,y,a,J_y^*,J_{yy}^*)\right) da\right] = 0, \\ &J^*(T,y) = \beta \cdot h(y). \end{aligned} \tag{20}$$

Note that we do not attempt to solve this PDE because the Hamiltonian H involves the unknown true score function (we never solve any HJB equations in the realm of RL). However, (20) illustrates the impacts of the terminal reward function h as well as the temperature parameter θ and the score $\nabla \log p_{T-t}(y)$ on the mean of the optimal Gaussian policy (19). We also observe from (18) that the optimal Gaussian policy has a covariance matrix $\frac{\theta}{2g^2(T-t)} \cdot I_d$, whose magnitude is, naturally, proportional to the temperature parameter $\theta > 0$. Meanwhile, the exploration level is inversely proportional to $g^2(T-t)$. This is intuitive because

g represents the strength of the noise we add to blur the original samples. The higher this noise the less *additional* noise we need for exploration.

Inspired by Proposition 4, in our RL algorithm to be presented in Section 6, we only need to consider Gaussian policies in the following form:

$$\boldsymbol{\pi}^{\psi}(\cdot|t,y) \sim \mathcal{N}\left(\mu^{\psi}(t,y), \frac{\theta}{2g^2(T-t)} \cdot I_d\right) \quad \text{for all } (t,y), \tag{21}$$

where the mean $\mu^{\psi}(t,y)$ is parameterised by some vector ψ.

5. Reward Signals

Before presenting our algorithm, we introduce a new component essential to its design. In general RL problems, we either have access to the reward function or have its (noisy) observations for learning. A unique feature of the current problem is that the running/instantaneous reward function (13) involves the unknown true score $\nabla \log p_{T-t}(\cdot)$. To obtain signals from this unknown running reward (or score), we express a score function value as the ratio of two expectations with respect to the data distribution.

Specifically, note that

$$p_t(\mathbf{x}) = \int_{\mathbb{R}^d} p_{t|0}(\mathbf{x}|\mathbf{x}_0) p_0(\mathbf{x}_0) d\mathbf{x}_0 = \mathbb{E}_{\mathbf{x}_0 \sim p_0}[p_{t|0}(\mathbf{x}|\mathbf{x}_0)],$$

where we recall that $p_{t|0}(\cdot|\mathbf{x}_0)$ is the conditional density of $\mathbf{x}_t$ given $\mathbf{x}_0$. By the forward process (2), we know that $p_{t|0}(\cdot|\mathbf{x}_0)$ is Gaussian with

$$p_{t|0}(\mathbf{x}|\mathbf{x}_0) = \frac{1}{\left(2\pi \int_0^t e^{-2\int_s^t f(v)dv} (g(s))^2 ds\right)^{d/2}} \exp\left(-\frac{\|\mathbf{x} - e^{-\int_0^t f(s)ds}\mathbf{x}_0\|^2}{2\int_0^t e^{-2\int_s^t f(v)dv}(g(s))^2 ds}\right). \tag{22}$$

It follows that

$$\begin{aligned}
\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) &= \frac{\nabla_{\mathbf{x}} p_t(\mathbf{x})}{p_t(\mathbf{x})} = \frac{\mathbb{E}_{\mathbf{x}_0 \sim p_0}[\nabla_{\mathbf{x}} p_{t|0}(\mathbf{x}|\mathbf{x}_0)]}{\mathbb{E}_{\mathbf{x}_0 \sim p_0}[p_{t|0}(\mathbf{x}|\mathbf{x}_0)]} \\
&= \frac{\mathbb{E}_{\mathbf{x}_0 \sim p_0}\left[p_{t|0}(\mathbf{x}|\mathbf{x}_0) \cdot \frac{-(\mathbf{x} - e^{-\int_0^t f(s)ds}\mathbf{x}_0)}{\int_0^t e^{-2\int_s^t f(v)dv}(g(s))^2 ds}\right]}{\mathbb{E}_{\mathbf{x}_0 \sim p_0}[p_{t|0}(\mathbf{x}|\mathbf{x}_0)]} \\
&= \frac{1}{\int_0^t e^{-2\int_s^t f(v)dv}(g(s))^2 ds} \cdot \left(-\mathbf{x} + \frac{\mathbb{E}_{\mathbf{x}_0 \sim p_0}\left[p_{t|0}(\mathbf{x}|\mathbf{x}_0) \cdot \mathbf{x}_0\right]}{\mathbb{E}_{\mathbf{x}_0 \sim p_0}[p_{t|0}(\mathbf{x}|\mathbf{x}_0)]} \cdot e^{-\int_0^t f(s)ds}\right).
\end{aligned} \tag{23}$$

Therefore, given m i.i.d. samples $(\mathbf{x}_0^i)$ from the data distribution p_0, a simple ratio estimator for the true score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ at given $(t,\mathbf{x})$ is

$$\widehat{\nabla_{\mathbf{x}} \log p_t}(\mathbf{x}) = \frac{1}{\int_0^t e^{-2\int_s^t f(v)dv}(g(s))^2 ds} \cdot \left(-\mathbf{x} + \frac{\sum_{i=1}^m \left[p_{t|0}(\mathbf{x}|\mathbf{x}_0^i) \cdot \mathbf{x}_0^i\right]}{\max\{\sum_{i=1}^m [p_{t|0}(\mathbf{x}|\mathbf{x}_0^i)], m\epsilon\}} \cdot e^{-\int_0^t f(s)ds}\right), \tag{24}$$

where $\epsilon > 0$ is some prespecified small constant. Note that we use $\max\{\sum_{i=1}^m [p_{t|0}(\mathbf{x}|\mathbf{x}_0^i)], m\epsilon\}$ instead of $\sum_{i=1}^m [p_{t|0}(\mathbf{x}|\mathbf{x}_0^i)]$ in (24) to avoid division by extremely small numbers for numerical stability. In view of the running reward function r in (13), if we define

$$\hat{r}(t, \mathbf{y}, a) = -g^2(T-t) \cdot |\nabla \widehat{\log p_{T-t}}(\mathbf{y}) - a|^2, \tag{25}$$

then a noisy observation of the running reward at time t is

$$\hat{r}_t = \hat{r}(t, \mathbf{y}_t, a_t). \tag{26}$$

For fixed $(t, \mathbf{x})$, the simple ratio estimator in (24) provides a nonparametric approach to estimate the true score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$. It is efficient to compute the estimator, because $p_{t|0}(\mathbf{x}|\mathbf{x}_0)$ has an explicit form. In addition, while this ratio estimator is generally biased, it is asymptotically exact as the number of data samples $m \to \infty$ by the strong law of large numbers. On the other hand, the true score function is a function of *all* $(t, \mathbf{x})$. Thus, using this method to accurately estimate the score function for every $(t, \mathbf{x})$ is simply impossible. In our study, the estimator (24) allows us to compute the reward signal (26) as we go, namely we compute it only when a state–action pair $(\mathbf{y}_t, a_t)$ is visited at time $t < T$. On the other hand, it is even unnecessary to use up all the data samples from p_0 each time in computing (24), though in theory a larger sample size will generally lead to a lower variance of the running reward signal (26). In the training process, suppose that we have M training samples in total, it is more practical to use $m < M$ samples to estimate the score function value and compute the reward signal, so that the computational cost can be significantly reduced. Indeed, later in an image generation task we will show that even $m = 1$ suffices. Details will be discussed in the subsequent experiments.

Remark 5 *Given the estimator* (24) *of the score function value* $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$*, it seems tempting, at least for the pure score-matching problem (without preference), to consider directly running the reverse process* (6) *with the ratio estimator to generate new samples, i.e. to produce samples* $\mathbf{y}_T$ *via simulating*

$$d\mathbf{y}_t = \left[f(T-t)\mathbf{y}_t + (g(T-t))^2 \nabla \widehat{\log p_{T-t}}(\mathbf{y}_t) \right] dt + g(T-t) dW_t, \quad \mathbf{y}_0 \sim \nu. \tag{27}$$

However, this approach essentially replicates those training samples rather than generating new *samples from the target distribution. Indeed, assuming we always use all the* M *training samples to estimate the score function values* $\nabla_{\mathbf{x}} \widehat{\log p_t}(\mathbf{x})$ *in* (24)*, according to the derivation in* (23) *we have*

$$\begin{aligned} \nabla_{\mathbf{x}} \widehat{\log p_t}(\mathbf{x}) &= \frac{1}{\int_0^t e^{-2\int_s^t f(v)dv} (g(s))^2 ds} \cdot \left(-\mathbf{x} + \frac{\mathbb{E}_{\mathbf{x}_0 \sim \hat{p}_0} \left[p_{t|0}(\mathbf{x}|\mathbf{x}_0) \cdot \mathbf{x}_0 \right]}{\mathbb{E}_{\mathbf{x}_0 \sim \hat{p}_0} [p_{t|0}(\mathbf{x}|\mathbf{x}_0)]} \cdot e^{-\int_0^t f(s)ds} \right) \\ &= \frac{\mathbb{E}_{\mathbf{x}_0 \sim \hat{p}_0} [\nabla_{\mathbf{x}} p_{t|0}(\mathbf{x}|\mathbf{x}_0)]}{\mathbb{E}_{\mathbf{x}_0 \sim \hat{p}_0} [p_{t|0}(\mathbf{x}|\mathbf{x}_0)]} = \nabla_{\mathbf{x}} \log \hat{p}_t(\mathbf{x}) \end{aligned}$$

where $\hat{p}_0$ *is the empirical distribution of the samples* $\mathbf{x}_0^1, \cdots \mathbf{x}_0^M$ *and* $\hat{p}_t$ *is the time marginal distribution of the forward process* (1) *but starting from* $\mathbf{x}_0 \sim \hat{p}_0$*. Then running the reverse process* (27) *essentially generates samples from* $\hat{p}_0$*, i.e. replicating samples in the training*

dataset. It is therefore not a truly generative model for it can not generate new and diverse samples that are different from the training data. In our continuous-time RL approach for training reward-directed diffusion models (which cover the pure score-matching as a special case), we use the ratio estimator only for generating running reward signals, and employ a neural network to parameterize the policy which is related to the true score; see (21) *and* (19).

6. q-Learning Algorithm and Convergence

In this section, we present an algorithm to solve the continuous-time RL problem (16) and discuss its convergence.

6.1 q-Learning Algorithm

Our algorithm is based on the general q-learning algorithms developed recently in Jia and Zhou (2023, 2025), which are of the actor–critic type. For the reader's convenience, we first introduce some definitions and theoretical results in Jia and Zhou (2023, 2025) that are important for developing our algorithm.

Following Jia and Zhou (2023), we define the so-called optimal q-function by

$$q^*(t,y,a) = \frac{\partial J^*(t,y)}{\partial t} + H(t,y,a,J_y^*(t,y),J_{yy}^*(t,y),J^*(t,y)), \quad (t,y,a) \in [0,T]\times\mathbb{R}^d\times\mathbb{R}^d,$$

where J^* is the optimal value function given in (16), and the Hamiltonian function H is defined in (17). One can readily infer from (20) that (see Proposition 8 in Jia and Zhou, 2023)

$$\int_{\mathbb{R}^d} \exp\left(\frac{1}{\theta}q^*(t,y,a)\right) da = 1, \quad \text{for all } (t,y)\in[0,T]\times\mathbb{R}^d,$$

and the optimal stochastic policy $\boldsymbol{\pi}^*(\cdot|t,y)$ in Proposition 4 is simply

$$\boldsymbol{\pi}^*(a|t,y) = \exp\left(\frac{1}{\theta}q^*(t,y,a)\right).$$

Note that the exploratory (or "averaged") dynamic in (14) is introduced mainly for the purpose of theoretical analysis. Its solutions are not observable (i.e. not data), and hence can not be used in the actual implementation of RL algorithms. We next introduce the discretely sampled state processes, which can be observed based on the agent interactions with the environment. Given an admissible feedback policy $\boldsymbol{\pi}$ and $(t,y)\in[0,T)\times\mathbb{R}^d$, consider a time grid $\mathbb{S} = \{t = t_0 < t_1 < \ldots < t_K = T\}$ of $[t,T]$. In algorithmic implementations, we sample actions from $\boldsymbol{\pi}$ only at the grid points in $\mathbb{S}$. The corresponding state process, which is referred to as the *discretely sampled state process*, satisfies the following SDE: for all $i = 0,...,K-1$ and all $s\in[t_i,t_{i+1})$,

$$d\mathbf{y}_s^{\boldsymbol{\pi},\mathbb{S}} = [f(T-s)\mathbf{y}_s^{\boldsymbol{\pi},\mathbb{S}} + (g(T-s))^2\mathbf{a}_{t_i}^{\boldsymbol{\pi}}]ds + g(T-s)dW_s, \tag{28}$$

where $\mathbf{a}_{t_i}^{\boldsymbol{\pi}} = \phi(t_i,\mathbf{y}_{t_i}^{\boldsymbol{\pi},\mathbb{S}},U_{i+1})$. Here, the function $\phi : (t,y,u)\in[0,T]\times\mathbb{R}^d\times[0,1]^d \to \phi(t,y,u)\in\mathbb{R}^d$ is given such that $\phi(t,y,U)\sim\boldsymbol{\pi}(\cdot|t,y)$ where U is a uniform random vector

on $[0,1]^d$. The random vectors (U_i) are i.i.d d-dimensional uniform random vectors, and they are independent of the Brownian motion W. We also define the action process $a_s^{\boldsymbol{\pi},\mathbb{S}} = \mathbf{a}_{t_i}^{\boldsymbol{\pi}}$ for $s \in [t_i, t_{i+1})$. Hence, the action is a constant over each subinterval $[t_i, t_{i+1})$. Finally, we denote by $\mathcal{F}^{\mathbb{S}}$ the (right-continuous) filtration generated by the Brownian motion W and the random vectors (U_i). For further details about the underlying probability space associated with the discretely sampled state process, we refer readers to Jia and Zhou (2025).

We next state the martingale condition that characterizes the optimal value function J^* and the optimal q-function; see Theorem 9 in (Jia and Zhou, 2025). This result is essential because it provides the theoretical foundation for designing our q-learning algorithm.

Proposition 6 *Let a function $\hat{J}^* \in C^{1,2}([0,T) \times \mathbb{R}_+) \cap C([0,T] \times \mathbb{R}_+)$ and a continuous function $\hat{q}^* : [0,T] \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be given satisfying*

$$\hat{J}^*(T,y) = h(y), \quad \int_{\mathbb{R}^d} \exp\left(\frac{1}{\theta}\hat{q}^*(t,y,a)\right) da = 1, \quad \text{for all } (t,y) \in [0,T] \times \mathbb{R}^d.$$

Assume that $\hat{J}^*$ and $\hat{J}_y^*$ both have polynomial growth. Then

(i) If $\hat{J}^*$ and $\hat{q}^*$ are respectively the optimal value function and the optimal q-function, then for any $\boldsymbol{\pi} \in \boldsymbol{\Pi}$, for all $(t,y) \in [0,T] \times \mathbb{R}^d$ and any grid $\mathbb{S}$ of $[t,T]$, the following process

$$\hat{J}^*(s, \mathbf{y}_s^{\boldsymbol{\pi},\mathbb{S}}) + \int_t^s [r(\tau, \mathbf{y}_\tau^{\boldsymbol{\pi},\mathbb{S}}, a_\tau^{\boldsymbol{\pi},\mathbb{S}}) - \hat{q}^*(\tau, \mathbf{y}_\tau^{\boldsymbol{\pi},\mathbb{S}}, a_\tau^{\boldsymbol{\pi},\mathbb{S}})] d\tau \tag{29}$$

is an $\mathcal{F}^{\mathbb{S}}$-martingale, where $\mathbf{y}^{\boldsymbol{\pi},\mathbb{S}} = \{\mathbf{y}_s^{\boldsymbol{\pi},\mathbb{S}} : t \le s \le T\}$ satisfies (28) with $\mathbf{y}_t^{\boldsymbol{\pi},\mathbb{S}} = y$.

(ii) If there exists one $\boldsymbol{\pi} \in \boldsymbol{\Pi}$ such that for all (t,y) and any grid $\mathbb{S}$ of $[t,T]$, the process (29) is an $\mathcal{F}^{\mathbb{S}}$-martingale where $\mathbf{y}_t^{\boldsymbol{\pi},\mathbb{S}} = y$, then $\hat{J}^*$ and $\hat{q}^*$ are respectively the optimal value function and the optimal q-function.

We are now ready to develop and state the algorithm to solve our continuous-time RL problem (16). Consider the parameterized Gaussian policies $\boldsymbol{\pi}^\psi(\cdot|t,y)$ in (21). It is useful to note that

$$\boldsymbol{\pi}^\psi(a|t,y) = \exp\left(\frac{1}{\theta}q^\psi(t,y,a)\right),$$

where

$$q^\psi(t,y,a) = -g^2(T-t) \cdot |a - \mu^\psi(t,y)|^2 - \frac{\theta d}{2}\log\left(\frac{\pi\theta}{g^2(T-t)}\right). \tag{30}$$

This function q^ψ is a resulting parameterization of the optimal q-function q^* which satisfies the equation $\int_{\mathbb{R}^d} \exp\left(\frac{1}{\theta}q^\psi(t,y,a)\right) da = 1$. We also choose the value function approximator J^Θ parametrized by Θ for the optimal value function J^*. We aim to learn the optimal value function and q-function simultaneously by updating (Θ, ψ), based on the martingale characterization in Proposition 6. In particular, we follow Jia and Zhou (2023) and use the so-called martingale orthogonality conditions. These conditions state that a diffusion

process M is a (square-integrable) martingale with respect to a filtration $\mathcal{F}$ if and only if $\mathbb{E}\int_0^T H_t dM_t = 0$ for any process H (called a test function) on $[0,T]$ that is progressively measurable (with respect to $\mathcal{F}$) with $\mathbb{E}[\int_0^T |H_t|^2 d\langle M\rangle_t] < \infty$, where $\langle M\rangle_t$ denotes the quadratic variation of the martingale M. See (Jia and Zhou, 2022a, Section 4.2).

Because the optimal value function J^* and q-function q^* are both infinite-dimensional, theoretically one should vary all possible test functions to obtain infinitely many equations from the martingale orthogonality conditions in order to determine (J^*, q^*). This is naturally infeasible for computer implementation. Numerically we need only to determine the parameters $(\Theta, \psi) \in \mathbb{R}^k$, for some k, of the function approximators (J^Θ, q^ψ). This calls for k test functions. We choose the following test functions ξ_t and ζ_t for our problem:

$$\xi_t = \frac{\partial J^\Theta}{\partial \Theta}(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}), \quad \zeta_t = \frac{\partial q^\psi}{\partial \psi}(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi}) = \frac{\partial}{\partial \psi}\log \pi^\psi(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi}), \tag{31}$$

where we use the fact that $q^\psi(t,y,a) = \theta \log \boldsymbol{\pi}^\psi(a|t,y)$. The above choice of ξ_t is motivated by the TD(0) method in discrete-time RL (e.g. Bhandari et al., 2021; see Jia and Zhou, 2022a, Section 4.2 for a detailed discussion), whereas that of ζ_t is inspired by the policy gradient method (Schulman et al., 2015; Jia and Zhou, 2022b).

In our implementation, we execute stochastic policies on a uniform time grid with step size Δt; that is, the grid $\mathbb{S} = \{0 = t_0 < t_1 < \ldots < t_K = T\}$, where $t_{i+1} - t_i = \Delta t$ for all i. For notational simplicity, in (31) and the discussions below we denote the discretely sampled state process $\mathbf{y}_t^{\boldsymbol{\pi}^\psi, \mathbb{S}}$ by $\mathbf{y}_t^{\boldsymbol{\pi}^\psi}$. Likewise, the (piecewise constant) action process is denoted by $a_t^{\boldsymbol{\pi}^\psi}$.

The martingale characterization in Proposition 6 together with the martingale orthogonality conditions lead to the following system of equations in (Θ, ψ):

$$\mathbb{E}\left[\int_0^T \xi_t \left(dJ^\Theta(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}) + r(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})dt - q^\psi(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})dt\right)\right] = 0, \tag{32}$$

$$\mathbb{E}\left[\int_0^T \zeta_t \left(dJ^\Theta(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}) + r(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})dt - q^\psi(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})dt\right)\right] = 0. \tag{33}$$

Equations (32)-(33) have k equations used to determine $(\Theta, \psi) \in \mathbb{R}^k$. We solve these equations using stochastic approximation (SA) and replace the true (unknown) running reward function r by its noisy estimate $\hat{r}$ in (25) to update (Θ, ψ):

$$\Theta \leftarrow \Theta + \alpha_\Theta \int_0^T \xi_t \left(dJ^\Theta(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}) + \hat{r}(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})dt - q^\psi(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})dt\right), \tag{34}$$

$$\psi \leftarrow \psi + \alpha_\psi \int_0^T \zeta_t \left(dJ^\Theta(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}) + \hat{r}(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})dt - q^\psi(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})dt\right). \tag{35}$$

The above updating rule leads to an actor–critic type algorithm based on the temporal difference $dJ^\Theta(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi})$. Algorithm 1 gives the pseudocode. We also remark that it is possible to consider other test functions in (32)–(33), which will lead to different algorithms; see Jia and Zhou (2022a, 2023) for further discussions.

Algorithm 1 q-Learning Algorithm (SDE-based unconditional generation)

Inputs: M samples from data distribution, number of samples m that used to estimate the score function, horizon T, time step Δt, number of episodes N, number of mesh grids $K = T/\Delta t$, initial learning rates $\alpha_\Theta, \alpha_\psi$ and a learning rate schedule function $l(\cdot)$ (a function of the number of episodes), functional forms of parameterized value function $J^\Theta(\cdot,\cdot)$ and $\mu^\psi(\cdot,\cdot)$, temperature parameter θ, functions f, g in (1), and ϵ in (24).

Required program: environment simulator $(y', \hat{r}) = \mathit{Environment}_{\Delta t}(t, y, a, m)$ that takes current time–state pair (t, y) and action a as inputs and generates state y' (by a numerical solver of SDE (28)) at time $t+\Delta t$ and sample instantaneous reward $\hat{r}$ (by randomly selecting m samples from the dataset and implementing (26)) at time t as outputs. Policy $\boldsymbol{\pi}^\psi(\cdot|t, y)$ in (21), and q-function $q^\psi(t, y, a)$ in (30).

Learning procedure:

Initialize Θ, ψ.

for episode $j = 1$ **to** N **do**

Initialize $k = 0$. Sample initial state $y_0 \sim \nu$ and store $y_{t_k} \leftarrow y_0$.

while $k < K$ **do**

Generate action $a_{t_k} \sim \boldsymbol{\pi}^\psi(\cdot|t_k, y_{t_k})$.

Apply a_{t_k} to environment simulator $(y, \hat{r}) = \mathit{Environment}_{\Delta t}(t_k, y_{t_k}, a_{t_k}, m)$, and observe new state y and reward $\hat{r}$ as outputs. Store $y_{t_{k+1}} \leftarrow y$ and $r_{t_k} \leftarrow \hat{r}$.

Update $k \leftarrow k + 1$.

end while

For every $i = 0, 1, \cdots, K-1$, compute and store test functions

$$\xi_{t_i} = \frac{\partial J^\Theta}{\partial \Theta}(t_i, y_{t_i}), \quad \zeta_{t_i} = \frac{\partial q^\psi}{\partial \psi}(t_i, y_{t_i}, a_{t_i}).$$

Compute

$$\Delta\Theta = \sum_{i=0}^{K-1} \xi_{t_i}\big[J^\Theta(t_{i+1}, y_{t_{i+1}}) - J^\Theta(t_i, y_{t_i}) + r_{t_i}\Delta t - q^\psi(t_i, y_{t_i}, a_{t_i})\Delta t\big],$$

$$\Delta\psi = \sum_{i=0}^{K-1} \zeta_{t_i}\big[J^\Theta(t_{i+1}, y_{t_{i+1}}) - J^\Theta(t_i, y_{t_i}) + r_{t_i}\Delta t - q^\psi(t_i, y_{t_i}, a_{t_i})\Delta t\big].$$

Update Θ and ψ by

$$\Theta \leftarrow \Theta + l(j)\alpha_\Theta \Delta\Theta, \tag{36}$$

$$\psi \leftarrow \psi + l(j)\alpha_\psi \Delta\psi. \tag{37}$$

end for

Remark 7 *In Algorithm 1, the update rule for* (Θ, ψ) *in* (36)-(37) *is essentially the stochastic gradient descent (SGD) type method. To see this, define*

$$G(t_i; \Theta, \psi) := \hat{J}(t_{i+1}, y_{t_{i+1}}) - J^{\Theta}(t_i, y_{t_i}) + r_{t_i}\Delta t - q^{\psi}(t_i, y_{t_i}, a_{t_i})\Delta t,$$

where $\hat{J}(t_{i+1}, y_{t_{i+1}}) := J^{\Theta}(t_{i+1}, y_{t_{i+1}})$, $i = 0, ..., K-1$, *which however are treated as constants free of parameter* Θ. *Then*

$$\Delta\Theta = -\frac{1}{2}\sum_{i=0}^{K-1}\frac{\partial}{\partial\Theta}\left[G(t_i; \Theta, \psi)\right]^2, \quad \Delta\psi = -\frac{1}{2\Delta t}\sum_{i=0}^{K-1}\frac{\partial}{\partial\psi}\left[G(t_i; \Theta, \psi)\right]^2.$$

Therefore, the update of (Θ, ψ) *in* (36)-(37) *can be considered as a gradient descent method for minimizing the following mean-square loss function:*

$$L(\Theta, \psi) := \sum_{i=0}^{K-1}\left[G(t_i; \Theta, \psi)\right]^2. \tag{38}$$

As a result, instead of the SA in (36)-(37), *we can also optimize the loss function* (38) *by applying other optimization methods such as the Adam optimizer (Kingma and Ba, 2015) to update these parameters. In our experiments, we use Adam because it converges much faster than SA/SGD. In addition, to further accelerate the training process, the above loss function could be estimated more accurately through the empirical mean of a batch of losses under the same* (Θ, ψ). *Specifically, we collect a batch of* B *trajectories simultaneously, calculate the above loss for each of the* B *sample paths, denoted by* $L^{(b)}(\Theta, \psi), b = 1, ..., B$, *and then use the following batch loss for optimization:*

$$L_{batch}(\Theta, \psi) := \frac{1}{B}\sum_{b=1}^{B} L^{(b)}(\Theta, \psi).$$

Remark 8 *In the classical setting of training diffusion models using score matching, the training data (i.e. samples from* p_0*) are used in Monte Carlo approximation of the expectation in the (denoising) score matching objective* (8). *In our RL formulation, however, the training data are used for approximating the score values via the ratio estimators in* (24) *in order to obtain noisy samples of the running reward. This again highlights one of the key differences between the two approaches.*

6.2 Convergence

In this section, we discuss the convergence of the q-learning algorithm. We focus on the analysis of the SA scheme in (34)-(35) for updating (Θ, ψ). In particular, we do not consider errors of time discretization of the integrals in actual implementations of the $q-$learning algorithm, Algorithm 1, because including these errors would only complicate the analysis and cloud the key insights from $q-$learning. In implementation, such errors can be controlled by appropriately choosing the time steps; see a recent paper Jia et al. (2025) for a detailed study.

To analyze the scheme (34)-(35) for updating (Θ, ψ), we rewrite it as follows: for each episode n,

$$(\Theta_{n+1}, \psi_{n+1}) = (\Theta_n, \psi_n) + (\alpha_\Theta(n+1), \alpha_\psi(n+1)) \cdot \mathcal{H}\bigg((\Theta_n, \psi_n), \underbrace{(\mathbf{y}_t^{\boldsymbol{\pi}^{\psi_n}}, a_t^{\boldsymbol{\pi}^{\psi_n}})_{t\in[0,T]}}_{\Xi_{n+1}}, \mathcal{Z} \bigg). \tag{39}$$

Here, we have

$$\mathcal{H}\big((\Theta, \psi), \Xi, \mathcal{Z}\big) := \left(\int_0^T \xi_t d\mathcal{G}(t; \Theta, \psi), \int_0^T \zeta_t d\mathcal{G}(t; \Theta, \psi) \right),$$

with ξ_t and ζ_t being the test functions in (31), $\Xi = (\mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})_{t\in[0,T]}$,

$$d\mathcal{G}(t; \Theta, \psi) := dJ^\Theta(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}) + \hat{r}(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})dt - q^\psi(t, \mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})dt,$$

and $\mathcal{Z} = (\mathcal{Z}_t)_{t\in\mathbb{S}}$ where $\mathcal{Z}_t$ is a mini batch of m data points independently sampled from the whole dataset (of size M) at each grid point t in order to estimate the score function value there, and $\mathcal{Z}$ is independent of Ξ_{n+1} for all n. Note that we allow time-varying learning rates $(\alpha_\Theta(n+1), \alpha_\psi(n+1))$, which helps in the convergence analysis of SA schemes (Kushner and Yin, 2003). The update scheme (39) is an instance of the Robbins–Monro algorithm (see e.g. p.343 of Benveniste et al., 2012). This is because in each episode n, the initial state $\mathbf{y}_0$ is sampled independently from a Gaussian distribution ν in (5), which implies that the conditional distribution of $(\Xi_{n+1}, \mathcal{Z})$, knowing the past, depends only on ψ_n. Define

$$e(\Theta, \psi) := \mathbb{E}\big[\mathcal{H}((\Theta, \psi), \Xi, \mathcal{Z})\big|\Theta, \psi\big],$$

where the expectation is taken with respect to the distribution of $\mathcal{Z} = (\mathcal{Z}_t)_{t\in\mathbb{S}}$ and the path measure induced by $\Xi = (\mathbf{y}_t^{\boldsymbol{\pi}^\psi}, a_t^{\boldsymbol{\pi}^\psi})_{t\in[0,T]}$ when Θ and ψ are given.

We can now state a convergence result for (39), which follows from Chapter 5.3 in Part II of Benveniste et al. (2012).

Proposition 9 *Assume the following conditions hold:*

1. *There exist a twice continuously differentiable Lyapunov function V with a uniformly bounded second derivative and (Θ^*, ψ^*) satisfying*

$$\sup_{\epsilon \le |(\Theta-\Theta^*, \psi-\psi^*)| \le 1/\epsilon} \nabla V(\Theta, \psi) \cdot e(\Theta, \psi) < 0, \quad \text{for all } \epsilon > 0.$$

   Moreover, $\mathbb{E}\big[|\mathcal{H}((\Theta, \psi,), \Xi, \mathcal{Z})|^2\big|\Theta, \psi\big] \le C(1 + V(\Theta, \psi))$ for some $C > 0$.

2. *The learning rates satisfy $\sum_n \alpha_\gamma(n) = \infty$ and $\sum_n \alpha_\gamma(n)^2 < \infty$ for $\gamma = \Theta, \psi$.*

Then the sequence (Θ_n, ψ_n) in (39) converges to (Θ^*, ψ^*) almost surely.

The second assumption in Proposition 9 is standard that guides the choices of learning rates. Verifying the first assumption, however, is nontrivial. The difficulty arises mainly from the fact that our setting involves continuous time and continuous state/action spaces, and the Lyapunov function depends on the specific choice of the function approximations (neural networks) of the value function and the q-function (i.e. J^Θ and q^ψ). As such, a rigorous verification of this assumption (i.e. finding the desired Lyapunov function) is beyond the scope of this work and is left for future investigation.

7. Experiments for Two Toy Examples

In this section, we implement Algorithm 1 and conduct "proof-of-concept" experiments for two toy examples with synthetic training data, one involving a one-dimensional (1D) Gaussian mixture distribution and the other a two-dimensional (2D) Swiss rolls dataset. We further show the effectiveness of our RL approach by comparing its performance against that of two state-of-the-art RL fine-tuning methods.

7.1 General setup

We first provide some details on the implementation, including the environment (i.e. SDE model (28)) and its simulator, as well as the structure of the neural networks used for the parameterized actor π^{ψ} and critic J^{Θ}.

- **SDE/Environment simulator.**

  The implementation of Algorithm 1 requires an environment simulator describing the (controlled) denoising process. This corresponds to a numerical approximation of the controlled SDE (28).

  For (28), we take

  $$f(t) \equiv 1, \quad g(t) \equiv \sqrt{2}, \quad 0 \le t \le T, \tag{40}$$

  for both the 1D and 2D examples, leading to the prior distribution ν in (5) as

  $$\nu := \mathcal{N}\left(\mathbf{0}, \left(1 - e^{-2T}\right) \cdot I_d\right), \quad d = 1, 2.$$

  Other forms of f and g can also be considered, but we choose (40) because it is simple and such a choice already yields satisfactory numerical results as we will see below. We numerically solve the controlled SDE (28) via the standard Euler–Maruyama discretization, i.e., for some small $\Delta t > 0$ and $t < T$, $\mathbf{y}^{\boldsymbol{\pi}}_{t+\Delta t} - \mathbf{y}^{\boldsymbol{\pi}}_t \approx \left[f(T-t)\mathbf{y}^{\boldsymbol{\pi}}_t + (g(T-t))^2 a^{\boldsymbol{\pi}}_t\right] \cdot \Delta t + g(T-t)\sqrt{\Delta t} \cdot \xi$, where $\xi \sim \mathcal{N}(\mathbf{0}, I_d)$. The SDE (or environment) simulator under a policy $\boldsymbol{\pi}$ is given by (with a slight abuse of notation for $\mathbf{y}^{\boldsymbol{\pi}}_t$)

  $$\mathbf{y}^{\boldsymbol{\pi}}_{t+\Delta t} - \mathbf{y}^{\boldsymbol{\pi}}_t = \left[\mathbf{y}^{\boldsymbol{\pi}}_t + 2a^{\boldsymbol{\pi}}_t\right] \Delta t + \sqrt{2\Delta t} \cdot \xi, \quad \mathbf{y}^{\boldsymbol{\pi}}_0 \sim \nu. \tag{41}$$

- **Neural network approximators.**

  We use two neural networks (NNs) for the policy μ^{ψ} (i.e., the mean of the Gaussian policy in (21)) and value function J^{Θ}, respectively. Because we consider low-dimensional examples, the structure of those two NNs are similar, except that the output dimensions are different; see Table 1 below for details. As noted in Remark 7, we choose the Adam optimizer to optimize these two NNs, and we will specify the hyperparameters values for the training processes later.

Layer ID	Input Dimension	Output Dimension	Activation Function
1 (Input)	$d+1$	64	Tanh
2 (Hidden)	64	64	ReLU
3 (Output)	64	$\mu^\psi : d, \quad J^\Theta : 1$	

Table 1: Neural networks setting

7.2 1D example - mixed Gaussian

We first consider an example where the RL agent is provided with $m = 300$ samples drawn from a mixture of two 1D Gaussians:

$$p_0 = \frac{1}{2}\mathcal{N}(-3, 1) + \frac{1}{2}\mathcal{N}(3, 1).$$

We consider a reward function

$$h(y) = -(y - 6)^2, \tag{42}$$

indicating the agent's preference for the generated samples to be close to 6 (which is located at the right tail of p_0).

In our experiment, we consider different β's and implement Algorithm 1 under hyperparameters shown in Table 2. For each trained model with a particular value of β, we generate 300 samples (i.e. 300 terminal states y_T from the SDE simulator (41)). The resulting probability density functions (PDFs), computed via kernel density estimation using the Python seaborn library, are plotted in Figure 1.

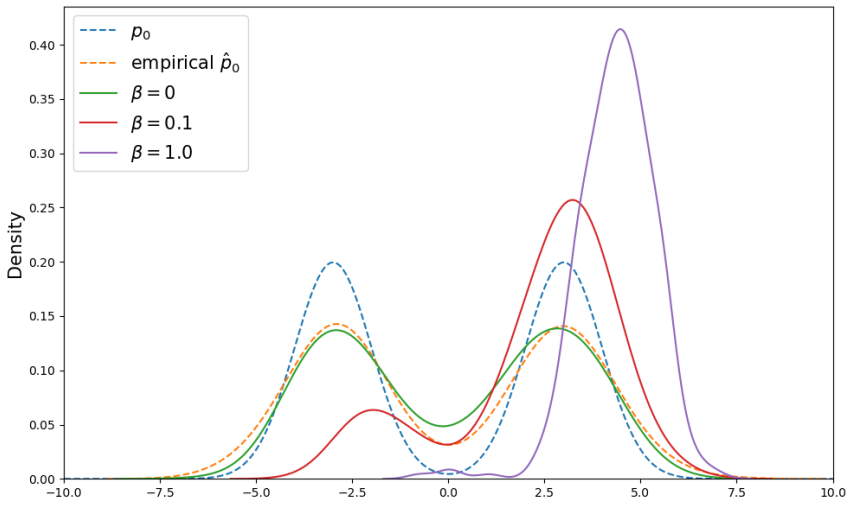


Figure 1: Learned probability densities of generated samples under different β's in 1D example

We have the following observations from Figure 1. First, when $\beta = 0$, our continuous-time RL formulation (9) essentially reduces to score matching, and our RL algorithm indeed generates samples with a PDF that matches closely with the empirical PDF $\hat{p}_0$ that are

computed using the 300 samples from the Gaussian mixture distribution. It is worthwhile to note that the empirical PDF still deviates away from the true density p_0 due to the small number of samples ($m = 300$) taken, and the two densities will become closer with a larger sample size. Second, when β is 0.1, our RL algorithm generates more samples that are closer to the right mode of the Gaussian mixture distribution, due to the specific reward function (42). Finally, when β is large, say 1, the reward term $\beta \cdot h$ dominates the score matching term in (9). In this case, the algorithm generates samples that are closer to 6, and the generated distribution is significantly different from the original data distribution.

Inputs/Hyperparameters	Setting
Sample Size m	300
Terminal Time T	5
Time Step Δt	0.25
Number of Episodes	50,000
Batch Size B	300
Learning Rate α_ψ	0.001
Learning Rate α_Θ	0.001
Scheduler $l(episode)$	1
Temperature θ	5
Lower Bound ϵ	10^{-20}

Table 2: Hyperparameters

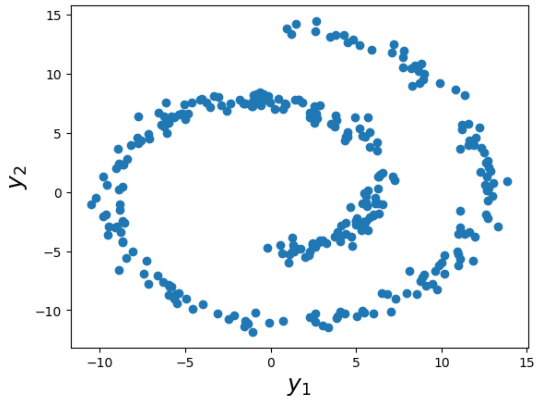


Figure 2: 300 samples from Swiss roll data

Before proceeding, we briefly discuss the choice of hyperparameters presented in Table 2. Key hyperparameters that impact the RL training process include $T, \Delta t, \alpha_\psi, \alpha_\Theta$, and θ. We employ a grid search to determine the optimal combination of these hyperparameters. The search space included: $T \in \{0.1, 0.5, 1, 2, 5, 10\}$, $\Delta t \in \{0.01, 0.025, 0.05, 0.1, 0.25, 0.5\}$, $\alpha_\psi = \alpha_\Theta \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$, and $\theta \in \{j \times 10^k : \forall j = 1, 5, k = -3, -2, -1, 0, 1\}$. By tuning these key hyperparameters, we aim to achieve a balance between sample quality and computational efficiency in our training process.

7.3 2D example - Swiss roll

We next consider 2D Swiss Rolls data, and plot all $M = 300$ training samples in the dataset in Figure 2.[4] Consider a non-differentiable reward function

$$h(y_1, y_2) = 1_{y_1 \in [-5,6]}, \quad (y_1, y_2) \in \mathbb{R}^2.$$

This reward function encourages generated samples to stay within the rectangular region $[-5, 6] \times \mathbb{R}$.

We again consider different β's and implement Algorithm 1 with hyperparameters shown in Table 2. The numerical results are visualized in Figure 3. Similar to the 1D example, when $\beta = 0$, our RL algorithm produces samples distributed close to the data distribution

4. We use the dataset provided by the sklearn package of Python in `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_swiss_roll.html`. This dataset is also used in other studies on diffusion models; see e.g. Sohl-Dickstein et al. (2015b) and Lai et al. (2023).

shown in Figure 2. As we increase the value of β, the generated samples become more concentrated on the rectangular region $[-5, 6] \times \mathbb{R}$. When β is large enough, say 30, all the generated samples tend to stay inside the region where $y_1 \in [-5, 6]$, visually resembling a French croissant.

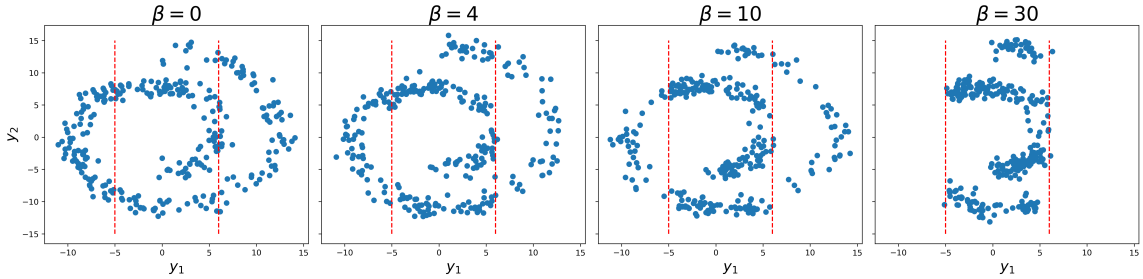


Figure 3: 300 samples generated from diffusion models trained under different terminal weights β for 2D Swiss rolls.

Sample Size m	$\beta = 0$	$\beta > 0$	
	KL$(p(\mathbf{y}_T)\|\|p_0) \downarrow$	Reward $\mathbb{E}[h(\mathbf{y}_T)]$	KL$(p(\mathbf{y}_T)\|\|p_0) \downarrow$
30	0.33 ± 0.017	0.80 ± 0.006	0.88 ± 0.026
60	0.32 ± 0.016	0.80 ± 0.004	0.51 ± 0.027
90	0.20 ± 0.019	0.80 ± 0.004	0.42 ± 0.025
120	$\mathbf{0.17 \pm 0.016}$	0.80 ± 0.004	0.40 ± 0.029
150	0.18 ± 0.016	0.80 ± 0.004	$\mathbf{0.38 \pm 0.025}$
300	$\mathbf{0.17 \pm 0.019}$	0.80 ± 0.003	0.39 ± 0.026

Table 3: Effect of sample size m.

We now discuss the effect of the sample size m in the score estimator (24) on the performance of our q-learning algorithm. As described in Algorithm 1, whenever the algorithm runs the environment simulator and generates the running reward signal $\hat{r}$, the simulator will first randomly select m samples from the dataset of M training samples, and then calculate $\nabla_{\mathbf{x}} \widehat{\log p_t}(\mathbf{x})$ and $\hat{r}$ according to (24) and (26), respectively. To see the effect of sample size m on the performance of Algorithm 1, we conduct two different experiments. In the first one, we run Algorithm 1 without terminal reward, i.e. $\beta = 0$, using various sample sizes of $m = 30, 60, 90, 120, 150$ and 300, and then compute the KL-divergences between the generated data and the Swiss roll training data. In the second experiment, we fix the expected terminal reward at 0.8 and compute the same KL-divergences with different sample sizes. This is achieved by choosing different values of the weight β for the reward in our RL formulation.

The results are summarized in Table 3. To obtain the performance metrics in the table, we generate 100 batches of m samples (i.e. $\mathbf{y}_T$), compute the KL divergence to the Swiss roll data p_0 for each batch via the estimation method introduced in Wang et al. (2009), and compute the 95% confidence interval based on the 100 batches. Table 3 shows that with a larger sample size in the ratio estimator, the generated data are generally of a

higher quality in the sense that they are more similar to the training data in terms of a smaller KL divergence for this example. On the other hand, a sample size of $m = 120$ or $m = 150$ is already enough to produce satisfactory performance of the q-learning algorithm. In particular, it is not necessary to always use all the available samples to estimate the score function values in our algorithm. This can help us reduce the computational time and speed up the training process. While this point is less prominent for this two-dimensional example because training is fast anyway, it will become significant in high-dimensional applications as we will see in image generation tasks later.

To further investigate how sample size m affects the quality of score estimation, we calculate the bias, variance, and mean-squared error (MSE) of the ratio estimators (24) for various m and time points t. To compute these metrics for each t, we randomly generate 10,000 states $\mathbf{x} \in \mathbb{R}^2$, and for each $\mathbf{x}$, we first use 500,000 samples from p_0 to calculate the score estimator in (24) and regard it as the true score function value $\nabla \log p_t(\mathbf{x})$. Then, we calculate the estimator $\widehat{\nabla \log p_t(\mathbf{x})}$ by randomly drawing m samples and repeat this estimation process for 100 times. This allows us to compute (approximately) for each state–time pair $(\mathbf{x}, t)$: (i) the MSE of $\widehat{\nabla \log p_t(\mathbf{x})}$ in (24), and (ii) the bias and variance for each dimension of $\widehat{\nabla \log p_t(\mathbf{x})}$. Since the score function value $\nabla \log p_t(\mathbf{x})$ is a two-dimensional vector in this example, the bias of the estimator is a 2D vector and the corresponding covariance is a 2×2 matrix. To focus on key summary statistics, we compute the L_1 norm of the bias vector, as well as the trace of the covariance matrix which is simply the sum of variances across the two dimensions of the score estimator (referred to as "variance" below). For each t, we average the bias, variance, and MSE over 10,000 states $\mathbf{x}$. The resulting bias, variance, and MSE values for the ratio estimator (24) are summarized in Table 4.

From Table 4, we observe that as the sample size m increases, the bias, variance, and MSE of the ratio estimator generally decrease. Moreover, the accuracy of the estimator depends on the time dimension of the true score function $\nabla \log p_t(\cdot)$, with smaller MSE values obtained for larger t. Intuitively, as $t \to T = 5$, the distribution p_T becomes approximately Gaussian, making the score easier to learn compared to the scenario where $t \to 0$, which corresponds to the true unknown data distribution.

The above analysis suggests that while using a larger sample size in the score estimator (24) in general and in theory increases accuracy, its benefit may not outweigh the resulting training cost as a smaller sample size may suffice to achieve satisfactory performance with a shorter training time.

7.4 Comparison with pretrain-then-fine-tune approach

In this section, we compare our continuous-time RL approach with those developed for fine-tuning pretrained discrete-time diffusion models.

Specifically, we consider two benchmark fine-tuning methods, DDPO and DPOK, proposed by Black et al. (2024) and Fan et al. (2023), respectively. Both DDPO and DPOK are online RL methods for fine-tuning a pretrained diffusion model to generate samples with higher terminal rewards. Note that the key difference between their formulations and ours is that the latter does not involve a pretrained model nor is it designed for fine-tuning. For reader's convenience, we first briefly review their formulations.

Sample Size m	$\|\text{Bias}\|_1$	Variance	MSE
$t=1$			
30	6.178×10^{-2}	5.036×10^{-2}	5.348×10^{-2}
90	3.346×10^{-2}	1.515×10^{-2}	1.938×10^{-2}
300	2.151×10^{-2}	4.458×10^{-3}	8.682×10^{-3}
$t=2$			
30	2.294×10^{-2}	7.504×10^{-3}	8.273×10^{-3}
90	1.393×10^{-2}	2.440×10^{-3}	3.213×10^{-3}
300	8.723×10^{-3}	7.302×10^{-4}	1.501×10^{-3}
$t=4$			
30	3.069×10^{-3}	1.015×10^{-3}	1.112×10^{-3}
90	2.524×10^{-3}	3.386×10^{-4}	4.504×10^{-4}
300	1.457×10^{-3}	1.024×10^{-4}	3.209×10^{-4}

Table 4: The bias, variance, and mean-squared error of the ratio estimator (24) for the 2D Swiss roll example.

Black et al. (2024) and Fan et al. (2023) formulate the denoising process of a discrete-time diffusion model, called the denoising diffusion probabilistic model (DDPM), as a Markov decision process (MDP). Specifically, denote by $\{\mathbf{y}_{t_k}\}_{k=0}^K$ the denoising process (with a slight abuse of notations), where $\mathbf{y}_0$ is sampled from a normal distribution, and let the one-step transition probabilities $\{p_\phi(\mathbf{y}_{t_k}|\mathbf{y}_{t_{k-1}})\}_{k=1}^K$ (often modeled as Gaussian) be parameterized by ϕ. Given a pretrained diffusion model $p_{pre} = p_{\phi_0}$, DDPO and DPOK aim to fine-tune it to maximize the expected reward of generated samples by updating ϕ. They regard the transition $p_\phi(\mathbf{y}_{t_{k+1}}|\mathbf{y}_{t_k})$ as a stochastic policy and apply the policy gradient algorithm. To do so, they formulate a finite-horizon MDP with state $\mathbf{s}$, action $\mathbf{a}$, policy $\boldsymbol{\pi}^\phi$, deterministic transition dynamics P and reward R as follows:

$$\mathbf{s}_{t_k} := \mathbf{y}_{t_k}, \quad \mathbf{a}_{t_k} := \mathbf{y}_{t_{k+1}}, \quad \boldsymbol{\pi}^\phi(\mathbf{a}_{t_k}|\mathbf{s}_{t_k}) := p_\phi(\mathbf{y}_{t_{k+1}}|\mathbf{y}_{t_k}), \quad P(\mathbf{s}_{t_{k+1}}|\mathbf{s}_{t_k}, \mathbf{a}_{t_k}) := 1_{\mathbf{s}_{t_{k+1}}=\mathbf{a}_{t_k}},$$
$$R(\mathbf{s}_{t_k}, \mathbf{a}_{t_k}) = 0, \ \forall k = 0, ..., K-2, \quad R(\mathbf{s}_{t_{K-1}}, \mathbf{a}_{t_{K-1}}) = h(\mathbf{a}_{t_{K-1}}) = h(\mathbf{y}_{t_K}).$$

Denote by $p_\phi(\mathbf{y}_{0:K}) := p(\mathbf{y}_0) \prod_{k=1}^K p_\phi(\mathbf{y}_{t_k}|\mathbf{y}_{t_{k-1}})$ the joint distribution of the denoising trajectory $\{\mathbf{y}_{t_k}\}_{k=0}^K$. The objective of DDPO and DPOK is given below, with $\gamma = 0$ for DDPO (i.e. purely reward-directed) and $\gamma > 0$ for DPOK:

$$\min_\phi \mathbb{E}_{p_\phi(\mathbf{y}_{0:K})} \left[-\beta \cdot h(\mathbf{y}_{t_K}) + \gamma \cdot \sum_{k=1}^K \text{KL}\left(p_\phi(\mathbf{y}_{t_k}|\mathbf{y}_{t_{k-1}}) || p_{\text{pre}}(\mathbf{y}_{t_k}|\mathbf{y}_{t_{k-1}}) \right) \right]. \tag{43}$$

Here, $p_{\text{pre}}(\mathbf{y}_{t_k}|\mathbf{y}_{t_{k-1}})$ is the one-step transition probability distribution under the pretrained diffusion model, $\text{KL}(\cdot||\cdot)$ is the KL-divergence between distributions, and β, γ are weights balancing the trade-off between the terminal reward h and the deviation from the pretrained model. One can easily see that our RL formulation is significantly different from the ones in Black et al. (2024) and Fan et al. (2023) at least in two aspects: First, we penalize the deviation from the (unknown) true score model, rather than a (known) pretrained score or

diffusion model. Second, the setting (i.e. continuous-time) and definitions of actions and policies in our formulation are entirely different from theirs.

We now compare the performance of our algorithm with those of DDPO and DPOK. For illustrations, we focus on the 2D Swiss roll data discussed in Section 7.3 along with the reward function $h(y_1, y_2) = 1_{y_1 \in [-5,6]}$. To adapt DDPO and DPOK to score-based diffusion models, we consider the discrete-time denoising process $\{\mathbf{y}_{t_k}\}_{k=0}^K$ similar to (41):

$$\mathbf{y}_{t_{k+1}} = \mathbf{y}_{t_k} + \left[\mathbf{y}_{t_k} + 2\mu^\psi(t_k, \mathbf{y}_{t_k})\right]\Delta t + \sqrt{2\Delta t} \cdot \xi, \quad \mathbf{y}_0 \sim \nu$$

where $t_{k+1} = t_k + \Delta t$, and μ^ψ is the policy or score neural network discussed earlier (see (21)). Then, the one-step transition $p_\psi(\mathbf{y}_{t_k}|\mathbf{y}_{t_{k-1}})$ is Gaussian with mean effectively parameterized by μ^ψ, and one can compute the KL-divergence in (43) explicitly. The pretrained score network μ^{pre}, as required by DDPO and DPOK, is obtained by applying our Algorithm 1 with hyperparameters shown in Table 2 and $\beta = 0$, although one can use other standard score matching methods. For our q-learning algorithm, the controlled state process $\mathbf{y}_t^{\mathbf{a}}$ is updated according to the dynamic (41). To make a fair comparison, the neural network μ^ψ used in all the three algorithms share the same structure which is the one displayed in Table 1. Moreover, the pretrained network μ^{pre} is used as the initialization of μ^ψ in our algorithm. Finally, for all the algorithms, we set $T = 10, \Delta t = 0.25$ and $K = T/\Delta t = 40$.

Algorithm	Reward ($\mathbb{E}[h(\mathbf{y}_T)]$) $\uparrow$	KL($p(\mathbf{y}_T)\|\|p_0$) $\downarrow$
Pretrained model A	0.52 $\pm$ 0.006	0.17 $\pm$ 0.019
DDPO	1.00 $\pm$ 0.001	2.97 $\pm$ 0.024
DPOK	0.70 $\pm$ 0.004	**0.21 $\pm$ 0.021**
q-learning	0.70 $\pm$ 0.005	0.29 $\pm$ 0.023
DPOK	0.80 $\pm$ 0.004	**0.38 $\pm$ 0.023**
q-learning	0.80 $\pm$ 0.003	0.39 $\pm$ 0.026
DPOK	0.90 $\pm$ 0.003	0.69 $\pm$ 0.024
q-learning	0.90 $\pm$ 0.003	**0.66 $\pm$ 0.024**

Table 5: Empirical mean rewards and KL-divergences (with 95% confidence) based on (100 batches of) 300 samples generated by diffusion models fine-tuned/trained with different algorithms, when a good pretrained model is available.

We first report the numerical results in Table 5 and Figure 4 for the scenario when a "good" pretrained model (referred to as Pretrained Model A) is given. Here, by "good" we mean that Model A generates samples whose distribution is very close to the true data distribution p_0 as measured by the KL divergence, which is obtained by implementing Algorithm 1 with $\beta = 0$ and running the algorithm with 50,000 iterations. Given this pretrained model, DDPO fine-tunes it for pure reward maximization (without any KL regularization). Table 5 shows that DDPO generates samples achieving the maximal empirical mean reward, but with a large KL divergence from the true data distribution. On the other hand, both DPOK and our q-learning method trade off two criteria: reward and KL-divergence. For a fair comparison, we compare the KL-divergence values of the two algorithms under the same level of expected reward of the generated samples. This is achieved by choosing

different values of the weight β for the reward in DPOK (while setting $\gamma = 1$ in its objective) and our model, because DPOK penalizes deviation from the pretrained model, while we penalize deviation from the true score model. Table 5 displays the empirical mean KL divergence to the data distribution p_0 (and the corresponding 95% confidence interval) of 300 samples obtained by DPOK and our algorithm, when the terminal expected rewards are set to be 0.7, 0.8 and 0.9, respectively. We can observe that the performances of our q-learning algorithm and DPOK are similar (if the latter is slightly better) in the current scenario of a good pretrained model being available. For visualization, Figure 4 plots 300 samples generated by different algorithms with a fixed expected reward level 0.8. It is clear that samples generated from DDPO diverge from the data distribution significantly, while samples obtained from DPOK and our q-learning algorithm have similar distributions to the data distribution.

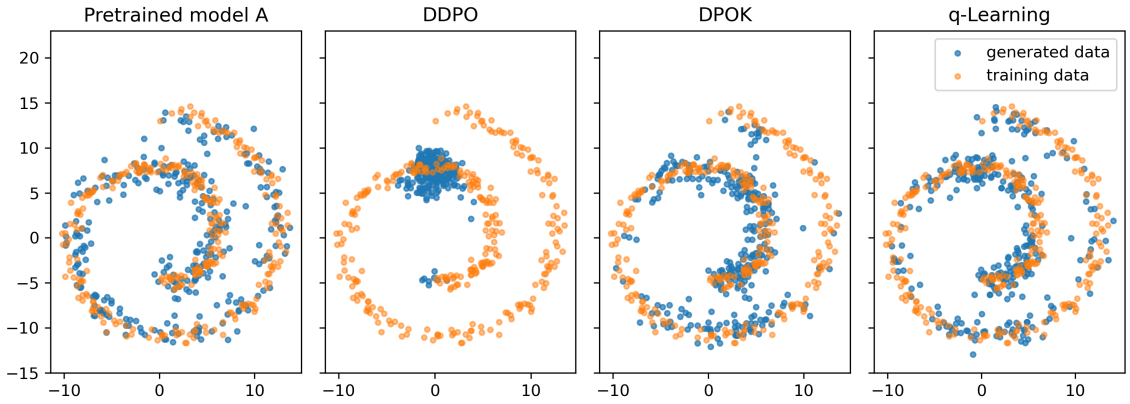


Figure 4: 300 samples generated by diffusion models fine-tuned/trained by different algorithms. DPOK and $q-$Learning have a fixed expected reward of 0.80. The samples obtained from (good) Pretrained Model A are also plotted for reference.

Algorithm	Reward ($\mathbb{E}[h(\mathbf{y}_T)]$) $\uparrow$	KL($p(\mathbf{y}_T)\|\|p_0$) $\downarrow$
Pretrained model B	0.55 $\pm$ 0.005	0.68 $\pm$ 0.026
DPOK	0.70 $\pm$ 0.004	0.90 $\pm$ 0.025
q-learning	0.70 $\pm$ 0.004	**0.30 $\pm$ 0.023**
DPOK	0.80 $\pm$ 0.004	1.03 $\pm$ 0.027
q-learning	0.80 $\pm$ 0.004	**0.42 $\pm$ 0.023**
DPOK	0.90 $\pm$ 0.003	1.21 $\pm$ 0.028
q-learning	0.90 $\pm$ 0.003	**0.64 $\pm$ 0.026**

Table 6: Empirical mean rewards and KL-divergences (with 95% confidence) based on (100 batches of) 300 samples generated by diffusion models fine-tuned/trained with different algorithms, when only a bad pretrained model is available.

While DPOK and q-learning have similar performances in the above experiment, the two are based on very different optimization objectives. DPOK is a fine-tuning RL method, which requires the fine-tuned model to be close to the pretrained one. As a result, DPOK

may not perform well, if the pretrained model is not sufficiently good. By contrast, our method does not rely on any pretrained model and is purely data driven, resulting in robust results. To illustrate, we repeat the above experiment but with a "bad" pretrained model. This is referred to as Pretrained Model B, which is obtained by implementing Algorithm 1 with $\beta = 0$ and running the algorithm with only 30,000 (instead of 50,000) iterations. In particular, the generated distribution from Model B has a much higher KL divergence (0.68 vs. 0.17 for Model A) with respect to the true data distribution; see also the first panel of Figure 5 for a visual. Table 6 reports the corresponding performances of DPOK and the q-learning algorithm. For a fixed target reward level, DPOK now performs much worse than our algorithm: the former generates samples with a distribution much further away from the true data distribution than the latter does. Interestingly, our algorithm can even improve the quality of a bad pretrained model in the sense of reducing the KL-divergence from the data distribution p_0, while DPOK generates samples with an even bigger divergence when applied for reward maximization. For visualization, Figure 5 shows 300 samples generated by different algorithms with a fixed expected reward level 0.8. The outperformance of our algorithm is evident in the case when only a low-quality pretrained model is available.

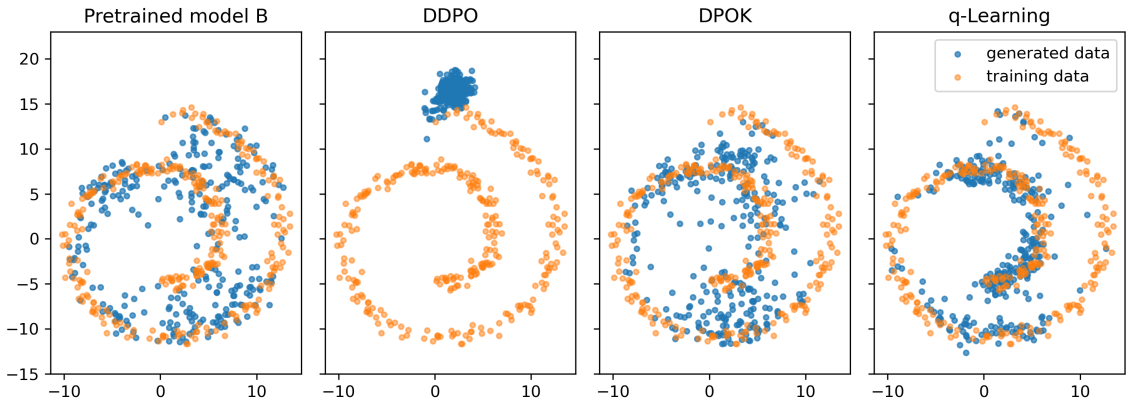


Figure 5: 300 samples generated by diffusion models fine-tuned/trained by different algorithms. DPOK and $q-$Learning have a fixed expected reward of 0.80. The samples obtained from (bad) Pretrained Model B are also plotted for reference.

To summarize the findings, DDPO of Black et al. (2024) suffers from the issue of reward over-optimization, where the generated distributions diverge too far from the original data distribution. DPOK of Fan et al. (2023) has a similar performance as our q-learning algorithm, *provided* that the pretrained model is of good quality. However, if the pretrained model is not good enough, our q-learning algorithm outperforms DPOK significantly.

8. Image Generations

In this section, we test Algorithm 1 on a reward-directed image generation task. The training samples are 50,000 RGB images of size 32×32 from the CIFAR-10 dataset (Krizhevsky, 2009), some of which are displayed in Figure 6(a). We consider a reward function from Black et al. (2024) based on imcompressibility, where the reward is the file size of generated

images after JPEG compression. Note that the resolution of generated samples is fixed at 32×32 pixels, so the file size is determined solely by the compressibility of the images. In the following, we first describe the experimental settings (which are different from the settings for the two toy examples discussed earlier) and then report the numerical results.

8.1 Experimental Settings

- **SDE/Environment simulator.**

  In this experiment, we consider the following environment simulator inspired by the celebrated DDPM model (Ho et al., 2020) for image generations: for some small Δt and $t < T = 1$,

$$\mathbf{y}^{\boldsymbol{\pi}}_{t+\Delta t} = \frac{1}{\sqrt{1-\beta(1-t)\cdot\Delta t}}\left(\mathbf{y}^{\boldsymbol{\pi}}_t + \beta(1-t)\cdot\Delta t\cdot a^{\boldsymbol{\pi}}_t\right) + \sqrt{\beta(1-t)\cdot\Delta t}\cdot\xi, \qquad (44)$$

  where $\xi \sim \mathcal{N}(\mathbf{0}, I_d)$, $\beta(t)$ is a linear function of t, and $\mathbf{y}^{\boldsymbol{\pi}}_0 \sim \mathcal{N}(\mathbf{0}, I_d)$. The simulator (44) corresponds to an appropriate discretization of the controlled dynamic (28) where $f(t) = \frac{1}{2}g(t)^2$ and f is linear in t, as well as to the VP-SDE model introduced in Song et al. (2021b). To match the settings in Ho et al. (2020), we use the linear schedule $\beta(t)$ such that $\beta(\frac{1}{1000}) = 10^{-1}\cdot\frac{1}{1000\Delta t}$ and $\beta(1) = 20\cdot\frac{1}{1000\Delta t}$. In particular, if we use $\Delta t = 0.001$ and simulate (44) in $K = \frac{T}{\Delta t} = 1000$ steps, the simulator (with the action replaced by the trained score function obtained from denoising score matching) corresponds to the sampler used in the DDPM model in (Ho et al., 2020). In the implementation of our RL algorithm, we do not use $K = 1000$ because it is time-consuming to simulate 1000 steps in each episode. Instead, we consider $\Delta t = 0.05$ so that we run (44) with $K = 20$ steps in each episode. In this case, the linear schedule we use becomes $\beta(t) = \frac{389}{999}t + \frac{8}{4995}$. We find that $K = 20$ sampling steps are enough to generate images of an acceptable quality. This is demonstrated in Figure 6(b), where we display image samples generated by running (44) with $K = 20$ steps, and replacing the action $a^{\boldsymbol{\pi}}_t$ by the score function approximator trained by DDPM.

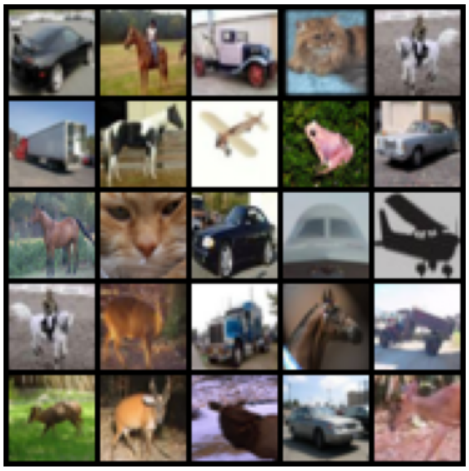
- **Actor and Critic Networks**

  For the actor network $\mu^{\psi}(\cdot,\cdot)$, we adopt U-Net (Ronneberger et al., 2015), which is a popular neural network for image generation tasks; see e.g. Ho et al. (2020); Song et al. (2021b). We initialize the actor network with the pre-trained DDPM model in the implementation of our q-learning algorithm, which speeds up the training process.

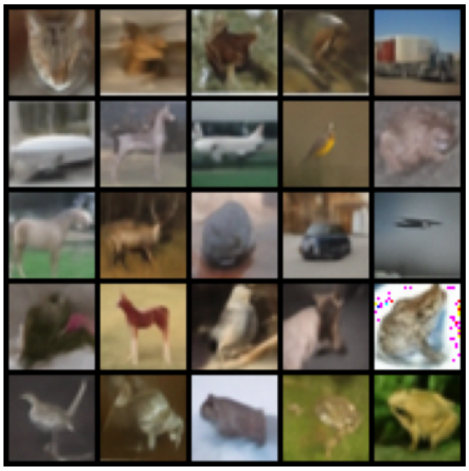
  The critic network $J^{\Theta}(\cdot,\cdot)$ consists of a U-Net that shares the same architecture with μ^{ψ} which we denote by U^{Θ_1}, a convolution layer Conv^{Θ_2} that has one output channel, and a multilayer perception denoted by MLP^{Θ_3} for which we use a dropout function to deal with the output of each layer to prevent overfitting. Technically, we have

$$J^{\Theta}(\mathbf{y}, t) := \mathrm{MLP}^{\Theta_3}(\mathrm{Conv}^{\Theta_2}(U^{\Theta_1}(\mathbf{y}, t))),$$

  where $\Theta = (\Theta_1, \Theta_2, \Theta_3)$. Further details about these component networks are given in Table 7.



(a) Original CIFAR-10 Samples



(b) Samples from a Pre-trained Model

Figure 6: Sample images from the CIFAR-10 dataset (left) and those generated by the pre-trained DDPM model (right) with 20 sampling steps.

Layer ID	Input Dimension	Output Dimension
U-Net U^{Θ_1}	$3 \times 32 \times 32 + 1$	$3 \times 32 \times 32$
Conv^{Θ_2}	$3 \times 32 \times 32$	$1 \times 25 \times 25$
MLP^{Θ_3}: Input	25×25	256
MLP^{Θ_3} - Hidden 1	256	64
MLP^{Θ_3} - Hidden 2	64	16
MLP^{Θ_3} - Output	16	1

Table 7: Components of the critic network for the experiment on CIFAR-10

- **Hyperparameters.**

  We set the temperature parameter $\theta = 10^{-3}$, the learning rates $\alpha_\psi = 1 \times 10^{-5}$ and $\alpha_\Theta = 2 \times 10^{-5}$. These key hyperparamters are tuned via grid search over the space $\{j \times 10^{-k} : \ j, k = 1, 2, 3, 4, 5\}$ to optimize the training process. We use a batch size of $B = 32$ in each episode. We also test $B = 64$ and $B = 128$, and the results are similar. Finally, we use a sample size of $m = 1$ to calculate the score estimator (24). This choice will be discussed in details later in Section 8.3. Note that when $m = 1$, the lower bound parameter ϵ in (24) no longer matters, which will be dropped in implementing the q-learning algorithm.

8.2 Numerical results

We now report the numerical results and compare the performance of our algorithm with that of DPOK in (43). We do not consider DDPO here because it is prone to reward over-optimization as we have already discussed in the 2D example in Section 7.3. For DPOK, we also consider $K = 20$ sampling steps in each episode and set $\alpha_\psi = 1 \times 10^{-5}$ and $B = 32$ as

with our q-learning algorithm. Similar as in Section 7.4, we fix the same level of the expected reward (the incompressibility score) of the generated samples by choosing different value of the weight β for the terminal reward in the algorithms. We then compare the FID (Fréchet Inception Distance; Heusel et al., 2017) scores for the generated images under the two algorithms. We use FID instead of KL-divegence, because the former is a more widely-used performance metric that measures the similarity of generated images. The results of the q-learning algorithm and DPOK are presented in Table 8, where each reward is the average incompressibility score of 20,000 images generated by the corresponding algorithm, and the FID score is calculated based on 20,000 generated images and 20,000 images from the CIFAR-10 dataset.

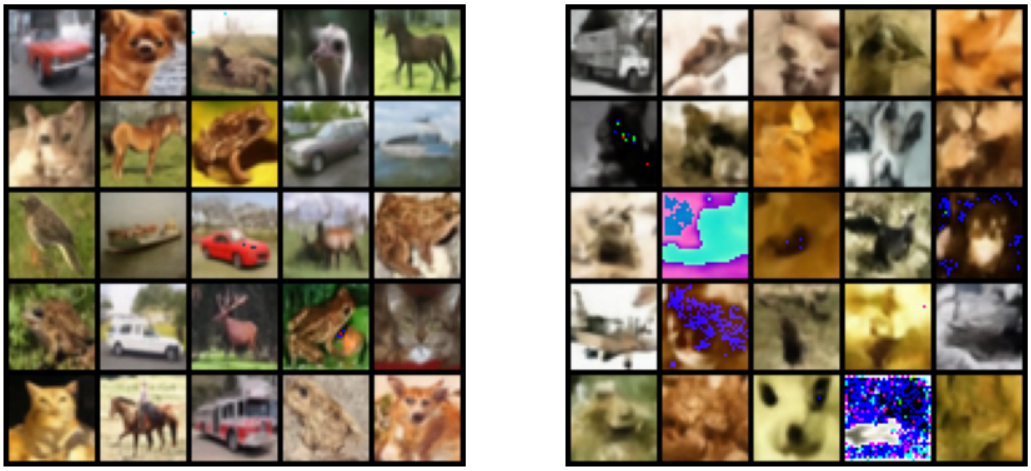
Table 8 shows that, with a fixed (terminal) reward level, our q-learning algorithm generates images with much lower FID scores (lower is better) than those from DPOK. For visualization, we also display some randomly selected samples of the generated images from our algorithm and DPOK in Figure 7, where the expected reward is fixed at 1.30 KB. We can see that the q-learning images have more details in their background compared with those generated from the pre-trained model in Figure 6. On the other hand, DPOK produces images of notably lower quality (higher FID), with a sizable portion barely recognizable or even unidentifiable. Indeed, approximately 15% of the 20,000 images generated by DPOK exhibit a substantial amount of noise, as evident in Figure 7(b). This is primarily because DOPK penalizes deviations from the pre-trained model, yet the samples generated from the pre-trained model is not of high quality (Figure 6(b)). By contrast, our RL formulation penalizes deviations from the true data distribution. When the expected reward level is fixed at 1.20 KB, we observe from Table 8 that q-learning even improves the pre-trained model by reducing the FID score while achieving actually a higher terminal reward (file size).

Algorithm	Reward (KB)	FID ↓	Training Time (hrs.) ↓
Pre-trained Model	1.10	29.90	-
q-learning	1.20	**26.64**	~ 8
DPOK		77.94	$\sim$ **4**
q-learning	1.30	**50.58**	~ 8
DPOK		107.99	$\sim$ **6**
q-learning	1.40	**89.79**	~ 9
DPOK		138.89	$\sim$ **7**

Table 8: Comparison between q-learning algorithm and DPOK.

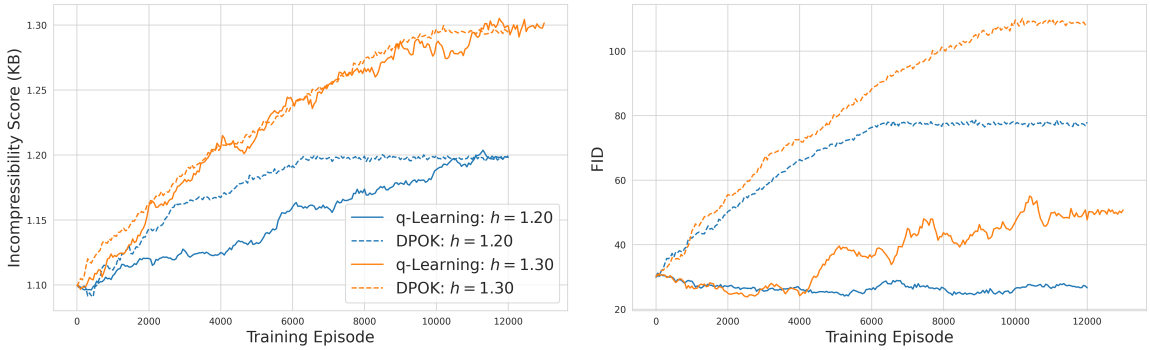
We next briefly discuss the training processes, including the training times for q-learning and DPOK[5]. The last column of Table 8 indicates that q-learning takes longer to converge than DPOK. Here, the training time of each algorithm is recorded when the performance metrics including the reward and FID curve converge or stabilize. The longer training

5. Our experiments are conducted on a 64-bit Linux operating system with 2 Intel Xeon Gold 5320 CPUs @ 2.20GHz and 4 A30 GPUs, each with 24GB of memory. Each algorithm in the experiments is trained on a single GPU, and the corresponding training time is recorded.



(a) q-learning: $h = 1.30$, FID=50.58 (b) DPOK: $h = 1.30$, FID=107.99

Figure 7: Sample images generated by diffusion models trained by q-learning algorithm (left) and DPOK (right). The average reward is fixed at 1.30 KB.



(a) Reward Curve (b) FID (to CIFAR-10) Curve

Figure 8: Training processes of q-learning (solid lines) and DPOK (dashed lines). The blue and orange curves correspond to final average terminal reward at 1.20 KB and 1.30 KB, respectively.

time of the q-learning algorithm is primarily due to the additional value/critic network involved. In particular, while DPOK only optimizes the policy/actor network, q-learning has to update parameters of both the actor and critic networks at each episode, inevitably increasing its computational time. In our experiment, it takes around 2.55 and 2.02 seconds to run one episode for q-learning and DPOK, respectively. Figure 8 further displays the training processes (learning curves) of the two methods in terms of reward and FID, where each reward and FID value is estimated based on 20,000 images generated by models saved during the training process. Figure 8 (a) shows a slower convergence of q-learning to the desired incompressibility score of 1.20 KB (while the convergence speeds are similar for the score of 1.30 KB): DPOK only needs less than 7,000 episodes while q-learning takes around

11,500 episodes to reach the target of 1.20 KB. Again, the reason is because optimizing the actor and critic networks simultaneously compels more episodes for q-learning to converge. Despite the longer training time, however, Figure 8 (b) indicates that q-learning outperforms DPOK decisively and significantly in terms of keeping the generated images closer to the original CIFAR-10 dataset.

8.3 Choice of sample size m in ratio estimator (24)

In this subsection, we discuss the impact of the sample size m in the ratio estimator (24) on the performance of the q-learning algorithm in image generations.

For the results presented in Table 8 we use a sample size $m = 1$ in (24) for obtaining the running reward signal during training. In general, images are considered samples from a high-dimensional distribution; as such using a larger sample size m will significantly increase training time which may greatly outweigh the benefit of an increased quality of the trained model. To see this, we conduct an experiment where three diffusion models with $m = 1, 10$ and 100 are trained via q-learning to reach a terminal incompressibility score of 1.30 KB by tuning the parameter β. The results, including performance metrics (FID) and training times, are summarized in Table 9. To generate this table, we calculate the FID score using 20,000 original and generated CIFAR-10 images, and the average time cost per episode is calculated by dividing the total computational time of the 10,000 episodes by 10,000. We also track the number of episodes and total training time required for each model to converge.

Table 9 shows that diffusion models trained using q-learning under different sample sizes m yield comparable FID scores for their generated images, indicating similar image qualities. (In fact, $m = 1$ even yields the smallest FID score!) However, increasing the sample size m significantly prolongs the time spent per episode, thereby substantially enlarging the total training time despite faster convergence (i.e. fewer episodes to converge). This experiment suggests that a sample size of $m = 1$ is sufficient for this image generation task, allowing us to train a high-quality diffusion model for achieving a target terminal reward level in the shortest amount of time.

Sample Size m	FID $\downarrow$	Time per Episode (sec.) $\downarrow$	Number of Episodes $\downarrow$	Training Time (hrs.) $\downarrow$
1	**50.58**	**2.55**	$\sim 11,000$	$\sim$ **8**
10	52.72	3.74	$\sim 10,000$	~ 11
100	51.34	5.61	$\sim$ **9,000**	~ 15

Table 9: Impact of sample size m in the ratio estimator (24) on the performance of the q-learning algorithm. The expected terminal reward level is fixed at 1.30 (KB).

This seemingly surprising result can be explained from two perspectives, one at a general/conceptual level and the other at a specific/technical one. The conceptual one is due to the RL approach employed. Although at a given time the algorithm generates a reward signal (i.e. a ratio estimator) using only one sample at random, at the next time point it generates another signal using another sample. When the number of episodes is large and/or the time step is small, a large number of samples will actually be used to produce

these signals during the *entire* learning process. Consequently, all the noises in the reward signals will be eventually averaged out. The RL agent can improve her policies by interacting with the environment and observing such reward (reinforcement) signals, and each of these signals does not need to be accurate and can be noisy.

The technical reason is specific to the image generation task and the associated score estimator (24). Ignoring ϵ, we note that the score estimator $\widehat{\nabla_{\mathbf{x}} \log p_t}(\mathbf{x})$ in (24) is based on the weighted combination of m samples:

$$\sum_{i=1}^{m} \frac{p_{t|0}(\mathbf{x}|\mathbf{x}_0^i)}{\sum_{i=1}^{m} p_{t|0}(\mathbf{x}|\mathbf{x}_0^i)} \cdot \mathbf{x}_0^i. \tag{45}$$

By (22), the conditional density $p_{t|0}(\mathbf{x}|\mathbf{x}_0^i) \propto \exp\left(-\|\mathbf{x} - e^{-\int_0^t f(s)ds}\mathbf{x}_0^i\|^2\right)$. However, in a high-dimensional setting such as CIFAR-10 images with dimension 3072 ($3 \times 32 \times 32$), the data points are typically sparse. Hence for a given $\mathbf{x}$, $p_{t|0}(\mathbf{x}|\mathbf{x}_0^i)$ are typically very small and thus the weights $\frac{p_{t|0}(\mathbf{x}|\mathbf{x}_0^i)}{\sum_{i=1}^{m} p_{t|0}(\mathbf{x}|\mathbf{x}_0^i)}$ close to zero, except for the data point, say $\mathbf{x}_0^j$, that is *closest* to $\mathbf{x} \cdot e^{\int_0^t f(s)ds}$. To illustrate this, consider a concrete example where we have only $m = 2$ samples $\mathbf{x}_0^1$ and $\mathbf{x}_0^2$, i.e., 2 images sampled from the CIFAR-10 dataset, along with a specific initial state $\mathbf{x}$ sampled from the standard multivariate normal distribution. The specific values of $\mathbf{x}, \mathbf{x}_0^1$ and $\mathbf{x}_0^2$ are not given explicitly here because they are all vectors in $\mathbb{R}^{3072}$. We find that $p_{1|0}(\mathbf{x}|\mathbf{x}_0^1) = \exp(-4200.7139)$, and $p_{1|0}(\mathbf{x}|\mathbf{x}_0^2) = \exp(-4229.2939)$. This yields a ratio $p_{1|0}(\mathbf{x}|\mathbf{x}_0^1)/p_{1|0}(\mathbf{x}|\mathbf{x}_0^2) = \exp(28.5799) \approx 10^{12}$, indicating that the weighted combination in (45) is essentially determined by $\mathbf{x}_0^1$ only. This observation holds for $t < T = 1$ and $m > 2$ as well. This provides an intuition for why in our training process, a very small number of samples or indeed even one sample may be sufficient for computing the score estimator in (24) in high-dimensional spaces, when the number of episodes is large and/or the time step is small.

The preceding argument also suggests that one might enhance the accuracy of the score estimator by using nearest-neighbor search to compute (45), i.e., finding the sample image $\mathbf{x}_0^i$ closest to $\mathbf{x} \cdot e^{\int_0^t f(s)ds}$ at $(\mathbf{x}, t)$. However, as expected this method is computationally costly. To demonstrate this, we conduct an experiment where we train our model using the q-learning algorithm with a fixed expected terminal reward level of 1.30 KB, employing nearest-neighbor search to compute the score estimator. Although the resulting model achieves a slightly lower FID score of 47.17, the time per episode increases significantly to 15.05 seconds, and the entire training process takes approximately 20 hours to complete (in contrast to the results reported in Table 9). In comparison, our simplified approach of randomly selecting a data point (with $m = 1$) reduces computational overhead substantially while keeping a good image quality because it still makes use of the valuable insights from our analysis.

9. Extensions

In this section, we discuss two extensions of our SDE-based formulation.

9.1 ODE-based formulation

In addition to the SDE-based implementation of diffusion models, another mainstream approach for sample generation is the probability flow ODE implementation (Song et al., 2021b). In this subsection, we show that our SDE-based RL framework for reward maximization can be easily extended to the ODE-based formulation.

Recall that the forward process $(\mathbf{x}_t)_{t\in[0,T]}$ satisfies the following SDE:

$$d\mathbf{x}_t = -f(t)\mathbf{x}_t dt + g(t)d\mathbf{B}_t, \qquad \mathbf{x}_0 \sim p_0, \tag{46}$$

and $p_t(\cdot)$ denotes the probability density function of $\mathbf{x}_t$ in (46). Song et al. (2021b) show that there exists an ODE:

$$\frac{d\bar{\mathbf{x}}_t}{dt} = f(T-t)\bar{\mathbf{x}}_t + \frac{1}{2}(g(T-t))^2\nabla_{\mathbf{x}}\log p_{T-t}(\bar{\mathbf{x}}_t), \qquad \bar{\mathbf{x}}_0 \sim p_T, \tag{47}$$

whose solution at time $t \in [0,T]$, $\bar{\mathbf{x}}_t$, is distributed according to p_{T-t}, i.e., the ODE (47) induces the same *marginal* probability density function as the SDE in (4). In particular, $\bar{\mathbf{x}}_T \sim p_0$. The ODE (47) is called the *probability flow ODE*.

Motivated by the SDE-based problem formulation (9), we now consider an ODE-based formulation (with a slight abuse of notation):

$$\max_{\mathbf{a}=(a_t:0\le t\le T)} \left\{\beta\cdot\mathbb{E}[h(\mathbf{y}_T^{\mathbf{a}})] - \mathbb{E}\left[\int_0^T (g(T-t))^2\cdot|\nabla\log p_{T-t}(\mathbf{y}_t^{\mathbf{a}}) - a_t|^2 dt\right]\right\} \tag{48}$$

where

$$d\mathbf{y}_t^{\mathbf{a}} = \left[f(T-t)\mathbf{y}_t^{\mathbf{a}} + \frac{1}{2}(g(T-t))^2 a_t\right]dt, \quad \mathbf{y}_0 \sim \nu. \tag{49}$$

Because the dynamics $(\mathbf{y}_t^{\mathbf{a}})$ is described by a controlled ODE, the Hamiltonian for this problem becomes

$$H(t,y,a,p) = -(g(T-t))^2|\nabla\log p_{T-t}(y) - a|^2 + [f(T-t)y + \frac{1}{2}(g(T-t))^2 a]\circ p. \tag{50}$$

Remark 10 *Because* (49) *is an ODE, instead of an SDE, the second term in the objective* (48) *can no longer be directly interpreted as the KL divegence between two path measures as in* (11)*. However, we still keep this term in the objective as in the SDE-based formulation to encourage the action process not to deviate too much away from the true score function.*

Similar to Section 3.2, we can write down the exploratory RL formulation of the above problem. With slight abuse of notations, the exploratory dynamic is given by the following ODE with random/normal initialization:

$$d\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}} = \left[f(T-s)\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}} + \frac{1}{2}g^2(T-s)\int_{\mathbb{R}^d} a\boldsymbol{\pi}(a|s,\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}})da\right]ds, \quad \tilde{\mathbf{y}}_0^{\boldsymbol{\pi}} \sim \nu.$$

The associated entropy-regularized control objective is given by:

$$J(t,y,\boldsymbol{\pi}) = \mathbb{E}_{t,y}\bigg[\int_t^T\int_{\mathbb{R}^d}\big(r(s,\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}},a) \ -\theta\log\boldsymbol{\pi}(a|s,\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}})\big)\boldsymbol{\pi}(a|s,\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}})dads + \beta h(\tilde{\mathbf{y}}_T^{\boldsymbol{\pi}})\bigg],$$

where r is given in (13). The goal of RL is to solve the following optimization problem:

$$\max_{\boldsymbol{\pi}\in\boldsymbol{\Pi}} \int J(0, y, \boldsymbol{\pi}) d\nu(y), \tag{51}$$

where $\boldsymbol{\Pi}$ stands for the set of admissible stochastic policies. As in (14), for actual execution of a stochastic policy $\boldsymbol{\pi}$ in the RL algorithm, we need the discretely sampled state process, which satisfies the following ODE: for all $i = 0, ..., K-1$ and all $s \in [t_i, t_{i+1})$,

$$d\mathbf{y}_s^{\boldsymbol{\pi},\mathbb{S}} = [f(T-s)\mathbf{y}_s^{\boldsymbol{\pi},\mathbb{S}} + \frac{1}{2}(g(T-t))^2 \mathbf{a}_{t_i}^{\boldsymbol{\pi}}] ds, \tag{52}$$

where $\mathbb{S} = (t_i)_{i=0,...,K}$ is a grid of $[0, T]$ and $\mathbf{a}_{t_i}^{\boldsymbol{\pi}} \sim \boldsymbol{\pi}(\cdot|t_i, \mathbf{y}_{t_i}^{\boldsymbol{\pi},\mathbb{S}})$.

It is easy to see that the theory and algorithm for the SDE-based formulation can be developed for the ODE-based formulation analogously. For instance, because the Hamiltonian in (50) is still a quadratic function of the action a, we deduce that the optimal stochastic policy for (51) is still a Gaussian distribution, which has the same form as (18). We can still apply Algorithm 1 to solve the RL problem (51), except one importance difference. In Algorithm 1, the environment simulator (i.e. the sampler) is based on the discretization (e.g. Euler–Maruyama) of the SDE in (28). For the ODE-based formulation, we instead use a numerical solver of the ODE (52) as the sampler. Many ODE solvers have been developed and employed in practice. For instance, one can use Euler (Song et al., 2021b), DDIM (Song et al., 2020a), Heun's 2nd order method (Karras et al., 2022), DPM solver (Lu et al., 2022), exponential integrator (Zhang and Chen, 2023), among others.

To test our method experimentally, we consider the Swiss roll dataset and implement Algorithm 1 for two ODE-based samplers/simulators: ODE-Euler of Song et al. (2021b) and DDIM of Song et al. (2020a). Following Song et al. (2020a), we set $f \equiv 0$ and $g \equiv \sqrt{2}$ in our experiment. Then from (52) we can obtain that for a uniform grid with size Δt, the update rule for the ODE-Euler simulator is given by (again with a slight abuse of notation and suppressing grid notation for clarity)

$$\mathbf{y}_{t+\Delta t}^{\boldsymbol{\pi}} = \mathbf{y}_t^{\boldsymbol{\pi}} + \mathbf{a}_t^{\boldsymbol{\pi}} \cdot \Delta t.$$

Moreover, the DDIM simulator, which is simply Euler's method applied to a reparameterization of the ODE (52) with $f \equiv 0$ and $g \equiv \sqrt{2}$, is given as follows (see Song et al., 2020a)

$$\mathbf{y}_{t+\Delta t}^{\pi} = \mathbf{y}_t^{\pi} + \left(\sqrt{\frac{1-\alpha(t)}{\alpha(t)}} - \sqrt{\frac{1-\alpha(t+\Delta t)}{\alpha(t+\Delta t)}} \right) \cdot \sqrt{2(T-t)} \mathbf{a}_t^{\boldsymbol{\pi}},$$

where $\alpha(t) = \frac{1}{2(T-t)+1}$ and $\alpha(t) = 1$ for all $t \geq T$. The DDIM simulator behaves similarly as ODE-Euler when $\Delta t \to 0$, but differs from the latter when Δt is not small (i.e. with fewer sampling steps). For comparison purposes, when implementing Algorithm 1, we also include the SDE simulator, which takes the following form when $f \equiv 0$ and $g \equiv \sqrt{2}$ (similar to (41)):

$$\mathbf{y}_{t+\Delta t}^{\pi} = \mathbf{y}_t^{\pi} + 2\mathbf{a}_t^{\boldsymbol{\pi}} \cdot \Delta t + \sqrt{2\Delta t} \cdot \xi, \quad \xi \sim \mathcal{N}(0, I_d).$$

All the three simulators share the same prior distribution $\nu := \mathcal{N}(0, 2T \cdot I_d)$, which is obtained from (5).

In our experiment, we fix $T = 5$ and vary Δt (or equivalently the number of sampling steps $K := T/\Delta t$). The implementation of Algorithm 1 with three different simulators/samplers is still based on the settings described in Tables 1 and 2. We first consider $\beta = 0$, in which the algorithm aims to generate samples that are close to the original data without concerning the reward. Table 10 shows how the choice of a simulator affects the output quality of the diffusion model. When there are 50 sampling steps, the RL algorithms trained under either the ODE or SDE simulator all generate reasonably good samples, although the two ODE-based ones have slightly better performances. When the number of sampling steps is reduced, the quality of samples generated from all the simulators decrease, but the SDE-based one decreases more. Figure 9 displays visually the generated samples by different simulators.

Simulator	Sampling Steps (Δt)	Reward ($\mathbb{E}[h(\mathbf{y}_T)]$) $\uparrow$	KL($p(\mathbf{y}_T)\|\|p_0$) $\downarrow$
DDIM	50 (0.1)	0.44 $\pm$ 0.005	0.14 $\pm$ 0.018
ODE-Euler		0.44 $\pm$ 0.006	**0.11 $\pm$ 0.016**
SDE		0.45 $\pm$ 0.005	0.18 $\pm$ 0.020
DDIM	10 (0.5)	0.45 $\pm$ 0.005	**0.15 $\pm$ 0.019**
ODE-Euler		0.44 $\pm$ 0.006	0.23 $\pm$ 0.022
SDE		0.46 $\pm$ 0.006	0.35 $\pm$ 0.024
DDIM	5 (1.0)	0.46 $\pm$ 0.005	0.72 $\pm$ 0.022
ODE-Euler		0.44 $\pm$ 0.006	**0.56 $\pm$ 0.024**
SDE		0.46 $\pm$ 0.005	0.83 $\pm$ 0.026

Table 10: Empirical mean rewards and KL-divergence (with 95% confidence) based on (100 batches of) 300 samples generated by our algorithm with ODE-based and SDE-based samplers with different sampling steps and $\beta = 0$.

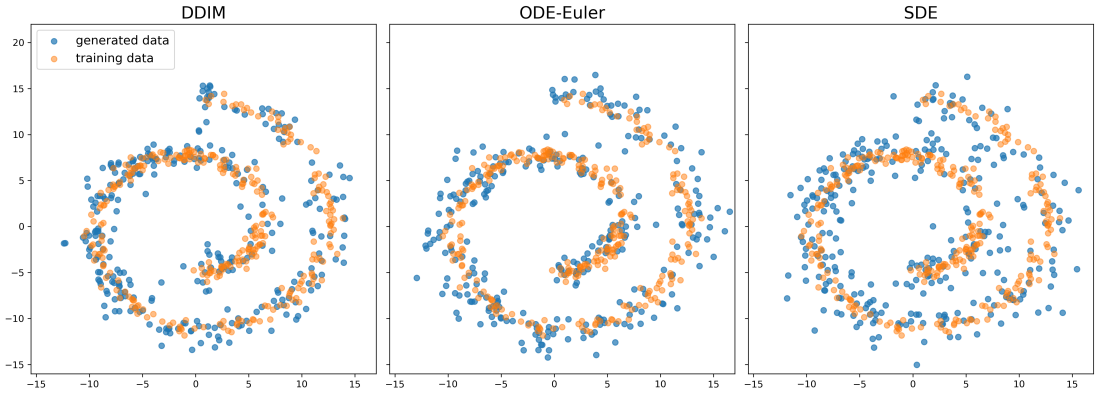


Figure 9: 300 samples generated by our algorithm for three simulators when $\beta = 0$ and $\Delta t = 0.5$.

We next consider the reward-directed generative task with the same reward function $h(\mathbf{y}) = 1_{y_1 \in [-5,6]}$ as before, and study the performance of Algorithm 1 with the three different simulators. Similar to Section 7.4, we train the diffusion model with a properly chosen

reward weight β so that the samples generated from the different simulators will earn the same mean reward of 0.80. The numerical results are presented in Table 11. We can see that when there are 50 sampling steps, the SDE-based sampler performs the best with a significantly small KL divergence between the generated distribution and the data distribution. However, as we increase Δt, the performance of the SDE-based sampler deteriorates and becomes inferior to the ODE-based samplers. Figure 10 displays the generated samples out of the three simulators with a fixed expected reward of 0.8 and $\Delta t = 0.5$.

To conclude this subsection, the ODE simulators prove to be effective when employed in conjunction with our RL formulation and q-learning algorithm for reward-directed diffusion models. With a smaller number of sampling steps, the ODE simulators significantly reduce the sample generation time while maintaining an acceptable level of sample quality and reward. In our Algorithm 1, running the SDE-based simulator consumes a significant proportion of the total training time. Therefore, the ODE formulation is a competitive contender for our reward-directed RL diffusion model.

Simulator	Sampling Steps (Δt)	Reward ($\mathbb{E}[h(\mathbf{y}_T)]$) $\uparrow$	KL($p(\mathbf{y}_T)\|\|p_0$) $\downarrow$
DDIM	50 (0.1)	0.80 $\pm$ 0.004	0.50 $\pm$ 0.027
ODE-Euler		0.80 $\pm$ 0.005	0.62 $\pm$ 0.022
SDE		0.80 $\pm$ 0.005	**0.24 $\pm$ 0.024**
DDIM	10 (0.5)	0.80 $\pm$ 0.004	**0.61 $\pm$ 0.025**
ODE-Euler		0.80 $\pm$ 0.004	0.72 $\pm$ 0.024
SDE		0.80 $\pm$ 0.004	0.77 $\pm$ 0.024
DDIM	5 (1.0)	0.80 $\pm$ 0.004	1.22 $\pm$ 0.028
ODE-Euler		0.80 $\pm$ 0.005	**1.07 $\pm$ 0.025**
SDE		0.80 $\pm$ 0.004	1.29 $\pm$ 0.027

Table 11: Empirical mean rewards and KL-divergence (with 95% confidence) based on (100 batches of) 300 samples generated by our algorithm with ODE-based and SDE-based samplers with different sampling steps when the (fixed) expected reward is 0.8.

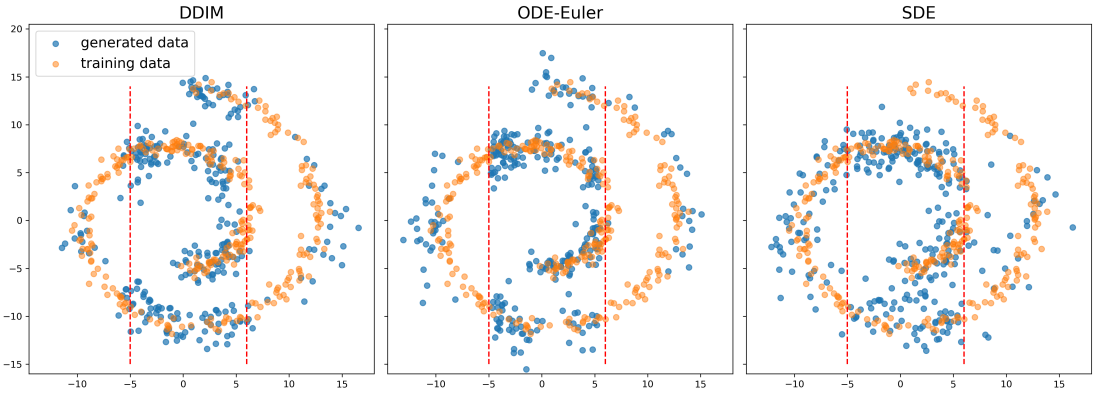


Figure 10: 300 samples generated by our algorithm for three simulators with a (fixed) expected of 0.80 and $\Delta t = 0.5$.

9.2 Conditional diffusion models

Diffusion models are often used for conditional data generations, e.g., in text-to-image models. In this section, we briefly discuss the extension of our framework to conditional diffusion models. We take the SDE-based formulation as an illustration.

We first introduce some notations. Let $(\mathbf{x}_0, \mathbf{C})$ be a random vector in $\mathbb{R}^d \times \mathbb{R}^{d_c}$, where $\mathbf{x}_0$ represents the data (e.g. images) and $\mathbf{C}$ represents the condition/context (e.g. a text prompt or a class label). Denote by $P_{\mathbf{C}}$ the (marginal) distribution of $\mathbf{C}$, which is assumed to be either known or accessible through i.i.d. samples. Denote $p_0(\cdot|c) := \mathbb{P}(\mathbf{x}_0 \in \cdot|\mathbf{C} = c)$ the conditional distribution of $\mathbf{x}_0$. Given $\mathbf{C} = c$, the standard conditional diffusion model aims to generate samples from the (unknown) conditional data distribution $p_0(\cdot|c)$.

For a conditional diffusion model, the forward process $\mathbf{x}_t$ (with a slight abuse of notations) at time t is given by (see e.g. Section 2.2 of Chen et al., 2024):

$$d\mathbf{x}_t = -f(t)\mathbf{x}_t dt + g(t)d\mathbf{B}_t, \quad \mathbf{x}_0 \sim p_0(\cdot|c),$$

where the noise is added to the data $\mathbf{x}_0$, *but not to the context* c. Denote by $p_t(\cdot|c)$ the conditional density function of $\mathbf{x}_t$ at time t given $\mathbf{C} = c$. Then the reverse-time SDE $(\mathbf{z}_t)$ satisfies

$$d\mathbf{z}_t = \left[f(T-t)\mathbf{z}_t + (g(T-t))^2 \nabla \log p_{T-t}(\mathbf{z}_t|c)\right] dt + g(T-t)dW_t, \quad \mathbf{z}_0 \sim \nu, \tag{53}$$

where the initialization distribution ν is still chosen to follow the normal distribution in (5), which is independent of $\mathbf{C} = c$. In (53), the quantity $\nabla \log p_{T-t}(\cdot|c)$ is referred to as the conditional score function. In text-to-image models, classifier guidance (Dhariwal and Nichol, 2021) or classifier-free guidance methods (Ho and Salimans, 2022) are often used to estimate such conditional score functions.

To adapt standard conditional diffusion models to reward maximization, we consider the following problem:

$$\max_{\mathbf{a}=(a_t:0\le t\le T)} \left\{ \beta \cdot \mathbb{E}[h(\mathbf{y}_T^{\mathbf{a}}, \mathbf{C})] - \mathbb{E}\left[\int_0^T (g(T-t))^2 \cdot |\nabla \log p_{T-t}(\mathbf{y}_t^{\mathbf{a}}|\mathbf{C}) - a_t|^2 dt\right] \right\} \tag{54}$$

where

$$d\mathbf{y}_t^{\mathbf{a}} = \left[f(T-t)\mathbf{y}_t^{\mathbf{a}} + (g(T-t))^2 a_t\right] dt + g(T-t)dW_t, \quad \mathbf{y}_0 \sim \nu, \tag{55}$$

and the expectation is taken with respect to the randomness in $(\mathbf{y}_t^{\mathbf{a}})$ and $\mathbf{C}$. In contrast to the unconditional diffusion model, here the reward function h now depends also on the condition $\mathbf{C}$. Moreover, the state at time t can be viewed as $(t, \mathbf{y}_t, \mathbf{C})$, and hence an optimal action process will also depend on the condition $\mathbf{C}$. Define the running reward for this problem as follows:

$$r(t, \mathbf{y}, c, a) := -(g(T-t))^2 \cdot |\nabla \log p_{T-t}(\mathbf{y}|c) - a|^2. \tag{56}$$

The exploratory formulation can be defined similarly as in Section 3.2. We let $\boldsymbol{\pi} : (t, y, c) \to \boldsymbol{\pi}(\cdot|t, y, c) \in \mathcal{P}(\mathbb{R}^d)$ be a given stochastic feedback policy, which now depends also on the condition c. The exploratory value function is defined by

$$J(t, y, c, \boldsymbol{\pi})$$

$$=\mathbb{E}_{t,y,c}\bigg[\int_t^T\int_{\mathbb{R}^d}\big(r(s,\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}},c,a)\ -\theta\log\boldsymbol{\pi}(a|s,\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}},c)\big)\boldsymbol{\pi}(a|s,\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}},c)dads+\beta h(\tilde{\mathbf{y}}_T^{\boldsymbol{\pi}},c)\bigg],$$

where $\{\tilde{\mathbf{y}}_s^{\boldsymbol{\pi}}:0\le s\le T\}$ is the exploratory state process satisfying the SDE (14), and $\mathbb{E}_{t,y,c}$ denotes the expectation conditioned on $(t,\mathbf{y}_t^{\boldsymbol{\pi}},\mathbf{C})=(t,y,c)$. We aim to solve the following optimization problem:

$$\max_{\boldsymbol{\pi}}\iint J(0,y,c,\boldsymbol{\pi})d\nu(y)dP_{\mathbf{C}}(c).$$

This problem can be solved similarly as in the unconditional setting. The running reward signal can be obtained in a similar manner as in the unconditional diffusion model discussed earlier. Specifically, given condition $\mathbf{C}=c$, we have

$$\begin{aligned}\nabla_{\mathbf{x}}\log p_t(\mathbf{x}|c)&=\frac{\nabla_{\mathbf{x}}p_t(\mathbf{x}|c)}{p_t(\mathbf{x}|c)}=\frac{\mathbb{E}_{\mathbf{x}_0\sim p_0(\cdot|c)}[\nabla_{\mathbf{x}}p_{t|0}(\mathbf{x}|\mathbf{x}_0)]}{\mathbb{E}_{\mathbf{x}_0\sim p_0(\cdot|c)}[p_{t|0}(\mathbf{x}|\mathbf{x}_0)]}\\&=\frac{\mathbb{E}_{\mathbf{x}_0\sim p_0(\cdot|c)}\left[p_{t|0}(\mathbf{x}|\mathbf{x}_0)\cdot\frac{-(\mathbf{x}-e^{-\int_0^t f(s)ds}\mathbf{x}_0)}{\int_0^t e^{-2\int_s^t f(v)dv}(g(s))^2ds}\right]}{\mathbb{E}_{\mathbf{x}_0\sim p_0(\cdot|c)}[p_{t|0}(\mathbf{x}|\mathbf{x}_0)]}\\&=\frac{1}{\int_0^t e^{-2\int_s^t f(v)dv}(g(s))^2ds}\cdot\left(-\mathbf{x}+\frac{\mathbb{E}_{\mathbf{x}_0\sim p_0(\cdot|c)}\left[p_{t|0}(\mathbf{x}|\mathbf{x}_0)\cdot\mathbf{x}_0\right]}{\mathbb{E}_{\mathbf{x}_0\sim p_0(\cdot|c)}[p_{t|0}(\mathbf{x}|\mathbf{x}_0)]}\cdot e^{-\int_0^t f(s)ds}\right).\end{aligned}$$

Therefore, given m i.i.d. samples $(\mathbf{x}_0^i)$ from the conditional data distribution $p_0(\cdot|c)$, a simple ratio estimator for the true score $\nabla_{\mathbf{x}}\log p_t(\mathbf{x}|c)$ at given t and $\mathbf{x}$ is given by

$$\widehat{\nabla_{\mathbf{x}}\log p_t}(\mathbf{x}|c)=\frac{1}{\int_0^t e^{-2\int_s^t f(v)dv}(g(s))^2ds}\cdot\left(-\mathbf{x}+\frac{\sum_{i=1}^m\left[p_{t|0}(\mathbf{x}|\mathbf{x}_0^i)\cdot\mathbf{x}_0^i\right]}{\max\{\sum_{i=1}^m[p_{t|0}(\mathbf{x}|\mathbf{x}_0^i)],m\epsilon\}}\cdot e^{-\int_0^t f(s)ds}\right),$$

where $\epsilon>0$ is some prespecified small value. It follow that we can obtain a noisy sample of the instantaneous reward (56) at time t given by

$$\hat{r}_t=-g^2(T-t)\cdot|\widehat{\nabla\log p_{T-t}}(\mathbf{y}_t|c)-a_t|^2. \tag{57}$$

From the problem (54), the Hamiltonian becomes

$$\begin{aligned}&H(t,y,c,a,p,q)\\&=-(g(T-t))^2|\nabla\log p_{T-t}(y|c)-a|^2+[f(T-t)y+(g(T-t))^2a]\circ p+\frac{1}{2}(g(T-t))^2\circ q.\end{aligned}$$

This is still a quadratic function of a, and hence the optimal stochastic policy is still Gaussian as in Proposition 4. Hence, given the condition c, we can consider Gaussian policies in the RL algorithm design:

$$\boldsymbol{\pi}^{\psi}(\cdot|t,y,c)\sim\mathcal{N}\left(\mu^{\psi}(t,y,c),\frac{\theta}{2g^2(T-t)}\cdot I_d\right)\quad\text{for all }(t,y,c). \tag{58}$$

We also use q^{ψ} as the function approximator for the optimal q-function which is given below:

$$q^{\psi}(t,y,c,a)=-g^2(T-t)\cdot|a-\mu^{\psi}(t,y,c)|^2-\frac{\theta d}{2}\log\left(\frac{\pi\theta}{g^2(T-t)}\right). \tag{59}$$

Algorithm 2 summarizes the resulting algorithm for reward maximization in the conditional diffusion model.

Algorithm 2 q-Learning Algorithm (SDE-based conditional generation)

Inputs: condition/context distribution $P_{\mathbf{C}}$, horizon T, time step Δt, number of episodes N, number of mesh grids $K = T/\Delta t$, initial learning rates $\alpha_\Theta, \alpha_\psi$ and a learning rate schedule function $l(\cdot)$ (a function of the number of episodes), functional forms of parameterized value function $J^\Theta(\cdot,\cdot,\cdot)$ and $\mu^\psi(\cdot,\cdot,\cdot)$, temperature parameter θ, functions f, g in (1), and ϵ in (24).

Required program: environment simulator $(y', \hat{r}) = \mathit{Environment}_{\Delta t}(t, y, a)$ that takes current time–state pair (t, y) and action a as inputs and generates state y' (by a numerical solver of SDE (55)) at time $t + \Delta t$ and sample instantaneous reward $\hat{r}$ (see (57)) at time t as outputs. Policy $\boldsymbol{\pi}^\psi(\cdot|t, y, c)$ in (58), and q-function $q^\psi(t, y, c, a)$ in (59).

Learning procedure:

Initialize Θ, ψ.

for episode $j = 1$ **to** N **do**

Initialize $k = 0$. Sample $c \sim P_{\mathbf{C}}$. Sample initial state $y_0 \sim \nu$ and store $y_{t_k} \leftarrow y_0$.

while $k < K$ **do**

Generate action $a_{t_k} \sim \boldsymbol{\pi}^\psi(\cdot|t_k, y_{t_k}, c)$.

Apply a_{t_k} to environment simulator $(y, \hat{r}) = \mathit{Environment}_{\Delta t}(t_k, y_{t_k}, a_{t_k})$, and observe new state y and reward $\hat{r}$ as outputs. Store $y_{t_{k+1}} \leftarrow y$ and $r_{t_k} \leftarrow \hat{r}$.

Update $k \leftarrow k + 1$.

end while

For every $i = 0, 1, \cdots, K-1$, compute and store test functions

$$\xi_{t_i} = \frac{\partial J^\Theta}{\partial \Theta}(t_i, y_{t_i}, c), \quad \zeta_{t_i} = \frac{\partial q^\psi}{\partial \psi}(t_i, y_{t_i}, c, a_{t_i}).$$

Compute

$$\Delta\Theta = \sum_{i=0}^{K-1} \xi_{t_i} \big[J^\Theta(t_{i+1}, y_{t_{i+1}}, c) - J^\Theta(t_i, y_{t_i}, c) + r_{t_i}\Delta t - q^\psi(t_i, y_{t_i}, c, a_{t_i})\Delta t \big],$$

$$\Delta\psi = \sum_{i=0}^{K-1} \zeta_{t_i} \big[J^\Theta(t_{i+1}, y_{t_{i+1}}, c) - J^\Theta(t_i, y_{t_i}, c) + r_{t_i}\Delta t - q^\psi(t_i, y_{t_i}, c, a_{t_i})\Delta t \big].$$

Update Θ and ψ by

$$\Theta \leftarrow \Theta + l(j)\alpha_\Theta \Delta\Theta,$$
$$\psi \leftarrow \psi + l(j)\alpha_\psi \Delta\psi.$$

end for

10. Conclusions

In this paper, we provide a continuous-time RL framework for adapting score-based diffusion models to generate samples that maximize some reward function, without requiring the availability of a pretrained score function or attempting to learn the score function. The key idea is that of *model-free* and *data-driven*: optimization is based on a stream of (possibly noisy) score signals, instead of on a trained score model obtained from score matching. Our framework is general and applicable to both SDE and probability flow ODE based implementations of diffusion models. It also can be adapted readily to cover both pure score matching and fine-tuning pretrained models as special cases. Numerically, the resulting RL algorithms are shown to perform well on both low-dimensional synthetic data sets and high-dimensional CIFAR-10 image datasets. Our approach is conceptually different from the hitherto mainstream method of pretraining score functions followed by fine-tuning, offering an overarching way of accomplishing generate AI tasks via diffusion models.

There are a couple of notable limitations of the paper that beg for further investigations. As discussed in Section 8, while q-learning does not need to learn the score function which is the most expensive part in diffusion models, its actor–critic structure requires substantial training time to achieve convergence at least for image generations. A promising future research direction lies in accelerating the training process, particularly when dealing with complex, high-dimensional datasets and diverse reward functions. On the other hand, the convergence analysis of the q-learning algorithm in Section 6.2 is very preliminary, with the result relying on certain assumptions, including the existence of Lyapunov functions, that are hard to verify. A specific interesting question is to identify structural properties of neural network approximations for actor/critic and of data distributions that would enable the validation of these assumptions. In general, convergence and regret analysis in continuous-time RL for diffusion processes is an uncharted territory where most exciting research awaits.

Acknowledgement. We thank the two anonymous referees and the Action Editor for very helpful and constructive comments that have led to a much improved manuscript. This work was presented at the 2024 Asian Quantitative Finance Conference in Taipei, the 2024 International Workshop on Probability Theory and Stochastic Analysis in Weihai, the 2024 Conference on Stochastic Control and Games for Risk and Regulation in Hammamet, the 2024 BIRS Workshop on Modeling, Learning and Understanding: Modern Challenges between Financial Mathematics, Financial Technology and Financial Economics in Banff, the 2024 International Conference on Stochastic Analysis and Differential Equations in Zhuhai, the 2024 Conference on Quantitative Methods in Finance in Sydney, the 2025 INI Workshop on Bridging Stochastic Control And Reinforcement Learning in Cambridge, the 2025 Mostly OM Workshop in Beijing, the 2025 Workshop on Data-driven Techniques in Operations Research in Shenzhen, and at seminars at University of Melbourne, Shanghai University of Finance and Economics, Hong Kong Polytechnic University, Peking University, Luiss University, University of Toronto, and Imperial College. We are grateful to the participants at these events for their comments. Gao acknowledges support from the Hong Kong Research Grants Council [GRF 14201424, 14200123, 14212522]. Zhou gratefully acknowledges financial supports through the Nie Center for Intelligent Asset Management at Columbia University.

References

Brian D. O. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023.

Albert Benveniste, Michel Métivier, and Pierre Priouret. *Adaptive algorithms and stochastic approximations*, volume 22. Springer Science & Business Media, 2012.

Jalaj Bhandari, Daniel Russo, and Raghav Singal. A finite time analysis of temporal difference learning with linear function approximation. *Operations Research*, 69(3):950–973, 2021.

Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024.

Patrick Cattiaux, Giovanni Conforti, Ivan Gentil, and Christian Léonard. Time reversal of diffusion processes under a finite entropy condition. *Annales de l'Institut Henri Poincaré (B) Probabilités et Statistiques*, 59(4):1844–1881, 2023.

Minshuo Chen, Song Mei, Jianqing Fan, and Mengdi Wang. An overview of diffusion models: Applications, guided generation, statistical rates and optimization. *arXiv preprint arXiv:2404.07771*, 2024.

Kevin Clark, Paul Vicol, Kevin Swersky, and David J Fleet. Directly fine-tuning diffusion models on differentiable rewards. In *The Twelfth International Conference on Learning Representations*, 2024.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

Ying Fan, Olivia Watkins, Yuqing Du, Hao Liu, Moonkyung Ryu, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, Kangwook Lee, and Kimin Lee. Reinforcement learning for fine-tuning text-to-image diffusion models. *Advances in Neural Information Processing Systems*, 36, 2023.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

Ulrich G Haussmann and Etienne Pardoux. Time reversal of diffusions. *The Annals of Probability*, pages 1188–1205, 1986.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International Conference on Machine Learning*, pages 8867–8887. PMLR, 2022.

Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4):695–708, 2005.

Yanwei Jia and Xun Yu Zhou. Policy evaluation and temporal-difference learning in continuous time and space: A martingale approach. *Journal of Machine Learning Research*, 23:(154):1–55, 2022a.

Yanwei Jia and Xun Yu Zhou. Policy gradient and actor-critic learning in continuous time and space: Theory and algorithms. *Journal of Machine Learning Research*, 23:(275):1–50, 2022b.

Yanwei Jia and Xun Yu Zhou. q-learning in continuous time. *Journal of Machine Learning Research*, 24(161):1–61, 2023.

Yanwei Jia and Xun Yu Zhou. Erratum to "q-learning in continuous time". *Forthcoming in Journal of Machine Learning Research. Available at* `https://www.columbia.edu/~xz2574/download/err.pdf`, 2025.

Yanwei Jia, Du Ouyang, and Yufei Zhang. Accuracy of discretely sampled stochastic policies in continuous-time reinforcement learning. *arXiv preprint arXiv:2503.09981*, 2025.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.

Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

H Kushner and G Yin. Stochastic approximation and recursive algorithms. In *Stochastic Modelling and Applied Probability*, volume 35. Springer-Verlag NY, 2003.

Salem Lahlou, Tristan Deleu, Pablo Lemos, Dinghuai Zhang, Alexandra Volokhova, Alex Hernández-Garcıa, Léna Néhale Ezzine, Yoshua Bengio, and Nikolay Malkin. A theory of continuous generative flow networks. In *International Conference on Machine Learning*, pages 18269–18300. PMLR, 2023.

Chieh-Hsin Lai, Yuhta Takida, Naoki Murata, Toshimitsu Uesaka, Yuki Mitsufuji, and Stefano Ermon. FP-Diffusion: Improving score-based diffusion models by enforcing the underlying score fokker-planck equation. In *International Conference on Machine Learning*, pages 18365–18398. PMLR, 2023.

Holden Lee, Jianfeng Lu, and Yixin Tan. Convergence for score-based generative modeling with polynomial complexity. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Kimin Lee, Hao Liu, Moonkyung Ryu, Olivia Watkins, Yuqing Du, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, and Shixiang Shane Gu. Aligning text-to-image models using human feedback. *arXiv preprint arXiv:2302.12192*, 2023.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In *Advances in Neural Information Processing Systems*, volume 35, pages 5775–5787, 2022.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*, 2022.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, volume 37, pages 2256–2265. PMLR, 2015a.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015b.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2020a.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation. In *Uncertainty in Artificial Intelligence*, pages 574–584. PMLR, 2020b.

Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum likelihood training of score-based diffusion models. *Advances in Neural Information Processing Systems*, 34: 1415–1428, 2021a.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021b.

Wenpin Tang. Fine-tuning of diffusion models via stochastic control: entropy regularization and beyond. *arXiv preprint arXiv:2403.06279*, 2024.

Wenpin Tang, Yuming Paul Zhang, and Xun Yu Zhou. Exploratory hjb equations and their convergence. *SIAM Journal on Control and Optimization*, 60(6):3191–3216, 2022.

Masatoshi Uehara, Yulai Zhao, Kevin Black, Ehsan Hajiramezanali, Gabriele Scalia, Nathaniel Lee Diamant, Alex M Tseng, Tommaso Biancalani, and Sergey Levine. Fine-tuning of continuous-time diffusion models as entropy-regularized control. *arXiv preprint arXiv:2402.15194*, 2024.

Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011.

Haoran Wang, Thaleia Zariphopoulou, and Xun Yu Zhou. Reinforcement learning in continuous time and space: A stochastic control approach. *Journal of Machine Learning Research*, 21:(198):1–34, 2020.

Qing Wang, Sanjeev R Kulkarni, and Sergio Verdú. Divergence estimation for multidimensional densities via k-nearest-neighbor distances. *IEEE Transactions on Information Theory*, 55(5):2392–2405, 2009.

Lemeng Wu, Chengyue Gong, Xingchao Liu, Mao Ye, and Qiang Liu. Diffusion-based molecule generation with informative prior bridges. *Advances in Neural Information Processing Systems*, 35:36533–36545, 2022.

Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Yingxia Shao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *ACM Copmuting Surveys*, 56(4):1–39, 2023.

Jiongmin Yong and Xun Yu Zhou. *Stochastic controls: Hamiltonian systems and HJB equations*, volume 43. Springer Science & Business Media, 2012.

Dinghuai Zhang, Ricky TQ Chen, Nikolay Malkin, and Yoshua Bengio. Unifying generative models with gflownets and beyond. *arXiv preprint arXiv:2209.02606*, 2022.

Dinghuai Zhang, Yizhe Zhang, Jiatao Gu, Ruixiang Zhang, Josh Susskind, Navdeep Jaitly, and Shuangfei Zhai. Improving GFlowNets for text-to-image diffusion alignment. *arXiv preprint arXiv:2406.00633*, 2024.

Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. In *International Conference on Learning Representations*, 2023.

Hanyang Zhao, Chen Haoxian, Ji Zhang, David Yao, and Wenpin Tang. Scores as actions: a framework of fine-tuning diffusion models by continuous-time reinforcement learning. *arXiv preprint arXiv:2409.08400*, 2024.