GUIDING THE TRAINING OF DIFFUSION MODELS: A REPLICATION OF DDPO

Gao, Qiaosong
New York University, Shanghai
qg2089@nyu.edu

Shi, Xingyu
New York University, Shanghai
xs2337@nyu.edu

Zhang, Haoqi
New York University, Shanghai
hz3223@nyu.edu

ABSTRACT

Diffusion models have emerged as a class of effective generative models, trained with an approximation to the log-likelihood objective. However, this objective sometimes doesn't best reflect the objective of empirical use cases of diffusion models such as human-perceived image quality. The paper we want to replicate: *Training Diffusion Models with Reinforcement Learning* [1], aim to improve the capabilities of diffusion models for such empirical objectives through reinforcement learning methods. They describe denoising as a multi-step decisionmaking problem, which enables a class of policy gradient algorithms, which they refer to as denoising diffusion policy optimization (DDPO). We replicate DDPO, and based on this, propose our own insights. The code for our replication and implementation can be found at Fall-2025-Reinforcement-Learning-Final-Project.

1 Introduction

Diffusion models are widely used to generate high-quality images and videos. A key challenge in training empirically effective diffusion models is that the evaluation metrics for generated samples (such as perceptual image quality) are not directly defined in terms of the likelihood-based objectives that diffusion models optimize during training.

To better train diffusion models that align with the objectives of empirical tasks, Black et al. [1] frames denoising as a multi-step decision-making task, introducing reinforcement learning methods. Specifically, they present a policy gradient algorithm, or Denoising Diffusion policy Optimization (DDPO), which is able to optimize a diffusion model for downstream tasks using different reward functions. Various reward functions are also designed for different downstream tasks. The downstream tasks they aim to optimize are: Compressibility, Aesthetic Quality, and Prompt-Image Alignment. DDPO and these reward functions are used for training on different downstream objectives and demonstrate their effectiveness compared to alternative reward-weighted likelihood methods.

We've successfully replicated the DDPO algorithm, and achieved similar results on similar downstream objectives. We've also proposed our improvement based on one class of the DDPO algorithm and achieved faster convergence and avoided overoptimization in some tasks.

2 Environment

The environment of the DDPO reinforcement learning method is text-to-image diffusion. Text-to-image diffusion "serves as a valuable test environment for reinforcement learning due to the availability of large pretrained models and the versatility of using diverse and visually interesting reward functions [1]. "

In this section, we describe the environment of this reinforcement learning method, including:

1. The Markov Decision Proces (MDP) formulation
2. Description of a selection of downstream tasks and corresponding reward functions

2.1 Markov Decision Process Formulation

A Markov decision process (MDP) is a formalization of sequential decision-making problems. An MDP is defined by a tuple (S, A, ρ_0, P, R), in which S is the state space, A is the action space, ρ_0 is the distribution of initial states, P is the transition kernel, and R is the reward function. At each timestep t, the agent observes a state $s_t \in S$, takes an action $a_t \in A$, receives a reward $R(s_t, a_t)$, and transitions to a new state $s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$. An agent acts according to a policy $\pi(a_t \mid s_t)$.

As the agent acts in the MDP, it produces trajectories, which are sequences of states, actions $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$. The reinforcement learning (RL) objective is for the agent to maximize $J_{\text{RL}}(\pi)$, the expected cumulative reward over trajectories sampled from its policy:

$$J_{\text{RL}}(\pi) = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[\sum_{t=0}^{T} R(s_t, a_t) \right].$$

2.2 MDP formulation of the denoising process of diffusion models

Now, we consider the **MDP formulation of the denoising process of diffusion models**. We cast conditional diffusion sampling as an episodic finite–horizon Markov decision process (MDP). Each episode corresponds to one reverse diffusion trajectory that starts from pure noise and ends at a synthesized image.

State space. Let $c \in \mathcal{C}$ denote a text prompt and $x_t \in \mathbb{R}^d$ the latent (noisy image) at diffusion step $t \in \{0, \ldots, T\}$. We define the state at time t as

$$s_t \triangleq (c, t, x_t) \in \mathcal{S} = \mathcal{C} \times \{0, \ldots, T\} \times \mathbb{R}^d.$$

The agent therefore observes the current prompt, the current diffusion time index, and the current noisy latent.

Action space. At each step $t > 0$ the agent chooses the next latent x_{t-1}. The action space is the same Euclidean space as the latent:

$$a_t \triangleq x_{t-1} \in \mathcal{A} = \mathbb{R}^d.$$

Policy. The policy is given by the reverse diffusion model p_θ:

$$\pi_\theta(a_t \mid s_t) = \pi_\theta(x_{t-1} \mid c, t, x_t) \triangleq p_\theta(x_{t-1} \mid x_t, c, t),$$

which is typically a Gaussian with mean $\mu_\theta(x_t, c, t)$ and fixed variance $\sigma_t^2 I$ (possibly after classifier–free guidance).

Transition dynamics. The environment transition simply decreases the time index and copies the chosen latent:

$$P(s_{t-1} \mid s_t, a_t) = P\big((c', t', x'_{t-1}) \mid (c, t, x_t), x_{t-1}\big) = \delta_c(c')\, \delta_{t-1}(t')\, \delta_{x_{t-1}}(x'_{t-1}),$$

i.e., the prompt c is fixed throughout the episode, the time index counts down from T to 0, and the next latent is deterministically exactly the chosen action.

Initial state distribution. At the beginning of each episode we sample a prompt and an initial noise:

$$c \sim p(c), \qquad x_T \sim \mathcal{N}(0, I),$$

and set the initial state to

$$s_T = (c, T, x_T), \qquad \rho_0(s_T) = p(c)\, \mathcal{N}(x_T; 0, I).$$

Reward. The episode terminates after producing x_0 at $t = 0$. We only assign a terminal reward based on the final image and prompt:

$$R(s_t, a_t) = \begin{cases} r(x_0, c), & t = 0, \\ 0, & t > 0, \end{cases}$$

where $r(x_0, c)$ is given by an external reward model (e.g., aesthetic score, compression ratio, or a VLM).

Objective. Let $\tau = (s_T, a_T, \ldots, s_0, a_0)$ denote a full trajectory induced by π_θ. The DDRL objective is to maximize the expected terminal reward

$$J_{\text{DDRL}}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \big[\, r(x_0, c) \,\big].$$

One-step RWR view. For reward-weighted regression (RWR), this MDP can be collapsed to a one-step formulation:

$$s \triangleq c, \quad a \triangleq x_0, \quad \pi_\theta(a \mid s) \triangleq p_\theta(x_0 \mid c), \quad R(s,a) \triangleq r(x_0, c),$$

which leads to a weighted minimization of the diffusion MSE loss objective for $p_\theta(x_0 \mid c)$.

2.3 A Selection of Tasks and Corresponding Reward Functions

This paper mainly evaluate the methods on the following selection of downstream of tasks and corresponding reward functions:

1. Compressibility and Incompressibility
2. Aesthetic Quality

2.3.1 Compressibility and Incompressibility

Images are rarely captioned with their file size, making it impossible to specify a desired file size via prompting. This limitation makes reward functions based on file size a convenient case study: they are simple to compute, but not controllable through the conventional methods of likelihood maximization and prompt engineering. We fix the resolution of diffusion model samples at 512x512, such that the file size is determined solely by the compressibility of the image. We define two tasks based on file size: compressibility, in which the file size of the image after JPEG compression is minimized, and incompressibility, in which the same measure is maximized.

We adopt this task setting and reward function (image file size) in our replication.

2.3.2 Aesthetic Quality

To capture a reward function that would be useful to a human user, we define a task based on perceived aesthetic quality. We use the LAION aesthetics predictor [2], which is trained on 176,000 human image ratings. Annotations range between 1 and 10, with the highest-rated images mostly containing artwork.

We adopt this task setting and reward function (LAION aesthetics predictor rating) in our replication.

(a) Aesthetics

(b) Compressibility

(c) Incompressibility

Figure 1: Training progression for different tasks

(a) DDPO IS

(b) DDPO SF

Figure 2: Training progression for compressibility with DDPO IS and SF

3 Algorithm and Proposed Improvement

This section includes the algorithms we choose to replicate and run.In Black et al.'s paper, they proposed the following four algorithms[1]:

1. Reward-Weighted Regression (RWR) (See section 3.1)
2. Sparse Reward-Weighted Regression (See section 3.2)
3. DDPO_{SF} (See section 3.3)
4. DDPO_{IS} (See section 3.4)

We also propose our improvement based on DDPO_{IS}, introducing KL-Divergence regularization between Model in training p_θ and a pretrained stable diffusion model p_{ref}. Through this improvement, we achieved faster convergence and avoided overoptimization in some tasks. Details and peusdo-code can be found in

5. $\text{DDPO}_{\text{ISKL}}$ (See section 3.5)

3.1 Reward-Weighted Regression (RWR)

We have The DDRL objective:

$$J_{\text{DDRL}}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\, r(x_0, c)\,\right].$$

Now, with training data $x_0 \sim p_0(x_0|c)$ and an added weighting that depends on the reward $r(x_0, c)$. This approach can be performed for multiple rounds of alternating sampling and training, leading to an online RL method. We refer to this general class of algorithms as reward-weighted regression (RWR).

We have the following standard weighting scheme uses exponentiated rewards to ensure non-negativity,

$$w_{\text{RWR}}(x_0, c) = \frac{1}{Z}\exp\left(\beta\, r(x_0, c)\right),$$

where β is an inverse temperature and Z is a normalization constant.

We have the following pseudo-code for RWR (a policy gradient method):

Algorithm 1 Denoising Diffusion with Reward-Weighted Regression (RWR)

Input: initial diffusion parameters θ; reward function $r(x, c)$; prompt distribution $p(c)$; timestep distribution $p(t)$.
Hyperparameters: reward temperature β; number of gradient updates per iteration N_{updates}; number of diffusion steps T; CFG weight W_c; number of samples per iteration N_{samples}; number of inference steps N_{inf}; batch size B; noise schedule endpoints $\beta_{\text{start}}, \beta_{\text{end}}$; learning rate η; Adam parameters (β_1, β_2); Adam numerical constant ε; weight decay λ; gradient clipping norm C_{clip}.
repeat
  Initialize empty dataset $\mathcal{D}$
  Use a diffusion scheduler with $(T, \beta_{\text{start}}, \beta_{\text{end}})$ to generate the cumulative factors $\{\bar{\alpha}_t\}_{t=0}^{T-1}$
  for $i = 1$ to N_{samples} **do**
    Sample prompt $c_i \sim p(c)$
    Run policy $\pi_\theta(\cdot \mid c_i, w)$ to get a trajectory with N_{inf} time steps, the reward in each step is $r_{\text{raw}}^{(i)} \leftarrow r(x_0^{(i)}, c_i)$
    Collect the terminal $(c_i, x_0^{(i)}, r_{\text{raw}}^{(i)})$ to $\mathcal{D}$
  end for
  for all prompts c in $\mathcal{D}$ **do**
    Let $\mathcal{I}_c$ be indices with prompt c
    $Z_c \leftarrow \sum_{j \in \mathcal{I}_c} \exp(\beta r_{\text{raw}}^{(j)})$
    for all $i \in \mathcal{I}_c$ **do**
      $w_i \leftarrow \exp(\beta r_{\text{raw}}^{(i)})/Z_c$
    end for
  end for
  for $u = 1$ to N_{updates} **do**
    Sample a minibatch $\mathcal{B} \subset \mathcal{D}$ of size B
    $L \leftarrow 0$
    for all $i \in \mathcal{B}$ **do**
      Sample timestep $t_i \sim p(t)$ over $\{0, \ldots, T-1\}$
      Sample noise $\epsilon_i \sim \mathcal{N}(0, I)$
      Construct noised latent $x_{t_i}^{(i)} \leftarrow \sqrt{\bar{\alpha}_{t_i}}\, x_0^{(i)} + \sqrt{1 - \bar{\alpha}_{t_i}}\, \epsilon_i$
      Compute conditional prediction $\hat{\epsilon}_\theta^{\text{cond}} \leftarrow \epsilon_\theta(x_{t_i}^{(i)}, c_i, t_i)$
      Compute unconditional prediction $\hat{\epsilon}_\theta^{\text{uncond}} \leftarrow \epsilon_\theta(x_{t_i}^{(i)}, \varnothing, t_i)$
      Apply classifier-free guidance:

$$\hat{\epsilon}_\theta \leftarrow \hat{\epsilon}_\theta^{\text{uncond}} + W_c\big(\hat{\epsilon}_\theta^{\text{cond}} - \hat{\epsilon}_\theta^{\text{uncond}}\big)$$

      Compute per-sample loss $\ell_i \leftarrow \|\epsilon_i - \hat{\epsilon}_\theta\|^2$
      $L \leftarrow L + w_i \cdot \ell_i$
    end for
    Update θ by one AdamW step with η and (β_1, β_2) using $\nabla_\theta L$ clipped by GlobalNorm with C_{clip}
  end for
until convergence

3.2 Sparse Reward-Weighted Regression

Sparse Reward-Weighted Regression ($\text{RWR}_{\text{sparse}}$) is similar to RWR. We consider a simplified weighting scheme that uses binary weights,

$$w_{\text{sparse}}(x_0, c) = 1_{\{r(x_0, c) \geq C\}},$$

where C is a reward threshold determining which samples are used for training.

We have the following pseudo-code for $\text{RWR}_{\text{sparse}}$ (a policy gradient method):

Algorithm 2 Denoising Diffusion with Sparse Reward-Weighted Regression ($\text{RWR}_{\text{sparse}}$)

Input: initial diffusion parameters θ; reward function $r(x, c)$; prompt distribution $p(c)$; timestep distribution $p(t)$.
Hyperparameters: reward temperature β; Sparse Cap C; number of gradient updates per iteration N_{updates}; number of diffusion steps T; CFG weight W_c; number of samples per iteration N_{samples}; number of inference steps N_{inf}; batch size B; noise schedule endpoints $\beta_{\text{start}}, \beta_{\text{end}}$; learning rate η; Adam parameters (β_1, β_2); Adam numerical constant ε; weight decay λ; gradient clipping norm C_{clip}.
repeat
  Initialize empty dataset $\mathcal{D}$
  Use a diffusion scheduler with $(T, \beta_{\text{start}}, \beta_{\text{end}})$ to generate the cumulative factors $\{\bar{\alpha}_t\}_{t=0}^{T-1}$
  for $i = 1$ to N_{samples} **do**
    Sample prompt $c_i \sim p(c)$
    Run policy $\pi_\theta(\cdot \mid c_i, w)$ to get a trajectory with N_{inf} time steps, the reward in each step is $r_{\text{raw}}^{(i)} \leftarrow r(x_0^{(i)}, c_i)$
    Collect the terminal $(c_i, x_0^{(i)}, r_{\text{raw}}^{(i)})$ to $\mathcal{D}$
  end for
  for all $(c_i, x_0^{(i)}, r_{\text{raw}}^{(i)}) \in \mathcal{D}$ **do**
    $w_i \leftarrow \mathbf{1}\{r_{\text{raw}}^{(i)} \geq C\}$
  end for
  for $u = 1$ to N_{updates} **do**
    Sample a minibatch $\mathcal{B} \subset \mathcal{D}$ of size B
    $L \leftarrow 0$
    for all $i \in \mathcal{B}$ **do**
      Sample timestep $t_i \sim p(t)$ over $\{0, \ldots, T-1\}$
      Sample noise $\epsilon_i \sim \mathcal{N}(0, I)$
      Construct noised latent $x_{t_i}^{(i)} \leftarrow \sqrt{\bar{\alpha}_{t_i}}\, x_0^{(i)} + \sqrt{1 - \bar{\alpha}_{t_i}}\, \epsilon_i$
      Compute conditional prediction $\hat{\epsilon}_\theta^{\text{cond}} \leftarrow \epsilon_\theta(x_{t_i}^{(i)}, c_i, t_i)$
      Compute unconditional prediction $\hat{\epsilon}_\theta^{\text{uncond}} \leftarrow \epsilon_\theta(x_{t_i}^{(i)}, \varnothing, t_i)$
      Apply classifier-free guidance:

$$\hat{\epsilon}_\theta \leftarrow \hat{\epsilon}_\theta^{\text{uncond}} + W_c\big(\hat{\epsilon}_\theta^{\text{cond}} - \hat{\epsilon}_\theta^{\text{uncond}}\big)$$

      Compute per-sample loss $\ell_i \leftarrow \|\epsilon_i - \hat{\epsilon}_\theta\|^2$
      $L \leftarrow L + w_i \cdot \ell_i$
    end for
    Update θ by one AdamW step with η and (β_1, β_2) using $\nabla_\theta L$ clipped by GlobalNorm with C_{clip}
  end for
until convergence

3.3 DDPO_{SF}

DDPO is a form of Policy gradient estimation. With access to likelihoods and likelihood gradients, we can make direct Monte Carlo estimates of $\nabla_\theta J_{\text{DDRL}}$. DDPO alternates collecting denoising trajectories $\{x_T, x_{T-1}, \ldots, x_0\}$ via sampling and updating parameters via gradient descent.

The first variant of DDPO, which we call DDPO_{SF}, uses the score function policy gradient estimator, also known as the likelihood ratio method or REINFORCE [3]:

$$\nabla_\theta J_{\text{DDRL}} = \mathbb{E}\left[\sum_{t=0}^{T} \nabla_\theta \log p_\theta(x_{t-1} \mid x_t, c)\, r(x_0, c)\right] \quad (\text{DDPO}_{\text{SF}}) \tag{1}$$

where the expectation is taken over denoising trajectories generated by the current parameters θ.

We have the following pesudo-code for DDPO_{SF}:

Algorithm 3 Denoising Diffusion Policy Optimization with Score-Function Estimator (DDPO_{SF})

Input: initial diffusion parameters θ; reward function $r(x_0, c)$; prompt distribution $p(c)$.
Hyperparameters: number of denoising steps T; number of trajectories per iteration N_{samples}; number of gradient updates per iteration N_{updates}; classifier-free guidance weight W_c; batch size B; noise schedule endpoints $\beta_{\text{start}}, \beta_{\text{end}}$; learning rate η; AdamW parameters $(\beta_1, \beta_2, \varepsilon, \lambda)$; gradient clipping norm C_{clip}.
repeat
  Initialize empty dataset $\mathcal{D} \leftarrow \varnothing$
  Use a diffusion scheduler with $(T, \beta_{\text{start}}, \beta_{\text{end}})$ to generate variances $\{\sigma_t^2\}_{t=1}^T$ and $\{\beta_t\}_{t=1}^T$
  for $i = 1$ to N_{samples} **do**
    Sample prompt $c_i \sim p(c)$
    Sample initial latent $x_T^{(i)} \sim \mathcal{N}(0, I)$
    Initialize log-likelihood accumulator $\ell^{(i)} \leftarrow 0$
    for $t = T, T-1, \ldots, 1$ **do**
      Compute conditional prediction $\hat{\epsilon}_\theta^{\text{cond}} \leftarrow \epsilon_\theta(x_t^{(i)}, c_i, t)$
      Compute unconditional prediction $\hat{\epsilon}_\theta^{\text{uncond}} \leftarrow \epsilon_\theta(x_t^{(i)}, \varnothing, t)$
      Apply classifier-free guidance:

$$\hat{\epsilon}_\theta \leftarrow \hat{\epsilon}_\theta^{\text{uncond}} + W_c\big(\hat{\epsilon}_\theta^{\text{cond}} - \hat{\epsilon}_\theta^{\text{uncond}}\big)$$

      Compute DDPM mean using the scheduler:

$$\mu_t^{(i)} \leftarrow \frac{1}{\sqrt{\alpha_t}}\left(x_t^{(i)} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\hat{\epsilon}_\theta\right)$$

      Sample next latent $x_{t-1}^{(i)} \sim \mathcal{N}(\mu_t^{(i)}, \sigma_t^2 I)$
      Accumulate log-likelihood term

$$\ell^{(i)} \leftarrow \ell^{(i)} + \log \mathcal{N}\big(x_{t-1}^{(i)}; \mu_t^{(i)}, \sigma_t^2 I\big)$$

    end for
    Let $x_0^{(i)}$ be the final latent of the trajectory
    Compute terminal reward $r_{\text{raw}}^{(i)} \leftarrow r(x_0^{(i)}, c_i)$
    Store $(\ell^{(i)}, r_{\text{raw}}^{(i)}, c_i)$ in $\mathcal{D}$
  end for
  for $u = 1$ to N_{updates} **do**
    Sample a minibatch $\mathcal{B} \subset \mathcal{D}$ of size B
    $L \leftarrow 0$
    for all $i \in \mathcal{B}$ **do**
      $L \leftarrow L - \tilde{r}_i \cdot \ell^{(i)}$
    end for
    Update θ by one AdamW step with parameters $(\eta, \beta_1, \beta_2, \varepsilon, \lambda)$, using $\nabla_\theta L$ clipped by global norm C_{clip}
  end for
until convergence

3.4 DDPO_{IS}

To perform multiple steps of optimization, we may use an importance sampling estimator:

$$\nabla_\theta J_{\text{DDRL}} = \mathbb{E}\left[\sum_{t=0}^{T} \frac{p_\theta(x_{t-1} \mid x_t, c)}{p_{\theta_{\text{old}}}(x_{t-1} \mid x_t, c)} \nabla_\theta \log p_\theta(x_{t-1} \mid x_t, c)\, r(x_0, c)\right] \quad (\text{DDPO}_{\text{IS}}) \tag{2}$$

where the expectation is taken over denoising trajectories generated by the parameters θ_{old}. This estimator becomes inaccurate if p_θ deviates too far from $p_{\theta_{\text{old}}}$, which can be addressed using trust regions to constrain the size of the update. In practice, we implement the trust region via clipping.

Algorithm 4 Denoising Diffusion Policy Optimization with Importance Sampling (DDPO_{IS})

Input: initial diffusion parameters θ; reward function $r(x_0, c)$; prompt distribution $p(c)$.
State: behaviour parameters $\theta_{\text{beh}} \leftarrow \theta$.
Hyperparameters: number of denoising steps T; number of trajectories per iteration N_{samples}; number of gradient updates per iteration N_{updates}; classifier-free guidance weight W_c; batch size B; noise schedule endpoints $\beta_{\text{start}}, \beta_{\text{end}}$; learning rate η; AdamW parameters $(\beta_1, \beta_2, \varepsilon, \lambda)$; gradient clipping norm C_{clip}.
repeat
  Initialize empty dataset $\mathcal{D} \leftarrow \varnothing$
  Set behaviour parameters to the current policy snapshot: $\theta_{\text{beh}} \leftarrow \theta$
  Use a diffusion scheduler with $(T, \beta_{\text{start}}, \beta_{\text{end}})$ to generate variances $\{\sigma_t^2\}_{t=1}^T$ and $\{\beta_t\}_{t=1}^T$
  for $i = 1$ to N_{samples} **do**
    Sample prompt $c_i \sim p(c)$
    Sample initial latent $x_T^{(i)} \sim \mathcal{N}(0, I)$
    Initialize behaviour log-likelihood accumulator $\ell_{\text{beh}}^{(i)} \leftarrow 0$
    for $t = T, T-1, \ldots, 1$ **do**
      Compute conditional prediction $\hat{\epsilon}_{\theta_{\text{beh}}}^{\text{cond}} \leftarrow \epsilon_{\theta_{\text{beh}}}(x_t^{(i)}, c_i, t)$
      Compute unconditional prediction $\hat{\epsilon}_{\theta_{\text{beh}}}^{\text{uncond}} \leftarrow \epsilon_{\theta_{\text{beh}}}(x_t^{(i)}, \varnothing, t)$
      Apply classifier-free guidance:

$$\hat{\epsilon}_{\theta_{\text{beh}}} \leftarrow \hat{\epsilon}_{\theta_{\text{beh}}}^{\text{uncond}} + W_c\big(\hat{\epsilon}_{\theta_{\text{beh}}}^{\text{cond}} - \hat{\epsilon}_{\theta_{\text{beh}}}^{\text{uncond}}\big)$$

      Compute DDPM mean using the scheduler:

$$\mu_t^{(i)} \leftarrow \frac{1}{\sqrt{\alpha_t}}\left(x_t^{(i)} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\hat{\epsilon}_{\theta_{\text{beh}}}\right)$$

      Sample next latent $x_{t-1}^{(i)} \sim \mathcal{N}(\mu_t^{(i)}, \sigma_t^2 I)$
      Accumulate behaviour log-likelihood term

$$\ell_{\text{beh}}^{(i)} \leftarrow \ell_{\text{beh}}^{(i)} + \log \mathcal{N}\big(x_{t-1}^{(i)}; \mu_t^{(i)}, \sigma_t^2 I\big)$$

    end for
    Let $x_0^{(i)}$ be the final latent of the trajectory
    Compute terminal reward $r_{\text{raw}}^{(i)} \leftarrow r(x_0^{(i)}, c_i)$
    Store trajectory $\tau_i = \{x_t^{(i)}\}_{t=0}^T$ and stats $\big(\tau_i, c_i, \ell_{\text{beh}}^{(i)}, r_{\text{raw}}^{(i)}\big)$ in $\mathcal{D}$
  end for
  for $u = 1$ to N_{updates} **do**
    Sample a minibatch $\mathcal{B} \subset \mathcal{D}$ of size B
    $L \leftarrow 0$
    Compute any normalized rewards $\{\tilde{r}_i\}_{i \in \mathcal{B}}$ from $\{r_{\text{raw}}^{(i)}\}$
    for all $i \in \mathcal{B}$ **do**
      Recompute trajectory log-likelihood under current parameters: $\ell_\theta^{(i)} \leftarrow \log p_\theta(\tau_i)$
      Compute importance weight

$$\tilde{w}_i \leftarrow \exp\big(\ell_\theta^{(i)} - \ell_{\text{beh}}^{(i)}\big)$$

      Accumulate loss with IS-corrected score-function term:

$$L \leftarrow L - (\tilde{w}_i \tilde{r}_i) \cdot \ell_\theta^{(i)}$$

    end for
    Update θ by one AdamW step with parameters $(\eta, \beta_1, \beta_2, \varepsilon, \lambda)$, using $\nabla_\theta L$ clipped by global norm C_{clip}
  end for
until convergence

3.5 $\text{DDPO}_{\text{ISKL}}$

Here, we propose our own improved algorithm: $\text{DDPO}_{\text{ISKL}}$. Our $\text{DDPO}_{\text{ISKL}}$ is based on DDPO_{IS}, but we add a KL-Divergence regularization term between a pretrained diffusion model θ_{ref} as reference, and our model in training θ.

Algorithm 5 Denoising Diffusion Policy Optimization with Importance Sampling and KL ($\text{DDPO}_{\text{ISKL}}$)

Input: initial diffusion parameters θ; fixed pretrained reference parameters θ_{ref}; reward function $r(x_0, c)$; prompt distribution $p(c)$.
Hyperparameters: number of denoising steps T; number of trajectories per iteration N_{samples}; number of gradient updates per iteration N_{updates}; classifier-free guidance weight W_c; batch size B; noise schedule endpoints $\beta_{\text{start}}, \beta_{\text{end}}$; learning rate η; AdamW parameters $(\beta_1, \beta_2, \varepsilon, \lambda)$; gradient clipping norm C_{clip}; KL coefficient λ_{KL}.
repeat
  Set behaviour parameters to the current policy snapshot: $\theta_{\text{beh}} \leftarrow \theta$
  Initialize empty dataset $\mathcal{D} \leftarrow \varnothing$
  Use a diffusion scheduler with $(T, \beta_{\text{start}}, \beta_{\text{end}})$ to generate variances $\{\sigma_t^2\}_{t=1}^T$ and $\{\beta_t\}_{t=1}^T$
  for $i = 1$ to N_{samples} **do**
    Sample prompt $c_i \sim p(c)$
    Sample initial latent $x_T^{(i)} \sim \mathcal{N}(0, I)$
    Initialize behaviour log-likelihood accumulator $\ell_{\text{beh}}^{(i)} \leftarrow 0$
    for $t = T, T-1, \ldots, 1$ **do**
      Compute conditional prediction $\hat{\epsilon}_{\theta_{\text{beh}}}^{\text{cond}} \leftarrow \epsilon_{\theta_{\text{beh}}}(x_t^{(i)}, c_i, t)$
      Compute unconditional prediction $\hat{\epsilon}_{\theta_{\text{beh}}}^{\text{uncond}} \leftarrow \epsilon_{\theta_{\text{beh}}}(x_t^{(i)}, \varnothing, t)$
      Apply classifier-free guidance:

$$\hat{\epsilon}_{\theta_{\text{beh}}} \leftarrow \hat{\epsilon}_{\theta_{\text{beh}}}^{\text{uncond}} + W_c\big(\hat{\epsilon}_{\theta_{\text{beh}}}^{\text{cond}} - \hat{\epsilon}_{\theta_{\text{beh}}}^{\text{uncond}}\big)$$

      Compute DDPM mean using the scheduler:

$$\mu_t^{(i)} \leftarrow \frac{1}{\sqrt{\alpha_t}}\left(x_t^{(i)} - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\hat{\epsilon}_{\theta_{\text{beh}}}\right)$$

      Sample next latent $x_{t-1}^{(i)} \sim \mathcal{N}(\mu_t^{(i)}, \sigma_t^2 I)$
      Accumulate behaviour log-likelihood term

$$\ell_{\text{beh}}^{(i)} \leftarrow \ell_{\text{beh}}^{(i)} + \log \mathcal{N}\big(x_{t-1}^{(i)}; \mu_t^{(i)}, \sigma_t^2 I\big)$$

    end for
    Let $x_0^{(i)}$ be the final latent of the trajectory
    Compute terminal reward $r_{\text{raw}}^{(i)} \leftarrow r(x_0^{(i)}, c_i)$
    Store trajectory $\tau_i = \{x_t^{(i)}\}_{t=0}^T$ and stats $\big(\tau_i, c_i, \ell_{\text{beh}}^{(i)}, r_{\text{raw}}^{(i)}\big)$ in $\mathcal{D}$
  end for
  for $u = 1$ to N_{updates} **do**
    Sample a minibatch $\mathcal{B} \subset \mathcal{D}$ of size B
    $L \leftarrow 0$
    Compute any normalized rewards $\{\tilde{r}_i\}_{i \in \mathcal{B}}$ from $\{r_{\text{raw}}^{(i)}\}$
    for all $i \in \mathcal{B}$ **do**
      Recompute trajectory log-likelihood under current parameters: $\ell_\theta^{(i)} \leftarrow \log p_\theta(\tau_i)$
      Compute reference log-likelihood under fixed pretrained parameters: $\ell_{\text{ref}}^{(i)} \leftarrow \log p_{\theta_{\text{ref}}}(\tau_i)$
      Compute importance weight

$$\tilde{w}_i \leftarrow \exp\big(\ell_\theta^{(i)} - \ell_{\text{beh}}^{(i)}\big)$$

      Accumulate loss with IS-corrected score-function term and KL penalty:

$$L \leftarrow L - (\tilde{w}_i \tilde{r}_i) \cdot \ell_\theta^{(i)} + \frac{\lambda_{\text{KL}}}{2}\big(\ell_\theta^{(i)} - \ell_{\text{ref}}^{(i)}\big)^2$$

    end for
    Update θ by one AdamW step with parameters $(\eta, \beta_1, \beta_2, \varepsilon, \lambda)$, using $\nabla_\theta L$ clipped by global norm C_{clip}
  end for
until convergence

4 Experiment and Results

4.1 Experimental Setup

We run the three algorithms :DDPO$_{\text{SF}}$, DDPO$_{\text{IS}}$, DDPO$_{\text{ISKL}}$ (RWR, RWR$_{\text{Sparse}}$ are omitted because they are not really a part of this paper, but from previous literature) described above across three different downstream tasks (Compresibility and Incompressibility, Aesthetic Quality).

For DDPO$_{\text{ISKL}}$, we also choose two sets of hyperparameter for configuration of model training, one with fairly-optimal results and one with sub-optimal results. We also plot the reward and trajectories (against the number of iterations) for each configuration.

The following table 1 is a full documentation of our configuration of hyperparameters for the model to work well:

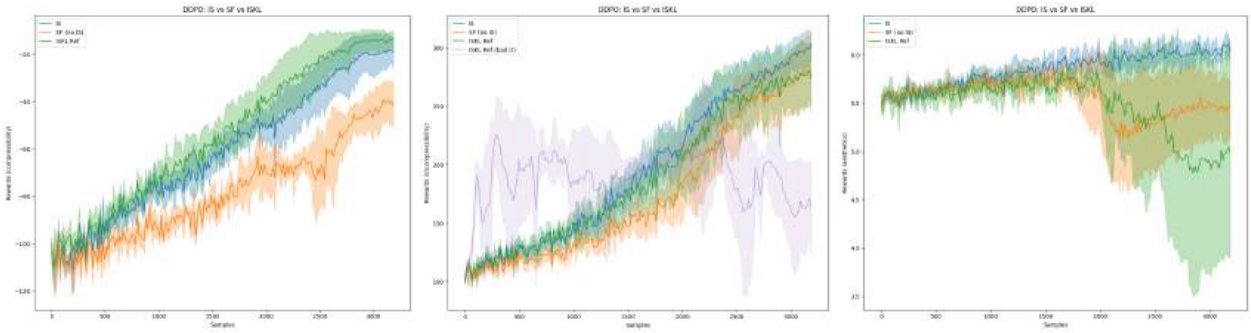
Table 1: A set of Hyperparameter that work well

	DDPO$_{\text{IS}}$	**DDPO$_{\text{SF}}$**	**DDPO$_{\text{ISKL}}$**
Diffusion			
Denoising steps (T)	50	50	50
Guidance weight (w)	5.0	5.0	5.0
Optimization			
Optimizer	AdamW	AdamW	AdamW
Learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
Weight decay	1×10^{-4}	1×10^{-4}	1×10^{-4}
β_1	0.9	0.9	0.9
β_2	0.999	0.999	0.999
ϵ	1×10^{-8}	1×10^{-8}	1×10^{-8}
Gradient clip norm	1.0	1.0	1.0
DDPO			
Batch size	16	16	16
Samples per iteration	8	16	8
Gradient updates per iteration	2	4	2
Clip range	1×10^{-4}	-	1×10^{-4}

For sub-optimal hyperparamters (Meaning bad hyperparameters), the only change we make compared to the hyperparameters above is we change the learning rate to 3×10^{-3} (from 3×10^{-4}).

4.2 Experiment Results

We report the results of the experiments of the three algorithms DDPO$_{\text{SF}}$, DDPO$_{\text{IS}}$, and DDPO$_{\text{ISKL}}$ across some selected tasks in the following format:



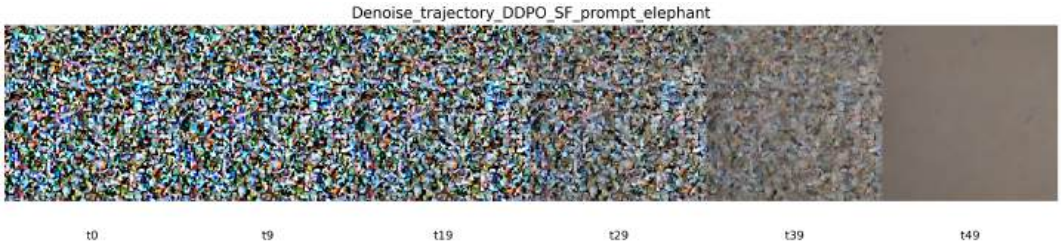
(a) Task: Compressibility (b) Task: Incompressibility (c) Task: Aesthetics

Figure 3: Rewards of three algorithms v.s. number of samples for downstream tasks, reported with mean value and standard deviation for 5 runs per algo

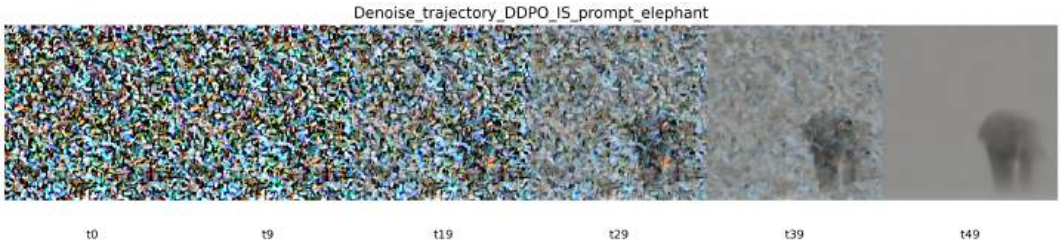
Figure 3 shows the plots of rewards against the number of samples. Each plot shows the rewards against samples for each one of three tasks. Each plot contains three curves that are the rewards plotted against number of samples for the three algorithms. The rewards are averaged over 5 runs with shading area representing the standard deviation form the mean. This is with good hyperparameters from above.

Figure 3b shows the plots of rewards against the iterations of algorithms, but with bad hyperparameters (learning rate 3×10^{-3}) included. The algorithm chosen to demonstrate bad hyperparameters in $\text{DDPO}_{\text{ISKL}}$.

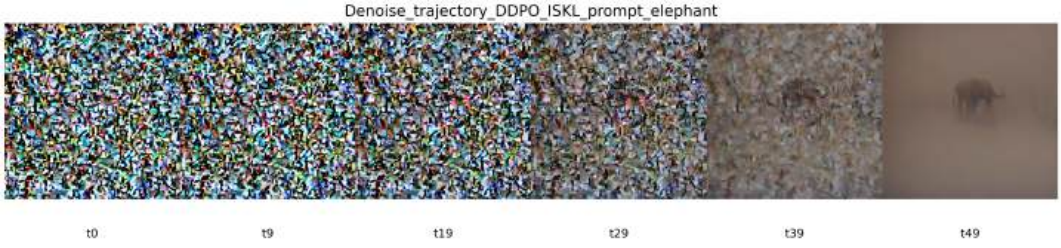
For $\text{DDPO}_{\text{ISKL}}$, we also report two groups of plots, each group is configured with a different set of hyperparameters, one working well, the other less optimal.



(a) DDPO SF

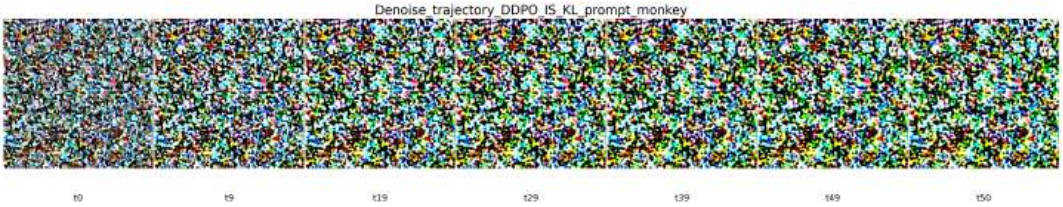


(b) DDPO IS



(c) DDPO ISKL

Figure 4: Denoising trajectories for three algorithms with prompt "elephant" with good parameters



(a) DDPO ISKL

Figure 5: Some Denoising trajectories for $\text{DDPO}_{\text{ISKL}}$ with bad hyperparameters

Figure 4 shows the plots of sampled trajectories with the learned policy against time steps for the selected task Compressibility. All three algorithms are shown. This is with good hyperparameters from above.

Firgure 5 shows the plots of sampled trajectories with the learned policy against time steps for the selected task Compressibility, but with bad hyperparameters (change the learning rate to 3×10^{-3}). The algorithm chosen to demonstrate the hyperparameters is $\text{DDPO}_{\text{ISKL}}$.

5 Conclusion

We have successfully replicated the paper: *Training Diffusion Models with Reinforcement Learning*; Training diffusion models different reinforcement learning algorithms across various downstream tasks. We also proposed our own improvement to the algorithm DDPO_{IS}, hence formulating our own algorithm $\text{DDPO}_{\text{ISKL}}$, achieving faster convergence and avoiding overoptimization in some tasks.

For all three tasks we observed that with Important Sampling, DDPO converges faster and show more stable training and less variance. The effect can be attribute to 2 reasons: 1) Off-policy allow more frequent gradient update from one round of data sampling 2) important sampling with clipping make the gradient less noisy thus resulting more stable training process

The added KL constraint with respect to a reference Stable Diffusion model is intended to keep the learned distribution close to the pretrained one, preventing the policy from collapsing into nonsense generations such as the SF algorithm's output shown here. It has a high score of compressibility, but completely loses the text-to-image alignment. Empirically, combining KL with importance sampling (IS) leads to noticeably better visual quality. Intuitively, the KL penalty regularizes the updated distribution and discourages overly aggressive parameter updates that would otherwise produce meaningless noise. As a result, the model is guided toward more desirable sampling trajectories, reducing the failure of completely losing alignment for pursuing high rewards. On task that are harder to optimize, such as aesthetics score in our case, the KL introduced extra complexity to gradient update. This could disturb the training when gradient is unstable due to our small batch size, where only DDPO IS has not suffered from collapsing during training.

References

[1] Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning, 2024.

[2] Zejian Li, Chenye Meng, Yize Li, Ling Yang, Shengyuan Zhang, Jiarui Ma, Jiayi Li, Guang Yang, Changyuan Yang, Zhiyuan Yang, Jinxiong Chang, and Lingyun Sun. Laion-sg: An enhanced large-scale dataset for training complex image-text models with structural annotations, 2024.

[3] Junzi Zhang, Jongho Kim, Brendan O'Donoghue, and Stephen Boyd. Sample efficient reinforcement learning with reinforce, 2020.