Sampling from Energy-based Policies using Diffusion

Vineet Jain, Tara Akhound-Sadegh, Siamak Ravanbakhsh

Keywords: Energy-based policies, Boltzmann policies, diffusion models.

Summary

Energy-based policies offer a flexible framework for modeling complex, multimodal behaviors in reinforcement learning (RL). In maximum entropy RL, the optimal policy is a Boltzmann distribution derived from the soft Q-function, but direct sampling from this distribution in continuous action spaces is computationally intractable. As a result, existing methods typically use simpler parametric distributions, like Gaussians, for policy representation limiting their ability to capture the full complexity of multimodal action distributions. In this paper, we introduce a diffusion-based approach for sampling from energy-based policies, where the negative Q-function defines the energy function. Based on this approach, we propose an actor-critic method called Diffusion Q-Sampling (DQS) that enables more expressive policy representations, allowing stable learning in diverse environments. We show that our approach enhances sample efficiency in continuous control tasks and captures multimodal behaviors, addressing key limitations of existing methods.

Contribution(s)

1. We develop a novel actor-critic reinforcement learning algorithm such that the policy samples actions from the Boltzmann distribution of the Q-function. We achieve this by using a diffusion model to parameterize the policy that explicitly learns the score function of the target Boltzmann density.
   Context: Boltzmann policies are a popular choice in discrete action spaces. However, sampling from these policies in continuous action spaces is generally intractable. Prior work (Psenka et al., 2023) used Langevin sampling to address this challenge. Other applications of diffusion models (Wang et al., 2024) backpropagate the gradient through the entire diffusion chain to maximize Q-values. To the best of our knowledge, our method is the first to use diffusion models to explicitly sample from Boltzmann policies.
2. Experiments on continuous control tasks demonstrate improved sample efficiency of our method compared to relevant baselines.
   Context: We observe higher returns with fewer number of environment interactions (compared to our baselines) on a majority of tasks.
3. We demonstrate that our proposed method can learn multimodal behaviors in maze navigation tasks.
   Context: Our setup consists of a maze with two possible goals. Multimodality in this context refers to the ability of an agent to reach both goals from some initial state, and discover multiple paths (if they exist) to a goal. We qualitatively examine the trajectories of a trained agent and compare them with respect to goal coverage and diversity of paths.

Sampling from Energy-based Policies using Diffusion

Vineet Jain, Tara Akhound-Sadegh, Siamak Ravanbakhsh
{jain.vineet, tara.akhoundsadegh, siamak.ravanbakhsh}@mila.quebec

School of Computer Science, McGill University
Mila - Quebec AI Institute

Abstract

Energy-based policies offer a flexible framework for modeling complex, multimodal behaviors in reinforcement learning (RL). In maximum entropy RL, the optimal policy is a Boltzmann distribution derived from the soft Q-function, but direct sampling from this distribution in continuous action spaces is computationally intractable. As a result, existing methods typically use simpler parametric distributions, like Gaussians, for policy representation — limiting their ability to capture the full complexity of multimodal action distributions. In this paper, we introduce a diffusion-based approach for sampling from energy-based policies, where the negative Q-function defines the energy function. Based on this approach, we propose an actor-critic method called DIFFUSION Q-SAMPLING (DQS) that enables more expressive policy representations, allowing stable learning in diverse environments. We show that our approach enhances sample efficiency in continuous control tasks and captures multimodal behaviors, addressing key limitations of existing methods.

1 Introduction

Deep reinforcement learning (RL) is a powerful paradigm for learning complex behaviors in diverse domains, from strategy-oriented games (Silver et al., 2016; Berner et al., 2019; Schrittwieser et al., 2020) to fine-grained control in robotics (Kober et al., 2013; Sünderhauf et al., 2018; Wu et al., 2023). In the RL framework, an agent learns to make decisions by interacting with an environment and receiving feedback in the form of reward. The agent aims to learn a policy that maximizes the cumulative sum of rewards over time by exploring actions and exploiting known information about the environment's dynamics.

The parameterization of the policy is a crucial design choice for any RL algorithm. Under the conventional notion of optimality, under full observability, there always exists an optimal deterministic policy that maximizes the long-term return (Sutton & Barto, 2018). However, this is only true when the agent has explored sufficiently and has nothing to learn about the environment. Exploration requires a stochastic policy to experiment with different potentially rewarding actions. Moreover, even in the exploitation phase, there may be more than one way of performing a task, and we might be interested in mastering all of them. This diversification is motivated by the robustness of the resulting policy to environment changes; if certain pathways for achieving a task become infeasible due to a change of the dynamics or reward, some may remain feasible, and the agent has an easier time in adapting to this change by exploiting and improving the viable options. This argument also suggests that such policies can serve as effective initialization for fine-tuning on specific tasks.

While exploration, diversity and robustness motivate stochastic policies, representing such policies in continuous action spaces remains challenging. As a result, stochasticity is often introduced by noise injection (Lillicrap et al., 2015b) or using an arbitrary parametric family (Schulman et al., 2015) which lacks expressivity. Orthogonal to the difficulty of representing such policies is their

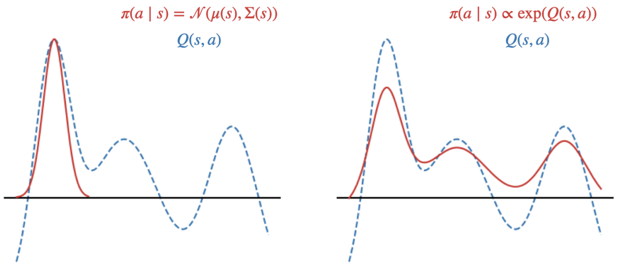

Figure 1: Illustration comparing Gaussian and Boltzmann policies.

training objective; policies are often optimized to maximize the Q-function, and stochasticity is introduced to encourage exploration as an afterthought. However, our argument for stochasticity favours multi-modal policies; instead of learning the *single best* way to solve a task, we want to learn *all reasonably good* ways to solve the task.

We address both of these issues by explicitly sampling from energy-based policies of the form,

$$\pi(a \mid s) \propto \exp(Q^{\pi}(s,a)).$$

This is also known as the Boltzmann distribution of the Q-function. The optimal policy in the maximum entropy RL framework is also known to be of this form, except it uses the soft Q-function (Haarnoja et al., 2017). Such a policy has several benefits. First, it offers a principled way to balance exploration and exploitation in continuous action spaces. By sampling from this distribution, the policy still prioritizes actions with high Q-values but also has a non-zero probability of sampling sub-optimal actions. While the use of Boltzmann policies is common in the discrete setting, it is challenging in continuous spaces. This sampling problem is often tackled with Markov Chain Monte Carlo (MCMC) techniques, which can be computationally expensive and suffer from exponential mixing time. Second, this formulation naturally incorporates multimodal behavior, since the policy can sample one of multiple viable actions at any given state. However, such policies are generally intractable to sample from in continuous action spaces, requiring approximations in policy parameterization often at the cost of expressivity.

Diffusion models offer a potential solution to the policy parameterization problem since they are expressive and can produce high-quality samples from complex distributions. Indeed, they have been extensively applied to solve sequential decision-making tasks, especially in offline settings where they can model multimodal datasets from suboptimal policies or diverse human demonstrations. A few studies have applied these models in the online setting, focusing on deriving training objectives for policy optimization via diffusion. Yang et al. (2023) uses the gradient of the Q-function to refine actions sampled from a diffusion policy; however, the exact form of the policy is unspecified, and it is unknown what distribution the diffusion models sample from. Psenka et al. (2023) samples from the Boltzmann distribution of the Q-function using Langevin dynamics, which may suffer from insufficient mode coverage in high dimensions. Wang et al. (2024) uses diffusion policy within the maximum entropy framework, where the entropy is approximated using a mixture of Gaussians. In contrast, our approach directly samples from $\exp(Q^{\pi}(s,a))$ by constructing a diffusion process that estimates the score of this target Boltzmann density at different noise scales.

Our contributions in this work are as follows:

- We propose a novel actor-critic algorithm, DIFFUSION Q-SAMPLING (DQS), for sequential decision-making using diffusion models for sampling from energy-based policies.
- We show that DQS approximately samples from the Boltzmann density of the Q-function.
- We demonstrate that DQS is more sample efficient compared to both classical actor-critic methods and more recent diffusion-based methods in continuous control tasks.
- We demonstrate that DQS can learn multimodal behaviors in maze navigation tasks.

2 Related work

Our work is related to two distinct sub-areas of reinforcement learning: the relatively new and actively explored line of work on applying diffusion models in the RL setting, and the classical maximum entropy RL framework.

Diffusion models in RL. Early work on applying diffusion models for RL was focused on behavior cloning in the offline setting (Chi et al., 2023). This setting more closely matches the original purpose of diffusion models - to match a distribution to a given dataset. Janner et al. (2022); Ajay et al. (2023) use a diffusion model trained on the offline data for trajectory optimization, while Reuss et al. (2023); Jain & Ravanbakhsh (2024) apply diffusion models to offline goal-reaching tasks by additionally conditioning the score function on the goal. Within the behavior cloning setting, there is some existing work on learning a stochastic state dynamics model using diffusion (Li et al., 2022).

Beyond behavior cloning, offline RL methods incorporate elements from Q-learning to learn a value function from the offline dataset and leverage the learned Q-function to improve the diffusion policy. A large body of work exists in this sub-field, where the most common approach is to parameterize the policy using a diffusion model and propose different training objectives to train the diffusion policy. Wang et al. (2023); Kang et al. (2024) add a Q-function maximizing term to the diffusion training objective, and Hansen-Estruch et al. (2023) use an actor-critic framework based on a diffusion policy and Implicit Q-learning (Kostrikov et al., 2021). Lu et al. (2023) take an energy-guidance approach, where they frame the problem as using the Q-function to guide the behavior cloning policy to high reward trajectories.

The application of diffusion models has been relatively less explored in the online setting (Ding & Jin, 2023). DIPO (Yang et al., 2023) modifies actions in the replay buffer based on the gradient of the Q-function, then trains a diffusion model on the modified actions. QSM (Psenka et al., 2023) directly trains a neural network to match the gradient of the Q-function, then uses Langevin diffusion for sampling. In contrast, our method models the policy as a Boltzmann distribution using a diffusion model that can represent complex distributions.

Maximum entropy RL. In contrast to standard RL, where the goal is to maximize expected returns, in the maximum entropy RL framework, the value function is augmented by Shannon entropy of the policy. Ziebart et al. (2008) applied such an approach in the context of inverse reinforcement learning and Haarnoja et al. (2017) generalized this approach by presenting soft Q-learning to learn energy-based policies. A follow-up work, Haarnoja et al. (2018a), presented the well-known soft actor-critic (SAC) algorithm. This line of work proposes to learn a soft value function by adding the entropy of the policy to the reward. The optimal policy within this framework is a Boltzmann distribution, where actions are sampled based on the exponentiated soft Q-values. Some recent works use diffusion models within this framework, such as DACER (Wang et al., 2024), which uses a diffusion policy to represent a maximum entropy policy and estimates the entropy using a mixture of Gaussians. A recent work (Ishfaq et al., 2025) uses Langevin Monte Carlo to improve critic learning through uncertainty estimation over policy optimization.

A separate but related line of work on generative flow networks (GFlowNets), originally defined in the discrete case (Bengio et al., 2021; 2023), learns a policy that samples terminal states proportional to the Boltzmann density corresponding to some energy function. They have been extended to the continuous setting (Lahlou et al., 2023) and under certain assumptions, they are equivalent to maximum entropy RL (Tiapkin et al., 2024; Deleu et al., 2024). They can effectively sample from the target distribution using off-policy exploration, however, they encounter challenges in credit assignment and exploration efficiency (Malkin et al., 2022; Madan et al., 2023; Rector-Brooks et al., 2023; Shen et al., 2023). Our approach is distinct as we sample the action at each step from the Boltzmann density of the Q-function, instead of the terminal states based on the reward.

3 Preliminaries

3.1 Reinforcement learning

We consider a finite-horizon Markov Decision Process (MDP) denoted by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where the state space $\mathcal{S}$ and action space $\mathcal{A}$ are continuous. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ denotes the transition probability of the next state $\mathbf{s}_{t+1} \in \mathcal{S}$ given the current state $\mathbf{s}_t \in \mathcal{S}$ and action $\mathbf{a}_t \in \mathcal{A}$. The reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is assumed to be bounded $r(\mathbf{s}, \mathbf{a}) \in [r_{\min}, r_{\max}]$. $\gamma \in [0, 1]$ is the discount factor.

A policy $\pi : \mathcal{S} \to \mathcal{A}$ produces an action for every state $s \in \mathcal{S}$. In the standard RL framework, the objective is to learn a policy that maximizes the expected sum of rewards $\sum_t \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)} \left[\gamma^t r(s_t, a_t)\right]$.

The actor-critic framework is a commonly used approach for learning such policies. It involves optimizing a policy (the actor) to choose actions that maximize the action value function, also known as the Q-function (the critic). The Q-function is defined as the sum of expected future rewards starting from a given state-action pair, and thereafter following some policy π until terminal time step T:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k) \mid s_t = s, a_t = a \right].$$

The optimal policy is defined as the policy that maximizes the sum of rewards along a trajectory:

$$\pi^* = \operatorname*{argmax}_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right].$$

3.2 Diffusion models

Denoising diffusion. Denoising diffusion (Dinh et al., 2016; Ho et al., 2020; Song et al., 2021) refers to a class of generative models which relies on a stochastic process which progressively transforms the target data distribution to a Gaussian distribution. The time-reversal of this diffusion process gives the generative process which can be used to transform noise into samples from the target data distribution.

The forward noising process is a stochastic differential equation:

$$dx_\tau = -\alpha(\tau) x_\tau d\tau + g(\tau) dw_\tau \,, \tag{1}$$

where w_τ denotes Brownian motion. In this paper, we consider the Variance Exploding (VE) SDE where the decay rate, α, is set to $\alpha(\tau) = 0$. This noising process starts with samples from the target density $x_0 \sim p_0$ and progressively adds noise to them over a diffusion time interval $\tau \in [0, 1]$. The marginal probability distribution at time τ is denoted by p_τ and is the convolution of the target density p_0 with a normal distribution with a time-dependent variance, σ_τ^2. For the VE setting we consider, these marginal distributions are given by:

$$p_\tau(x_\tau) = \int_0^\tau p_0(x_0) \mathcal{N}(x_\tau; x_0, \sigma_\tau^2) dx_0 \,, \tag{2}$$

where the variance is related to the diffusion coefficient, $g(\tau)$ via $\sigma_\tau^2 = \int_0^\tau g(\xi)^2 d\xi$.

The generative process corresponding to the corresponding to Equation (1) is an SDE with Brownian motion $\bar{w}_\tau$, given by:

$$dx_\tau = [-\alpha(\tau) x_\tau - g(\tau)^2 \nabla \log p_\tau(x_\tau)] d\tau + g(\tau) d\bar{w}_\tau \,. \tag{3}$$

Therefore, to be able to generate data, we need to estimate the score of the intermediate distributions, $\nabla \log p_\tau(x_\tau)$.

Iterated Denoising Energy Matching. Recently, Akhound-Sadegh et al. (2024) proposed an algorithm known as iDEM (Iterated Denoising Energy Matching) for sampling from a Boltzmann-type target distribution, $p_0(x) \propto \exp(-\mathcal{E}(x))$, where $\mathcal{E}$ denotes the energy. iDEM is a diffusion-based neural sampler, which estimates the diffusion score, $\nabla \log p_\tau$ using a Monte Carlo estimator. Given the VE diffusion path defined above, iDEM rewrites the score of the marginal densities as:

$$\begin{aligned}\nabla \log p_\tau &= \frac{\int \nabla \exp(-\mathcal{E}(x_0))\mathcal{N}(x_\tau; x_0, \sigma_\tau^2)dx_0}{\int \exp(-\mathcal{E}(x_0))\mathcal{N}(x_\tau; x_0, \sigma_\tau^2)dx_0} \\ &= \frac{\mathbb{E}_{\tilde{x}\sim\mathcal{N}(x_\tau, \sigma_\tau^2)}\left[\nabla \exp(-\mathcal{E}(x)\right]}{\mathbb{E}_{\tilde{x}\sim\mathcal{N}(x_\tau, \sigma_\tau^2)}\left[\exp(-\mathcal{E}(x)\right]}.\end{aligned} \tag{4}$$

By observing that the above equation can be written as the gradient of a logarithm leads to the K-sample Monte-Carlo estimator of the score:

$$S_k(x_\tau, \tau) = \nabla_{x_\tau} \log \sum_{i=1}^{K} \exp\left(-\mathcal{E}(\tilde{x}^{(i)})\right), \quad \tilde{x}^{(i)} \sim \mathcal{N}(x_\tau, \sigma_\tau^2). \tag{5}$$

A score-network, f_ϕ is trained to regress to the MC estimator, S_K. The network is trained using a bi-level iterative scheme: (1) in the outer-loop a replay buffer is populated with samples that are generated using the model and (2) in the inner-loop the network is regressed $S_k(x_\tau, \tau)$ where x_τ are noised samples from the replay buffer.

4 An actor-critic algorithm for Boltzmann policies

Our objective is to learn general policies of the form $\pi(\mathbf{a} \mid \mathbf{s}) \propto \exp(-\mathcal{E}(\mathbf{s}, \mathbf{a}))$, where $\mathcal{E}$ represents an energy function which specifies the desirability of state-action pairs. By setting the Q-function, $Q(\mathbf{s}, \mathbf{a})$ as the negative energy, we get what is known as the Boltzmann policy:

$$\pi(\mathbf{a} \mid \mathbf{s}; T) = \frac{\exp(\frac{1}{T}Q(\mathbf{s}, \mathbf{a}))}{\int_{\mathbf{a}} \exp(\frac{1}{T}Q(\mathbf{s}, \mathbf{a}))d\mathbf{a}}. \tag{6}$$

Choosing such a policy gives us a principled way to balance exploration and exploitation. Specifically, by scaling the energy function with a temperature parameter T and annealing it to zero, we get a policy that initially explores to collect more information about the environment and over time exploits the knowledge it has gained.

4.1 Diffusion Q-Sampling

We propose an off-policy actor-critic algorithm, which we call Diffusion Q-Sampling (DQS), based on the above formulation. Being an off-policy method means DQS can reuse past interactions with the environment by storing them in a replay buffer $\mathcal{D}$, improving sample efficiency.

Let Q_θ denote the Q-function and π_ϕ a parametric policy, where θ, ϕ represent the parameters of a neural network. The Q-function is learned using standard temporal difference learning:

$$J(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)\sim\mathcal{D}}\left[\left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}(\mathbf{s}_t, \mathbf{a}_t)\right)^2\right], \tag{7}$$

where

$$\hat{Q}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\, \mathbb{E}_{\mathbf{s}_{t+1}\sim\mathcal{P}, \mathbf{a}_{t+1}\sim\pi_\phi}\left[Q_{\bar{\theta}}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})\right].$$

The target Q-values, $\hat{Q}$, make use of a target Q-network denoted by $Q_{\bar{\theta}}$, where the parameters $\bar{\theta}$ are usually an exponentially moving average of the Q-network parameters θ. Also, in practice, the expectation over next states $\mathbf{s}_{t+1}$ is estimated using only a single sample.

We parameterize the policy using a diffusion process and use iDEM (Akhound-Sadegh et al., 2024) to sample actions from the target density $\pi(\cdot|\mathbf{s}_t) \propto \exp(Q_\theta(\mathbf{s}_t, \mathbf{a}_t))$.

Algorithm 1: Diffusion Q-Sampling (DQS)

```
Initialize: Initialize Q-function parameters $\theta$, policy parameters $\phi$, target network $\bar{\theta} \leftarrow \theta$,
            replay buffer $\mathcal{D}$
for each iteration do
    // Environment Interaction
    for each environment step do
        Observe state $\mathbf{s}_t$ and sample action $\mathbf{a}_t$ via reverse diffusion using $f_\phi$
        Execute $\mathbf{a}_t$, observe reward $r_t$ and next state $\mathbf{s}_{t+1}$
        Store transition $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ in $\mathcal{D}$
    end
    // Parameter Updates
    for each gradient step do
        Sample minibatch $\mathcal{B} = \{(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})\}$ from $\mathcal{D}$
        // Update Q-function parameters $\theta$
        Compute target Q-values: $\hat{Q}_t = r_t + \gamma Q_{\bar{\theta}}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}), \quad \mathbf{a}_{t+1} \sim \pi_\phi(\mathbf{s}_{t+1})$
        Update $\theta$ by minimizing: $J(\theta) = \frac{1}{|\mathcal{B}|}\sum_{\mathcal{B}}\left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}_t\right)^2$
        // Update policy parameters $\phi$
        for each $(\mathbf{s}_t, \mathbf{a}_t)$ in $\mathcal{B}$ do
            Sample diffusion time $\tau \sim \mathcal{U}[0,1]$
            Sample noisy action $\mathbf{a}_{t,\tau} \sim \mathcal{N}(\mathbf{a}_t, \sigma_\tau^2 \mathbf{I})$
            Sample $\{\tilde{\mathbf{a}}_t^{(i)}\}_{i=1}^K$, where $\tilde{\mathbf{a}}_t^{(i)} \sim \mathcal{N}(\mathbf{a}_{t,\tau}, \sigma_\tau^2 \mathbf{I})$
            Estimate score: $S_t = \nabla_{\mathbf{a}_{t,\tau}} \log \sum_{i=1}^K \exp\left(Q_\theta(\mathbf{s}_t, \tilde{\mathbf{a}}_t^{(i)})\right)$
            Update $\phi$ by minimizing: $J(\phi) = \|f_\phi(\mathbf{s}_t, \mathbf{a}_{t,\tau}, \tau) - S_t\|^2$
        end
        // Update target network
        Update $\bar{\theta} \leftarrow \eta\,\theta + (1-\eta)\,\bar{\theta}$
    end
end
```

Forward process. Given $(\mathbf{s}_t, \mathbf{a}_t) \in \mathcal{S} \times \mathcal{A}$, we progressively add Gaussian noise to the action following some noise schedule. Let $\mathbf{a}_{t,\tau}$ denote the noisy action at diffusion step $\tau \in [0,1]$, such that:

$$\mathbf{a}_{t,0} = \mathbf{a}_t; \qquad \mathbf{a}_{t,\tau} \sim \mathcal{N}(\mathbf{a}_t, \sigma_\tau^2 I).$$

We choose a geometric noise schedule $\sigma_\tau = \sigma_{\text{min}}\left(\frac{\sigma_{\text{max}}}{\sigma_{\text{min}}}\right)^\tau$, where σ_{min} and σ_{max} are hyperparameters. We found it sufficient to set $\sigma_{\text{min}} = 10^{-5}$ and $\sigma_{\text{max}} = 1.0$ for all our experiments.

Reverse process. Given noisy action samples, we iteratively denoise them using a learned score function to produce a sample from the target action distribution $\pi(\cdot|\mathbf{s}_t) \propto \exp(Q^\pi(\mathbf{s}_t, \cdot))$. We train a neural network, f_ϕ to match iDEM's K-sample Monte Carlo estimator of the score, defined in Equation (5), by setting the negative Q-function as the energy function. The score function takes as input the noisy action and diffusion time, while also being conditioned on the current state. The loss function is given by:

$$J(\phi) = \mathbb{E}_{\substack{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}, \tau \sim U[0,1], \\ \mathbf{a}_{t,\tau} \sim \mathcal{N}(\mathbf{a}_t, \sigma_\tau^2 I)}}\left[\left\| f_\phi(\mathbf{s}_t, \mathbf{a}_{t,\tau}, \tau) - \nabla_{\mathbf{a}_{t,\tau}} \log \sum_{i=1}^K \exp(Q_\theta(\mathbf{s}_t, \tilde{\mathbf{a}}_t^{(i)}))\right\|^2\right], \tag{8}$$

where $\tilde{\mathbf{a}}_t^{(i)} \sim \mathcal{N}(\mathbf{a}_{t,\tau}, \sigma_\tau^2 I)$.

Summarizing, to sample an action $\mathbf{a}_t$ for the current state $\mathbf{s}_t$ such that $\pi_\phi(\mathbf{a}_t|\mathbf{s}_t) \propto \exp(Q^{\pi_\phi}(\mathbf{s}_t, \mathbf{a}_t))$, we first sample noise from the prior (corresponding to diffusion time $\tau = 1$) $\mathbf{a}_{t,1} \sim \mathcal{N}(0, \sigma_1^2)$. We then use Equation (3) in the VE setting (i.e. $\alpha(\tau) = 0$) by using the trained score function f_ϕ in place of $\nabla \log p_\tau$ to iteratively denoise samples produce the action sample $\mathbf{a}_t$. The full algorithm is presented in Algorithm 1.

Temperature. We can incorporate the temperature parameter T from Equation (6) within our framework by simply scaling the Q-function in Equation (8) and regressing to the estimated score of the temperature-scaled Boltzmann distribution. To enable the score network to model this temperature scaling accurately, we additionally condition f_ϕ on the current temperature. Generally, the temperature is set to a high value initially, and is annealed over time such that at $t \to \infty$, we have $T \to 0$. In practice, the temperature is annealed to a sufficiently small value for large time steps. This ensures that the policy explores initially and as it collects more information about the environment, starts exploiting more and more as time passes.

4.2 An illustrative experiment

In this section, we aim to answer the question: *does* DQS *effectively learn a Boltzmann policy?*

Since the policy learning is based on iDEM, one may assume that simply optimizing Equation (8) should produce a policy that samples from the Boltzmann distribution of $Q(s, a)$. However, the learning dynamics in the interactive setting are fundamentally different from the sampling setting.

In the sampling case, the diffusion model is trained to sample from a *known, fixed* energy function. This means that the target score in Equation (8) corresponds to a static function. In the actor-critic algorithm described above, the diffusion policy tracks the Q-function of the current policy. Since this Q-function is learned simultaneously along with the policy, it is a *moving* target, hence it is not immediately obvious whether the proposed algorithm leads to stable learning.

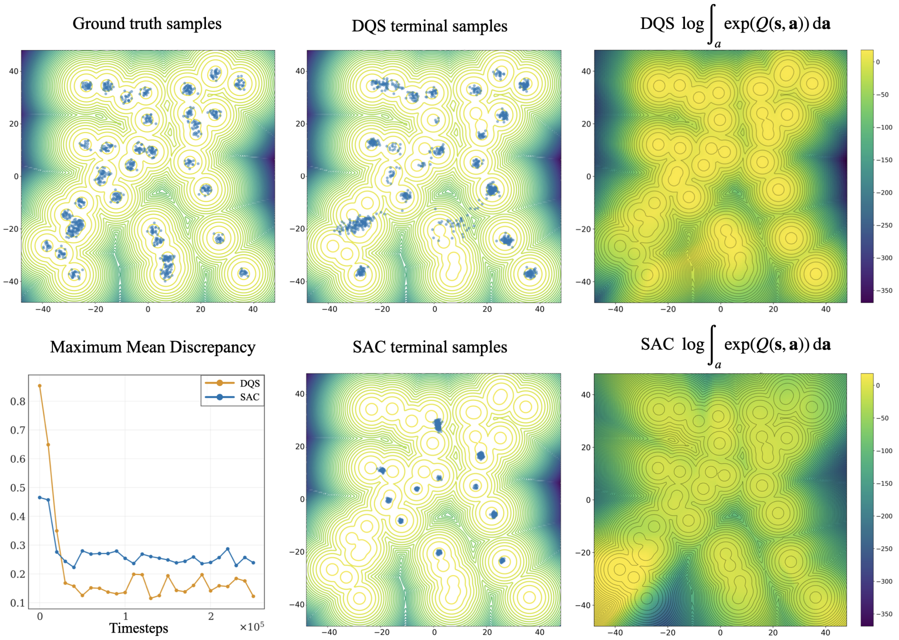

Figure 2: Ground truth, DQS , and SAC terminal samples for the Gaussian mixture experiment. The right panels show the log partition function based on the learned Q-functions for DQS and SAC, with contours showing the ground truth density for reference. We evaluate the samples using maximum mean discrepancy (MMD) with the ground truth samples (bottom left).

We test DQS in a controlled setting where we can qualitatively and quantitatively compare with a known ground truth distribution. Consider 2-dimensional state space (x, y), an action space $(\Delta x, \Delta y)$ with the actions normalized to be unit length, and an episode length of 100 steps. The reward function is the log likelihood of samples under a mixture of 40 Gaussian distributions. We train a Q-function and diffusion policy using DQS (Algorithm 1) and measure the Maximum Mean Discrepancy (MMD) between the final policy samples and the ground truth samples. Figure 2 plots the final samples at the end of the episode and the log partition function $\log Z = \log \int_{\mathbf{a}} Q(\mathbf{s}_t, \mathbf{a}) \mathrm{d}\mathbf{a}$. To demonstrate the benefit of using a diffusion-based policy, we use soft actor-critic (SAC) with a Gaussian policy as a representative baseline. We observe that DQS approximates the ground truth samples more closely and covers most modes, which is also corroborated by a lower MMD compared to SAC.

5 Experiments

We perform experiments to answer the following major questions:

- Does DQS offer improved sample efficiency in continuous control tasks?
- Can DQS learn multimodal behaviors, i.e., learn multiple ways to solve a task?

Baselines. We test our method against a number of relevant methods. This includes classical RL algorithms such as (1) Soft Actor-Critic (SAC) (Haarnoja et al., 2018a), a maximum entropy RL method that is widely used for continuous state-action spaces; (2) Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015a) which uses a deterministic policy and directly backpropagates gradients through the Q-function; (3) Proximal Policy Optimization (PPO) (Schulman et al., 2017), an on-policy policy gradient algorithm that uses a clipped objective for stable updates; and (4) Twin Delayed DDPG (TD3) (Fujimoto et al., 2018), an off-policy actor-critic method that mitigates overestimation bias by training two critics and delaying policy updates.

We also compare against some recent diffusion-based RL algorithms, including (5) Q-Score Matching (QSM) (Psenka et al., 2023), a diffusion-based approach that trains a score function to match the gradient of the Q-function and uses this score function to sample actions; (6) Diffusion Actor-Critic with Entropy Regulator (DACER) (Wang et al., 2024), which uses a diffusion-based maximum entropy policy along with Gaussian mixture models to estimate entropy; and (7) Diffusion Policy (DIPO) (Yang et al., 2023) which samples actions from a diffusion policy and performs gradient ascent using Q-functions to improve the actions.

All methods were trained with $250k$ environment interactions and one network update per environment step. For a fair comparison, all policy/score networks are MLPs with two hidden layers of dimension 256 each, and the learning rate for all networks is 3×10^{-4}. We tune hyperparameters for each baseline and select the one that gives the overall best performance across tasks. We apply the double Q-learning trick, a commonly used technique, where two Q-networks are trained independently and their minimum value is used for policy evaluation to avoid overestimation bias.

5.1 Continuous control tasks

Figure 3: Domains from DeepMind Control Suite considered in our experiments - cheetah, finger, fish, reacher, hopper, quadruped and walker.

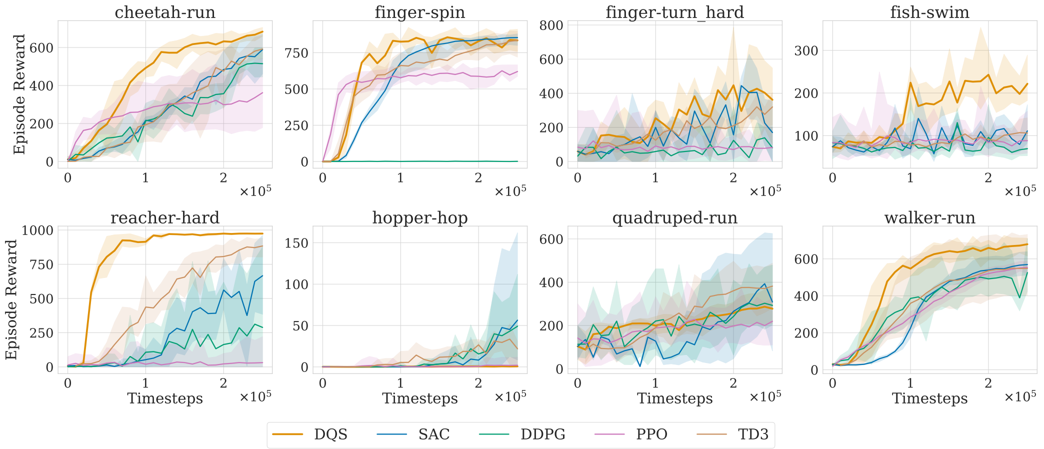

Figure 4: Experimental results for classic RL algorithms on 8 tasks from different domains from the DeepMind Control Suite. Each result is averaged over 100 evaluation episodes across 10 seeds, with the shaded regions showing minimum and maximum values. For PPO, the x-axis represents the number of network updates.

Table 1: Mean episode returns for 100 evaluation episodes for classic RL algorithms, averaged over 10 seeds. The highest mean values in each row are highlighted and values within one standard deviation are underlined.

	Task	DQS (Ours)	SAC	DDPG	PPO	TD3
100k steps	cheetah-run	492.61±30.99	216.88 ±18.51	214.02 ±64.55	273.52 ±101.92	193.63 ±68.43
	finger-spin	826.00 ±6.24	681.72 ±68.96	0.36 ±0.64	577.47 ±25.96	661.70 ±61.30
	finger-turn_hard	253.72 ±104.84	200.02 ±85.60	58.58 ±119.76	91.15 ±120.00	124.59 ±32.54
	fish-swim	224.27 ±38.96	69.31 ±11.95	87.40 ±53.15	83.86 ±29.02	77.88 ±7.69
	hopper-hop	0.33 ±0.35	0.01 ±0.02	1.36 ±2.62	0.50 ±0.99	5.34 ±7.47
	quadruped-run	199.26 ±32.79	127.94 ±32.06	220.66 ±159.40	188.86 ±85.45	191.11 ±21.99
	reacher-hard	914.15 ±18.28	52.76 ±50.88	107.20 ±124.32	19.63 ±46.40	437.10 ±138.98
	walker-run	547.39 ±32.58	226.52 ±37.38	386.05 ±98.73	285.89 ±25.10	359.85 ±158.52
250k steps	cheetah-run	683.64 ±18.51	588.91 ±84.52	514.91 ±61.35	362.27 ±145.64	592.13 ±49.08
	finger-spin	835.00 ±61.36	854.04 ±31.92	1.10 ±1.12	620.32 ±31.55	835.45 ±71.52
	finger-turn_hard	361.46 ±179.32	169.58 ±141.64	80.48 ±80.64	82.34 ±78.69	319.80 ±75.90
	fish-swim	221.67 ±42.59	111.09 ±35.72	69.33 ±21.20	88.13 ±39.20	106.06 ±21.31
	hopper-hop	0.66 ±0.08	56.47 ±61.47	49.10 ±44.49	2.00 ±4.71	21.91 ±12.10
	quadruped-run	277.63 ±66.45	308.61 ±216.94	292.94 ±110.62	219.69 ±86.76	381.84 ±82.94
	reacher-hard	974.05 ±1.64	666.12 ±230.56	288.54 ±350.28	31.21 ±88.40	883.55 ±54.84
	walker-run	679.17 ±38.63	569.28 ±50.41	525.64 ±113.60	553.79 ±26.28	548.09 ±131.90

We evaluate the performance of DQS on several continuous control tasks via the DeepMind Control Suite. We choose eight tasks from different domains to cover tasks of varying complexity and dynamics. These tasks typically involve controlling the torques applied at the joints of robots to reach a specific configuration or location, or for locomotion.

Since we are interested in evaluating the data efficiency of DQS , we limit the number of environment interactions to $250k$ steps. Figure 4 shows the performance of various classic methods on these different tasks. On most tasks, DQS performs on par or outperforms the baseline methods. In particular, on five out of the eight tasks considered (cheetah-run, finger-spin, fish-swim, reacher-hard and walker-run) DQS reaches higher reward much faster than competing methods, demonstrating

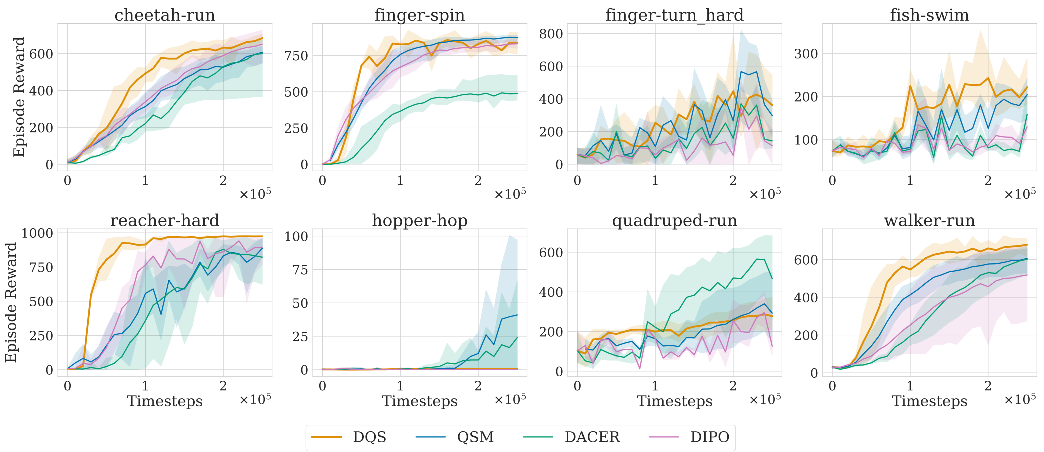

Figure 5: Experimental results for diffusion-based RL algorithms on 8 tasks from different domains from the DeepMind Control Suite. Each result is averaged over 100 evaluation episodes across 10 seeds, with the shaded regions showing minimum and maximum values.

Table 2: Mean episode returns for 100 evaluation episodes for diffusion-based RL methods, averaged over 10 seeds. The highest mean values in each row are highlighted and values within one standard deviation are underlined.

	Task	DQS (Ours)	QSM	DACER	DIPO
100k steps	cheetah-run	492.61 ±30.99	313.34 ±55.38	221.14 ±33.26	338.93 ±37.90
	finger-spin	826.00 ±6.24	757.15 ±59.20	366.06 ±44.68	670.22 ±71.68
	finger-turn_hard	253.72 ±104.84	108.12 ±100.52	36.04 ±37.44	57.84 ±48.80
	fish-swim	224.27 ±38.96	81.73 ±18.83	78.20 ±14.72	72.12 ±3.86
	hopper-hop	0.33 ±0.35	0.04 ±0.01	0.48 ±0.52	0.41 ±0.59
	quadruped-run	199.26 ±32.79	163.60 ±32.42	221.39 ±72.44	164.08 ±5.37
	reacher-hard	914.15 ±18.28	557.12 ±178.20	358.48 ±303.20	766.20 ±53.16
	walker-run	547.39 ±32.58	413.53 ±47.69	195.30 ±43.95	252.88 ±134.23
250k steps	cheetah-run	683.64 ±18.51	607.16 ±36.45	599.64 ±123.50	650.98 ±71.03
	finger-spin	835.00 ±61.36	874.52 ±19.38	486.66 ±67.52	830.36 ±27.16
	finger-turn_hard	361.46 ±179.32	297.66 ±90.00	142.66 ±47.16	113.32 ±85.20
	fish-swim	221.67 ±42.59	204.20 ±38.65	158.82 ±37.19	129.76 ±23.50
	hopper-hop	0.66 ±0.08	40.97 ±38.86	24.01 ±26.32	0.00 ±0.01
	quadruped-run	277.63 ±66.45	293.12 ±141.69	466.77 ±169.98	126.09 ±75.00
	reacher-hard	974.05 ±1.64	887.26 ±54.04	821.88 ±134.16	894.88 ±79.32
	walker-run	679.17 ±38.63	605.98 ±52.12	602.12 ±61.36	518.49 ±162.78

improved exploration. From the numerical results in Table 1, we see that DQS particularly shines in very low environment interaction budgets. When all agents are limited to $100k$ environment steps, DQS is much more performant than other methods. Note that for PPO, the step number represents the number of network updates.

We perform a similar analysis in Figure 5 and Table 2, where we compare the performance of DQS with more recent diffusion-based RL methods. We describe these approaches in more detail in Section 2. We observe a similar trend as the previous set of experiments, where QSM achieves higher returns quicker than other methods on a majority of tasks, which is especially marked when considering performance at $100k$ environment steps. This improved sample efficiency could possibly a result of better ability to handle exploration and exploitation, owing to the use of a Boltzmann pol-

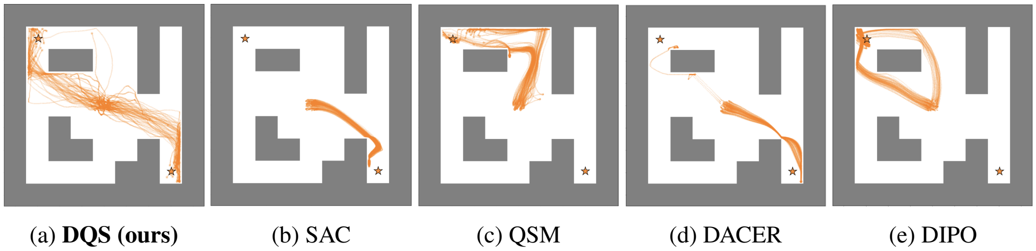
(a) **DQS (ours)** (b) SAC (c) QSM (d) DACER (e) DIPO

Figure 6: Trajectories for 100 evaluation episodes after $250k$ training steps. The starting states are sampled from a Gaussian distribution centered at $(0, 0)$

icy that samples high Q-value actions while maintaining some probability of sampling exploratory actions.

We use a single fixed temperature of $T = 0.05$ across tasks. Note that SAC and DACER use automatic temperature tuning which allows them to influence the policy entropy over the course of training. The performance of DQS may be further improved by fine-tuning the temperature schedule on each individual task.

5.2 Goal reaching maze navigation

We use a custom maze environment to evaluate the ability of our method to reach multiple goals. The agent is tasked with manipulating a ball to reach some unknown goal position in the maze. The state consists of the ball's (x, y) position and the velocity vector. The action is the force vector applied to the ball.

The initial state of the ball is at the center of the maze, with some noise added for variability. We define two potential goal states for the ball - the top left and the bottom right corners respectively. The negative Euclidean distance between the desired goal and the achieved state gives the reward function.

For DQS , we used temperature annealing with an initial temperature of $T = 10$, which is decayed exponentially with the number of training steps to a value of $T = 1$ after $250k$ steps. SAC uses automatic temperature tuning (Haarnoja et al., 2018b), where the entropy co-efficient is automatically tuned using gradient descent to maintain the desired level of entropy.

Figure 6 plots the trajectories of the ball over 100 evaluation episodes after $250k$ training steps. As seen in Figure 6a, DQS learns to reach both goals, owing to the proposed sampling approach which can effectively capture multimodal behavior. Moreover, it discovers both paths to reach the top left goal. In contrast, SAC (Figure 6b), QSM (Figure 6c), and DIPO (Figure 6e) can only reach one of the goals. Since SAC models the policy using a Gaussian, there is little variability between different trajectories. QSM produces slightly more varied behavior, since it uses Langevin sampling to sample actions, but ultimately fails to learn distinct behaviors. DIPO, on the other hand, managers to learn distinct paths to reach the same goal. DACER in Figure 6d discovers the second mode but is heavily skewed towards one mode.

6 Discussion

In this work, we showcase the benefits of using energy-based policies as an expressive class of policies for deep reinforcement learning. Such policies arise in different RL frameworks, but their application has been limited in continuous action spaces owing to the difficulty of sampling in this setting. We alleviate this problem using a diffusion-based sampling algorithm, Diffusion Q-Sampling (DQS), can sample multimodal behaviors and improve sample efficiency, possibly owing to better handling of the exploration-exploitation trade off.

While diffusion methods offer high expressivity, they often come with increased computation. This is particularly true in the online RL setting, where using a diffusion policy means that each environment step requires multiple function evaluations to sample from the diffusion model. There is a growing body of work on efficient SDE samplers (Jolicoeur-Martineau et al., 2021), which aim to reduce the number of function evaluations required to obtain diffusion-based samples while maintaining high accuracy. Incorporating such techniques with Boltzmann policies can greatly reduce the computational cost, especially in high-dimensional state-action spaces.

A crucial aspect of energy-based policies is the temperature parameter, which defines the shape of the sampling distribution. Our method enables annealing of the temperature from some starting value to lower values, as is typically done when applying Boltzmann policies in deep RL. However, this temperature schedule has to be manually tuned. Haarnoja et al. (2018b) proposes an automatic temperature tuning method for SAC, which maintains the temperature so that the entropy of the current policy is close to some target entropy. While such an approach could be applied to DQS in principle, it is computationally expensive to compute the likelihoods of samples under a diffusion model.

Finally, as we argued in the introduction, Boltzmann policies based on their own value function are attractive choices for pre-training of RL agents for later fine-tuning and multi-task settings. We hope to investigate this exciting potential in the future.

References

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B Tenenbaum, Tommi S Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *The Eleventh International Conference on Learning Representations*, 2023.

Tara Akhound-Sadegh, Jarrid Rector-Brooks, Avishek Joey Bose, Sarthak Mittal, Pablo Lemos, Cheng-Hao Liu, Marcin Sendera, Siamak Ravanbakhsh, Gauthier Gidel, Yoshua Bengio, Nikolay Malkin, and Alexander Tong. Iterated denoising energy matching for sampling from boltzmann densities. *arXiv*, 2024.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *The Journal of Machine Learning Research*, 24(1):10006–10060, 2023.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

Tristan Deleu, Padideh Nouri, Nikolay Malkin, Doina Precup, and Yoshua Bengio. Discrete probabilistic inference as control in multi-path environments. In *The 40th Conference on Uncertainty in Artificial Intelligence*, 2024.

Zihan Ding and Chi Jin. Consistency models as a rich and efficient policy class for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2023.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pp. 1352–1361. PMLR, 2017.

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

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. Idql: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Haque Ishfaq, Guangyuan Wang, Sami Nur Islam, and Doina Precup. Langevin soft actor-critic: Efficient exploration through uncertainty-driven critic learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Vineet Jain and Siamak Ravanbakhsh. Learning to reach goals via diffusion. In *Forty-first International Conference on Machine Learning*, 2024.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, pp. 9902–9915. PMLR, 2022.

Alexia Jolicoeur-Martineau, Ke Li, Rémi Piché-Taillefer, Tal Kachman, and Ioannis Mitliagkas. Gotta go fast when generating data with score-based models. *arXiv preprint arXiv:2105.14080*, 2021.

Bingyi Kang, Xiao Ma, Chao Du, Tianyu Pang, and Shuicheng Yan. Efficient diffusion policies for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

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

Gene Li, Junbo Li, Anmol Kabra, Nati Srebro, Zhaoran Wang, and Zhuoran Yang. Exponential family model-based reinforcement learning via score matching. *Advances in Neural Information Processing Systems*, 35:28474–28487, 2022.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015a.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015b.

Cheng Lu, Huayu Chen, Jianfei Chen, Hang Su, Chongxuan Li, and Jun Zhu. Contrastive energy prediction for exact energy-guided diffusion sampling in offline reinforcement learning. In *International Conference on Machine Learning*, pp. 22825–22855. PMLR, 2023.

Kanika Madan, Jarrid Rector-Brooks, Maksym Korablyov, Emmanuel Bengio, Moksh Jain, Andrei Cristian Nica, Tom Bosc, Yoshua Bengio, and Nikolay Malkin. Learning gflownets from partial episodes for improved convergence and stability. In *International Conference on Machine Learning*, pp. 23467–23483. PMLR, 2023.

Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. Trajectory balance: Improved credit assignment in gflownets. *Advances in Neural Information Processing Systems*, 35:5955–5967, 2022.

Michael Psenka, Alejandro Escontrela, Pieter Abbeel, and Yi Ma. Learning a diffusion model policy from rewards via q-score matching. *arXiv preprint arXiv:2312.11752*, 2023.

Jarrid Rector-Brooks, Kanika Madan, Moksh Jain, Maksym Korablyov, Cheng-Hao Liu, Sarath Chandar, Nikolay Malkin, and Yoshua Bengio. Thompson sampling for improved exploration in gflownets. *arXiv preprint arXiv:2306.17693*, 2023.

Moritz Reuss, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. Goal-conditioned imitation learning using score-based diffusion policies. *arXiv preprint arXiv:2304.02532*, 2023.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

John Schulman, Sergey Levine, P. Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. *ArXiv*, abs/1502.05477, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Max W Shen, Emmanuel Bengio, Ehsan Hajiramezanali, Andreas Loukas, Kyunghyun Cho, and Tommaso Biancalani. Towards understanding and improving gflownet training. In *International Conference on Machine Learning*, pp. 30956–30975. PMLR, 2023.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

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

Niko Sünderhauf, Oliver Brock, Walter Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, et al. The limits and potentials of deep learning for robotics. *The International journal of robotics research*, 37(4-5):405–420, 2018.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Daniil Tiapkin, Nikita Morozov, Alexey Naumov, and Dmitry P Vetrov. Generative flow networks as entropy-regularized rl. In *International Conference on Artificial Intelligence and Statistics*, pp. 4213–4221. PMLR, 2024.

Yinuo Wang, Likun Wang, Yuxuan Jiang, Wenjun Zou, Tong Liu, Xujie Song, Wenxuan Wang, Liming Xiao, Jiang Wu, Jingliang Duan, et al. Diffusion actor-critic with entropy regulator. *arXiv preprint arXiv:2405.15177*, 2024.

Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023.

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In *Conference on robot learning*, pp. 2226–2240. PMLR, 2023.

Long Yang, Zhixiong Huang, Fenghao Lei, Yucun Zhong, Yiming Yang, Cong Fang, Shiting Wen, Binbin Zhou, and Zhouchen Lin. Policy representation via diffusion probability model for reinforcement learning. *arXiv preprint arXiv:2305.13122*, 2023.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pp. 1433–1438. Chicago, IL, USA, 2008.

Supplementary Materials

The following content was not necessarily subject to peer review.

Implementation details

The score function is parameterized as an MLP with two hidden layers of 256 units each with the ReLU activation function, except for the final layer. The MLP has skip connections as is typical for denoising score functions. The input to the policy comprises the state, noised action, the diffusion time step, and the temperature. The diffusion time step and the temperature are encoded using sinusoidal positional embeddings of 256 dimensions. The action is sampled following Equation (3) and the $\tanh(\cdot)$ function is applied to the sampled action followed by multiplication with the maximum value of the action space to ensure the value is within the correct range. The Q-network is also an MLP with two hidden layers of 256 units each with the ReLU activation function, except for the final layer. We use two Q-networks for the double Q-learning technique, and take the minimum of the two values.

The score function and the Q-network are trained for $250k$ environment steps with one mini-batch update per environment step. Optimization is performed using the Adam optimizer (Kingma, 2014) with a learning rate of 3×10^{-4} and a batch size of 256.

Table 3: Hyperparameters.

Parameter	**Value**
Number of hidden layers	2
Number of hidden units per layer	256
Sinusoidal embedding dimension	256
Activation function	ReLU
Optimizer	Adam
Learning rate	$3 \cdot 10^{-4}$
Batch size	256
Replay buffer size	250000
Discount factor	0.99
Gradient updates per step	1
Target smoothing co-efficient	0.005
Target update period	1
Seed training steps	10^4
σ_{min}	0.00001
σ_{max}	1
Number of Monte Carlo samples	1000
Number of integration steps	1000