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

Xuefeng Gao[*] Jiale Zha[†] Xun Yu Zhou[‡]

September 7, 2024

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, our formulation does not involve any pretrained model for the unknown score functions of the noise-perturbed data distributions. We present an entropy-regularized continuous-time RL problem 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. 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. Finally, we discuss extensions of our RL formulation to probability flow ODE implementation of diffusion models and to conditional diffusion models.

Keywords: Generative AI, 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. 2021). 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.

[*]Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong, China. E-mail: xfgao@se.cuhk.edu.hk

[†]Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong, China. E-mail: jialezha@link.cuhk.edu.hk

[‡]Department of Industrial Engineering and Operations Research and The Data Science Institute, Columbia University, New York, NY 10027, USA. Email: xz2574@columbia.edu

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. 2020). 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 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. (2021), 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 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. 2021); 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), 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

[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.

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 from the denoising process with 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 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).

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 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, allowing us to obtain a reinforcement signal from the running reward whenever an action is applied. This procedure is not computationally expensive as we do not need to generate new samples from any distributions.

To solve the resulting RL problem for continuous-time diffusion models, we adapt the approach in Jia and Zhou (2023) 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. Based on this insight, we develop an actor–critic type algorithm, based on the (little) q-learning theory established in Jia and Zhou (2023), to solve our RL problem. We then implement the algorithm and conduct experiments for two examples with synthetic training data: a one-dimensional Gaussian mixture distribution and a two-dimensional Swiss rolls dataset (Sohl-Dickstein et al. 2015b, Lai et al. 2023). We find that our RL algorithm performs well in these experiments.

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 generations, in addition to the SDE-based one; see, e.g., Song et al. (2020), Song et al. (2021), 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 by 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. (2020). 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 generations 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 they 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 of 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 – the 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 are to incentivize the samples to stay close to the pretrained data distribution which is not necessarily always of the 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 (e.g. text-to-image models), 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.

Finally, we 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 our problem an RL one instead of a stochastic control as in Uehara et al. (2024). In sum, our theory and algorithms are conceptually different from these two papers.

The rest 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 5. In Section 6, we present the results of the numerical experiments. Section 7 highlights extensions to ODE models and conditional diffusion models. Finally, Section 8 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. 2021). 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 (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. (2021) 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

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

as a proxy to p_T, where I_d is the $d-$dimensional identity matrix; see Song et al. (2021), 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. 2021), sliced score matching (Song et al. 2020) 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. (2021) 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 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)

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

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*

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

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. (2021) for details. In particular, it was shown in Song et al. (2021) 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) essentially reduces to a score matching problem.

Denote

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

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 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 $\mathbb{R}^d$. Assume that the probability space is rich enough to support $\{Z_t, 0 \leq t \leq T\}$, a process of mutually independent copies of a random variable uniformly distributed over $[0,1]$, which is independent of the Brownian motion in (10). Let $\mathcal{G}_s = \mathcal{F}_s^W \vee \sigma(Z_t, 0 \leq t \leq s) \vee \sigma(\nu)$ be the sigma-algebra at time s that includes the information $\mathcal{F}_s^W$ generated by the Brownian motion, that generated by the copies of the uniform random variable (for action randomizations) and that generated by an independent normal random variable ν. At each time s, an action a_s is sampled from the distribution $\boldsymbol{\pi}(\cdot|s, \mathbf{y}_s^{\mathbf{a}})$. Denote by $a^{\boldsymbol{\pi}} = \{a_s^{\boldsymbol{\pi}} : 0 \le s \le T\}$ the action process generated by $\boldsymbol{\pi}$. Consider the sample state process $\mathbf{y}^{\boldsymbol{\pi}} = \{\mathbf{y}_s^{\boldsymbol{\pi}} : 0 \le s \le T\}$ corresponding to a specific copy of $a^{\boldsymbol{\pi}}$. By (10), it follows the SDE

$$d\mathbf{y}_s^{\boldsymbol{\pi}} = \left[f(T-s)\mathbf{y}_s^{\boldsymbol{\pi}} + (g(T-t))^2 a_s^{\boldsymbol{\pi}}\right] ds + g(T-s)dW_s \quad s \in (0,T], \quad \mathbf{y}_0^{\boldsymbol{\pi}} \sim \nu. \tag{14}$$

To encourage exploration, we follow Wang et al. (2020), Jia and Zhou (2023) and add an entropy regularizer to the running reward, and define

$$\begin{aligned} J(t,y,\boldsymbol{\pi}) &= \mathbb{E}_{t,y}^{\bar{\mathbb{P}}}\left[\int_t^T \left(r(s,\mathbf{y}_s^{\boldsymbol{\pi}}, a_s^{\boldsymbol{\pi}}) - \theta \log \boldsymbol{\pi}(a_s^{\boldsymbol{\pi}}|s, \mathbf{y}_s^{\boldsymbol{\pi}})\right) ds + \beta h(\mathbf{y}_T^{\boldsymbol{\pi}})\right] \qquad (15) \\ &= \mathbb{E}_{t,y}^{\bar{\mathbb{P}}}\left[\int_t^T \left(-g^2(T-t)\cdot|\nabla \log p_{T-s}(\mathbf{y}_s^{\boldsymbol{\pi}}) - a_s^{\boldsymbol{\pi}}|^2 - \theta \log \boldsymbol{\pi}(a_s^{\boldsymbol{\pi}}|s,\mathbf{y}_s^{\boldsymbol{\pi}})\right) ds + \beta h(\mathbf{y}_T^{\boldsymbol{\pi}})\right], \end{aligned}$$

where $\mathbb{E}_{t,y}^{\bar{\mathbb{P}}}$ is the expectation conditioned on $\mathbf{y}_t^{\boldsymbol{\pi}} = y$, and taken with respect to the randomness in the Brownian motion and action randomization. Here $\theta > 0$ is the temperature parameter that controls the level of exploration.[3] The function $J(\cdot,\cdot,\boldsymbol{\pi})$ in (15) is called the 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 defined below, following Jia and Zhou (2023).

Definition 1. *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

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

(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 $q \ge 1$, $\int_{\mathbb{R}^d} |a|^q \boldsymbol{\pi}(a|t,y)da \le C_q(1+|y|^p)$ for some $p \ge 1$ and C_q is a positive constant that can depend on q.

For our analysis it is also useful to define

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

Before we proceed to present our theoretical result, we discuss a key ingredient required in our RL algorithm design. In general RL problems, we either have access to the reward function or have

[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).

noisy observations from it for our learning. In our problem, 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, we express the score function 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

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

It follows that

$$\begin{aligned}
\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) &= \frac{\nabla_{\mathbf{x}} p_t(\mathbf{x})}{p_t(\mathbf{x})} = \frac{\mathbb{E}_{\mathbf{x}_0 \sim p_0}[\nabla_{\mathbf{x}} p_{t|0}(\mathbf{x}|\mathbf{x}_0)]}{\mathbb{E}_{\mathbf{x}_0 \sim p_0}[p_{t|0}(\mathbf{x}|\mathbf{x}_0)]} \\
&= \frac{\mathbb{E}_{\mathbf{x}_0 \sim p_0}\left[p_{t|0}(\mathbf{x}|\mathbf{x}_0) \cdot \frac{-(\mathbf{x} - e^{-\int_0^t f(s)ds}\mathbf{x}_0)}{\int_0^t e^{-2\int_s^t f(v)dv}(g(s))^2 ds}\right]}{\mathbb{E}_{\mathbf{x}_0 \sim p_0}[p_{t|0}(\mathbf{x}|\mathbf{x}_0)]} \\
&= \frac{1}{\int_0^t e^{-2\int_s^t f(v)dv}(g(s))^2 ds} \cdot \left(-\mathbf{x} + \frac{\mathbb{E}_{\mathbf{x}_0 \sim p_0}\left[p_{t|0}(\mathbf{x}|\mathbf{x}_0) \cdot \mathbf{x}_0\right]}{\mathbb{E}_{\mathbf{x}_0 \sim p_0}[p_{t|0}(\mathbf{x}|\mathbf{x}_0)]} \cdot e^{-\int_0^t f(s)ds}\right).
\end{aligned} \tag{17}$$

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

$$\nabla_{\mathbf{x}} \widehat{\log p_t}(\mathbf{x}) = \frac{1}{\int_0^t e^{-2\int_s^t f(v)dv}(g(s))^2 ds} \cdot \left(-\mathbf{x} + \frac{\sum_{i=1}^m \left[p_{t|0}(\mathbf{x}|\mathbf{x}_0^i) \cdot \mathbf{x}_0^i\right]}{\max\{\sum_{i=1}^m [p_{t|0}(\mathbf{x}|\mathbf{x}_0^i)], m\epsilon\}} \cdot e^{-\int_0^t f(s)ds}\right), \tag{18}$$

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 (18) to avoid division by extremely small numbers for numerical stability. It directly follows from (13) that we can obtain a noisy observation of the running reward at time t given by

$$\hat{r}_t = -g^2(T-t) \cdot |\nabla \widehat{\log p_{T-t}}(\mathbf{y}_t) - a_t|^2. \tag{19}$$

For fixed $(t, \mathbf{x})$, the simple ratio estimator in (18) 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, and no new samples need to be generated. 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 $(t, \mathbf{x})$ can be computational expensive, and this is not what we pursue. In our study, we only use (18) as a noisy score, which allows us to compute the reward signal (19) as we go, namely we compute it only when a state–action pair $(\mathbf{y}_t, a_t)$ is visited at time $t < T$. Finally, it is also unnecessary to use all the data samples from p_0 in computing (18), though a larger sample size will generally lead to a lower variance of the running reward signal (19).

4 Theory

In this section, we present our main theoretical results, which provide the optimal stochastic policy to the RL problem in (16). To this end, we first recall the system dynamics (10) and the running reward (13). We then 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 (9)–(10) (see e.g. Yong and Zhou 2012):

$$\begin{aligned} &H(t,y,a,p,q) \\ &= -(g(T-t))^2|\nabla \log p_{T-t}(y) - a|^2 + [f(T-t)y + (g(T-t))^2 a] \circ p + \frac{1}{2}(g(T-t))^2 \circ q. \end{aligned} \tag{20}$$

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

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

However, H in our case, (20), is quadratic in a, leading to the following result.

Proposition 1. *The optimal stochastic policy $\boldsymbol{\pi}^*(\cdot|t,y)$ is a Gaussian distribution in $\mathbb{R}^d$:*

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

where

$$\mu^*(t,y) = \nabla \log p_{T-t}(y) + \frac{1}{2} \cdot J^*_y(t,y). \tag{22}$$

This result provides some 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 HJB equation, a nonlinear PDE (see Equation (14) of Jia and Zhou 2023):

$$\frac{\partial J^*}{\partial t}(t,y) + \theta \log\left[\int_{\mathbb{R}^d} \exp\left(\frac{1}{\theta} H(t,y,a,J^*_y,J^*_{yy})\right) da\right] = 0, \tag{23}$$

$$J^*(T,y) = \beta \cdot h(y). \tag{24}$$

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 eqautions in the realm of RL). However, (23) 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 (22). We also observe from (21) 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 1, in our RL algorithms to be presented in Section 5, we only need to consider Gaussian policies in the following form:

$$\boldsymbol{\pi}^\psi(\cdot|t,y) \sim \mathcal{N}\left(\mu^\psi(t,y), \frac{\theta}{2g^2(T-t)} \cdot I_d\right) \quad \text{for all } (t,y), \tag{25}$$

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

5 q-Learning Algorithm

In this section, we present an algorithm to solve our continuous-time RL problem (16). In particular, we adapt the q-learning algorithms developed recently in Jia and Zhou (2023), which are of actor–critic type. For reader's convenience, we first introduce some definitions and theoretical result in Jia and Zhou (2023) 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 (20). One can readily infer from (23) that (see Proposition 8 in Jia and Zhou 2023)

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

and the optimal stochastic policy $\boldsymbol{\pi}^*(\cdot|t,y)$ in Proposition 1 is simply

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

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 (2023). This result is essential because it provides the theoretical foundation for designing our q-learning algorithm.

Proposition 2. *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*

$$\hat{J}^*(T,y) = h(y), \quad \int_{\mathbb{R}^d} \exp\left(\frac{1}{\theta}\hat{q}^*(t,y,a)\right) da = 1, \quad \textit{for all } (t,y) \in [0,T]\times\mathbb{R}^d. \tag{26}$$

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 $\boldsymbol{\pi} \in \mathbf{\Pi}$ and for all $(t,y) \in [0,T]\times\mathbb{R}^d$, the following process

$$\hat{J}^*(s,\mathbf{y}^{\boldsymbol{\pi}}_s) + \int_t^s [r(v,\mathbf{y}^{\boldsymbol{\pi}}_v,a^{\boldsymbol{\pi}}_v) - \hat{q}^*(v,\mathbf{y}^{\boldsymbol{\pi}}_v,a^{\boldsymbol{\pi}}_v)]dv \tag{27}$$

is an $(\{\mathcal{G}_s\}_{s\geq 0},\bar{\mathbb{P}})$ martingale, where $\mathbf{y}^{\boldsymbol{\pi}} = \{\mathbf{y}^{\boldsymbol{\pi}}_s : t \leq s \leq T\}$ satisfies (14) with $\mathbf{y}^{\boldsymbol{\pi}}_t = y$.

(ii) If there exists $\boldsymbol{\pi} \in \mathbf{\Pi}$ such that for all (t,y), the process (27) is an $(\{\mathcal{G}_s\}_{s\geq 0},\bar{\mathbb{P}})$ martingale where $\mathbf{y}^{\boldsymbol{\pi}}_t = y$, 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 (25). It is useful to note that

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

where

$$q^{\psi}(t,y,a) = -g^2(T-t)\cdot|a-\mu^{\psi}(t,y)|^2 - \frac{\theta d}{2}\log\left(\frac{\pi\theta}{g^2(T-t)}\right). \tag{28}$$

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 2. In particular, we follow Jia and Zhou (2023) and use the so-called martingale orthogonality conditions. These conditions state that a 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 $\mathbb{E}[\int_0^T |H_t|^2 dt] < \infty$. For detailed discussions, see (Jia and Zhou 2022a, Section 4.2).

For our problem, we choose the following $\mathcal{G}_t$−adapted test functions ξ_t and ζ_t:

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

The martingale characterization in Theorem 2 leads to the following system of equations in (Θ,ψ):

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

We solve these equations using stochastic approximation to update (Θ,ψ):

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

Because $q^{\psi}(t,y,a) = \theta\log\boldsymbol{\pi}^{\psi}(a|t,y)$, the above updating rule is essentially an actor–critic policy gradient method based on temporal difference $dJ^{\Theta}(t,\mathbf{y}_t^{\boldsymbol{\pi}^{\psi}})$. Algorithm 1 gives the pseudo codes.

Remark 3. *In Algorithm 1, the update rule for (Θ,ψ) in (29)-(30) 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

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

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

Inputs: m samples from data distribution, 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 (18).

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 (14)) at time $t + \Delta t$ and sample instantaneous reward $\hat{r}$ (see (19)) at time t as outputs. Policy $\boldsymbol{\pi}^\psi(\cdot|t, y)$ in (25), and q-function $q^\psi(t, y, a)$ in (28).

Learning procedure:

```
Initialize $\Theta, \psi$.
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 $(y, \hat{r}) = \mathit{Environment}_{\Delta t}(t_k, y_{t_k}, a_{t_k})$, 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 $\Theta$ and $\psi$ by
      $$\Theta \leftarrow \Theta + l(j)\alpha_\Theta \Delta\Theta, \quad (29)$$
      $$\psi \leftarrow \psi + l(j)\alpha_\psi \Delta\psi. \quad (30)$$
end for
```

Therefore, the update of (Θ, ψ) *in* (29)-(30) *can be considered as a gradient descent method for minimizing the following mean-square loss function:*

$$L(\Theta, \psi) := \sum_{i=0}^{K-1} \left[G(t_i; \Theta, \psi)\right]^2 . \tag{31}$$

As a result, instead of the SGD in (29)-(30)*, we can also optimize the loss function* (31) *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 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 could 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 4. *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 via the ratio estimator in* (18) *in order to obtain noisy samples of the running reward.*

6 Experiments

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 those of two state-of-the-art RL fine-tuning methods. All the experiments are conducted on a MacBook Pro with 4 Intel i5 Cores.

6.1 General setup

We first provide some details on the implementation, including the environment (i.e. SDE model (14)) 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 solver of the controlled SDE (14).

  For (14), we take

$$f(t) \equiv 1, \quad g(t) \equiv \sqrt{2}, \quad 0 \le t \le T, \tag{32}$$

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

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

Other forms of f and g can also be considered, but we choose (32) because it is simple and such a choice already yields satisfactory numerical results as we will see below. We numerically solve the controlled SDE (14) 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 $\xi \sim \mathcal{N}(0, I_d)$. The SDE (or environment) simulator under a policy $\boldsymbol{\pi}$ is given by (with a slight abuse of notation for $\mathbf{y}^{\boldsymbol{\pi}}_t$)

$$\mathbf{y}^{\boldsymbol{\pi}}_{t+\Delta t} - \mathbf{y}^{\boldsymbol{\pi}}_t = \left[\mathbf{y}^{\boldsymbol{\pi}}_t + 2a^{\boldsymbol{\pi}}_t\right] \Delta t + \sqrt{2\Delta t} \cdot \xi, \quad \mathbf{y}^{\boldsymbol{\pi}}_0 \sim \nu. \tag{33}$$

- **Neural network approximators.**

  We use two neural networks (NNs) for the policy μ^{ψ} (i.e., the mean of the Gaussian policy in (25)) 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 3, 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

6.2 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

$$h(y) = -(y-6)^2, \tag{34}$$

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 (33)). The resulting probability density functions (PDFs), computed via kernel density estimation using the Python seaborn library, are plotted in Figure 1.

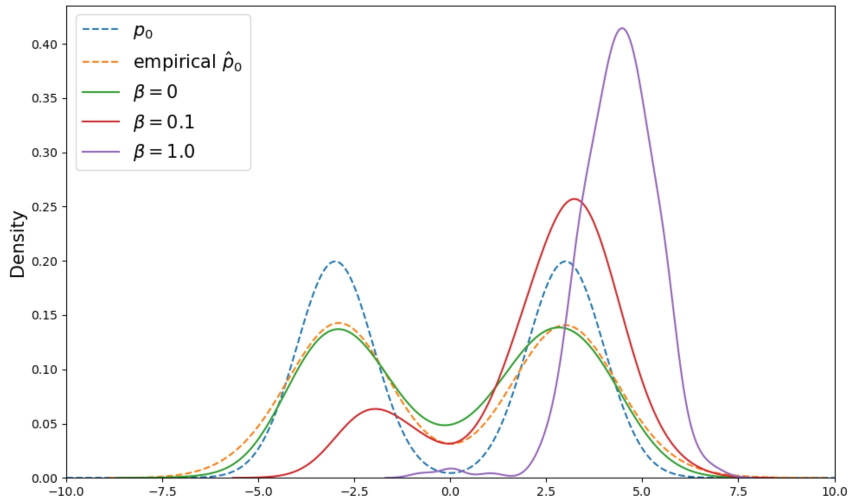

Figure 1: Learned probability densities of generated samples under different β's in 1D example

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 (34). 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

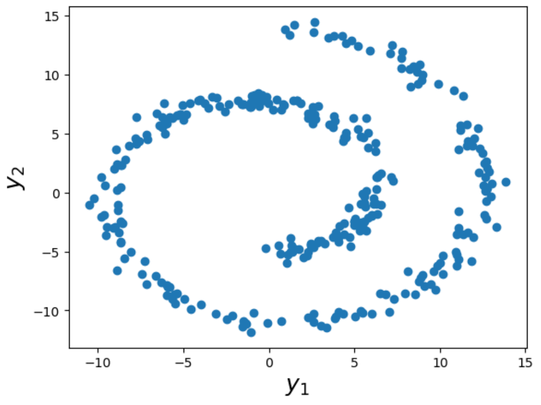

Figure 2: 300 samples from Swiss roll data

6.3 2D example - Swiss roll

We next consider 2D Swiss Rolls data, and plot $m = 300$ training samples drawn from 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 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.

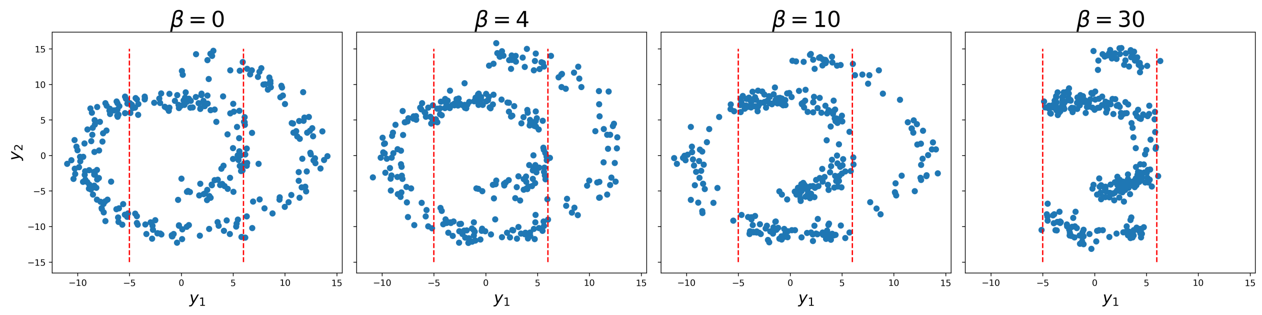

Figure 3: 300 generated samples under different β's for 2D Swiss rolls

6.4 Comparison with the 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 this is the key difference of their formulations compared with ours: our formulation does not involve a pretrained model and our algorithm is not designed for fine-tuning. For reader's convenience, we first briefly review their formulations.

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

[4]We use the dataset provided by 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).

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:

$$\min_{\phi} \mathbb{E}_{p_{\phi}(\mathbf{y}_{0:K})} \left[-\beta \cdot h(\mathbf{y}_{t_K}) + \gamma \cdot \sum_{k=1}^{K} \mathrm{KL} \left(p_{\phi}(\mathbf{y}_{t_k}|\mathbf{y}_{t_{k-1}}) || p_{\mathrm{pre}}(\mathbf{y}_{t_k}|\mathbf{y}_{t_{k-1}}) \right) \right]. \tag{35}$$

Here, $p_{\mathrm{pre}}(\mathbf{y}_{t_k}|\mathbf{y}_{t_{k-1}})$ is the one-step transition probability distribution under the pretrained diffusion model, $\mathrm{KL}(\cdot||\cdot)$ 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 6.3 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 (33):

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

where $t_{k+1} = t_k + \Delta t$, and μ^{ψ} is the policy or score neural network discussed earlier (see (25)). 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 (35) 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 dynamics (33). 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 $K = T/\Delta t = 40$.

We first report the numerical results in Table 3 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. Given this pretrained model, DDPO fine-tunes it for pure reward maximization (without any KL regularization). Table 3 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 3 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. To get those metrics, we generate 100 batches of 300 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. We can observe that the performances of our q-learning algorithm and DPOK are similar in the current scenario of a good pretrained model. 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.

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

Table 3: 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.

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 Pretrained Model B, which is obtained by implementing Algorithm 1 with $\beta = 0$ and running the algorithm with only 30,000 iterations (instead of 50,000 iterations used in obtaining Pretrained Model A). 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. Table 4 reports the corresponding performances of DPOK and our 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 a even bigger divergence when applied

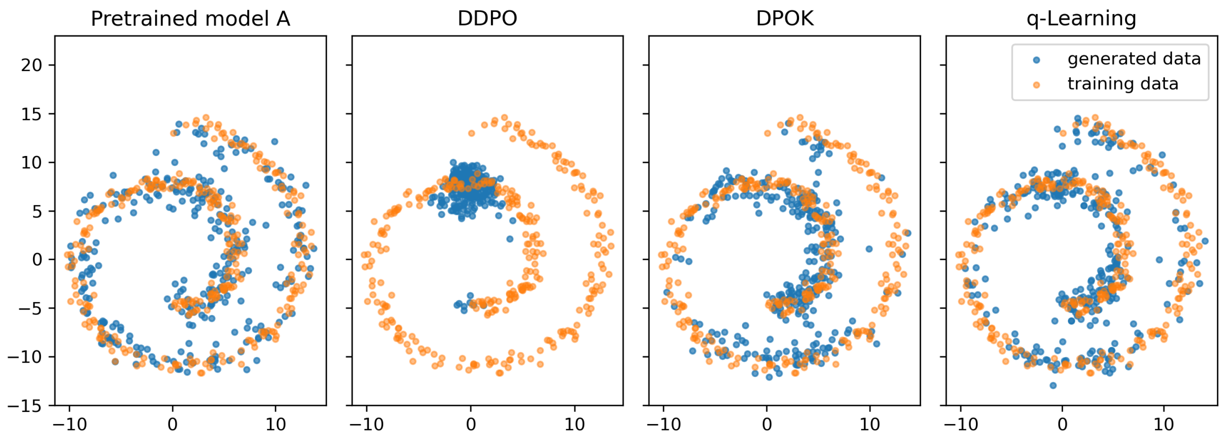

Figure 4: 300 samples generated by diffusion models fine-tuned/trained by different algorithms with a fixed expected reward of 0.80. The samples obtained from (good) Pretrained Model A are also plotted for reference.

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.

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

Table 4: 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.

To summarize the findings from our experiments, 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 algirithm, *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.

7 Extensions

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

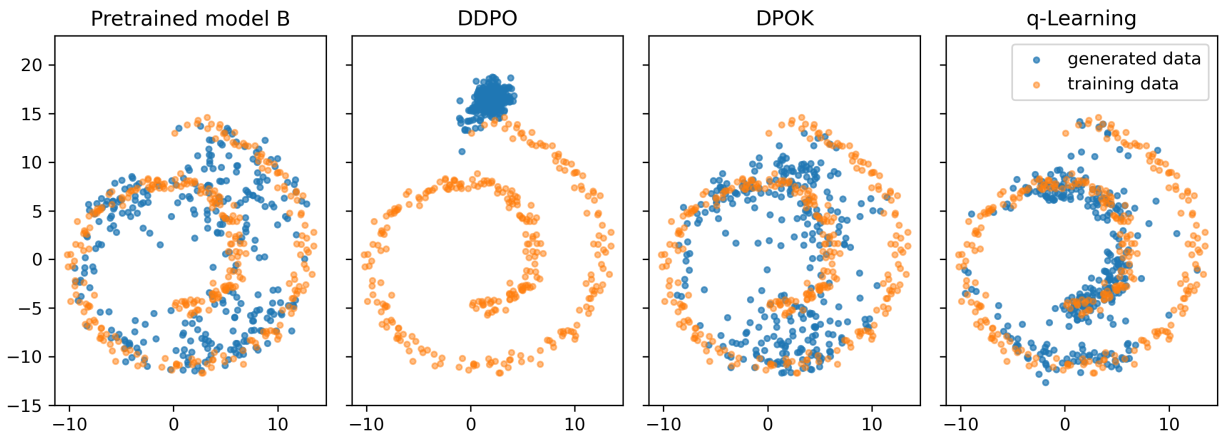

Figure 5: 300 samples generated by diffusion models fine-tuned/trained by different algorithms with a fixed expected reward of 0.80. The samples obtained from (bad) Pretrained Model B are also plotted for reference.

7.1 ODE-based formulation

In addition to the SDE-based implementation of diffusion models, another mainstream approach for sample generations is the probability flow ODE implementation (Song et al. 2021). 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:

$$d\mathbf{x}_t = -f(t)\mathbf{x}_t dt + g(t)d\mathbf{B}_t, \qquad \mathbf{x}_0 \sim p_0, \tag{36}$$

and $p_t(\cdot)$ denotes the probability density function of $\mathbf{x}_t$ in (36). Song et al. (2021) show that there exists an ODE:

$$\frac{d\bar{\mathbf{x}}_t}{dt} = f(T-t)\bar{\mathbf{x}}_t + \frac{1}{2}(g(T-t))^2\nabla_{\mathbf{x}} \log p_{T-t}(\bar{\mathbf{x}}_t), \qquad \bar{\mathbf{x}}_0 \sim p_T, \tag{37}$$

whose solution at time $t \in [0, T]$, $\bar{\mathbf{x}}_t$, is distributed according to p_{T-t}, i.e., the ODE (37) induces the same *marginal* probability density function as the SDE in (4). In particular, $\bar{\mathbf{x}}_T \sim p_0$. The ODE (37) 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):

$$\max_{\mathbf{a}=(a_t:0\le t\le T)} \left\{ \beta \cdot \mathbb{E}[h(\mathbf{y}_T^{\mathbf{a}})] - \mathbb{E}\left[\int_0^T (g(T-t))^2 \cdot |\nabla \log p_{T-t}(\mathbf{y}_t^{\mathbf{a}}) - a_t|^2 dt\right]\right\} \tag{38}$$

where

$$d\mathbf{y}_t^{\mathbf{a}} = \left[f(T-t)\mathbf{y}_t^{\mathbf{a}} + \frac{1}{2}(g(T-t))^2 a_t\right] dt, \quad \mathbf{y}_0 \sim \nu. \tag{39}$$

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

$$H(t, y, a, p) = -(g(T-t))^2|\nabla \log p_{T-t}(y) - a|^2 + [f(T-t)y + \frac{1}{2}(g(T-t))^2 a] \circ p. \tag{40}$$

Remark 5. *Because* (39) *is an ODE, instead of an SDE, the second term in the objective* (38) *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. The sample state process $\mathbf{y}^{\boldsymbol{\pi}} = \{\mathbf{y}_s^{\boldsymbol{\pi}} : 0 \leq s \leq T\}$, corresponding to a specific copy of the stochastic policy $a^{\boldsymbol{\pi}}$, follows the ODE with random/normal initialization:

$$d\mathbf{y}_s^{\boldsymbol{\pi}} = \left[f(T-s)\mathbf{y}_s^{\boldsymbol{\pi}} + \frac{1}{2}(g(T-t))^2 a_s^{\boldsymbol{\pi}} \right] ds, \quad s \in (0, T], \quad \mathbf{y}_0^{\boldsymbol{\pi}} \sim \nu. \tag{41}$$

The entropy regularized value function is given by

$$J(t, y, \boldsymbol{\pi}) = \mathbb{E}_{t,y}^{\bar{\mathbb{P}}} \left[\int_t^T \left(-g^2(T-t) \cdot |\nabla \log p_{T-s}(\mathbf{y}_s^{\boldsymbol{\pi}}) - a_s^{\boldsymbol{\pi}}|^2 - \theta \log \boldsymbol{\pi}(a_s^{\boldsymbol{\pi}} | s, \mathbf{y}_s^{\boldsymbol{\pi}}) \right) ds + \beta h(\mathbf{y}_T^{\boldsymbol{\pi}}) \right].$$

The goal of RL is to solve the following optimization problem:

$$\max_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \int J(0, y, \boldsymbol{\pi}) d\nu(y), \tag{42}$$

where $\boldsymbol{\Pi}$ stands for the set of admissible stochastic policies.

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 (40) is still a quadratic function of the action a, we deduce that the optimal stochastic policy for (42) is still a Gaussian distribution, which has the same form as (21). We can still apply Algorithm 1 to solve the RL problem (42), 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 (14). For the ODE-based formulation, we instead use a numerical solver of the ODE (41) as the sampler. Many ODE solvers have been developed and employed in practice. For instance, one can use Euler (Song et al. 2021), DDIM (Song et al. 2020), 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. (2021) and DDIM of Song et al. (2020). Following Song et al. (2020), we set $f \equiv 0$ and $g \equiv \sqrt{2}$ in our experiment. Then the controlled ODE (41) becomes

$$d\mathbf{y}_t^{\boldsymbol{\pi}} = \mathbf{a}_t^{\boldsymbol{\pi}} dt, \quad t \in [0, T] \tag{43}$$

where the action plays the role of the score function. The update rule for the ODE-Euler simulator is given by (again with a slight abuse of notation)

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

Moreover, the DDIM simulator, which is simply Euler's method applied to a reparameterization of the ODE (43), is given as follows (see Song et al. 2020)

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

where $\alpha(t) = \frac{1}{2(T-t)+1}$ and $\alpha(t) = 1$ for all $t \geq T$. The DDIM simulator converges to (43) 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 purpose, 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 (33)):

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

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 5 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 6 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 ± 0.005	0.14 ± 0.018
ODE-Euler		0.44 ± 0.006	$\mathbf{0.11} \pm \mathbf{0.016}$
SDE		0.45 ± 0.005	0.18 ± 0.020
DDIM	10 (0.5)	0.45 ± 0.005	$\mathbf{0.15} \pm \mathbf{0.019}$
ODE-Euler		0.44 ± 0.006	0.23 ± 0.022
SDE		0.46 ± 0.006	0.35 ± 0.024
DDIM	5 (1.0)	0.46 ± 0.005	0.72 ± 0.022
ODE-Euler		0.44 ± 0.006	$\mathbf{0.56} \pm \mathbf{0.024}$
SDE		0.46 ± 0.005	0.83 ± 0.026

Table 5: 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 $\beta = 0$.

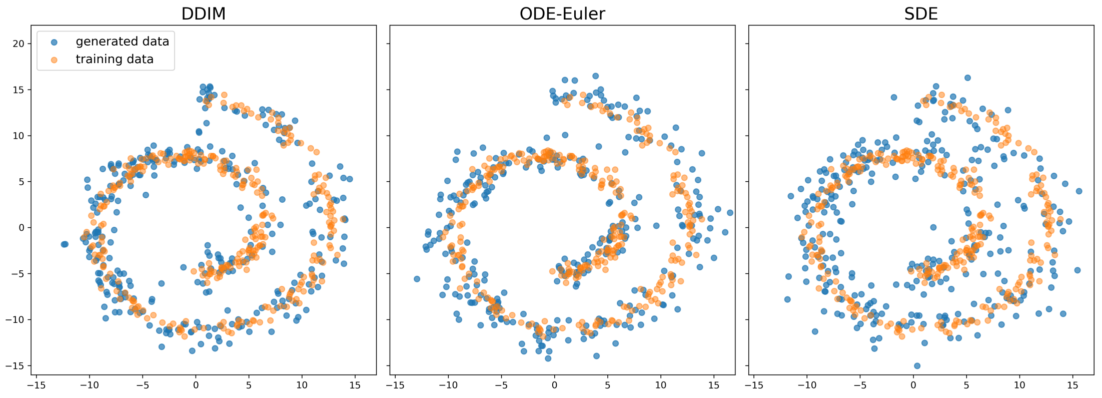

Figure 6: 300 samples generated by our algorithm for three simulators when $\beta = 0$ and $\Delta t = 0.5$.

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	50 (0.1)	0.80 $\pm$ 0.005	0.62 $\pm$ 0.022
SDE	50 (0.1)	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	10 (0.5)	0.80 $\pm$ 0.004	0.72 $\pm$ 0.024
SDE	10 (0.5)	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	5 (1.0)	0.80 $\pm$ 0.005	**1.07 $\pm$ 0.025**
SDE	5 (1.0)	0.80 $\pm$ 0.004	1.29 $\pm$ 0.027

Table 6: 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.

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 6.4, 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 6. 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 7 displays the generated samples out of the three simulators with a fixed expected reward of 0.8 and $\Delta t = 0.5$.

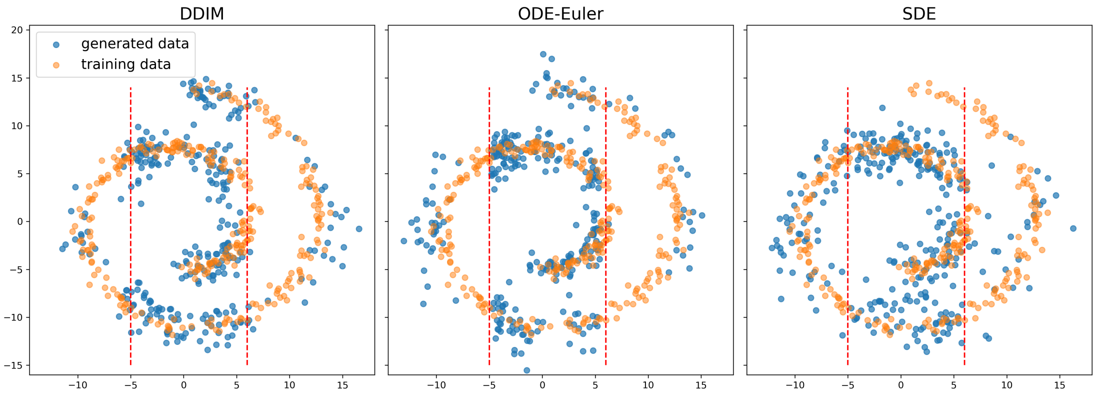

Figure 7: 300 samples generated by our algorithm for three simulators with a (fixed) expected of 0.80 and $\Delta t = 0.5$.

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.

7.2 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):

$$d\mathbf{x}_t = -f(t)\mathbf{x}_t dt + g(t) d\mathbf{B}_t, \quad \mathbf{x}_0 \sim p_0(\cdot|c), \tag{44}$$

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

$$d\mathbf{z}_t = \left[f(T-t)\mathbf{z}_t + (g(T-t))^2 \nabla \log p_{T-t}(\mathbf{z}_t|c)\right] dt + g(T-t) dW_t, \quad \mathbf{z}_0 \sim \nu, \tag{45}$$

where the initialization distribution ν is still chosen to follow the normal distribution in (5), which is independent of $\mathbf{C} = c$. In (45), 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:

$$\max_{\mathbf{a}=(a_t:0\leq t\leq T)} \left\{ \beta \cdot \mathbb{E}[h(\mathbf{y}_T^{\mathbf{a}}, \mathbf{C})] - \mathbb{E}\left[\int_0^T (g(T-t))^2 \cdot |\nabla \log p_{T-t}(\mathbf{y}_t^{\mathbf{a}}|\mathbf{C}) - a_t|^2 dt \right] \right\} \tag{46}$$

where

$$d\mathbf{y}_t^{\mathbf{a}} = \left[f(T-t)\mathbf{y}_t^{\mathbf{a}} + (g(T-t))^2 a_t \right] dt + g(T-t) dW_t, \quad \mathbf{y}_0 \sim \nu, \tag{47}$$

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:

$$r(t, \mathbf{y}, c, a) := -(g(T-t))^2 \cdot |\nabla \log p_{T-t}(\mathbf{y}|c) - a|^2. \tag{48}$$

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

$$\begin{aligned} &J(t, y, c, \boldsymbol{\pi}) \\ &= \mathbb{E}_{t,y,c}^{\bar{\mathbb{P}}} \left[\int_t^T \left(-g^2(T-t) \cdot |\nabla \log p_{T-s}(\mathbf{y}_s^{\boldsymbol{\pi}}|c) - a_s^{\boldsymbol{\pi}}|^2 - \theta \log \boldsymbol{\pi}(a_s^{\boldsymbol{\pi}}|s, \mathbf{y}_s^{\boldsymbol{\pi}}, c) \right) ds + \beta h(\mathbf{y}_T^{\boldsymbol{\pi}}, c) \right], \end{aligned}$$

where $\{\mathbf{y}_s^{\boldsymbol{\pi}} : 0 \leq s \leq T\}$ is the sample state process satisfying the SDE (14), and $\mathbb{E}_{t,y,c}^{\bar{\mathbb{P}}}$ 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

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

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

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

where $\epsilon > 0$ is some prespecified small value. It follow that we can obtain a noisy sample of the instantaneous reward (48) at time t given by

$$\hat{r}_t = -g^2(T-t) \cdot |\nabla \widehat{\log p_{T-t}}(\mathbf{y}_t|c) - a_t|^2. \tag{49}$$

From the problem (46), the Hamiltonian becomes

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

This is still a quadratic function of a, and hence the optimal stochastic policy is still Gaussian as in Proposition 1. Hence, given the condition c, we can consider Gaussian policies in the RL algorithm design:

$$\boldsymbol{\pi}^{\psi}(\cdot|t, y, c) \sim \mathcal{N}\left(\mu^{\psi}(t, y, c), \frac{\theta}{2g^2(T-t)} \cdot I_d\right) \quad \text{for all } (t, y, c). \tag{50}$$

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

$$q^{\psi}(t, y, c, a) = -g^2(T-t) \cdot |a - \mu^{\psi}(t, y, c)|^2 - \frac{\theta d}{2} \log\left(\frac{\pi\theta}{g^2(T-t)}\right). \tag{51}$$

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

8 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. The key idea is that of *data-driven*: optimization is based on a stream of score signals, instead of on an estimated score model. Our framework is general and applicable to both SDE and probability flow ODE based implementations of diffusion models. Numerically, the resulting RL algorithms is shown to perform well on at least low-dimensional synthetic data sets.

There are many open questions for future research, including speeding up the training processes, analyzing the convergence of the RL algorithms, adapting the framework to accommodate possibly several reward/cost functions, and performing high dimensional numerical experiments.

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 (18).

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 (47)) at time $t + \Delta t$ and sample instantaneous reward $\hat{r}$ (see (49)) at time t as outputs. Policy $\boldsymbol{\pi}^\psi(\cdot|t, y, c)$ in (50), and q-function $q^\psi(t, y, c, a)$ in (51).

Learning procedure:

```
Initialize $\Theta, \psi$.
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 $\Theta$ and $\psi$ by
      $$\Theta \leftarrow \Theta + l(j)\alpha_\Theta \Delta\Theta,$$
      $$\psi \leftarrow \psi + l(j)\alpha_\psi \Delta\psi.$$
end for
```

References

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

Black, K., M. Janner, Y. Du, I. Kostrikov, and S. Levine (2024). Training diffusion models with reinforcement learning. In *The Twelfth International Conference on Learning Representations*.

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

Chen, M., S. Mei, J. Fan, and M. Wang (2024). An overview of diffusion models: Applications, guided generation, statistical rates and optimization. *arXiv preprint arXiv:2404.07771*.

Clark, K., P. Vicol, K. Swersky, and D. J. Fleet (2024). Directly fine-tuning diffusion models on differentiable rewards. In *The Twelfth International Conference on Learning Representations*.

Dhariwal, P. and A. Nichol (2021). Diffusion models beat GANs on image synthesis. *Advances in Neural Information Processing Systems 34*, 8780–8794.

Fan, Y., O. Watkins, Y. Du, H. Liu, M. Ryu, C. Boutilier, P. Abbeel, M. Ghavamzadeh, K. Lee, and K. Lee (2023). Reinforcement learning for fine-tuning text-to-image diffusion models. *Advances in Neural Information Processing Systems 36*.

Haarnoja, T., A. Zhou, P. Abbeel, and S. Levine (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1861–1870. PMLR.

Haussmann, U. G. and E. Pardoux (1986). Time reversal of diffusions. *The Annals of Probability*, 1188–1205.

Ho, J., A. Jain, and P. Abbeel (2020). Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, Volume 33.

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

Hoogeboom, E., V. G. Satorras, C. Vignac, and M. Welling (2022). Equivariant diffusion for molecule generation in 3d. In *International conference on machine learning*, pp. 8867–8887. PMLR.

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

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

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

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

Karras, T., M. Aittala, T. Aila, and S. Laine (2022). Elucidating the design space of diffusion-based generative models. In *Advances in Neural Information Processing Systems*, Volume 35.

Kingma, D. P. and J. Ba (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*.

Lai, C.-H., Y. Takida, N. Murata, T. Uesaka, Y. Mitsufuji, and S. Ermon (2023). FP-Diffusion: Improving score-based diffusion models by enforcing the underlying score fokker-planck equation. In *International Conference on Machine Learning*, pp. 18365–18398. PMLR.

Lee, H., J. Lu, and Y. Tan (2022). Convergence for score-based generative modeling with polynomial complexity. In *Advances in Neural Information Processing Systems*, Volume 35.

Lee, K., H. Liu, M. Ryu, O. Watkins, Y. Du, C. Boutilier, P. Abbeel, M. Ghavamzadeh, and S. S. Gu (2023). Aligning text-to-image models using human feedback. *arXiv preprint arXiv:2302.12192*.

Lu, C., Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu (2022). DPM-solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In *Advances in Neural Information Processing Systems*, Volume 35, pp. 5775–5787.

Ramesh, A., P. Dhariwal, A. Nichol, C. Chu, and M. Chen (2022). Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*.

Rombach, R., A. Blattmann, D. Lorenz, P. Esser, and B. Ommer (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695.

Sohl-Dickstein, J., E. Weiss, N. Maheswaranathan, and S. Ganguli (2015a). Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, Volume 37, pp. 2256–2265. PMLR.

Sohl-Dickstein, J., E. Weiss, N. Maheswaranathan, and S. Ganguli (2015b). Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. PMLR.

Song, J., C. Meng, and S. Ermon (2020). Denoising diffusion implicit models. In *International Conference on Learning Representations*.

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

Song, Y. and S. Ermon (2019). Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, Volume 32.

Song, Y., S. Garg, J. Shi, and S. Ermon (2020). Sliced score matching: A scalable approach to density and score estimation. In *Uncertainty in Artificial Intelligence*, pp. 574–584. PMLR.

Song, Y., J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole (2021). Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*.

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

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

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

Wang, H., T. Zariphopoulou, and X. Y. Zhou (2020). Reinforcement learning in continuous time and space: A stochastic control approach. *Journal of Machine Learning Research 21*, (198):1–34.

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

Wu, L., C. Gong, X. Liu, M. Ye, and Q. Liu (2022). Diffusion-based molecule generation with informative prior bridges. *Advances in Neural Information Processing Systems 35*, 36533–36545.

Yang, L., Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, Y. Shao, W. Zhang, B. Cui, and M.-H. Yang (2023). Diffusion models: A comprehensive survey of methods and applications. *ACM Copmuting Surveys 56*(4), 1–39.

Yong, J. and X. Y. Zhou (2012). *Stochastic controls: Hamiltonian systems and HJB equations*, Volume 43. Springer Science & Business Media.

Zhang, Q. and Y. Chen (2023). Fast sampling of diffusion models with exponential integrator. In *International Conference on Learning Representations*.

Zhao, H., J. Zhang, X. Gu, D. Yao, and W. Tang (2024). Score as actions: Diffusion models alignment by continuous-time reinforcement learning. *Working paper*.