D3P: Dynamic Denoising Diffusion Policy via Reinforcement Learning

Shu'ang Yu[12], Feng Gao[1], Yi Wu[13], Chao Yu[1†], Yu Wang[1†],

[1]Tsinghua University [2]Shanghai AI Laboratory [3]Shanghai Qi Zhi Institute
[†] Corresponding Authors
{yuchao, yu-wang}@mail.tsinghua.edu.cn

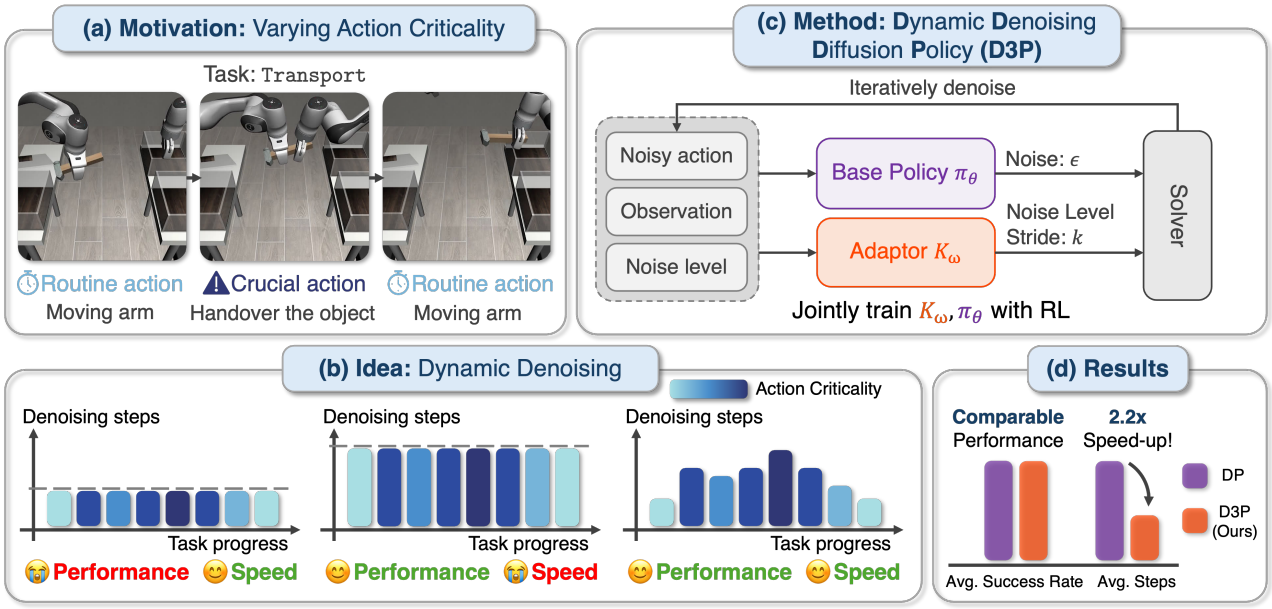


Figure 1: An overview of **D**ynamic **D**enoising **D**iffusion **P**olicy (D3P). **(a) Motivation:** Robotic tasks involve actions of varying criticality. Crucial actions, like object handover, have a greater impact on task success than routine actions. **(b) Idea:** Instead of using a fixed number of denoising steps, D3P dynamically allocates more denoising steps to crucial actions. **(c) Method:** D3P uses a base policy π_θ and a lightweight adaptor K_ω. The adaptor predicts the noise-level strides, which determines total denoising steps for current action. **(d) Results:** D3P achieves a comparable success rate, while providing a 2.2× speed-up to a normal fixed-step diffusion policy.

Abstract

Diffusion policies excel at learning complex action distributions for robotic visuomotor tasks, yet their iterative denoising process poses a major bottleneck for real-time deployment. Existing acceleration methods apply a fixed number of denoising steps per action, implicitly treating all actions as equally important. However, our experiments reveal that robotic tasks often contain a mix of *crucial* and *routine* actions, which differ in their impact on task success. Motivated by this finding, we propose **D**ynamic **D**enoising **D**iffusion **P**olicy **(D3P)**, a diffusion-based policy that adaptively allocates denoising steps across actions at test time. D3P uses a lightweight, state-aware adaptor to allocate the optimal number of denoising steps for each action. We jointly optimize the adaptor and base diffusion policy via reinforcement learning to balance task performance and inference efficiency. On simulated tasks, D3P achieves an averaged 2.2× inference speed-up over baselines without degrading success. Furthermore, we demonstrate D3P's effectiveness on a physical robot, achieving a 1.9× acceleration over the baseline.

Introduction

Diffusion policies have demonstrated remarkable promise in robotic visuomotor tasks (Janner et al. 2022; Pearce et al. 2023; Chi et al. 2023; Ze et al. 2024; Ma et al. 2024). By casting action generation as a conditional denoising diffusion process, they naturally model the full, often highly multimodal distribution of feasible actions while retaining stable

optimization dynamics (Chi et al. 2023). Concretely, action sampling integrates the reverse-time stochastic differential equation (SDE) that refines Gaussian noise into clean actions, following Ho, Jain, and Abbeel (2020) and Song et al. (2020). This procedure entails dozens of denoising steps, so diffusion policies typically run more slowly at inference than one-shot generators based on GANs (Goodfellow et al. 2020), VAEs (Kingma, Welling et al. 2013), or autoregressive models (Shafiullah et al. 2022; Zhao et al. 2023), which limits their deployment in real-time control.

To address these challenges, previous work has sought to accelerate inference through various strategies, such as reformulating the process as an ordinary differential equation (ODE) to allow fewer sampling steps (Song, Meng, and Ermon 2020; Lu et al. 2022a,b), distilling the policy into a single-step model (Prasad et al. 2024; Wang et al. 2024b), or employing streaming techniques that use action history as a prior (Høeg, Du, and Egeland 2024; Chen et al. 2025). These acceleration methods for diffusion policies typically use a uniform number of denoising steps per action, which implicitly assumes that all actions are equally important for task success.

However, our analysis of robotic tasks provides evidence that contradicts this assumption. We observe a stark non-uniformity in action criticality: task executions typically consist of pivotal **crucial actions** that largely determine success, and more forgiving **routine actions** with only a marginal impact. For instance, in a `Transport` task, the precise moment of object handover is crucial, whereas the arm movements before and after have less impact on task success. Treating all actions uniformly, therefore, incurs unnecessary computational overhead and restricts the ability to balance decision quality with inference efficiency. This insight motivates a new design principle: *Allocate denoising steps adaptively: spend more computation on crucial actions, and less on routine ones—to balance precision and efficiency.*

In this paper, we propose **D**ynamic **D**enoising **D**iffusion **P**olicy (D3P), a diffusion policy that dynamically adjusts the number of denoising steps during task execution. The D3P architecture consists of two components: a standard noise predicting network that serves as the base diffusion policy, and a lightweight adaptor (Fig. 1). The adaptor is designed to predict the *noise-level strides* based on the current observation, which eventually adapts the number of denoising steps for the current action. We mathematically formulate the dynamic denoising problem as a two-layer partially observable Markov decision process (POMDP). Then we use reinforcement learning (RL) with this two-layer POMDP to jointly train both the base diffusion policy and the adaptor. Specifically, a base policy is fine-tuned with DPPO (Ren et al. 2024) to maximize task success. Concurrently, the lightweight adaptor is trained from scratch with PPO (Schulman et al. 2017), where its reward function incentivizes minimizing denoising steps without compromising performance. A key challenge in this joint training process is to balance the contradictory objectives of task performance and inference efficiency. To address this, we introduce a three-stage training strategy that ensures stable convergence and robust performance.

We evaluate D3P on a range of simulated manipulation tasks. Using the same amount of training data, D3P achieves an average 2.2× inference speed-up over baseline methods while maintaining comparable performance. These results demonstrate that D3P effectively addresses the trade-off between performance and efficiency. Furthermore, we deploy D3P on the physical robot. It achieves a 1.9× acceleration in inference speed against a normal fixed-step diffusion policy. Overall, our main contributions are as follows:

1. Through an empirical study, we reveal that different actions contribute unequally to manipulation tasks. There exist some crucial actions significantly influencing task completion.
2. Building on this observation, we propose **D**ynamic **D**enoising **D**iffusion **P**olicy (**D3P**). Trained via RL, D3P features a noise-predicting network as the base diffusion policy and an adaptor to dynamically adjust its denoising steps during task execution.
3. We conduct experiments in eight simulated manipulation tasks, demonstrating that D3P achieves the best performance with an averaged 2.2× speed-up over baselines.
4. We demonstrate that D3P can be successfully deployed on a physical robot. It achieves a 1.9× acceleration against a standard fixed-step diffusion policy.

Preliminaries

Partially Observable Markov Decision Process (POMDP) We formulate the robot environment within the framework of a partially observable Markov decision Process (POMDP), defined by the tuple $\mathcal{M}_{\text{ENV}} := (\mathcal{S}_{\text{ENV}}, \mathcal{A}_{\text{ENV}}, \mathcal{O}_{\text{ENV}}, P_{\text{init,ENV}}, P_{\text{ENV}}, R_{\text{ENV}})$. Here, $\mathcal{S}_{\text{ENV}}$, $\mathcal{A}_{\text{ENV}}$, and $\mathcal{O}_{\text{ENV}}$ are the state, action, and observation space separately. The process begins with an initial state $s_0 \sim P_{\text{init,ENV}}$. At each environment timestep t, an agent receives an observation $o_t \in \mathcal{O}_{\text{ENV}}$, and takes an action $a_t \in \mathcal{A}_{\text{ENV}}$ according to a policy $\pi(a_t \mid o_t)$. The environment responds by transitioning to a new state $s_{t+1} \sim P_{\text{ENV}}(s_{t+1} \mid s_t, a_t)$ and providing a reward $r_t = R_{\text{ENV}}(s_t, a_t)$. The objective in RL is to learn a policy π_θ that maximizes the expected return, $\mathbb{E}_{\pi_\theta}\left[J_t(s_t, a_t)\right] := \mathbb{E}_{\pi_\theta}\left[\sum_{\tau=t}^{T-1} \gamma_{\text{ENV}}^{\tau-t} r_\tau \mid s_t, a_t\right]$, where $\gamma_{\text{ENV}} \in (0, 1)$ is a discount factor and T is the the episode horizon.

Diffusion Models Denoising diffusion probabilistic models (DDPMs) (Sohl-Dickstein et al. 2015; Ho, Jain, and Abbeel 2020) frame sample generation as an iterative denoising procedure, reversing a length-N diffusion chain $\{x^i\}_{i=0}^N$, where x^0 is a clean sample, $x^N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is pure Gaussian noise, and N is the number of denoising steps. The denoising is parameterized by a neural network, $\epsilon_\theta(x^i, i)$, trained to predict the noise component within a noisy sample x^i at noise-level i. At inference, this process starts with a sample of pure noise x^N and progressively refines it over N steps, generating a clean sample x^0.

Despite great performance, inference in DDPMs is slow because it must execute all N reverse steps. Denoising

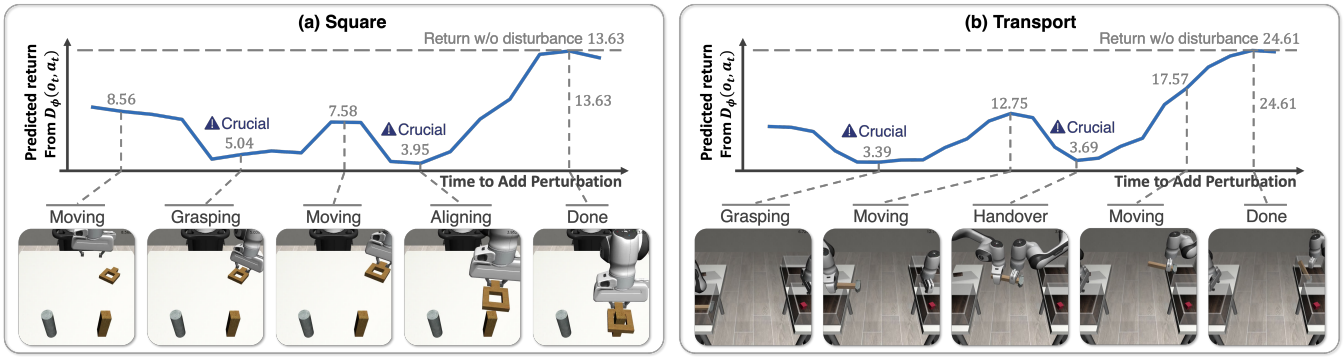


Figure 2: **Visualizing Action Criticality via Perturbed Returns.** The plots show the predicted perturbed return from $D_\phi(o_t, a_t)$ at different time of the task. A lower return indicates the action more crucial, as a perturbation is more likely to lead to task failure.

diffusion implicit models (DDIMs) (Song, Meng, and Ermon 2020) accelerate sampling by replacing the stochastic reverse process with a deterministic mapping that allows larger strides in the noise schedule. Using the same training loss as DDPMs, DDIMs traverse only a sparse set of noise levels $\tau_0 > \tau_1 > \cdots > \tau_S = 0$ where $S \ll N$. At a given level i, the sampler can skip $k > 1$ noise-levels and go directly to a less–noisy point x^{i-k}:

$$x^{i-k} \sim \mathcal{N}\big(\mu(x^i, \epsilon_i, i, k),\ \eta\,\sigma_i^2\mathbf{I}\big), \tag{1}$$

where σ_i comes from the predefined schedule. Setting $\eta = 0$ removes the stochastic term, making the process fully deterministic and much faster for inference

Diffusion Policies Diffusion policy (DP) (Chi et al. 2023) treats a diffusion model as the control policy π_θ. To preserve temporal consistency, DP predicts an action chunk $X_t = \{a_t, \ldots, a_{t+T_a-1}\}$ conditioned on the current observation o_t. With the DDIM sampler, the denoising update at noise level i is

$$X_t^{i-1} \sim \mathcal{N}\left(\mu\left(X_t^i, \epsilon_\theta(X_t^i, o_t, i), i, k\right),\ \eta\,\sigma_i^2\mathbf{I}\right), \tag{2}$$

with k the stride. This iterative procedure forms a POMDP $\mathcal{M}_{\text{DN}} := (\mathcal{S}_{\text{DN}}, \mathcal{A}_{\text{DN}}, \mathcal{O}_{\text{DN}}, P_{\text{init,DN}}, P_{\text{DN}}, R_{\text{DN}})$. This process starts with a state of pure noise $X_t^N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and terminates at $i = 0$ to produce the final action chunk.

Empirical Study: Identifying Crucial Actions

We conduct an empirical study to validate that not all actions in a task are equally important. An action is deemed crucial if a disturbance on it significantly degrades task performance. Otherwise, it is considered routine.

Experiment Design We start from a pre-trained expert policy π_{expert} that achieves over 90% success in the environment $\mathcal{M}_{\text{ENV}}$. At a randomly selected timestep $t \in \{0, \ldots, T-1\}$ we add Gaussian noise ξ to the expert action a_t, yielding a perturbed action $a'_t = a_t + \xi$. The episode then resumes under π_{expert}. The perturbation's impact is measured by the episode return, $J(s_t, a'_t) = \mathbb{E}_{\pi_{\text{expert}}}\left[\sum_{\tau=0}^{T} \gamma_{\text{ENV}}^{\tau-t} r_\tau \mid s_t, a'_t\right]$. A low J signals that the original a_t was crucial. We fit a lightweight predictor $D_\phi : \mathcal{O}_{\text{ENV}} \times \mathcal{A}_{\text{ENV}} \to \mathbb{R}$ that predicts J from the unperturbed pair (o_t, a_t). More implementation details of our empirical study appear in **Appendix A**.

Key Findings Fig. 2 plots D_ϕ's predicted return on the `Square` and `Transport` tasks from Robomimic (Mandlekar et al. 2021). The results demonstrate that the criticality of actions throughout a task is highly non-uniform. For `Square`, fine interactions, such as grasping the block and aligning it with the peg, receive the lowest predicted returns (5.04, 3.95), marking them as crucial. Whereas broad arm motions score higher (7.58, 8.56) and are routine. After completion, the return peaks at 13.63 because later actions no longer influence success. For `Transport`, grasping the hammer (3.39) and the bimanual hand-over (3.69) dominate task success, while transit motions are routine. Overall, crucial actions are concentrated in direct physical interactions, such as grasping and handover, while broader movements have a relatively minor impact on the results. This observation motivates our approach: adaptively allocating denoising steps to different actions, focusing more capacity on the crucial actions.

Method

In this section, we introduce **D**ynamic **D**enoising **D**iffusion **P**olicy (**D3P**), a method designed to dynamically adjust the number of denoising steps during task execution. D3P utilizes the noise-level stride scheme in DDIM solver (Song, Meng, and Ermon 2020), and augments a base diffusion policy π_θ with an adaptor K_ω trained to pick strides, as shown in Fig. 1(c). The adaptor observes $\left(o_t,\ X_t^i\right)$ and outputs a stride: large strides skip more noise levels to speed up routine actions, while small strides keep precise denoising for crucial actions. We frame adaptive denoising as a two-layer POMDP and train the base policy π_θ and adaptor K_ω jointly with RL. During the training, π_θ is fine-tuned to maximize task reward, while K_ω learns to cut denoising steps without degrading success. A three-stage training strategy is used to stabilize training. The full algorithm is summarized in Alg. 1.

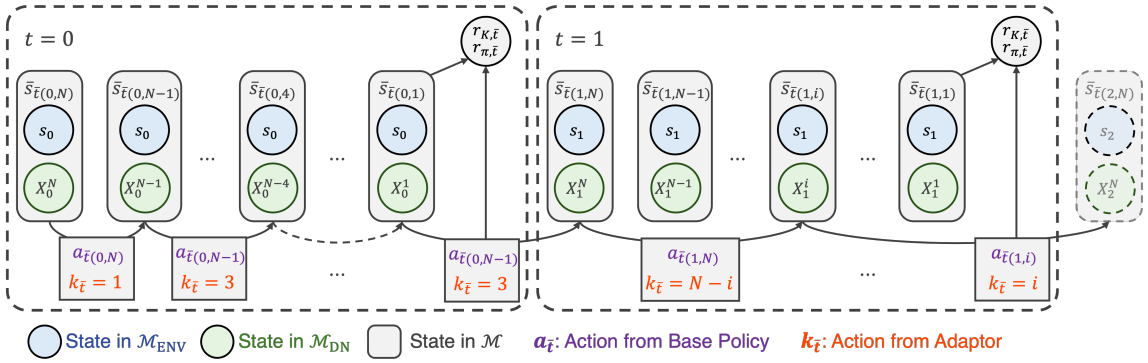


Figure 3: We formulate the dynamic denoising problem as a two-layer POMDP, where a denoising process ($\mathcal{M}_{\text{DN}}$) is nested within the environment ($\mathcal{M}_{\text{ENV}}$). At each step, the adaptor K_ω predicts the noise-level strides. The base diffusion policy and the adaptor are jointly updated via RL.

Problem Formulation

Following Psenka et al. (2023); Ren et al. (2024), we define a two-layer POMDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, P_{\text{init}}, P, R)$ that nests the denoising process $\mathcal{M}_{\text{DN}}$ inside the environment $\mathcal{M}_{\text{ENV}}$ (Fig. 3). In this two-layer PODMP $\mathcal{M}$, we denote a time index as $\bar{t}(t,i) = tN + (N - i - 1)$, where t is the environment step and $i \in [0, N]$ is the index of noise level. The states and observations in $\mathcal{M}$ are all tuples, denoted as $\bar{s}_{\bar{t}} = (s_t, X_t^i), \bar{o}_{\bar{t}} = (o_t, X_t^i)$, with $s_t \in \mathcal{S}_{\text{ENV}}$ and $X_t^i \in \mathcal{S}_{\text{DN}}$. An episode starts at $\bar{t}(0, N)$ with $\bar{s}_{\bar{t}(0,N)} \sim P_{\text{init}}$, and P_{init} is defined as

$$P_{\text{init}} = (P_{\text{init,ENV}},\ P_{\text{init,DN}})\,. \tag{3}$$

At each time step $\bar{t}$, the action tuple $(a_{\bar{t}}, k_{\bar{t}})$ is generated by the base policy π_θ and the adaptor K_ω following Eq. (4), where $\lfloor \cdot \rfloor$ indicates rounding down.

$$\begin{aligned} k_{\bar{t}(t,i)} &\sim K_\omega(k_{\bar{t}} \mid \bar{o}_{\bar{t}}),\ j_{\bar{t}} = \max\left(i - \lfloor k_{\bar{t}(t,i)} \rfloor,\ 0\right), \\ a_{\bar{t}} &\sim \pi_\theta(a_{\bar{t}} \mid \bar{o}_{\bar{t}}, i) := \mathcal{N}\left(\mu(X_t^i, \epsilon_\theta(\bar{o}_{\bar{t}}, i), i, k_{\bar{t}}), \eta\sigma_i^2 \mathbf{I}\right). \end{aligned} \tag{4}$$

Eq. (4) indicates that K_ω predicts the noise-level strides $k_{\bar{t}}$ to further control the total denoising steps. The η is set to 1 during training, making the Gaussian likelihood of π_θ computable, and is set to 0 when inference. After taking actions, $\mathcal{M}$ makes transition following Eq. (5).

$$\bar{s}_{\bar{t}+1} \sim \begin{cases} (\delta_{s_t}, \delta_{a_{\bar{t}}}) & ,\ j > 0, \\ \left(P_{\text{ENV}}(s_{t+1} \mid s_t, X_t^0), P_{\text{init,DN}}\right) & ,\ j = 0. \end{cases} \tag{5}$$

In Eq. (5), $j > 0$ signifies an incomplete denoising process, and both the environment state s_t and observastion o_t remain unchanged. When $j = 0$, the denoising process concludes and the resulting action is executed in the environment. Subsequently, $\mathcal{M}_{\text{ENV}}$ transitions to a new state according to P_{ENV}, and a new pure noise, X_{t+1}^N, is resampled from Gaussian distribution.

Dynamic Denoising

Following the problem formulation, we jointly train π_θ and K_ω on the same batch of rollout data.

To fine-tune the base policy, we apply DPPO (Ren et al. 2024) to optimize π_θ and its value critic V_Θ on the dynamic-denoising POMDP $\mathcal{M}$. The critic $V_\Theta(o_t)$, conditioned on the environment observation o_t, later serves as a proxy for task performance when training the adaptor.

The adaptor aims to reduce the total denoising steps without degrading performance. The most direct performance signal is the binary success flag $r_{s,\bar{t}} \in \{0, 1\}$, yet its sparsity and high variance impede stable learning. Instead, we employ the advantage of each state–action pair in $\mathcal{M}_{\text{ENV}}$ as a dense, low-variance, but slightly biased metric, computed as

$$\hat{A}_\Theta(X_t^0, o_t) = J_{\pi_\theta}(s_t, X_t^0) - V_\Theta(o_t), \tag{6}$$

where $J_{\pi_\theta}(s_t, X_t^0)$ is the discounted return defined in the **Preliminaries**. A larger $\hat{A}_\Theta$ indicates that the fully denoised action X_t^0 is beneficial for future returns.

To encourage shorter denoising chains, we introduce a discount factor $\gamma_s \in (0, 1)$ that penalizes longer denoising process. Balancing task performance against computational cost, we define the reward for adaptor as

$$r_{K,\bar{t}(t,i)} = \begin{cases} 0 & ,\ j > 0 \\ \alpha\ \hat{A}_\Theta\left(X_t^0, o_t\right)\ \gamma_s^{\text{sgn}_t \times \text{stp}_t} & \\ +\beta\ r_{s,\bar{t}}\ \gamma_s^{\text{stp}_t} & ,\ j = 0 \end{cases} \tag{7}$$

where α and β are weighting coefficients, sgn_t is the sign of the advantage $\hat{A}_\Theta(X_t^0, o_t)$, and stp_t denotes the total denoising steps for generating the clean action X_t^0. This formulation rewards advantageous actions and successful episodes while exponentially penalizing long denoising sequences.

We update K_ω with PPO (Schulman et al. 2017), minimizing the clipped PPO loss in Eq. (8).

$$\begin{aligned} -L^{\text{CLIP}}(\omega) = -\hat{\mathbb{E}}_{K_{\omega_{\text{old}}}} \Big[&\min\Big(\rho(\omega)\hat{A}_{\omega_{\text{old}}}(k_{\bar{t}}, \bar{o}_{\bar{t}}), \\ &\text{clip}\left(\rho(\omega), 1 - \epsilon_{\text{clip}}, 1 + \epsilon_{\text{clip}}\right) \hat{A}_{\omega_{\text{old}}}(k_{\bar{t}}, \bar{o}_{\bar{t}})\Big)\Big]. \end{aligned} \tag{8}$$

Here, $\rho(\omega) = \frac{K_\omega(k_{\bar{t}}|\bar{o}_{\bar{t}})}{K_{\omega_{\text{old}}}(k_{\bar{t}}|\bar{o}_{\bar{t}})}$ is the probability ratio between the current and old adaptor. The advantage $\hat{A}_{\omega_{\text{old}}}$ is computed using Generalized Advantage Estimation (GAE) (Schulman et al. 2015) based on the reward $r_{K,\bar{t}}$.

Algorithm 1: Dynamic Denoising Diffusion Policy (D3P)

Input: Pretrained base policy π_θ, dynamic denoising environment $\mathcal{M}$, discount factors $\gamma_{\text{ENV}}, \gamma_s$, environment horizon T, max denoise steps N, warm-up denoising steps c, update epochs e, e_{slow}, stage threshold ζ_1, ζ_2.
Initialize: Adaptor K_ω with mean of c, empty buffer $\mathcal{B}$.
while Average $r_{s,bart} < \zeta_1$ **do** ▷ Stage 1: Warm-up
    Warm-up π_θ using DPPO (Ren et al. 2024) with fixed denoising steps c
while not converged **do** ▷ Stage 2: Joint training
    Reset environment to $\bar{t}(0, N)$ as Eq. (3)
    while $t < T$ **do** ▷ Rollout data
        Sample action $(k_{\bar{t}}, \bar{X}_{\bar{t}})$ following Eq. (4). Get $\log K_{\bar{t}}, \log \pi_{\bar{t}}$
        Execute the actions. Perform transition as Eq. (5). Get $\bar{o}_{\bar{t}+1}$ and $r_{\pi,\bar{t}}$ following (Ren et al. 2024).
        Add $\left(\bar{o}_{\bar{t}}, (k_{\bar{t}}, a_{\bar{t}}), (\log K_\omega(k_{\bar{t}}), \log \pi_\theta(a_{\bar{t}})), r_{\pi,\bar{t}}\right)$ to buffer $\mathcal{B}$.
    Calculate the advantage $\hat{A}_\Theta$ using Eq. (6). Get success flag $r_{s,\bar{t}}$ refer to task results. ▷ Prepare for update
    Set $r_{K,\bar{t}}$ as Eq. (7), using $\hat{A}_\Theta$ and $\mathrm{r}_{s,\bar{t}}$.
    for epoch $= 0, 1, \ldots, e-1$ **do** ▷ Policy optimization
        Update parameter θ and Θ using DPPO (Ren et al. 2024)
        Update parameter ω with PPO loss in Eq. (8)
    if Average stp $< \zeta_2$ **then** $e \leftarrow e_{\text{slow}}$ ▷ Stage 3: Conservative fine-tuning
Return: Trained base policy π_θ and adaptor K_ω.

Training Strategy

Directly training the base policy and adaptor from scratch often causes instability and even collapse. To address this, D3P adopts a three-stage training strategy preceded by behavior cloning of the base policy π_θ on pre-collected datasets.

Stage 1: Base DP Warm-up. We first warm up π_θ with DPPO (Ren et al. 2024) while keeping the denoising steps fixed at c. The warming up proceeds until the task success rate exceeds a preset threshold ζ_1. This stage can be skipped if a sufficiently strong base policy is already available.

Stage 2: Joint Training. Next, we initialize the adaptor K_ω as a Gaussian policy with mean of c and variance of v^2. Then we train K_ω and π_θ jointly. Each iteration collects a batch of trajectories followed by e PPO update epochs for both modules.

Stage 3: Conservative Fine-tuning. When the average denoising steps satisfies $\mathbb{E}[\text{stp}_t] < \zeta_2$, we switch to a conservative stage. This stage mirrors Stage 2 but uses fewer update epochs per iteration. This precaution prevents the adaptor from shrinking the denoising steps so aggressively that it destabilizes the base policy and leads to a collapse.

The full algorithm of D3P, including the three-stage schedule, is summarized in Alg. 1.

Experiments

To evaluate D3P, we conduct comprehensive robot manipulation experiments in both simulation and the real world. We first detail the experimental setups, then benchmark D3P against several baselines. Subsequently, we present ablation studies to analyze the contribution of each component in our framework. Finally, we demonstrate the deployment of D3P onto a physical robot, highlighting its practical applicability.

Setups

Environments We evaluate our method on eight manipulation tasks from two benchmarks: Robomimic (Mandlekar et al. 2021) and Franka Kitchen (Gupta et al. 2019). These environments include simple pick-and-place tasks and challenging long-horizon, multi-stage assembly tasks.

Baselines We compare D3P against three representative baselines covering different paradigms: (1) DPPO (Ren et al. 2024), a state-of-the-art (SOTA) algorithm for *online fine-tuning* DPs, (2) consistency policy (CP) (Prasad et al. 2024), a *distillation-based acceleration* method, and (3) Falcon (Chen et al. 2025), a training-free *streaming* approach. To ensure a fair comparison, all policies are pre-trained on the same dataset. Using the pre-trained policy, we train D3P and the DPPO policy with an identical amount of online RL data. The Falcon and consistency policy baselines are subsequently derived from the DPPO fine-tuned policy.

Metrics We evaluate task performance using success rate and episodic return ($J = \sum_{\tau=0}^{T} r_\tau$) where higher values indicate better performance. To assess computational efficiency, we follow Prasad et al. (2024); Chen et al. (2025) and adopt the Number of Function Evaluations (NFE) per action as our primary metric. NFE provides a fair comparison of inference cost because all methods use the same base network architecture. For DPPO and CP, the NFE count is equivalent to the number of denoising steps. For Falcon, its selection mechanism requires an additional base policy inference, adding one NFE per action. For our method, we only count the NFE from the base policy, as our lightweight adaptor has fewer than $1/15$ the parameters of the base policy. All results are averaged over three random seeds, with evaluations conducted on 100 episodes per seed.

For additional details on the experimental setup, please refer to **Appendix B**.

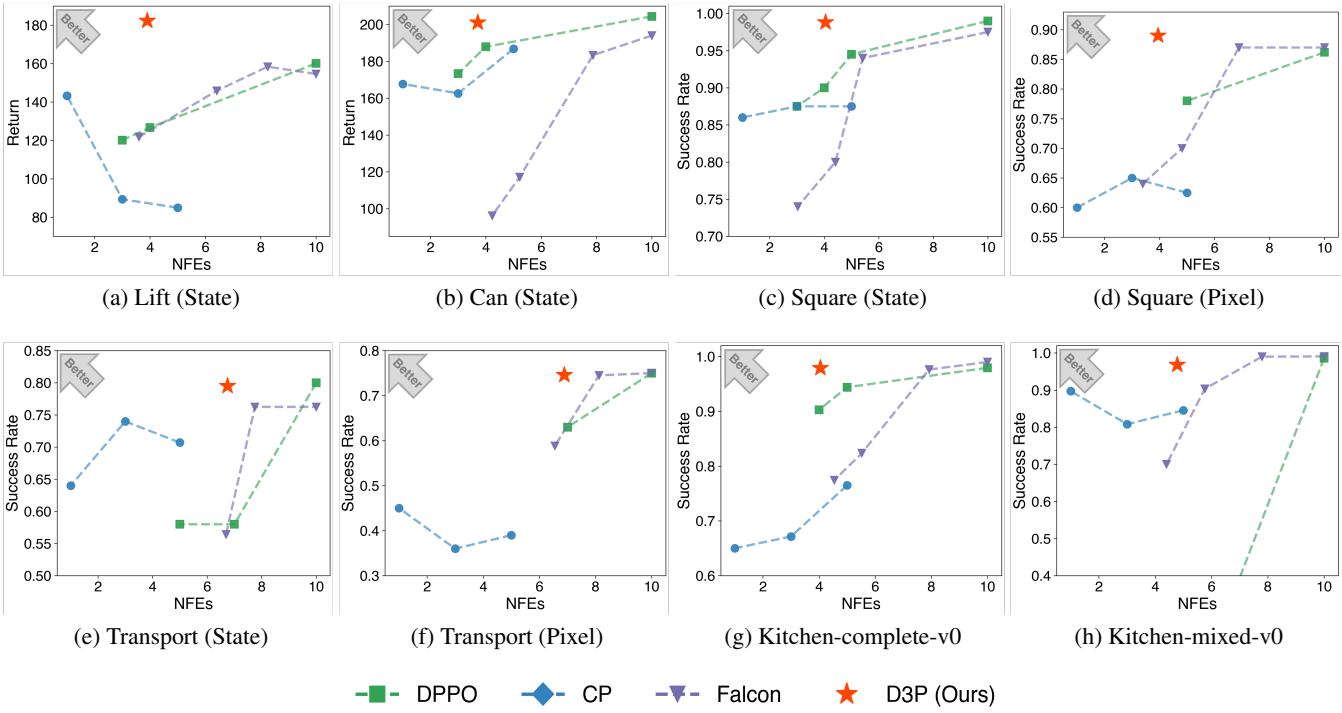


Figure 4: We plot success rate / episodic return against the Number of Function Evaluations (NFE) per action. We use episodic return for the simpler `Lift` and `Can` tasks because success rates for most methods saturate above 90%. D3P achieves the best in performance and efficiency across all tasks. D3P matches or surpasses the peak performance of the 10-step DPPO baseline while achieving an average 2.2× speed-up. All results are averaged over 3 seeds with 100 evaluation episodes per seed.

Performance Evaluation

Fig. 4 compares our D3P against the baselines by plotting their task performance versus inference cost. On the simpler `Lift` and `Can` tasks, the success rates for most methods saturate above 90% and we use episodic return as a more fine-grained performance metric. In this figure, an ideal method would occupy the upper-left corner, signifying high performance achieved at a low inference cost.

As expected, the performance of the DPPO fine-tuned policy correlates with its inference cost. Reducing the NFEs improves training and inference speed, yet leads to a clear performance drop in success rate and return. Similarly, while Falcon accelerates the 10-step DPPO policy, its performance consistently decreases as the acceleration ratio increases. The success rate (or return) of Falcon degrades sharply under 6 NFE. Consistency Policy, through distillation, enables few-step or even single-step inference. However, the distillation process creates a performance ceiling that prevents it from matching the optimal, multi-step teacher policy.

In contrast, D3P dynamically adapts its denoising effort, using more denoising steps only when necessary. This allows D3P to match or even exceed the peak performance of the 10-step DPPO policy while achieving an average inference speed-up of 2.2 times. The results unequivocally demonstrate that D3P establishes a better Pareto frontier, consistently achieving an optimal performance-efficiency trade-off than all baselines across all tasks. Due to space constraints, detailed training curves are provided in **Appendix C**.

Ablation study

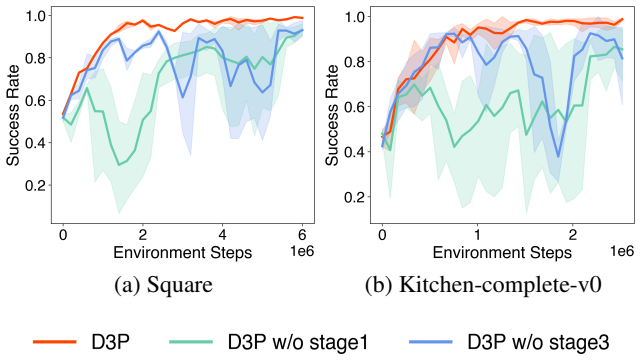
We perform ablation studies on the `Square` and `Kitchen-complete-v0` tasks to isolate the contrinbutions of D3P's key design choices.

Fig. 5 validates the importance of our three-stage training strategy. Removing stage 1 leads to a significant performance drop at the start of the training, while removing stage 3 causes unstable curves during later training. Our full three-stage approach effectively warms up the policy and then stabilizes the fine-tuning process, proving crucial for guiding a robust, optimal convergence.

Fig. 6 shows the analysis of our reward formulation in Eq. (7). Setting $\alpha = 0$ leaves only the unbiased but delayed success reward. This high-variance signal makes training unstable and prone to failure. Setting $\beta = 0$ leaves only the low-variance but biased advantage term. While the advantage term stabilizes training, the policy is more likely to converge to a suboptimal solution.

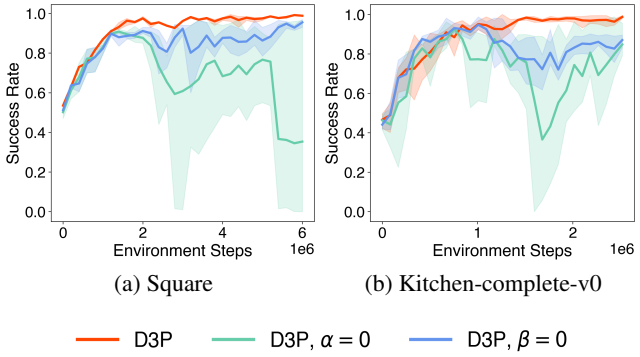
Real-world Deployment

To demonstrate D3P's effectiveness in the physical world, we deploy the policy on a Franka robot arm. All inference was performed on a consumer-grade desktop (i7-12900K CPU, RTX 2080 GPU). We mitigate the visual sim-to-real



(a) Square (b) Kitchen-complete-v0

Figure 5: Ablation of the three-stage training strategy. Removing Stage 1 impairs initial learning, while removing Stage 3 destabilizes final convergence. The full strategy is critical for achieving rapid and stable performance.



(a) Square (b) Kitchen-complete-v0

Figure 6: Ablation of our reward formulation. Relying solely on the success reward ($\alpha = 0$) causes training instability, while using only the advantage term ($\beta = 0$) leads to suboptimal convergence. Both components are essential for stable training towards an optimal policy.

gap with a latent diffusion model (Rombach et al. 2021) that aligns real-world images simulated ones. As illustrated in Fig. 13, D3P successfully performs the `Square` task. For crucial actions such as grasping and aligning, D3P increases its denoising steps to 8 and 6, to generate accurate actions. Conversely, for simpler motions, it reduces the step count to as low as 3. D3P achieves the control frequency of 33.68 Hz, a 1.92× speedup over the 17.59 Hz of a fixed 10-step diffusion policy. Additional deployment details are provided in **Appendix D**.

Related Work

Optimizing Diffusion Policies via RL To overcome the data dependency of imitation learning (Chi et al. 2023; Pearce et al. 2023; Wang et al. 2024b; Prasad et al. 2024), many methods use RL to optimize DPs. In offline RL, methods adapt DPs using techniques derived from Q-learning (Wang, Hunt, and Zhou 2022; Hansen-Estruch et al. 2023) or policy gradients (Kang et al. 2023). In the online setting, DPs are often trained within actor-critic frameworks (Wang et al. 2024a; Yang et al. 2023; Ren et al. 2024; Li et al. 2024) or with action-gradients from a learned Q-function (Psenka et al. 2023). However, these methods apply a fixed number of denoising steps for all actions, leaving the critical issue of slow inference speed unresolved.

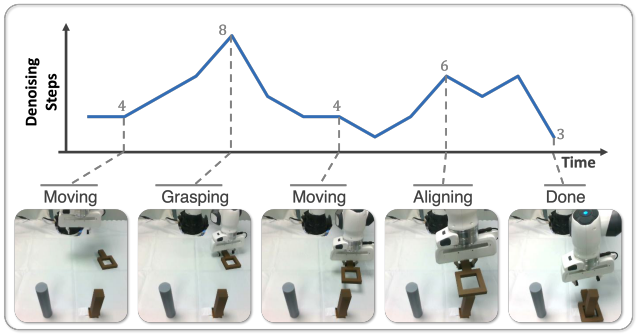


Figure 7: Real-world demonstration of D3P performing the `Square` task. The plot shows that D3P dynamically adjusts the number of denoising steps during task execution. It allocates more steps for crucial actions, such as grasping and aligning, and fewer for routine movements.

Accelerating Diffusion Policy Inference To improve the inference speed of DPs, a straightforward way is reduce the number of denoising steps. Prior work primarily use policy distillation (Prasad et al. 2024; Wang et al. 2024b) or streaming denoising (Høeg, Du, and Egeland 2024; Chen et al. 2025). A key limitation is that these methods treat all actions as equally important. In contrast, our method, D3P, adaptively adjusts the computational effort for each action.

While the concept of adaptive denoising exists for single-image generation (Ye et al. 2025), the sequential decision-making of robotics presents distinct challenges with long-term rewards and temporal dependencies. We address this by formulating the dynamic denoising problem as a two-layer POMDP and jointly optimizing the base diffusion policy and the adaptor. The training process is stabilized by a specialized reward and a three-stage training strategy.

Conclusion

In this work, we introduced Dynamic Denoising Diffusion Policy (D3P), a diffusion policy capitalizing on the varying action criticalities in robotic tasks. D3P employs a lightweight adaptor to dynamically adjust denoising steps, assigning more steps to crucial actions and fewer to routine ones. We use RL to joint optimize the base policy and the adaptor with a carefully-designed reward and a three-stage training strategy. Our simulation experiments demonstrate that D3P achieves an averaged 2.2× inference speedup over baselines without compromising task success. Furthermore, D3P is deployed on a physical robot, achieving a 1.9× inference acceleration against a fixed-step diffusion policy. These results underscore the potential of adaptive inference in robot learning, developing more efficient policies for real-time applications.

References

Chen, H.; Liu, M.; Ma, C.; Ma, X.; Ma, Z.; Wu, H.; Chen, Y.; Zhong, Y.; Wang, M.; Li, Q.; and Yang, Y. 2025. Falcon: Fast Visuomotor Policies via Partial Denoising. arXiv:2503.00339.

Chi, C.; Xu, Z.; Feng, S.; Cousineau, E.; Du, Y.; Burchfiel, B.; Tedrake, R.; and Song, S. 2023. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 02783649241273668.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2020. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144.

Gupta, A.; Kumar, V.; Lynch, C.; Levine, S.; and Hausman, K. 2019. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*.

Hansen-Estruch, P.; Kostrikov, I.; Janner, M.; Kuba, J. G.; and Levine, S. 2023. Idql: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33: 6840–6851.

Høeg, S. H.; Du, Y.; and Egeland, O. 2024. Streaming Diffusion Policy: Fast Policy Synthesis with Variable Noise Diffusion Models. *arXiv preprint arXiv:2406.04806*.

Janner, M.; Du, Y.; Tenenbaum, J.; and Levine, S. 2022. Planning with Diffusion for Flexible Behavior Synthesis. In Chaudhuri, K.; Jegelka, S.; Song, L.; Szepesvari, C.; Niu, G.; and Sabato, S., eds., *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 9902–9915. PMLR.

Kang, B.; Ma, X.; Du, C.; Pang, T.; and Yan, S. 2023. Efficient diffusion policies for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36: 67195–67212.

Kingma, D. P.; Welling, M.; et al. 2013. Auto-encoding variational bayes.

Li, S.; Krohn, R.; Chen, T.; Ajay, A.; Agrawal, P.; and Chalvatzaki, G. 2024. Learning multimodal behaviors from scratch with diffusion policy gradient. *Advances in Neural Information Processing Systems*, 37: 38456–38479.

Loshchilov, I.; and Hutter, F. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Lu, C.; Zhou, Y.; Bao, F.; Chen, J.; Li, C.; and Zhu, J. 2022a. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35: 5775–5787.

Lu, C.; Zhou, Y.; Bao, F.; Chen, J.; Li, C.; and Zhu, J. 2022b. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv preprint arXiv:2211.01095*.

Ma, X.; Patidar, S.; Haughton, I.; and James, S. 2024. Hierarchical diffusion policy for kinematics-aware multi-task robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 18081–18090.

Mandlekar, A.; Xu, D.; Wong, J.; Nasiriany, S.; Wang, C.; Kulkarni, R.; Fei-Fei, L.; Savarese, S.; Zhu, Y.; and Martín-Martín, R. 2021. What Matters in Learning from Offline Human Demonstrations for Robot Manipulation. In *arXiv preprint arXiv:2108.03298*.

Pearce, T.; Rashid, T.; Kanervisto, A.; Bignell, D.; Sun, M.; Georgescu, R.; Macua, S. V.; Tan, S. Z.; Momennejad, I.; Hofmann, K.; and Devlin, S. 2023. Imitating Human Behaviour with Diffusion Models. arXiv:2301.10677.

Prasad, A.; Lin, K.; Wu, J.; Zhou, L.; and Bohg, J. 2024. Consistency policy: Accelerated visuomotor policies via consistency distillation. *arXiv preprint arXiv:2405.07503*.

Psenka, M.; Escontrela, A.; Abbeel, P.; and Ma, Y. 2023. Learning a diffusion model policy from rewards via q-score matching. *arXiv preprint arXiv:2312.11752*.

Ren, A. Z.; Lidard, J.; Ankile, L. L.; Simeonov, A.; Agrawal, P.; Majumdar, A.; Burchfiel, B.; Dai, H.; and Simchowitz, M. 2024. Diffusion policy policy optimization. *arXiv preprint arXiv:2409.00588*.

Rombach, R.; Blattmann, A.; Lorenz, D.; Esser, P.; and Ommer, B. 2021. High-Resolution Image Synthesis with Latent Diffusion Models. arXiv:2112.10752.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shafiullah, N. M.; Cui, Z.; Altanzaya, A. A.; and Pinto, L. 2022. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35: 22955–22968.

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

Song, J.; Meng, C.; and Ermon, S. 2020. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*.

Song, Y.; Sohl-Dickstein, J.; Kingma, D. P.; Kumar, A.; Ermon, S.; and Poole, B. 2020. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*.

Wang, Y.; Wang, L.; Jiang, Y.; Zou, W.; Liu, T.; Song, X.; Wang, W.; Xiao, L.; Wu, J.; Duan, J.; and Li, S. E. 2024a. Diffusion Actor-Critic with Entropy Regulator. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *Advances in Neural Information Processing Systems*, volume 37, 54183–54204. Curran Associates, Inc.

Wang, Z.; Hunt, J. J.; and Zhou, M. 2022. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*.

Wang, Z.; Li, Z.; Mandlekar, A.; Xu, Z.; Fan, J.; Narang, Y.; Fan, L.; Zhu, Y.; Balaji, Y.; Zhou, M.; et al. 2024b. One-step diffusion policy: Fast visuomotor policies via diffusion distillation. *arXiv preprint arXiv:2410.21257*.

Yang, L.; Huang, Z.; Lei, F.; Zhong, Y.; Yang, Y.; Fang, C.; Wen, S.; Zhou, B.; and Lin, Z. 2023. Policy representation via diffusion probability model for reinforcement learning. *arXiv preprint arXiv:2305.13122*.

Ye, Z.; Chen, Z.; Li, T.; Huang, Z.; Luo, W.; and Qi, G.-J. 2025. Schedule on the fly: Diffusion time prediction for faster and better image generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 23412–23422.

Ze, Y.; Zhang, G.; Zhang, K.; Hu, C.; Wang, M.; and Xu, H. 2024. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. *arXiv preprint arXiv:2403.03954*.

Zhao, T. Z.; Kumar, V.; Levine, S.; and Finn, C. 2023. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*.

A Implementation Details of Empirical Study

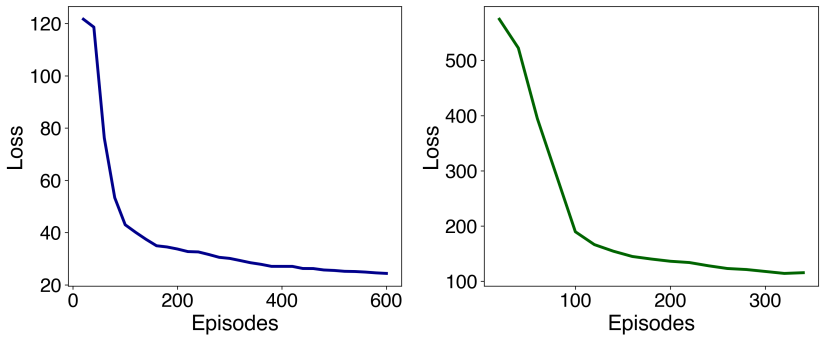
Our empirical study reveals that not all actions in a robotic task contribute equally to the results. This section details the specifics of our empirical study.

As introduced in the **Empirical Study**, we train a return predictor, denoted as $D_\phi : \mathcal{O}_{\text{ENV}} \times \mathcal{A}_{\text{ENV}} \to \mathbb{R}$, to predict the subsequent return based on current observation o_t and the unperturbed action a_t. This predictor is trained via supervised learning on a dataset collected from Monte-Carlo rollouts. The detailed procedure for data collection and training is presented in Alg. 2, with all notations consistent with the main text.

Algorithm 2: Train a Return Predictor D_ϕ

Input: Task environment $\mathcal{M}_{\text{ENV}}$, expert policy π_{expert}, number of episodes L, discount factor γ_{ENV}, episode horizon T, update interval l_{int}, update epoch Ep, max buffer length M, variance v.
Initialize: Network D_ϕ, Empty buffer $\mathcal{B}$ with max length M.
for $l = 0, 1, \ldots, L-1$ **do**
    Reset environment to get o_0
    Sample $t_l \sim \text{Uniform}\{0, 1, \ldots, T-1\}$
    for $t = 0, 1, \ldots, T-1$ **do**
        Get expert action $a_t \sim \pi_{\text{expert}}(\cdot \mid o_t)$
        if $t == t_l$ **then**
            $\xi \sim \mathcal{N}\left(0, v^2\right)$, $a'_t \leftarrow a_t + \xi$
            Execute a'_t to get o_{t+1} and r_t
        else
            Execute a_t to get o_{t+1} and r_t
    Calculate $J_l \leftarrow \sum_{\tau=0}^{T} \gamma_{\text{ENV}}^{\tau - t_l} r_\tau$
    Add (o_{t_l}, a_{t_l}, J_l) to $\mathcal{B}$
    if $l \,\%\, l_{\text{int}} == 0$ **then**
        for $e = 0, 1, \ldots, Ep-1$ **do**
            Update D_ϕ on $\mathcal{B}$ by minimizing the loss: $\mathcal{L}(\phi) = \frac{1}{L} \sum_{(o,a,J)\in\mathcal{B}} (D_\phi(o,a) - J)^2$
Return: Trained predictor D_ϕ.

We parameterize D_ϕ as a 5-layer MLP with hidden layer sizes of $[256, 512, 1024, 512, 256]$. For our experiments on the `Square` and `Transport` tasks, we use $L = 600$ and $L = 350$, respectively. The full training hyperparameters are provided in Tab. 1, with training curves shown in Fig. 8. Detailed settings for both tasks are available in **Appendix B**.

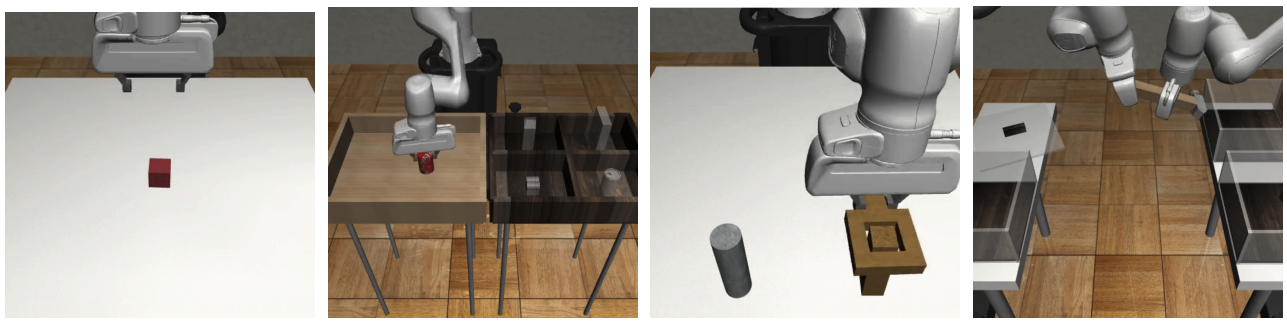


(a) Square (b) Transport

Figure 8: Training loss of the return predictor D_ϕ versus training episodes for the (a) `Square` and (b) `Transport` tasks

Hyperparameter	Value
max buffer size M	100000
number of parallel environments	10
discount factor γ_{ENV}	0.99
noise variance v	0.1
update interval l_{int}	20
update epoch Ep	6
learning rate	0.0003
weight decay	0.0001

Table 1: Hyperparameters for training the return predictor D_ϕ.



(a) Lift (b) Can (c) Square (d) Transport

Figure 9: Four manipulation tasks in Robomimic

B Additional details of simulation experiments

B.1 Environment and Dataset

Environment We evaluate D3P in two benchmarks:

1. **Robomimic** (Mandlekar et al. 2021). We evaluate our method on manipulation tasks from the Robomimic suite, including the simple pick-and-place tasks `Lift` and `Can`, the assembly task `Square`, and the bimanual handover task `Transport`. Fig. 9 demonstrate the four manipulation tasks.
2. **Franka Kitchen** (Gupta et al. 2019). We also use the Franka Kitchen environment, a benchmark for challenging long-horizon, multi-stage manipulation. As illustrated in Fig. 10, the robot need to complete 4 subtasks in sequence: open the microwave, move the kettle, flip the light switch, and slide open the cabinet door. We use two settings in this environmrnt: (1) `Kitchen-complete-v0`, where the policy is pretrained on a dataset of successful demonstrations, and (2) `Kitchen-mixed-v0`, where the pretraining dataset contains contains various subtasks being performed, but the 4 target subtasks are never completed in sequence together.

Figure 10: Franka Kitchen environment. In the environment, the robot need to complete 4 subtasks in sequence: open the microwave, move the kettle, flip the light switch, and slide open the cabinet door.

Benchmark	Task	Obs dim (State)	Obs dim (Pixel)	Act dim $\times T_a$	T	Sparse reward
Franka Kitchen	kitchen-complete-v0	60	-	9×4	280	Yes
	kitchen-mixed-v0	60	-	9×4	280	Yes
Robomimic	Lift	19	-	7×4	300	Yes
	Can	23	-	7×4	300	Yes
	Square	23	-	7×4	400	Yes
	Transport	59	-	14×8	800	Yes
	Square (Pixel)	9	$(3, 96, 96) \times 1$	7×4	400	Yes
	Transport (Pixel)	18	$(3, 96, 96) \times 2$	14×8	800	Yes

Table 2: We detail the task configurations in this table, where"Obs dim (State)" is the dimension of the state observation, "Obs dim (Image)" is the dimension of the pixel observation, "Act dim" is the dimension of a single action, T_a is the action chunck horizon, and T is the episode horizon.

We list the task configurations in Tab. 2. In this table, "Obs dim (State)" indicates the dimension of the state observation, "Obs dim (Image)" is the dimension of the pixel observation, Act dim $\times T_a$ shows the shape of an action chunk, and T is the episode horizon.

All tasks employ a sparse reward. In Robomimic tasks, the agent receives a reward of 0 for every timestep prior to task completion. Upon successful completion, the agent is awarded a +1 reward for all subsequent timesteps until the episode concludes. Therefore, the episodic return directly reflects the agent's quality. A higher return indicates faster task completion, whereas a lower return suggests the task was only finished near the end of the episode. If the task is not completed within the maximum episode length of T steps, the total episodic return is 0. For the Franka Kitchen environment, the agent receives a +1 reward upon the completion of each sub-task. As there are 4 sub-tasks, the maximum possible episodic return is 4.

Dataset We pre-train the policies on the dataset provided by one of our baseline, DPPO (Ren et al. 2024). As acknowledged by the authors of DPPO the dataset includes suboptimal data.

B.2 Baseline

We compare D3P against three representative baselines that cover different paradigms for improving or acclerating diffusion policies.

DPPO Diffusion policy policy optimization (DPPO) (Ren et al. 2024) is a state-of-the-art algorithm for *online fine-tuning* diffusion policies that operate with a fixed number of denoising steps. DPPO models the problem as a two-layer POMDP to leverage the sequential nature of the diffusion denoising process. It directly optimizes the entire denoising chain using an actor-critic framework. At each iteration, the diffusion policy π_θ is updated with the following PPO-style loss function:

$$\mathcal{L}_\theta = \mathbb{E}\left[\min\left(\hat{A}^{\pi_{\theta_{\text{old}}}}(\bar{o}_{\bar{t}}, a_{\bar{t}})\frac{\pi_\theta(\bar{o}_{\bar{t}}, a_{\bar{t}})}{\pi_{\theta_{\text{old}}}(\bar{o}_{\bar{t}}, a_{\bar{t}})}, \hat{A}^{\pi_{\theta_{\text{old}}}}(\bar{o}_{\bar{t}}, a_{\bar{t}})\ \text{clip}\left(\frac{\pi_\theta(\bar{o}_{\bar{t}}, a_{\bar{t}})}{\pi_{\theta_{\text{old}}}(\bar{o}_{\bar{t}}, a_{\bar{t}})}, 1-\epsilon_{\text{clip}}, 1+\epsilon_{\text{clip}}\right)\right)\right], \tag{9}$$

where $\bar{t}(t, i) = tN + (N - i - 1)$ is the time index in the two-layer POMDP, $\bar{o}_{\bar{t}}$ is the observation, and $a_{\bar{t}}$ is the action. The advantage $\hat{A}^{\pi_\theta}$ is estimated as shown in Eq. (10), using a discount factor of $\gamma^i_{\text{DENOISE}}$.

$$\hat{A}^{\pi_\theta}(\bar{o}_{\bar{t}}, a_{\bar{t}}) = \gamma^i_{\text{DENOISE}}\left(J_{\pi_\theta}(\bar{o}_{\bar{t}}, a_{\bar{t}}) - V_\Theta(\bar{o}_{\bar{t}})\right). \tag{10}$$

Notably, DPPO employs a dynamic clipping parameter ϵ_{clip}, whose value is determined by the denoising progress, $t = 1 - \frac{i}{N}$, as defined in Eq. (11).

$$\epsilon_{\text{clip}} = \epsilon_{\text{base}} + (\epsilon_{\text{coef}} - \epsilon_{\text{base}}) \cdot \frac{e^{\epsilon_{\text{rate}}\ t} - 1}{e^{\epsilon_{\text{rate}}} - 1}. \tag{11}$$

Here, ϵ_{base}, ϵ_{coef}, and ϵ_{rate} are hyperparameters. DPPO provides structured online exploration and enhances training stability. In experiments, we use the official hyperparameter settings from the original implementation, detailed in Tab. 3.

Consistency Policy A Consistency Policy (CP) (Prasad et al. 2024) is created by distilling a pretrained diffusion policy, which enforces self-consistency along the teacher model's learned trajectories. This distillation enables CP to sample actions using very few denoising steps. We denote the CP as $g_\theta(o_t, X^i_t, i, s)$, which is conditioned on the current observation o_t, the current action chunk X^i_t, the current noise level i, and a target noise level $s < i$. The distillation loss combines a Denoising Score Matching (DSM) loss and a Consistency Trajectory Model (CTM) loss:

$$\begin{aligned}
\mathcal{L}_{DSM} &= \mathbb{E}_{x_0,\, i}\left[d(x_0, g_\theta(o_t, x_i, i, 0))\right], \\
\mathcal{L}_{CTM} &= \mathbb{E}_{x_0,i,s}\left[d\left(g_\theta\left(o_t, g_\theta(o, X^i_t, i, s), s, 0\right),\ g_\theta\left(o_t, g_\theta(o, X^{i-1}_t, i-1, s), s, 0\right)\right)\right].
\end{aligned} \tag{12}$$

Hyperparameter	Value
Number of parallel environments	40
Condition horizon T_o	1
Reward discount factor γ_{ENV}	0.999
Advantage discount factor γ_{DENOISE}	0.99
GAE λ	0.95
Optimizer	AdamW
Actor learning rate	1e-4
Actor weight decay	0
Critic learning rate	1e-3
Critic weight decay	0
Batch size	10000
Value loss coefficient	0.5
Entropy loss coefficient	0
Clip coefficient base ϵ_{base}	0.001
Clip coefficient ϵ_{coef}	0.01
Clip coefficient rate ϵ_{rate}	3
Max gradient norm	10.0
Rollout steps	400
Update epoch	10
Denoising η	1.0

Table 3: Hyperparameters of DPPO

In these equations, X_t^{i-1} is generated from X_t^i by the teacher policy, and $d(x, y)$ is a pseudo-Huber loss function measuring the distance between x and y, as shown in Eq. (13)

$$d(x, y) = \sqrt{\|x - y\|_2^2 + c^2} - c. \tag{13}$$

During training, the final loss is a weighted sum of $\mathcal{L}_{DSM}$ and $\mathcal{L}_{CTM}$ with coefficients w_{DSM} and w_{CTM}, respectively. For our experiments, we distill CPs from our DPPO-finetuned policies. The hyperparameters for distillation are listed in Tab. 4.

Falcon Falcon (Chen et al. 2025) is a training-free *streaming* method that accelerates diffusion policies through partial denoising. The core insight of Falcon is to leverage the sequential dependency inherent in such tasks. Instead of initiating the denoising process from a standard Gaussian distribution for every action, Falcon reuses a partially denoised action from a latent buffer of historical actions, thereby significantly reducing the required number of sampling steps.

The selection of this action prior is managed by a thresholding mechanism. Falcon first uses the unexecuted action sequence from the previous timestep as a reference. Then, for each partially denoised action a X_τ^i in the latent buffer, Falcon computes a one-step estimation $\hat{X}_t^0$ conditioned on the current observation o_t with Tweedie's formula (Eq. (14)), where ϵ_θ is the noise-

Hyperparameter	Value
Weight of DSM loss w_{DSM}	1.0
Weight of CTM loss w_{CTM}	1.0
Optimizer	AdamW
Batch size	512
Training epoch	1500
Learning rate	1e-4
Weight decay	1e-6
Learning rate scheduler	CosineAnnealingWarmupRestart (Loshchilov and Hutter 2016)
First cycle steps	1500
Warmup steps	100
Minimum learning rate	1e-5

Table 4: Hyperparameters of consistency policy

predicting network defined in Preliminaries, and $\bar{\alpha}_i$ is a set of parameters determinated by a fixed noise schedule.

$$\hat{X}_t^0 = \mathbb{E}[\hat{X}_t^0 | o_t, X_t^i, i] = \frac{X_t^t - \sqrt{1-\bar{\alpha}_i}\epsilon_\theta(o_t, X_t^i, i)}{\sqrt{\bar{\alpha}_i}}. \tag{14}$$

Then we form a candidate set $\mathcal{S}$, consists of actions whose one-step estimations are within a certain distance ϵ_{Falcon} of the reference action. The final prior action X_t^i is then sampled from this set with a probability that favors lower noise levels, modulated by a temperature parameter κ. This thresholding mechanism allows Falcon to find the best action priors.

B.3 Hyperparameters of D3P

The D3P training hyperparameters are provided in Tabs. 5 and 6. The settings for the base policy are based on those from DPPO. In contrast, for the adaptor, we keep most hyperparameters unchanged but specifically tune several key parameters, such as ζ_1, ζ_2 and the reward weight β.

Fine-tuning Base policy π_θ		Training Adaptor K_ω	
Hyperparameter	Value	Hyperparameter	Value
Number of parallel environments	40	Condition horizon T_o	1
Condition horizon T_o	1	Denoising step discount γ_s	0.95
Reward discount factor γ_{ENV}	0.999	reward discount factor γ	0.99
Advantage discount factor γ_{DENOISE}	0.99	GAE λ	0.95
GAE λ	0.95	Optimizer	AdamW
Optimizer	AdamW	Adaptor weight decay	1e-3
Actor learning rate	1e-4	Batch size	40000
Actor weight decay	0	Value loss coefficient	1.0
Critic learning rate	1e-3	Entropy loss coefficient	0.01
Critic weight decay	0	Clip coefficient	0.01
Batch size	10000	Max gradient norm	10.0
Value loss coefficient	0.5	Update epoch	10
Entropy loss coefficient	0	Rollout steps	400
Clip coefficient base ϵ_{base}	0.001	Reward weight α	1.0
Clip coefficient ϵ_{coef}	0.01		
Clip coefficient rate ϵ_{rate}	3		
Max gradient norm	10.0		
Rollout steps	400		
Denoising η	1.0		

Table 5: Shared hyperparameters of D3P

Hyperparameter	Lift	Can	Square (State & Pixel)	Transport (State & Pixel)	kitchen-complete-v0	kitchen-mixed-v0
Update epoch	10	10	10	6	10	10
Adaptor learning rate	1e-4	1e-4	1e-4	3e-5	1e-4	1e-4
Threshold ζ_1	100.0	170.0	210.0	310.0	3.3	3.3
Threshold ζ_2	4.0	4.0	5.0	7.5	5.0	5.0
Reward weight β	0.2	0.2	0.06	0.1	0.4	0.4

Table 6: Task-specific hyperparameters of D3P

B.4 Computation

All experiments are conducted on our server cluster running Ubuntu 22.04. The training environment is built on Python 3.8, with specific dependencies listed in Listing 1. The hardware resources utilized for training are detailed in Tab. 7. As the Robomimic and Franka Kitchen simulation environments are primarily CPU-intensive, our training process consumes significant CPU and memory resources. Each training run is performed using a single GPU.

	Environment	CPU	Memory	GPU
Training	Robomimic (State)	45 Core	128G	RTX 3090 24G
	Robomimic (Pixel)	45 Core	256G	A800 40G
	Franka Kitchen	10 Core	64G	RTX 3090 24G
Evaluation	All environment	16 Core	32G	RTX 3090 24G

Table 7: Computing resources for training D3P

Listing 1: Dependencies for training D3P

```
av==12.3.0
einops==0.8.0
gdown==5.2.0
gym==0.22.0
hydra-core==1.3.2
imageio==2.35.1
matplotlib==3.7.5
omegaconf==2.3.0
pretty_errors==1.2.25
torch==2.4.0
tqdm==4.66.5
wandb==0.17.7

# for robomimic environment
cython<3
d4rl
patchelf
mujoco==3.1.6
robomimic
robosuite @ v1.4.1

# for franka kitchen
cython<3
d4rl
dm_control==1.0.16
mujoco==3.1.6
patchelf
```

C Additional Experiment Results

C.1 Performance Comparison

Fig. 11 presents the success rate and return versus Number of Function Evaluations (NFE) for all methods. In these plots, the upper-left corner represents the ideal trade-off: high performance with low inference cost. With an equivalent training budget, D3P consistently matches or surpasses the fixed 10-step diffusion policy baseline across both metrics.

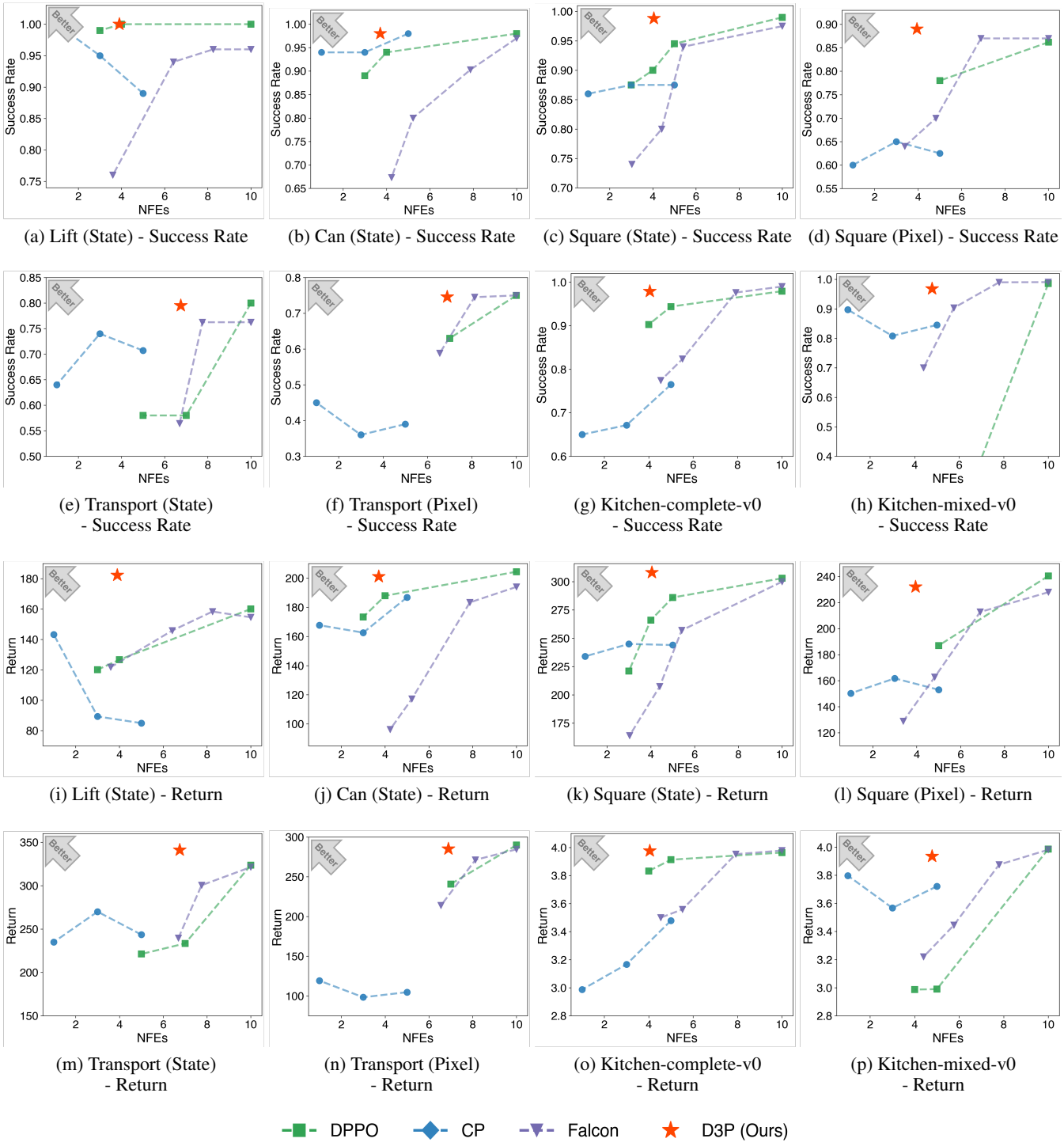
The performance of the DPPO baseline correlates directly with its inference cost, dropping significantly as NFE is reduced. The two acceleration baselines, CP and Falcon, exhibit distinct features. CP excels at few-step inference, outperforming Falcon on tasks like `Lift`, `Can`, and `Square` (State). In contrast, Falcon, a training-free accelerator, achieves a better performance-efficiency trade-off on more complex tasks such as `Square` (Pixel) and `Transport` (Pixel).

We present training curves in Fig. 12, and summarize quantitative results in Tab. 8. Across eight tasks, D3P achieves an average success rate of 0.917, nearly matching the 0.918 of the 10-step DPPO baseline. Meanwhile, D3P achieves a 2.2× mean speed-up. We calculate this per-task acceleration ratio, r_{acc}, using Eq. (15).

$$r_{acc} = \frac{\mathbb{E}_{\text{all episodes}}\left[\sum_{t=0}^{T} \text{stp}_{\text{DPPO},t}\right]}{\mathbb{E}_{\text{all episodes}}\left[\sum_{t=0}^{T} \text{stp}_{\text{D3P},t}\right]} \tag{15}$$

Task	DPPO			Falcon			CP			D3P (Ours)			
	NFE	SR	Return	NFE	SR	Return	NFE	SR	Return	NFE	SR	Return	Acc. Ratio
Lift (State)	10.0	1.00±0.00	160.1±4.4	10.0±0.0	0.96±0.00	154.6±4.2	5.00	0.89±0.01	84.9±2.4	3.90±0.12	1.00±0.01	182.4±23.2	2.56
	4.00	1.00±0.00	126.7±2.7	8.25±0.25	0.96±0.01	158.4±3.5	3.00	0.95±0.01	89.3±2.8				
	3.00	0.99±0.01	120.1±2.7	6.41±0.21	0.94±0.01	145.9±5.9	1.00	1.00±0.00	143.3±2.4				
				3.60±0.44	0.76±0.03	121.8±9.2							
Can (State)	10.0	0.98±0.00	204.4±3.4	10.0±0.0	0.97±0.01	194.0±5.8	5.00	0.98±0.01	186.8±3.7	3.71±0.37	0.98±0.02	201.1±8.0	2.70
	4.00	0.94±0.01	187.9±3.5	7.87±0.12	0.90±0.02	183.3±5.8	3.00	0.94±0.01	162.6±3.3				
	3.00	0.89±0.01	173.3±3.7	5.22±0.35	0.80±0.03	117.1±6.4	1.00	0.94±0.00	167.7±3.0				
				4.23±0.28	0.67±0.03	96.3±8.1							
Square (State)	10.0	0.99±0.00	303.1±2.2	10.0±0.0	0.98±0.00	300.4±6.3	5.0	0.88±0.01	244.3±8.4	4.05±0.20	0.99±0.01	308.7±5.5	2.47
	5.00	0.95±0.03	285.9±10.0	5.41±0.22	0.94±0.04	256.7±7.3	3.0	0.88±0.02	244.6±2.6				
	4.00	0.90±0.04	265.8±12.3	4.41±0.30	0.80±0.04	207.4±8.8	1.0	0.86±0.01	234.0±2.4				
	3.00	0.88±0.04	221.2±14.9	3.03±0.34	0.74±0.06	164.1±8.6							
Transport (State)	10.0	0.80±0.05	323.6±24.3	10.0±0.0	0.76±0.04	321.4±28.4	5.00	0.71±0.06	243.5±24.8	6.75±0.14	0.80±0.07	341.0±30.8	1.48
	7.00	0.58±0.04	233.4±25.0	7.75±0.50	0.76±0.04	300.4±31.1	3.00	0.74±0.03	270.0±19.9				
	5.00	0.58±0.05	221.2±22.3	6.69±0.89	0.56±0.06	239.6±31.5	1.00	0.64±0.03	235.0±22.8				
Square (Image)	10.0	0.86±0.05	240.5±14.2	10.0±0.0	0.87±0.04	228.2±14.9	5.00	0.63±0.06	153.1±7.4	3.95±0.74	0.89±0.03	232.0±3.2	2.53
	5.00	0.78±0.07	187.0±20.9	6.89±0.39	0.87±0.06	212.8±21.4	3.00	0.65±0.03	161.8±9.6				
				4.82±0.51	0.70±0.06	162.8±28.1	1.00	0.60±0.03	150.3±4.8				
				3.39±0.36	0.64±0.07	128.9±8.4							
Transport (Image)	10.0	0.75±0.02	289.9±13.5	10.0±0.0	0.75±0.02	284.6±9.7	5.00	0.39±0.07	104.8±8.4	6.89±0.52	0.75±0.03	285.0±9.8	1.45
	7.00	0.63±0.03	240.9±12.9	8.12±0.16	0.74±0.04	271.3±10.2	3.00	0.36±0.08	98.4±9.3				
				6.55±0.71	0.59±0.09	213.9±16.1	1.00	0.45±0.04	119.3±7.8				
Kitchen (Complete)	10.0	0.98±0.00	3.96±0.02	10.0±0.0	0.99±0.01	3.98±0.03	5.0	0.76±0.03	3.48±0.09	4.15±0.37	0.98±0.01	3.98±0.02	2.41
	5.00	0.94±0.01	3.91±0.03	7.92±0.18	0.98±0.01	3.95±0.07	3.0	0.67±0.01	3.17±0.02				
	4.00	0.90±0.01	3.83±0.03	5.52±0.39	0.82±0.05	3.56±0.08	1.0	0.65±0.01	2.99±0.02				
				4.54±0.31	0.77±0.05	3.50±0.07							
Kitchen (Mixed)	10.0	0.99±0.00	3.99±0.00	10.0±0.0	0.99±0.01	3.98±0.05	5.0	0.85±0.06	3.72±0.10	4.78±0.02	0.97±0.02	3.93±0.05	2.09
	5.00	0.0±0.0	2.99±0.01	7.79±0.15	0.99±0.01	3.88±0.04	3.0	0.81±0.06	3.57±0.09				
	4.00	0.0±0.0	2.99±0.00	5.76±0.21	0.90±0.05	3.44±0.09	1.0	0.90±0.05	3.80±0.07				
				4.40±0.37	0.70±0.07	3.22±0.10							

Table 8: Detailed results of all methods, fomulated as mean ± std. All results are averaged over 5 seeds.



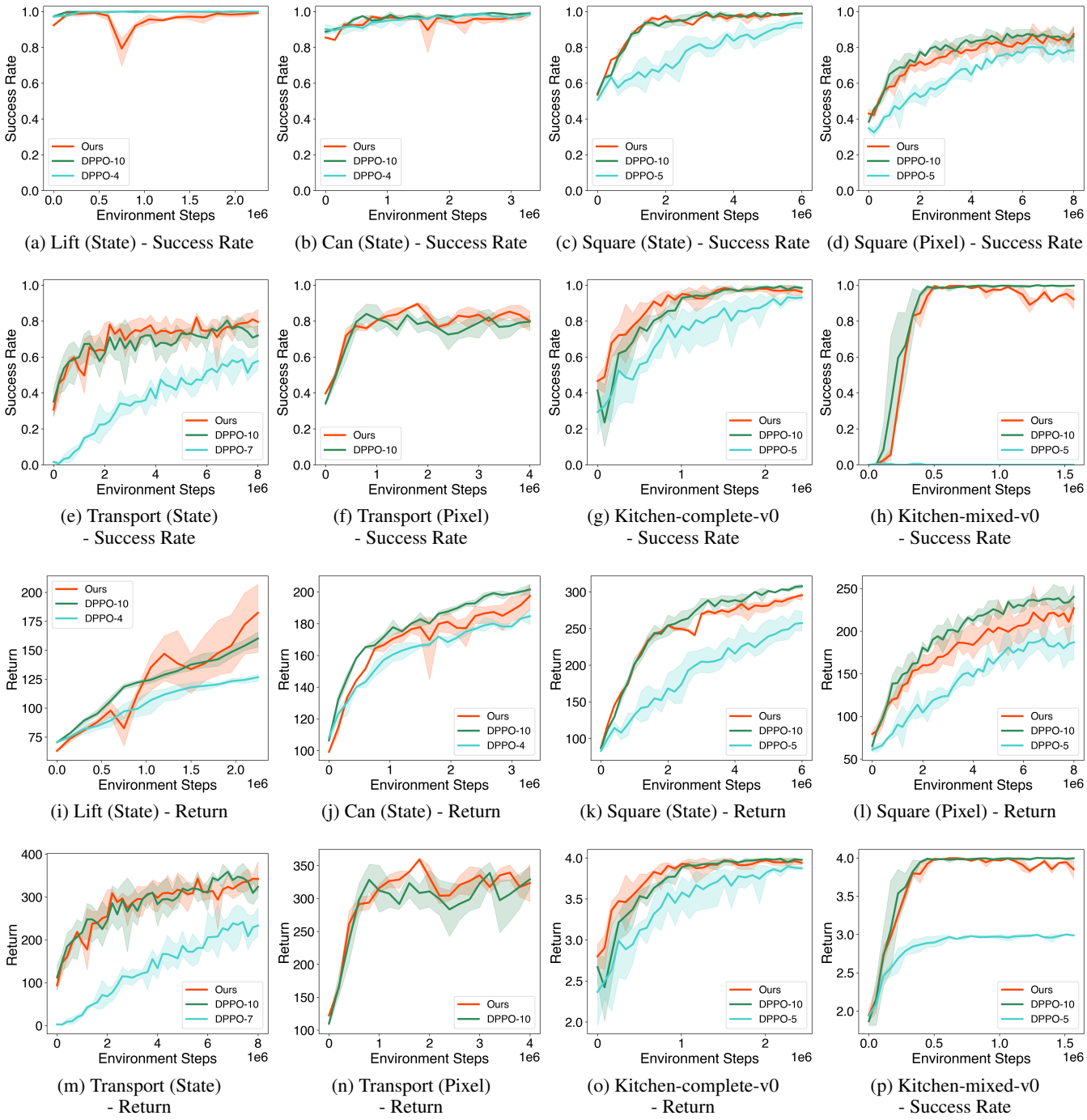
(a) Lift (State) - Success Rate (b) Can (State) - Success Rate (c) Square (State) - Success Rate (d) Square (Pixel) - Success Rate

(e) Transport (State) - Success Rate (f) Transport (Pixel) - Success Rate (g) Kitchen-complete-v0 - Success Rate (h) Kitchen-mixed-v0 - Success Rate

(i) Lift (State) - Return (j) Can (State) - Return (k) Square (State) - Return (l) Square (Pixel) - Return

(m) Transport (State) - Return (n) Transport (Pixel) - Return (o) Kitchen-complete-v0 - Return (p) Kitchen-mixed-v0 - Return

Figure 11: We plot success rate and episodic return against the NFE. An ideal method occupies the upper-left corner, representing high performance at a low inference cost. D3P matches or surpasses the peak performance of the 10-step DPPO baseline while achieving an average 2.2× speed-up. All results are averaged over 5 seeds.



(a) Lift (State) - Success Rate (b) Can (State) - Success Rate (c) Square (State) - Success Rate (d) Square (Pixel) - Success Rate

(e) Transport (State) - Success Rate (f) Transport (Pixel) - Success Rate (g) Kitchen-complete-v0 - Success Rate (h) Kitchen-mixed-v0 - Success Rate

(i) Lift (State) - Return (j) Can (State) - Return (k) Square (State) - Return (l) Square (Pixel) - Return

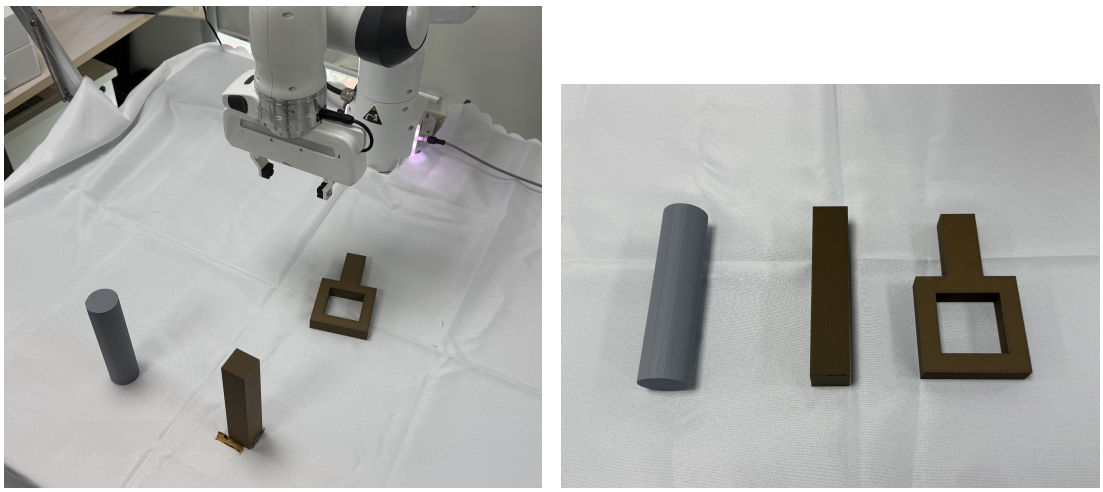
(m) Transport (State) - Return (n) Transport (Pixel) - Return (o) Kitchen-complete-v0 - Return (p) Kitchen-mixed-v0 - Success Rate

Figure 12: Training curves of eight tasks. Each curve is averaged over 5 random seeds.

D Real-world Deployment

We deploy D3P on a Franka robot arm to complete the `Square` task.

Setup We define the task environment as illustrated in Fig. 13a. In the `Square` task, the agent is required to successfully grasp the handle of the square nut and precisely mate it with the corresponding square peg. We use a Franka robot arm for executing, and a consumer-grade desktop (i7-12900K CPU, RTX 2080 GPU) for computing. We fabricated the physical objects required for the task using 3D printing, ensuring that their color, shape, and dimensions are consistent with simulated objects, as shown in Fig. 13b. For this task, D3P utilizes images and joint positions as input. The RGB images are captured by an Intel RealSense D435i camera positioned 0.92m in front of and 0.53m above the robot's base, providing a 45 degree downward viewing angle. Fig. 14 shows the progress of this task, and we include a video in the supplementary materials that showcases the agent executing this task.



(a) `Sqaure` Task Setting (b) We use 3D print to manufacture the objects

Figure 13: The `Square` task involves: (1) grasping the handle of the square, (2) inserting the square onto corresponding peg.

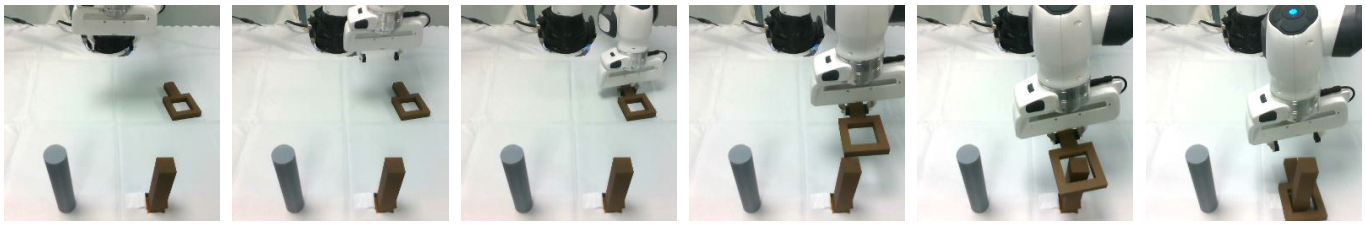
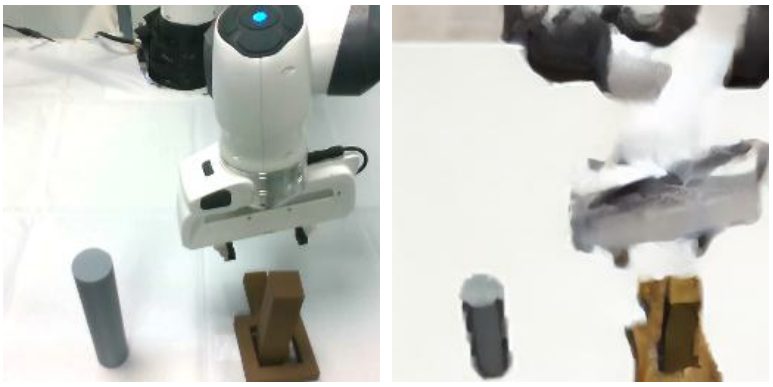


Figure 14: Demonstration of the `Square` task

Sim-to-real transfer To achieve sim-to-real transfer without real-world data, we employ a latent diffusion model (LDM) (Rombach et al. 2021) to convert real-world images into a style that approximates the simulated domain, as illustrated in Fig. 15. The process begins with a pretrained Variational Autoencoder (VAE) that encodes the input image into a compact latent feature. Subsequently, a diffusion model operates on this latent feature to perform the translation. Finally, the VAE's decoder reconstructs the processed feature into the converted output image.

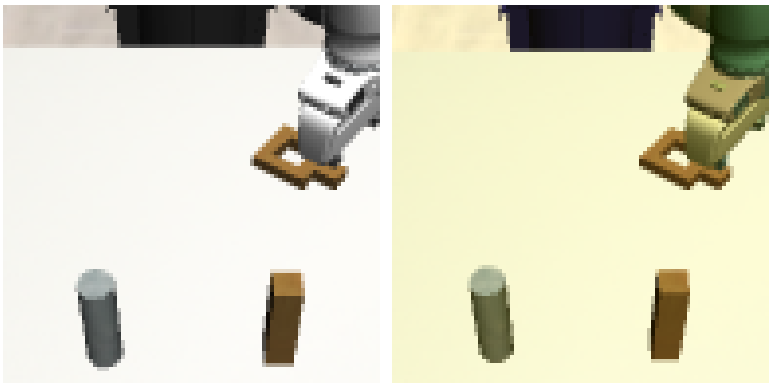
We train the LDM with paires of images collected in simulation by domain radomization. As illustrated in Fig. 16, we employ domain randomization in the simulation by varying the object materials and lighting conditions, from which we collect paired images of the canonical and randomized scenes.

Finally, to bridge the gap in camera parameters between the simulated and real-world environments, we introduce a curriculum learning strategy during RL training. This curriculum systematically adjusts the camera's intrinsic and extrinsic parameters, progressively transitioning them from the initial simulation settings to the physical hardware setting.



(a) Real-world Image (b) Converted Image

Figure 15: We use LDM to convert real-world images into the simulation style



(a) Canonical Image (b) Randomized Image

Figure 16: Images pair for training LDM