Steering Your Diffusion Policy with Latent Space Reinforcement Learning

Andrew Wagenmaker[*1] **Mitsuhiko Nakamoto**[*1] **Yunchu Zhang**[*2] **Seohong Park**[1]

Waleed Yagoub[2] **Anusha Nagabandi**[3] **Abhishek Gupta**[*2] **Sergey Levine**[*1]

Abstract: Robotic control policies learned from human demonstrations have achieved impressive results in many real-world applications. However, in scenarios where initial performance is not satisfactory, as is often the case in novel open-world settings, such behavioral cloning (BC)-learned policies typically require collecting additional human demonstrations to further improve their behavior—an expensive and time-consuming process. In contrast, reinforcement learning (RL) holds the promise of enabling autonomous online policy improvement, but often falls short of achieving this due to the large number of samples it typically requires. In this work we take steps towards enabling fast autonomous adaptation of BC-trained policies via efficient real-world RL. Focusing in particular on diffusion policies—a state-of-the-art BC methodology—we propose *diffusion steering via reinforcement learning* (DSRL): adapting the BC policy by running RL over its latent-noise space. We show that DSRL is highly sample efficient, requires only black-box access to the BC policy, and enables effective real-world autonomous policy improvement. Furthermore, DSRL avoids many of the challenges associated with finetuning diffusion policies, obviating the need to modify the weights of the base policy at all. We demonstrate DSRL on simulated benchmarks, real-world robotic tasks, and for adapting pretrained generalist policies, illustrating its sample efficiency and effective performance at real-world policy improvement. Website: https://diffusion-steering.github.io.

1 Introduction

Robotic learning methods have achieved impressive results on many real-world applications by shifting the burden of controller design from human engineering to data-driven end-to-end learning [1, 2, 3, 4, 5, 6]. Learning policies from expert data via supervised learning, often referred to as behavioral cloning (BC), has been particularly effective in translating large, offline demonstration datasets into robotic control policies. Such methods are appealing because of their simplicity, scalability, and applicability across a broad range of generative model parameterizations, ranging from diffusion or flow models [7, 6] to autoregressive transformers [8]. However, since these methods require expert data to learn, they are unable to directly utilize experience collected during deployment. In settings where the BC policy does not solve the goal task perfectly, as is common in many open-world deployments, utilizing this experience to further refine and improve the policy is highly desirable.

Reinforcement learning (RL) can, in principle, achieve this type of policy improvement. While learning policies via RL from scratch is often too inefficient for practical use, RL has served as an effective *finetuning* tool in domains such as language modeling, showing impressive results adapting models pretrained on large corpora of offline "demonstrations" [9, 10, 11, 12]. However, this finetuning process is still expensive in terms of both compute and time (i.e., samples), making it difficult to directly apply to the adaptation of real-world robotic control policies. Given this, developing meth-

[*]Core contributor. Please see appendix for individual contributions. [1] UC Berkeley, [2] University of Washington, [3] Amazon. Correspondence to: Andrew Wagenmaker {ajwagen@berkeley.edu}.

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

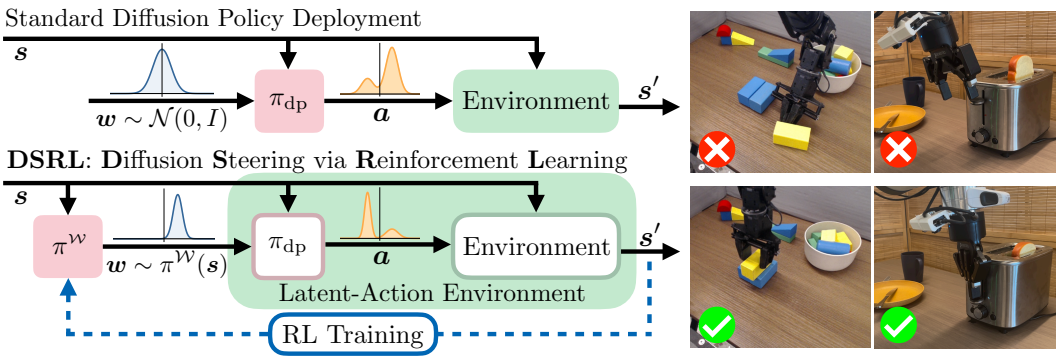


Figure 1: Overview of our proposed approach, **D**iffusion **S**teering via **R**einforcement **L**earning (DSRL). Standard deployment of a BC-trained diffusion policy π_{dp} first samples noise $\boldsymbol{w} \sim \mathcal{N}(0, I)$ that is then denoised through the reverse diffusion process to produce an action $\boldsymbol{a}$. We propose modifying the initial distribution of $\boldsymbol{w}$ with an RL-trained latent-noise space policy $\pi^{\mathcal{W}}$, that instead of choosing $\boldsymbol{w} \sim \mathcal{N}(0, I)$, chooses $\boldsymbol{w}$ to *steer* the distribution of actions produced by π_{dp} in a desirable way, enabling highly sample efficient real-world adaptation of robot policies.

ods for RL-based finetuning of pretrained BC policies that are simple, stable, broadly applicable, and efficient enough to apply to real-world robot learning has remained a persistent challenge.

In this work, we take steps towards addressing this challenge, and seek to understand how we can utilize novel experience to adapt pretrained robotic control policies in a simple and efficient way. Focusing in particular on BC policies parameterized as diffusion models [13, 14, 15], which has become the de-facto standard policy parameterization in robotics [7, 16, 5, 17, 18, 19, 4, 6, 20], we propose a lightweight approach that overcomes many of the challenges faced by existing RL-based finetuning methods. In particular, rather than modifying the *weights* of a pretrained diffusion-based BC policy, we instead modify its *sampling process*, altering the input noise distribution the diffusion model utilizes to generate samples. We show that, by running RL over this input noise distribution, we can quickly learn to steer the policy's behavior as new experience is collected online, and that this approach is stable and highly sample efficient, while being surprisingly expressive. Our approach relies only on forward passes of the denoising process and is fully black-box, not requiring direct access to the policy's weights at all. This allows for the adaptation of policies through pure API access and eliminates the need for potentially unstable back-propagation through the diffusion chain, a primary challenge in applying standard RL finetuning to diffusion policies [21, 22, 23]. Furthermore, our approach avoids the challenges associated with finetuning the large, complex policy architectures often utilized in modern robot learning settings, and instead only requires training much smaller latent-space policies. See Figure 1 for an overview of our approach.

Our main contribution is to formalize this *diffusion steering* process, which we refer to as *diffusion steering via reinforcement learning* (DSRL), and provide a practical instantiation that can be used for adapting robot policies in the real world. While we argue that virtually any RL algorithm can be utilized for diffusion steering, we develop an actor-critic-based procedure that takes advantage of structure in the diffusion policy to enable highly sample-efficient learning from both online and offline data. We empirically demonstrate the effectiveness of DSRL in both simulation and on real-world robotic embodiments, demonstrating state-of-the-art performance in adapting behavior both from online interaction and offline data. We show that our approach also leads to sample-efficient real-world refinement of robot policies, in some cases improving success from 20% to 90% with less than 50 episodes of online interaction, and enables online improvement of existing state-of-the-art generalist robot policies, in particular π_0 [6], in real-world deployment.

2 Related Work

Behavioral cloning and diffusion models in control & robotics. BC methods have seen widespread use throughout the control and robotics communities [24, 25, 26, 27, 28, 29]. While we focus primarily on diffusion-based BC, other approaches—for example, autoregressive next-

token prediction—have found success as well [1, 30, 31, 3, 2, 32, 33, 8]. Diffusion-based BC has led to state-of-the-art results in real-world robotic domains ranging from single-task applications [7, 16, 5, 17, 18, 19] to "generalist" policies [4, 6, 20]. Diffusion models have also been applied to control settings in a *model-based* fashion, enabling planning-based control [34, 35, 36, 37, 38, 39, 40, 41]. In contrast to these works, our focus is not on developing better methodology for training diffusion policies, but on adapting already-trained diffusion policies as new experience is collected.

Reinforcement learning with diffusion. Beyond the pure BC setting, many works have considered applying diffusion models in RL settings where the learner has a reward signal they wish to maximize. In the purely offline setting, approaches include training a diffusion policy on an objective weighted by the value of an action [42, 43, 44, 45], optimizing a diffusion policy directly to maximize a reward [46, 47, 48, 49, 50, 23], rejection sampling of the actions produced by the diffusion policy [51, 52, 53, 54], in addition to several other approaches [55, 56, 57, 58, 59]. In the online setting, past work has applied PPO [60] to finetune the first few steps of a diffusion policy's denoising process [21], trained the diffusion policy by matching the score of the Q-function [61], or applied iterative BC to clone actions that lead to large Q-function values [62, 63, 22]. Other works seek to post-process the outputs of a pretrained diffusion policy by learning an additive residual policy with RL [64, 65]. Compared to these works, we take a different approach and, instead of modifying the diffusion policy's weights or post-processing the actions, adjust the sampling process by modifying the input noise distribution.

Noise optimization in generative modeling. Our proposed approach optimizes the noise sampled at the start of the diffusion policy's denoising process. In image diffusion, it has been shown that optimizing the initial noise of the diffusion process to maximize some metric on image quality can lead to substantial gains in performance [66, 67, 68, 69, 70, 71] or mimic the effects of diffusion guidance [72]. In the control domain, [73] trains a *normalizing flow*-based policy [74] on offline data, then runs RL over the noise space of this policy to improve its performance online. The key difference between this work and ours is that we utilize a diffusion or flow matching model rather than a normalizing flow. Normalizing flows are invertible by construction, which ensures that the agent loses no expressivity over actions when optimizing the noise. As this property is not guaranteed to hold for diffusion models, it is much less clear that optimizing over the noise space of a diffusion policy is expressive enough for effective online adaptation. More importantly, diffusion models have seen much greater success in practice—while there are many examples successfully applying diffusion policies to robotic control, normalizing flows have been shown to perform poorly in such settings [75, 2]. Concurrent to this work, [76] propose optimizing the input noise distribution of a diffusion planner—a diffusion model trained to generate plausible trajectories—and using this to improve planning-based control. While their proposal of optimizing the input noise is similar to ours, they consider only offline learning, and their use of a diffusion planner instead of a diffusion policy differs from our setting.

3 Preliminaries

Markov decision processes. A Markov Decision Process (MDP) is denoted by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, p_0, r, \gamma)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ a set of actions (which we assume to be a subset of $\mathbb{R}^d$), $P : \mathcal{S} \times \mathcal{A} \to \triangle_{\mathcal{S}}$ a transition kernel, $p_0 \in \triangle_{\mathcal{S}}$ the initial state distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ a reward function, and $\gamma \in [0, 1]$ the discount factor. An episode begins by drawing state $\boldsymbol{s}_0 \sim p_0$, and proceeds with, at state $\boldsymbol{s}$, the agent choosing action $\boldsymbol{a}$, receiving reward $r(\boldsymbol{s}, \boldsymbol{a})$, and transitioning to $\boldsymbol{s}' \sim P(\cdot \mid \boldsymbol{s}, \boldsymbol{a})$. A policy $\pi : \mathcal{S} \to \triangle_{\mathcal{A}}$ denotes a mapping from states to actions. The Q-function for policy π, $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$, denotes the expected discounted return of policy π from taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$, $Q^\pi(\boldsymbol{s}, \boldsymbol{a}) := \mathbb{E}^\pi[\sum_{t \geq 0} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \mid \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}]$. The typical objective in RL is to find a policy that achieves maximum expected discounted return.

Diffusion models and flow matching. Diffusion models are generative models that aim to transform an easy-to-sample-from distribution into a more complex goal distribution $q(\boldsymbol{x}_0)$ [13, 14]. While various instantiations exist, denoising diffusion [14, 15] has emerged as perhaps the most popular. Denoising diffusion models this transformation process with a distribution $p_\theta(\boldsymbol{x}_0)$ parameterized

as $p_\theta(\boldsymbol{x}_0) = \int p_\theta(\boldsymbol{x}_{0:T})\mathrm{d}\boldsymbol{x}_{1:T}$, for $p_\theta(\boldsymbol{x}_{0:T}) = p_\theta(\boldsymbol{x}_T) \cdot \prod_{t=1}^{T} p_\theta^{(t)}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$, and typically takes $p_\theta^{(t)}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ to be a normal distribution. The *forward diffusion process* adds noise to samples $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$, and trains a network $\boldsymbol{\epsilon}_\theta^{(t)}$ to denoise these samples. The *reverse diffusion process*—the denoising or sampling process—begins by sampling $\boldsymbol{x}_T \sim \mathcal{N}(0, I)$, and then applies the update

$$\boldsymbol{x}_{t-1} = \alpha_t(\boldsymbol{x}_t - \beta_t \boldsymbol{\epsilon}_\theta^{(t)}(\boldsymbol{x}_t)) + \sigma_t \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(0, I), \tag{1}$$

for α_t, β_t, and σ_t some coefficients. This ultimately produces a "denoised" sample $\boldsymbol{x}_0$ distributed approximately as $q(\boldsymbol{x}_0)$. The reverse diffusion process can be run with both $\sigma_t > 0$ (typically referred to as DDPM sampling [14]) or $\sigma_t = 0$ (DDIM sampling [15]). Notably, DDIM sampling makes the reverse process *deterministic* given the initial noise sample $\boldsymbol{x}_T$ and, furthermore, can often lead to more efficient generation by reducing the number of required denoising steps [15], a property especially useful in real-world robotic deployment where it is critical that actions are generated at a high enough control frequency [7].

Beyond denoising diffusion, *flow matching* is a similar generative modeling technique—also based on an iterative denoising procedure—that has demonstrated impressive results across a range of domains [77, 78, 79, 6, 23]. While a full discussion of flow matching is beyond the scope of this work (see e.g. [80]) we highlight that, similar to denoising diffusion, flow matching operates by first sampling noise from some tractable distribution, and then processes this noise through a (by construction) deterministic process to produce a sample from a goal distribution.

Behavioral cloning with diffusion and flow policies. BC aims to learn a policy that models the state-conditional distribution of actions from a dataset of demonstrations $\mathfrak{D} = \{(\boldsymbol{s}^i, \boldsymbol{a}^i)\}_{i=1}^N$. This is simply a generative modeling problem and diffusion models or flow matching can be applied directly, with $\boldsymbol{x}_0 \leftarrow \boldsymbol{a}^i$ and the BC policy, π_{dp}, the entire denoising process, (1). As noted, for fixed input noise $\boldsymbol{x}_T \leftarrow \boldsymbol{w}$, a DDIM or flow-based policy is a deterministic process. Given this, for a diffusion policy π_{dp} utilizing DDIM sampling or a flow-based policy, we define $\pi_{\mathrm{dp}}^{\mathcal{W}} : \mathcal{S} \times \mathcal{W} \to \mathcal{A}$ the deterministic output of the denoising process at state $\boldsymbol{s}$ when input noise $\boldsymbol{w}$ initializes the denoising process, and denote by $\mathcal{W} := \mathbb{R}^d$ the latent-noise space.

Problem setting. In this work we assume we are given a pretrained diffusion policy π_{dp}, and our goal is to adapt its behavior to maximize some reward r in environment $\mathcal{M}$, the standard objective of RL. While we do not make explicit assumptions on the behavior of π_{dp}, for our approach to be effective π_{dp} must be "steerable" in a sense that will become clear (please see Section B for further discussion of when π_{dp} is steerable). We consider both the online setting—where the learner can interact with $\mathcal{M}$—as well as the offline setting—where the learner is given a dataset of transitions from $\mathcal{M}$. We assume we may select any $\boldsymbol{s} \in \mathcal{S}$ and $\boldsymbol{w} \in \mathcal{W}$ and observe $\pi_{\mathrm{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w})$, but do not assume any other access to $\pi_{\mathrm{dp}}^{\mathcal{W}}$ (for example, its weights or intermediate steps in its denoising process).

4 Diffusion Steering via Reinforcement Learning

In typical operation of a diffusion policy, $\boldsymbol{w}$ is sampled from a standard Gaussian, $\boldsymbol{w} \sim \mathcal{N}(0, I)$, and processed through $\pi_{\mathrm{dp}}^{\mathcal{W}}$ to produce an output action. $\pi_{\mathrm{dp}}^{\mathcal{W}}$ is trained so that the resulting action distribution, $\boldsymbol{a} \sim \pi_{\mathrm{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w})$ for $\boldsymbol{w} \sim \mathcal{N}(0, I)$, matches the action distribution of the demonstration data at $\boldsymbol{s}$. At deployment, this only holds true, however, if $\boldsymbol{w} \sim \mathcal{N}(0, I)$—if $\boldsymbol{w}$ is chosen some other way, π_{dp} will not necessarily produce actions that match the distribution of the demonstrator. This suggests a straightforward way to modify the behavior of π_{dp} is to modify the distribution of $\boldsymbol{w}$, and instead of $\boldsymbol{w} \sim \mathcal{N}(0, I)$, choose $\boldsymbol{w}$ so that the resulting action $\boldsymbol{a} \leftarrow \pi_{\mathrm{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w})$ leads to a desired result. This observation forms the backbone of our proposed approach, *diffusion steering*: rather than modifying the weights of the diffusion policy or postprocessing its output, diffusion steering simply "steers" the diffusion policy to produce desired actions by altering its input noise distribution.

4.1 Diffusion Steering as Latent-Noise Space Policy Optimization

While selecting $\boldsymbol{w}$ may give us some control over the output of π_{dp}, it remains to determine *which* $\boldsymbol{w}$ we should choose. To answer this, we propose reinterpreting the role of $\pi_{\mathrm{dp}}^{\mathcal{W}}$ from a *policy* to an *action space transformation*. At any state $\boldsymbol{s}$, we can produce an action $\boldsymbol{a} \in \mathcal{A}$ by selecting $\boldsymbol{w} \in \mathcal{W}$

and setting $\boldsymbol{a} \leftarrow \pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w})$. We can think of this mapping as transforming the action space of $\mathcal{M}$ from the original action space $\mathcal{A}$ to the latent-noise space $\mathcal{W}$. In particular, defining

$$P^{\mathcal{W}}(\cdot \mid \boldsymbol{s}, \boldsymbol{w}) := P(\cdot \mid \boldsymbol{s}, \pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w})) \quad \text{and} \quad r^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w}) := r(\boldsymbol{s}, \pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w})),$$

we see that $\mathcal{M}^{\mathcal{W}} := (\mathcal{S}, \mathcal{W}, P^{\mathcal{W}}, p_0, r^{\mathcal{W}}, \gamma)$ forms a transformed version of our original MDP, which we refer to as the *latent-action MDP*. Notably, we can treat interaction with $\mathcal{M}^{\mathcal{W}}$ just as we would treat interaction with $\mathcal{M}$, completely black-boxing the contribution of $\pi_{\text{dp}}^{\mathcal{W}}$—we now take actions in $\mathcal{W}$, and filter them through $\pi_{\text{dp}}^{\mathcal{W}}$ to obtain some action in $\mathcal{A}$ which we play in $\mathcal{M}$, but this filtering process is completely encapsulated in the transformed environment. Given this, we may consider $\pi_{\text{dp}}^{\mathcal{W}}$ simply part of the environment, and choosing $\boldsymbol{w}$ reduces to a standard policy optimization problem over $\mathcal{M}^{\mathcal{W}}$, allowing us to apply virtually *any* RL algorithm to learn a latent-noise space policy. We refer to this procedure—adapting the behavior of π_{dp} by running RL over its latent-noise space—as **D**iffusion **S**teering via **R**einforcement **L**earning (DSRL).

Computational efficiency of DSRL. Typical policy optimization procedures require back-propagating through a policy to compute a gradient direction in which to update the parameters. Such methods are challenging to apply to diffusion policies, however, as the diffusion policy generates actions via multi-step denoising (1), and back-propagation through this entire process is computationally intensive and numerically unstable [21, 22, 23]. DSRL avoids this issue entirely by lifting policy optimization from the diffusion policy itself to a secondary policy operating in the diffusion policy's latent-noise space. This latent-space policy can be parameterized in whatever form is most convenient, and only requires forward passes of the diffusion policy's denoising process to optimize. This also allows DSRL to circumvent the need to optimize large diffusion policies common in modern applications, yielding further computational improvements over standard finetuning procedures, and reduces the model access required by DSRL to only choosing $\boldsymbol{w} \in \mathcal{W}$ and computing $\boldsymbol{a} \leftarrow \pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w})$, rather than requiring access to the weights of $\pi_{\text{dp}}^{\mathcal{W}}$ or intermediate steps within the denoising process. Such weak access could enable, for example, adaptation of proprietary models where the base weights are not publicly available, or adaptation over API access.

4.2 Efficient DSRL with Noise Aliasing

While in principle DSRL can be instantiated with virtually any RL algorithm, in this section we introduce an approach which makes particular use of the diffusion policy's structure to increase sample efficiency. Our approach builds off standard actor-critic algorithms [81, 82, 83] which, in our setting, would require fitting a critic, $Q : \mathcal{S} \times \mathcal{W} \to \mathbb{R}$, that quantifies the value of latent-noise actions, and an actor, $\pi : \mathcal{S} \to \triangle_{\mathcal{W}}$, which maps states to high value latent-noise actions. Such an approach can be applied in online learning—where we generate latent-noise actions $\boldsymbol{w}$ ourself—but fails to apply in standard offline learning settings, where we are given a dataset of transitions $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$ labeled with actions in the *original* action space, rather than the latent-noise action space, as the above actor-critic procedure requires. A standard application of an actor-critic algorithm does not, furthermore, fully exploit the structure of the diffusion policy. There may exist $\boldsymbol{w}' \neq \boldsymbol{w}$ such that $\pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w}) \approx \pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w}')$—indeed, we would expect a diffusion policy to exhibit this behavior in settings where the demonstrator has a relatively narrow action distribution, causing many $\boldsymbol{w}$ to map to the same $\boldsymbol{a}$. As we assume sampling access to $\pi_{\text{dp}}^{\mathcal{W}}$, we can in principle infer that $\pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w}) \approx \pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w}')$, which could allow us, for example, to determine the behavior of $\mathcal{M}^{\mathcal{W}}$ when taking action $\boldsymbol{w}'$ without ever actually taking action $\boldsymbol{w}'$. Exploiting this aliasing behavior promises to reduce the amount of exploration over the latent-noise space required to learn in $\mathcal{M}^{\mathcal{W}}$, yet standard actor-critic algorithms fail to take advantage of this. To address these issues, we propose an instantiation of DSRL, *noise-aliased* DSRL, which we state in Algorithm 1.

DSRL-NA maintains two critics, $Q^{\mathcal{A}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $Q^{\mathcal{W}} : \mathcal{S} \times \mathcal{W} \to \mathbb{R}$. $Q^{\mathcal{A}}$ is trained via temporal-difference learning on the original action space, and is therefore able to incorporate standard offline data with $\mathcal{A}$-actions. $Q^{\mathcal{W}}$ operates on the latent-noise action space and distills $Q^{\mathcal{A}}$ by sampling $\boldsymbol{w} \sim \mathcal{N}(0, I)$ and generating the corresponding $\mathcal{A}$-action, $\boldsymbol{a} \leftarrow \pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w})$. This allows $Q^{\mathcal{W}}$ to internalize the dynamics information encoded within $Q^{\mathcal{A}}$ and, furthermore, map the value of latent-noise actions that have never been taken to the value of corresponding $\mathcal{A}$-actions that have been taken, fully exploiting the aliasing nature of $\pi_{\text{dp}}^{\mathcal{W}}$.

Algorithm 1 Noise-Aliased Diffusion Steering via Reinforcement Learning (DSRL-NA)

input: pretrained diffusion policy $\pi_{\text{dp}}^{\mathcal{W}}$, offline data $\mathfrak{D}_{\text{off}}$ and/or online environment $\mathcal{M}$
Initialize replay buffer $\mathfrak{B} \leftarrow \mathfrak{D}_{\text{off}}$, $\mathcal{A}$-critic $Q^{\mathcal{A}}$, latent-noise critic $Q^{\mathcal{W}}$, latent-noise actor $\pi^{\mathcal{W}}$
for $t = 1, \ldots, T$ **do**
  Update $Q^{\mathcal{A}}$: $\min_{Q^{\mathcal{A}}} \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')\sim\mathfrak{B},\boldsymbol{a}'\sim\pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s}',\pi^{\mathcal{W}}(\boldsymbol{s}'))}\left[(Q^{\mathcal{A}}(\boldsymbol{s},\boldsymbol{a}) - r - \gamma\bar{Q}^{\mathcal{A}}(\boldsymbol{s}',\boldsymbol{a}'))^2\right]$
  Update $Q^{\mathcal{W}}$: $\min_{Q^{\mathcal{W}}} \mathbb{E}_{\boldsymbol{s}\sim\mathfrak{B},\boldsymbol{w}\sim\mathcal{N}(0,I)}[(Q^{\mathcal{W}}(\boldsymbol{s},\boldsymbol{w}) - Q^{\mathcal{A}}(\boldsymbol{s},\pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s},\boldsymbol{w})))^2]$
  Update $\pi^{\mathcal{W}}$: $\max_{\pi^{\mathcal{W}}} \mathbb{E}_{\boldsymbol{s}\sim\mathfrak{B}}\left[Q^{\mathcal{W}}(\boldsymbol{s},\pi^{\mathcal{W}}(\boldsymbol{s}))\right]$
  if access to online environment $\mathcal{M}$ **then**
    Sample latent-noise action $\boldsymbol{w}_t \sim \pi^{\mathcal{W}}(\boldsymbol{s}_t)$ and compute $\boldsymbol{a}_t \leftarrow \pi_{\text{dp}}^{\mathcal{W}}(\boldsymbol{s}_t,\boldsymbol{w}_t)$
    Play $\boldsymbol{a}_t$ in $\mathcal{M}$, observe r_t and next state $\boldsymbol{s}_{t+1}$, and add $(\boldsymbol{s}_t,\boldsymbol{a}_t,r_t,\boldsymbol{s}_{t+1})$ to $\mathfrak{B}$

We remark that, in the purely offline setting, if π_{dp} is trained on the same offline dataset $\mathfrak{D}_{\text{off}}$ as DSRL-NA, then DSRL-NA naturally handles conservatism. When using standard diffusion policy training, by construction $\pi_{\text{dp}}^{\mathcal{W}}$ will only output in-distribution actions, which implies that any $\boldsymbol{w}$ will be mapped by $\pi_{\text{dp}}^{\mathcal{W}}$ to an in-distribution action. Since DSRL-NA only queries $Q^{\mathcal{A}}$ on actions in $\mathfrak{D}_{\text{off}}$ and actions produced by $\pi_{\text{dp}}^{\mathcal{W}}$, it therefore naturally avoids querying the value of unseen actions, and achieves this while allowing $\pi^{\mathcal{W}}$ to be optimized in an unrestricted manner over the latent-noise action space. Notably, this enables conservative policy optimization without explicitly enforcing a conservatism penalty, as is typically done in offline RL [84, 85, 86, 87].

5 Experiments

In our experiments, we evaluate DSRL on simulated environments—in online (Section 5.1) and offline (Section 5.2) settings (see Appendix B for offline-to-online results)—and real-world robotic control settings (Section 5.3). We also demonstrate that DSRL enables steering of generalist robot policies, and in both simulation and the real world (Section 5.4). Throughout this section we instantiate DSRL with DSRL-NA, as well as by applying SAC [83] directly to the latent-action MDP $\mathcal{M}^{\mathcal{W}}$ (which we refer to as DSRL-SAC), and evaluate DSRL's performance across both diffusion policies (with DDIM sampling) and flow-based policies. Please see the appendix for further details on all experiments, and Figure 7 for illustration of all our real-world experiments.

5.1 DSRL Enables Efficient Online Adaptation of Diffusion Policies

We first study the online adaptation setting, where we assume we are given a diffusion policy π_{dp} pretrained on demonstration data, and our goal is to adapt the behavior of π_{dp} to maximize a given reward using only online samples. We evaluate on the `Robomimic` benchmark [88], a challenging robotic manipulation benchmark, and three different tasks from the `OpenAI Gym` benchmark [89]. In all cases other than `Robomimic Square` and `Transport`, we set π_{dp} to the diffusion policy checkpoints provided by Ren et al. [21], while for `Square` and `Transport` we train a slightly higher-performing base diffusion policy. We compare against five other state-of-the-art methods for RL with diffusion policies, three of which—DPPO [21], IDQL [52], and DQL [46]—directly seek to adapt the behavior of π_{dp} online, and two—DIPO [62] and QSM [61]—which utilize diffusion policies but learn from scratch (see Section D.1 for additional details). Each experiment is averaged over 5 seeds, and all but `Transport` utilize DSRL-NA—for `Transport` we utilize DSRL-SAC. Our results are given in Figures 2 and 3. As these figures illustrate, DSRL is not only able to effectively modify the base policy's behavior, achieving near-optimal behavior on each task and demonstrating that diffusion steering is an expressive approach for policy adaptation, but also requires a substantially lower number of samples than existing approaches—at least a 5-10× improvement in efficiency—to reach this level of performance.

5.2 DSRL Enables Efficient Offline Adaptation of Diffusion Policies

We next turn to the offline setting, evaluating how well the DSRL-NA variant of DSRL can learn from offline data. We evaluate on 10 tasks from the `OGBench` offline RL benchmark [90], covering a variety of locomotion and manipulation settings. In all cases we utilize the provided offline dataset to train π_{dp}, then freeze π_{dp} and apply DSRL-NA on this same offline dataset. We compare against 10 different baselines: Gaussian-policy based BC ("BC ($\mathcal{N}$)"), IQL [85], and REBRAC [91],

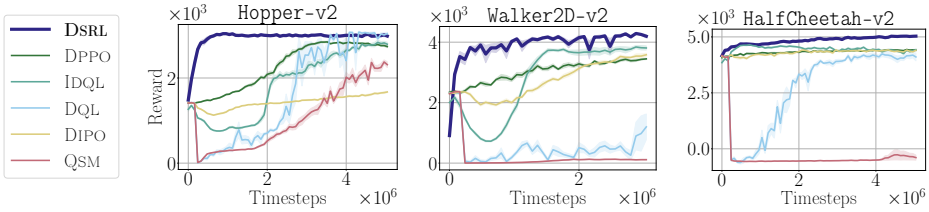


Figure 2: DSRL enables online adaptation of pretrained diffusion policies on `OpenAI Gym` [89].

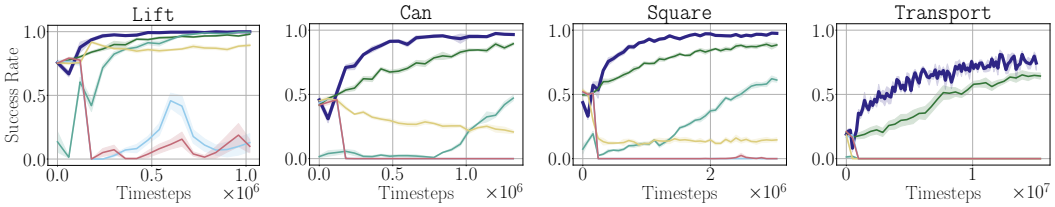


Figure 3: DSRL enables online adaptation of pretrained diffusion policies on `Robomimic` [88].

Task	BC ($\mathcal{N}$)	BC (π_{dp})	IQL	REBRAC	IDQL	SRPO	CAC	FAWAC	FBRAC	IFQL	FQL	**DSRL**
`antmaze-large-navigate-singletask`	0 ±0	0 ±0	48 ±9	**91** ±10	0 ±0	0 ±0	42 ±7	1 ±1	70 ±20	24 ±17	80 ±8	40 ±29
`antmaze-giant-navigate-singletask`	0 ±0	0 ±0	0 ±0	**27** ±22	0 ±0	0 ±0	0 ±0	0 ±0	0 ±1	0 ±0	4 ±5	0 ±0
`humanoidmaze-medium-navigate-singletask`	1 ±0	0 ±0	32 ±7	16 ±9	1 ±1	0 ±0	38 ±19	6 ±2	25 ±8	**69** ±19	19 ±12	34 ±20
`humanoidmaze-large-navigate-singletask`	0 ±0	0 ±0	3 ±1	2 ±1	0 ±0	0 ±0	1 ±1	0 ±0	0 ±1	6 ±2	7 ±6	**10** ±12
`antsoccer-arena-navigate-singletask`	1 ±0	0 ±0	3 ±2	0 ±0	0 ±1	0 ±0	0 ±0	12 ±3	24 ±4	16 ±9	**39** ±6	28 ±9
`cube-single-play-singletask`	3 ±1	3 ±3	85 ±8	92 ±4	**96** ±2	82 ±16	80 ±30	81 ±9	83 ±13	73 ±3	**97** ±2	**93** ±14
`cube-double-play-singletask`	0 ±0	0 ±0	1 ±1	7 ±3	16 ±10	0 ±0	2 ±2	2 ±1	22 ±12	9 ±5	36 ±6	**53** ±14
`scene-play-singletask`	1 ±1	0 ±0	12 ±3	50 ±13	33 ±14	2 ±2	50 ±40	18 ±8	46 ±10	0 ±0	76 ±9	**88** ±9
`puzzle-3x3-play-singletask`	1 ±1	0 ±0	2 ±1	2 ±1	0 ±0	0 ±0	0 ±0	1 ±1	2 ±2	0 ±0	**16** ±5	0 ±0
`puzzle-4x4-play-singletask`	0 ±0	0 ±0	5 ±2	10 ±3	26 ±6	7 ±4	1 ±1	0 ±0	5 ±1	21 ±11	11 ±3	**37** ±13

Table 1: DSRL enables effective adaptation of flow policies from offline data on `OGBench` [90].

diffusion-policy based IDQL [52], SRPO [58], and CAC [48], and flow-policy based FAWAC, FBRAC, IFQL (flow-based variants of [92], [46], and IDQL, respectively), FQL [23], and flow-based BC (the base diffusion policy used by DSRL, "BC (π_{dp})"). These baselines cover a wide range of standard offline RL methods, including methods that utilize diffusion or flow-based policies (e.g., FQL).

Our results are given in Table 1. All baseline values, with the exception of BC (π_{dp}), are taken from Park et al. [23], and DSRL and BC (π_{dp}) are averaged over 4 random seeds. We see that DSRL achieves state-of-the-art performance on approximately half the tasks, demonstrating its effectiveness at learning from offline data. We highlight in particular that DSRL substantially improves over BC (π_{dp})—the performance of π_{dp} before steering—again illustrating the large increase in policy performance diffusion steering can produce, even when utilizing offline data. It is notable that the same DSRL-NA method that we use in our online RL experiments directly serves as a highly effective offline RL method without modification.

5.3 Adapting Real-World Robot Policies with DSRL

In this section, we demonstrate that DSRL enables real-world improvement of robot control policies. We evaluate the performance of DSRL in steering a multi-task diffusion policy trained on a much more diverse set of demonstrations—see Appendix A for real-world results steering a single-task diffusion policy trained on a narrow set of expert demonstrations. We first train a diffusion policy on the Bridge V2 dataset [93], contains over 60,000 robotic teleoperation trajectories collected on a diverse set of manipulation tasks on the WidowX 250 6-DoF robot arm. Pretraining on Bridge endows π_{dp} with a variety of useful manipulation behaviors, yet such pretrained policies often still struggle when deployed in novel scenes. Here we parameterize π_{dp} with the DiT policy architecture proposed by Dasari et al. [19]—a state-of-the-art transformer-based diffusion policy.

We evaluate on three tasks—`Pick-and-Place`, `Drawer Closing`, and `Block Stacking`—as illustrated in Figure 7(b)-(d). For all tasks, we first roll out the pretrained diffusion policy π_{dp} 20 times, then begin the online RL training, utilizing DSRL-SAC and initializing the replay buffer with the data from these rollouts of π_{dp}. As a baseline, we run RLPD [94, 95, 96]. In all case we use a 0-1 sparse reward. Our results are given in Figure 4, where we plot the performance of the learned policies averaged over 20 evaluations for `Pick-and-Place`, and 10 for the other tasks. As can be seen, using a very small number of online adaptation steps (100-150 episodes), we are able to

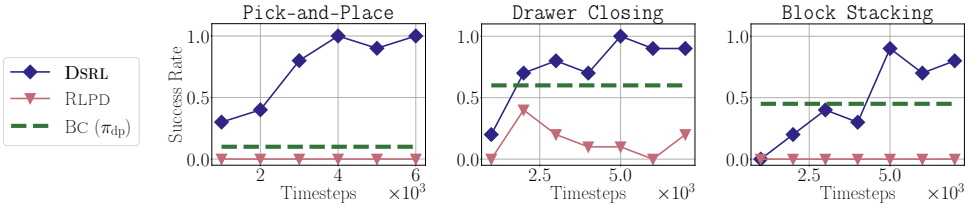


Figure 4: DSRL efficiently improves multi-task BC policy on real-world WidowX robot.

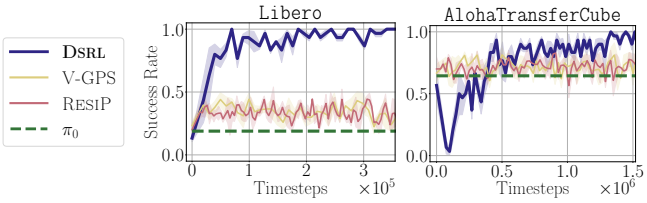


Figure 5: DSRL enables adaptation of π_0 [6] on `Libero` and `Aloha` simulation tasks.

Task	π_0	DSRL
Turn on toaster	5/20	**18/20**
Put spoon on plate	15/20	**19/20**

Figure 6: Real-world performance of π_0 and π_0 steered with DSRL.

improve the performance of π_{dp} significantly for each task; in contrast, RLPD is unable to learn at all. These results demonstrate the ability of DSRL to steer a real-world robotic control policy with sample efficiency high enough to make real-world adaptation practical.

5.4 Steering Pretrained Generalist Robot Policies with DSRL

Finally, we evaluate if DSRL can steer larger pretrained "generalist" policies. Here, we consider π_0 [6], a state-of-the-art generalist robot policy. π_0 is a 3.3B parameter model, trained to respond to language commands, and utilizes a VLM-based backbone and flow-based action head. Standard finetuning of policies of this size is challenging due to their scale and, furthermore, applying RL to π_0 is non-trivial due to its high-dimensional action space—π_0 uses an action chunk size of 50 and each action is 32-dimensional, resulting in a total action space dimension of 1,600.

We first evaluate DSRL-SAC steering of π_0 on two simulation tasks: (i) the `Libero` [97] manipulation benchmark, where we study a pick-and-place task from the `Libero-90` suite as the downstream task, and (ii) `AlohaTransferCube` [98], a challenging bimanual manipulation task that requires high precision. We compare against two baselines: RESIP [64], a residual RL-based method, and V-GPS [54]. Both RESIP and V-GPS require only black-box access to the pretrained policy, and aim to improve it by post-processing the actions. In all cases we use off-the-shelf public checkpoints for π_0 and average over 3 seeds. As shown in Figure 5, DSRL dramatically improves π_0's performance—for example, on `Libero`, improving from $\approx$20% to $\approx$100% after only $\approx$10,000 online samples—demonstrating the effectiveness of diffusion steering for adapting pretrained generalist policies. In contrast, neither RESIP nor V-GPS are able to significantly improve the performance of π_0. We would like to note that both tasks we study are extremely challenging for RL agents due to their complex nature—including high control frequency, long horizons, large observation and action spaces, and sparse rewards. The effectiveness of DSRL in such settings, as compared to approaches such as RESIP or V-GPS which utilize more traditional RL techniques, underscores the advantages of running RL over the latent-noise action space in settings where traditional RL is challenging, and DSRL's ability to more effectively exploit the pretrained policy's behavior than approaches which rely on post-processing the pretrained policy's actions.

We next evaluate DSRL-SAC steering of π_0 in the real world. We utilize the publicly available checkpoint of π_0 trained on the DROID dataset [99], and deploy on a Franka robot arm. For our evaluation we consider two tasks: turning on a toaster, and picking up a spoon and placing it on a plate (see Figure 7(e) and (f)). We run DSRL for 80 online episodes ($\sim$11000 total steps) on the toaster task, and 65 online episodes ($\sim$10000 total steps) on the pick-and-place task. We evaluate the performance of the base π_0 policy, and the final steered policy, and give the results in Figure 6. As these results illustrate, DSRL is able to significantly improve the performance of π_0 in a relatively small number of samples on real-world tasks. To the best of our knowledge, this is the first successful demonstration of real-world RL-based finetuning of π_0.

Limitations

Our experiments show that DSRL provides an effective and highly efficient method for improving diffusion and flow policies using either online or offline experience. However, our approach does have a number of limitations. First, the exploration capabilities of DSRL are inherently determined by the underlying diffusion policy, and while this seems to work well in practice, we do not have a guarantee that *all* diffusion policies are steerable, nor that all such policies provide adequate exploration. A highly concentrated action distribution (e.g., a policy trained on very narrow data) might not provide enough "options" for our method to select from, limiting its ability to improve, and our approach does not currently offer a clear way to quantify in advance how much improvement is possible with a given diffusion policy. Studying this question both empirically and theoretically could help us characterize the steerability of diffusion policies. Our method also has similar limitations to any RL approach: it requires reward signals, online rollouts (for the online RL variant), and resets (if learning in the real world). While the efficiency of our approach alleviates some of the challenges associated with RL in the real world, it still introduces additional requirements beyond the standard imitation learning methods that can be used to train the base diffusion policy. Reducing these limitations, for example by automating rewards or further improving sample efficiency, is an important direction for future work. We hope that studying these open questions will lead to even more practical methods for robotic RL, and that DSRL will provide a practical and useful tool for autonomous improvement in robotic learning.

Acknowledgments

This research was partly supported by ONR N00014-22-1-2773 and N00014-25-1-2060. YZ, AG and WY are partly supported by funding from Amazon. We would like to thank Pranav Atreya, Homer Walke, and Laura Smith for their advice on the π_0 setup. This research used the Savio computational cluster resource provided by the Berkeley Research Computing program at UC Berkeley.

References

[1] S. Stepputtis, J. Campbell, M. Phielipp, S. Lee, C. Baral, and H. Ben Amor. Language-conditioned imitation learning for robot manipulation tasks. *Advances in Neural Information Processing Systems*, 33:13139–13150, 2020.

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

[3] J. Gu, S. Kirmani, P. Wohlhart, Y. Lu, M. G. Arenas, K. Rao, W. Yu, C. Fu, K. Gopalakrishnan, Z. Xu, et al. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches. *arXiv preprint arXiv:2311.01977*, 2023.

[4] O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[5] T. Z. Zhao, J. Tompson, D. Driess, P. Florence, K. Ghasemipour, C. Finn, and A. Wahid. Aloha unleashed: A simple recipe for robot dexterity. *arXiv preprint arXiv:2410.13126*, 2024.

[6] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

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

[8] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[9] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[10] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[11] K. Team, A. Du, B. Gao, B. Xing, C. Jiang, C. Chen, C. Li, C. Xiao, C. Du, C. Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

[12] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

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

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

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

[16] L. Ankile, A. Simeonov, I. Shenfeld, and P. Agrawal. Juicer: Data-efficient imitation learning for robotic assembly. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5096–5103. IEEE, 2024.

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

[18] A. Sridhar, D. Shah, C. Glossop, and S. Levine. Nomad: Goal masked diffusion policies for navigation and exploration. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 63–70. IEEE, 2024.

[19] S. Dasari, O. Mees, S. Zhao, M. K. Srirama, and S. Levine. The ingredients for robotic diffusion transformers. *arXiv preprint arXiv:2410.10088*, 2024.

[20] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

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

[22] M. S. Mark, T. Gao, G. G. Sampaio, M. K. Srirama, A. Sharma, C. Finn, and A. Kumar. Policy agnostic rl: Offline rl and online rl fine-tuning of any class and backbone. *arXiv preprint arXiv:2412.06685*, 2024.

[23] S. Park, Q. Li, and S. Levine. Flow q-learning. *arXiv preprint arXiv:2502.02538*, 2025.

[24] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[25] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[26] M. Bojarski. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[27] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 5628–5635. IEEE, 2018.

[28] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3758–3765. IEEE, 2018.

[29] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

[30] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[31] S. Nair, E. Mitchell, K. Chen, S. Savarese, C. Finn, et al. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning*, pages 1303–1315. PMLR, 2022.

[32] Z. J. Cui, Y. Wang, N. M. M. Shafiullah, and L. Pinto. From play to policy: Conditional behavior generation from uncurated robot data. *arXiv preprint arXiv:2210.10047*, 2022.

[33] K. Black, M. Nakamoto, P. Atreya, H. Walke, C. Finn, A. Kumar, and S. Levine. Zero-shot robotic manipulation with pretrained image-editing diffusion models. *arXiv preprint arXiv:2310.10639*, 2023.

[34] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

[35] A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, T. Jaakkola, and P. Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

[36] Q. Zheng, M. Le, N. Shaul, Y. Lipman, A. Grover, and R. T. Chen. Guided flows for generative modeling and decision making. *arXiv preprint arXiv:2311.13443*, 2023.

[37] W. Li, X. Wang, B. Jin, and H. Zha. Hierarchical diffusion for offline decision making. In *International Conference on Machine Learning*, pages 20035–20064. PMLR, 2023.

[38] Z. Liang, Y. Mu, M. Ding, F. Ni, M. Tomizuka, and P. Luo. Adaptdiffuser: Diffusion models as adaptive self-evolving planners. *arXiv preprint arXiv:2302.01877*, 2023.

[39] H. T. Suh, G. Chou, H. Dai, L. Yang, A. Gupta, and R. Tedrake. Fighting uncertainty with gradients: Offline reinforcement learning via diffusion score matching. In *Conference on Robot Learning*, pages 2878–2904. PMLR, 2023.

[40] C. Chen, F. Deng, K. Kawaguchi, C. Gulcehre, and S. Ahn. Simple hierarchical planning with diffusion. *arXiv preprint arXiv:2401.02644*, 2024.

[41] L. Wang, J. Zhao, Y. Du, E. H. Adelson, and R. Tedrake. Poco: Policy composition from and for heterogeneous robot learning. *arXiv preprint arXiv:2402.02511*, 2024.

[42] C. Lu, H. Chen, J. Chen, H. Su, C. Li, and J. Zhu. Contrastive energy prediction for exact energy-guided diffusion sampling in offline reinforcement learning. In *International Conference on Machine Learning*, pages 22825–22855. PMLR, 2023.

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

[44] S. Zhang, W. Zhang, and Q. Gu. Energy-weighted flow matching for offline reinforcement learning. *arXiv preprint arXiv:2503.04975*, 2025.

[45] S. Ding, K. Hu, Z. Zhang, K. Ren, W. Zhang, J. Yu, J. Wang, and Y. Shi. Diffusion-based reinforcement learning via q-weighted variational policy optimization. *arXiv preprint arXiv:2405.16173*, 2024.

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

[47] L. He, L. Shen, L. Zhang, J. Tan, and X. Wang. Diffcps: Diffusion model based constrained policy search for offline reinforcement learning. *arXiv preprint arXiv:2310.05333*, 2023.

[48] Z. Ding and C. Jin. Consistency models as a rich and efficient policy class for reinforcement learning. *arXiv preprint arXiv:2309.16984*, 2023.

[49] S. E. Ada, E. Oztop, and E. Ugur. Diffusion policies for out-of-distribution generalization in offline reinforcement learning. *IEEE Robotics and Automation Letters*, 9(4):3116–3123, 2024.

[50] R. Zhang, Z. Luo, J. Sjölund, T. Schön, and P. Mattsson. Entropy-regularized diffusion policy with q-ensembles for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 37:98871–98897, 2024.

[51] H. Chen, C. Lu, C. Ying, H. Su, and J. Zhu. Offline reinforcement learning via high-fidelity generative behavior modeling. *arXiv preprint arXiv:2209.14548*, 2022.

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

[53] L. He, L. Shen, J. Tan, and X. Wang. Aligniql: Policy alignment in implicit q-learning through constrained optimization. *arXiv preprint arXiv:2405.18187*, 2024.

[54] M. Nakamoto, O. Mees, A. Kumar, and S. Levine. Steering your generalists: Improving robotic foundation models via value guidance. *arXiv preprint arXiv:2410.13816*, 2024.

[55] S. Venkatraman, S. Khaitan, R. T. Akella, J. Dolan, J. Schneider, and G. Berseth. Reasoning with latent diffusion in offline reinforcement learning. *arXiv preprint arXiv:2309.06599*, 2023.

[56] H. Chen, K. Zheng, H. Su, and J. Zhu. Aligning diffusion behaviors with q-functions for efficient continuous control. *arXiv preprint arXiv:2407.09024*, 2024.

[57] T. Chen, Z. Wang, and M. Zhou. Diffusion policies creating a trust region for offline reinforcement learning. *arXiv preprint arXiv:2405.19690*, 2024.

[58] H. Chen, C. Lu, Z. Wang, H. Su, and J. Zhu. Score regularized policy optimization through diffusion behavior. *arXiv preprint arXiv:2310.07297*, 2023.

[59] L. Mao, H. Xu, X. Zhan, W. Zhang, and A. Zhang. Diffusion-dice: In-sample diffusion guidance for offline reinforcement learning. *arXiv preprint arXiv:2407.20109*, 2024.

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

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

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

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

[64] L. Ankile, A. Simeonov, I. Shenfeld, M. Torne, and P. Agrawal. From imitation to refinement–residual rl for precise assembly. *arXiv preprint arXiv:2407.16677*, 2024.

[65] X. Yuan, T. Mu, S. Tao, Y. Fang, M. Zhang, and H. Su. Policy decorator: Model-agnostic online refinement for large policy model. *arXiv preprint arXiv:2412.13630*, 2024.

[66] M. Sarmad, H. J. Lee, and Y. M. Kim. Rl-gan-net: A reinforcement learning agent controlled gan network for real-time point cloud shape completion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5898–5907, 2019.

[67] Z. Tang, J. Peng, J. Tang, M. Hong, F. Wang, and T.-H. Chang. Inference-time alignment of diffusion models with direct noise optimization. *arXiv preprint arXiv:2405.18881*, 2024.

[68] L. Eyring, S. Karthik, K. Roth, A. Dosovitskiy, and Z. Akata. Reno: Enhancing one-step text-to-image models through reward-based noise optimization. *Advances in Neural Information Processing Systems*, 37:125487–125519, 2024.

[69] J. Mao, X. Wang, and K. Aizawa. The lottery ticket hypothesis in denoising: Towards semantic-driven initialization. In *European Conference on Computer Vision*, pages 93–109. Springer, 2024.

[70] C. Chen, L. Yang, X. Yang, L. Chen, G. He, C. Wang, and Y. Li. Find: Fine-tuning initial noise distribution with policy optimization for diffusion models. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 6735–6744, 2024.

[71] D. Samuel, R. Ben-Ari, S. Raviv, N. Darshan, and G. Chechik. Generating images of rare concepts using pre-trained diffusion models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 4695–4703, 2024.

[72] D. Ahn, J. Kang, S. Lee, J. Min, M. Kim, W. Jang, H. Cho, S. Paul, S. Kim, E. Cha, et al. A noise is worth diffusion guidance. *arXiv preprint arXiv:2412.03895*, 2024.

[73] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

[74] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[75] M. Dalal, D. Pathak, and R. R. Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems*, 34: 21847–21859, 2021.

[76] D. Ki, J. Oh, S.-W. Shim, and B.-J. Lee. Prior-guided diffusion planning for offline reinforcement learning. *arXiv preprint arXiv:2505.10881*, 2025.

[77] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[78] X. Liu, C. Gong, and Q. Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

[79] M. S. Albergo and E. Vanden-Eijnden. Building normalizing flows with stochastic interpolants. *arXiv preprint arXiv:2209.15571*, 2022.

[80] Y. Lipman, M. Havasi, P. Holderrieth, N. Shaul, M. Le, B. Karrer, R. T. Chen, D. Lopez-Paz, H. Ben-Hamu, and I. Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

[81] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[82] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[83] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.

[84] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems*, 33:1179–1191, 2020.

[85] I. Kostrikov, A. Nair, and S. Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[86] Y. Jin, Z. Yang, and Z. Wang. Is pessimism provably efficient for offline rl? In *International Conference on Machine Learning*, pages 5084–5096. PMLR, 2021.

[87] T. Xie, C.-A. Cheng, N. Jiang, P. Mineiro, and A. Agarwal. Bellman-consistent pessimism for offline reinforcement learning. *Advances in neural information processing systems*, 34: 6683–6694, 2021.

[88] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

[89] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[90] S. Park, K. Frans, B. Eysenbach, and S. Levine. Ogbench: Benchmarking offline goal-conditioned rl. *arXiv preprint arXiv:2410.20092*, 2024.

[91] D. Tarasov, V. Kurenkov, A. Nikulin, and S. Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36: 11592–11620, 2023.

[92] A. Nair, A. Gupta, M. Dalal, and S. Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[93] H. R. Walke, K. Black, T. Z. Zhao, Q. Vuong, C. Zheng, P. Hansen-Estruch, A. W. He, V. Myers, M. J. Kim, M. Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023.

[94] P. J. Ball, L. Smith, I. Kostrikov, and S. Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pages 1577–1594. PMLR, 2023.

[95] J. Luo, Z. Hu, C. Xu, Y. L. Tan, J. Berg, A. Sharma, S. Schaal, C. Finn, A. Gupta, and S. Levine. Serl: A software suite for sample-efficient robotic reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16961–16969. IEEE, 2024.

[96] J. Luo, C. Xu, J. Wu, and S. Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *arXiv preprint arXiv:2410.21845*, 2024.

[97] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

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

[99] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, P. D. Fagan, J. Hejna, M. Itkina, M. Lepert, Y. J. Ma, P. T. Miller, J. Wu, S. Belkhale, S. Dass, H. Ha, A. Jain, A. Lee, Y. Lee, M. Memmel, S. Park, I. Radosavovic, K. Wang, A. Zhan, K. Black, C. Chi, K. B. Hatch, S. Lin, J. Lu, J. Mercat, A. Rehman, P. R. Sanketi, A. Sharma, C. Simpson, Q. Vuong, H. R. Walke, B. Wulfe, T. Xiao, J. H. Yang, A. Yavary, T. Z. Zhao, C. Agia, R. Baijal, M. G. Castro, D. Chen, Q. Chen, T. Chung, J. Drake, E. P. Foster, J. Gao, V. Guizilini, D. A. Herrera, M. Heo, K. Hsu, J. Hu, M. Z. Irshad, D. Jackson, C. Le, Y. Li, K. Lin, R. Lin, Z. Ma, A. Maddukuri, S. Mirchandani, D. Morton, T. Nguyen, A. O'Neill, R. Scalise, D. Seale, V. Son, S. Tian, E. Tran, A. E. Wang, Y. Wu, A. Xie, J. Yang, P. Yin, Y. Zhang, O. Bastani, G. Berseth, J. Bohg, K. Goldberg, A. Gupta, A. Gupta, D. Jayaraman, J. J. Lim, J. Malik, R. Martín-Martín, S. Ramamoorthy, D. Sadigh, S. Song, J. Wu, M. C. Yip, Y. Zhu, T. Kollar, S. Levine, and C. Finn. Droid: A large-scale in-the-wild robot manipulation dataset. 2024.

[100] S. Haddadin, S. Parusel, L. Johannsmeier, S. Golz, S. Gabl, F. Walch, M. Sabaghian, C. Jähne, L. Hausperger, and S. Haddadin. The franka emika robot: A reference platform for robotics research and education. *IEEE Robotics & Automation Magazine*, 29(2):46–64, 2022.

[101] M. Nakamoto, S. Zhai, A. Singh, M. Sobol Mark, Y. Ma, C. Finn, A. Kumar, and S. Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. *Advances in Neural Information Processing Systems*, 36:62244–62269, 2023.

[102] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of machine learning research*, 22(268):1–8, 2021.

Contributions

Andrew Wagenmaker: Project lead. Core algorithm development; primary paper writer; led the simulated online, offline-to-online, and ablation experiments; led the real-world multi-task diffusion policy experiments; supported π_0 experiments.
Mitsuhiko Nakamoto: Led the π_0 experiments.
Yunchu Zhang: Led the real-world single-task diffusion policy experiments and supported the simulated experiments.
Seohong Park: Led the offline experiments and contributed to algorithm development.
Waleed Yagoub: Supported the real-world single-task diffusion policy experiments.
Anusha Nagabandi: Investigated the application of DSRL to dexterous manipulation.
Abhishek Gupta: Advised on the project and contributed to paper writing.
Sergey Levine: Advised on the project and contributed to paper writing.

A Additional Real-World Results

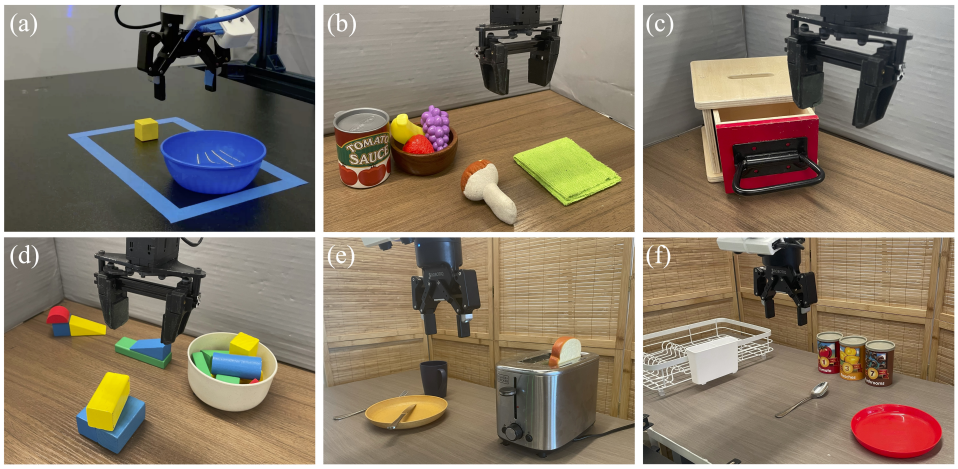


Figure 7: Real-world robot tasks for DSRL adaptation. **(a)**: task for single-task diffusion policy. **(b)**-**(d)**: tasks for multitask diffusion policies (Section 5.3). **(e)**-**(f)**: tasks for generalist policy (Section 5.4).

DSRL steering of single-task diffusion policy. Here we include an additional experiment, running DSRL on a diffusion policy trained on a narrow, single-task dataset. For this experiment, we consider a pick-and-place task, where the objective is to pick up a cube and place it in a bowl (see Figure 4(a)). We run on a Franka Emika Panda robot arm [100], and train the base policy on 10 human demonstrations. For the RLPD baseline, we utilize the human demonstrations as the offline dataset to initialize the algorithm. We also consider an additional variant of RLPD, where a human periodically intervenes to correct RLPD's actions, similar to the procedure described in [96]. We run all methods for 3500 online steps (approximately 40 episodes).

π_{dp}	RLPD	RLPD + interventions	DSRL
2/10	0/10	0/10	**9/10**

Table 2: Performance of single-task diffusion policy π_{dp}, RLPD, and DSRL steering of π_{dp}.

Our results are given in Table 2, where we see that DSRL is able to quickly improve the performance of π_{dp} from 20% to nearly 100%. In contrast, RLPD is unable to learn at all. We note that while augmenting RLPD with human interventions does lead to more effective behavior than standard RLPD, it is still unable to successfully solve the task. This not only shows that DSRL is sample-efficient enough to make adapting real-world diffusion policies practical, but also shows that, given some demonstration data, rather than using this data to initialize RLPD, a much more effective approach is to first train a diffusion policy on this data then run DSRL on this diffusion policy.

B Understanding Diffusion Steering

In this section, we seek to gain a deeper understanding of DSRL.

Does DSRL enables efficient offline-to-online adaptation of diffusion policies? The results from Sections 5.1 and 5.2 demonstrate that DSRL can effectively steer diffusion policies using both online interactions and offline data. Here we investigate whether DSRL is an effective algorithm for offline-to-online settings. We consider again the `Robomimic Can` and `Square` tasks of Section 5.1, and as baselines compare against two standard approaches for offline-to-online RL, RLPD [94] and CAL-QL [101]. We run DSRL exactly as described in Algorithm 1, with $\mathfrak{D}_{\text{off}}$ set to the `Robomimic` Multi-Human demonstration datasets. Our results are given in Figure 8. As can be seen, including the offline data improves the sample complexity of DSRL by a factor of approximately $2\times$, demonstrating that DSRL can indeed effectively leverage offline data to speed up learning. In contrast, standard offline-to-online RL approaches are not able to learn at all on these tasks.

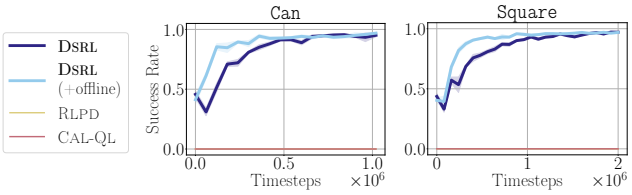


Figure 8: DSRL can make effective use of offline data to speed up online learning.

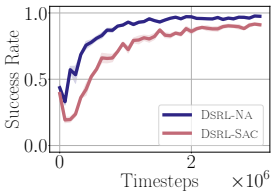


Figure 9: DSRL-NA improves on DSRL-SAC online (`Square`).

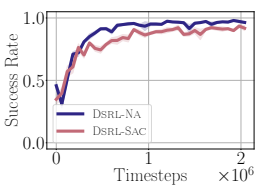


Figure 10: DSRL-NA improves on DSRL-SAC online (`Can`).

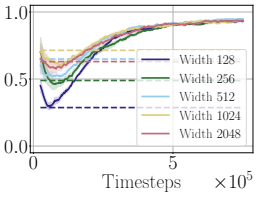


Figure 11: Sensitivity of DSRL to π_{dp}'s hidden layer width.

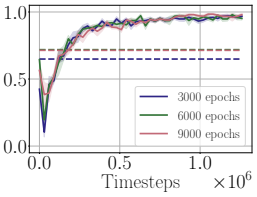


Figure 12: Sensitivity of DSRL to number of epochs π_{dp} trained on.

Comparing DSRL-NA and DSRL-SAC. While DSRL-NA has advantages over DSRL-SAC in that it can incorporate offline data, we are interested in whether it also improves the performance of purely online learning. In Figure 9 we compare DSRL-NA and DSRL-SAC on `Robomimic Square`, and in Figure 10 `Robomimic Can`. For DSRL-NA we use the hyperparameters given in Section D.1. For DSRL-SAC we swept over hyperparameters and included the results for best-performing ones, which were identical to those used for DSRL-NA except for using a hidden size of 128. As can be seen, while DSRL-SAC is still able to learn, it requires $\approx 2\times$ more samples than DSRL-NA, demonstrating the importance of noise aliasing for sample efficiency.

Does the size of π_{dp} affect DSRL? To investigate the effect of the size of π_{dp} on DSRL, we consider parameterizing π_{dp} as an MLP and varying its hidden layer width while keeping the training data fixed. Here we use the hyperparameters given in Section D.1 for `Robomimic Can` (with the exception of π_{dp}'s layer size, which we vary). For the plots shown here (and all following plots), the dashed line indicates the performance of π_{dp} without steering.

We illustrate our results in Figure 11, where we see that all model sizes can be successfully steered, despite the initial performance of π_{dp} varying significantly. We also highlight that at sizes of 512 and 1024 the performance of π_{dp} is essentially identical, yet DSRL is able to steer the 1024 model more quickly than the 512 model. We hypothesize that this may be due to the higher capacity of the 1024 model enabling it to more effectively capture diverse, potentially successful behaviors.

Does the number of epochs π_{dp} was trained affect DSRL? Next we consider whether the number of epochs π_{dp} was trained on affects the performance of π_{dp}. One might hypothesize that the

larger the number of epochs π_{dp} is trained for, the more difficult it will be to steer, as π_{dp} becomes increasingly overfit to the training data. To test this, we train π_{dp} for 3000, 6000, and 9000 epochs, and run DSRL on each. As can be seen in Figure 12, the number of epochs π_{dp} is trained on (while affecting the performance of π_{dp} somewhat) has almost no effect on the performance of DSRL, suggesting that DSRL is able to effectively steer policies that may be overfit to the demonstration data.

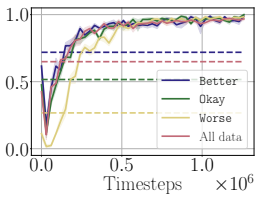


Figure 13: Sensitivity of DSRL to quality of data π_{dp} trained on.

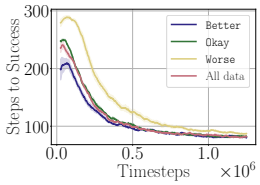


Figure 14: Sensitivity of DSRL's solution quality to quality of data π_{dp} trained on.

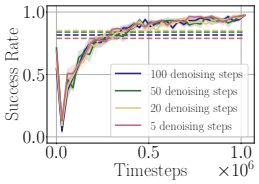


Figure 15: Sensitivity of DSRL to number of π_{dp} train denoising steps.

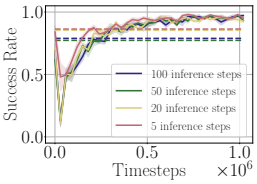


Figure 16: Sensitivity of DSRL to number of π_{dp} inference denoising steps.

Does the quality of data π_{dp} is trained on affect DSRL? To study the importance of data quality, we utilize the splits given in the Multi-Human dataset for `Robomimic Can`, which divides the data into sets collected by experienced operators ("`Better`"), adequate operators ("`Okay`"), and inexperienced operators ("`Worse`"). We train a new π_{dp} on each split, and use this as the base policy for DSRL. Our results are given in Figure 13. We see that the performance of π_{dp} is affected significantly by the quality of data it is trained on, yet the performance of DSRL only degrades for the `Worse` split, and quickly recovers to match its performance on the other splits. This demonstrates the robustness of DSRL to the quality of π_{dp}—while its performance may be somewhat affected initially if π_{dp} performs poorly, as long as π_{dp} still exhibits some task-solving behavior, DSRL is able to successfully steer it to solve the task.

To complement Figure 13, in Figure 14, we also plot the number of steps to success the steered policy requires. We see that the final steered policy DSRL converges to for π_{dp} trained on the `Worse` split is slightly less optimal in terms of time-to-success than the other cases. We conclude that the quality of π_{dp} does play a role in how quickly DSRL converges and the quality of the solution it converges to, yet find that DSRL is surprisingly robust to variation in the performance of π_{dp}.

We note as well that, compared to policies trained on only a single split of the data, the policy trained on all splits sees a more diverse set of behaviors, covering a larger number of possible solution paths. One may hypothesize that a policy could be more easily steered given the larger number of possible behaviors it has learned, but we do not find that to be the case here—training on all datasets does not lead to a performance improvement for DSRL as compared to training on a narrower dataset.

Does the number of denoising steps π_{dp} uses affect the performance of DSRL? Both diffusion and flow policies rely on an iterative denoising process, (1), to generate actions. An important hyperparameter in the denoising process is the number of steps that it uses, with typically more steps leading to higher quality generation at the expense of longer inference times. We can split the number of steps used in denoising into the value used in training (at what intervals to add the noise), and the number of steps taken at inference time. We test how the number of steps affects the behavior of DSRL, scaling both the number of train and inference steps. In Figure 15, we vary the number of steps we train on, while keeping the number of steps used at inference time fixed to 5, while in Figure 16 we train a single policy with 100 denoising steps, and vary the number of inference steps used. We see that neither the number of train steps or inference steps has virtually any impact on the performance of DSRL.

C Best Practices for DSRL

We found the following design choices led to the best results for DSRL. Several of these are consistent with current best practices for sample-efficient RL (e.g. [94]), while some are particular to DSRL:

- **High UTD**: In general we used a UTD of at least 20. We found that this led to faster convergence and higher sample efficiency, without sacrificing performance. Higher UTD seems to be particular valuable when running the DSRL-NA variant of DSRL.
- **Layer-norm**: We found applying layer norm to all but the final layer for both the critic and actor networks also led to improved performance.
- **Large policy and critic networks**: For all our experiments we instantiate our latent-noise actor and critics, $\pi^{\mathcal{W}}$ and $Q^{\mathcal{W}}$, as MLPs. We found that using larger MLPs led to better performance—in many experiments we used networks with layer size up to 2048.
- **Action magnitude**: When transforming the MDP from $\mathcal{M}$ to $\mathcal{M}^{\mathcal{W}}$, one design choice is how large a magnitude the actor can play in $\mathcal{W}$. If π_{dp} is trained by adding noise sampled from a standard Gaussian to the demonstration actions, as is typical, then only $\boldsymbol{w}$ reasonably likely under a standard Gaussian would be "in-distribution" for $\pi_{\text{dp}}(\boldsymbol{s}, \boldsymbol{w})$—$\pi_{\text{dp}}$ would have been unlikely to have denoised an example corresponding to $\boldsymbol{w}$ outside this range during training, and therefore its behavior on such examples may be unreliable. To incorporate this, we simply constrain the actions to a rectangle of some width $b_{\mathcal{W}}$, i.e. in practice we set $\mathcal{W} \leftarrow [-b_{\mathcal{W}}, b_{\mathcal{W}}]^d$. We found that choosing $b_{\mathcal{W}}$ in the range $[1, 3]$ generally worked well for online RL and $[0.5, 1.5]$ for offline RL. See below for the value of $b_{\mathcal{W}}$ used in each experiment.

D Additional Experimental Details

D.1 Details of Online Experiments (Section 5.1)

For all baselines, we use the implementations and hyperparameters given by Ren et al. [21] (please see Section E.3 of [21] for a detailed overview of baseline methods). As noted in the main text, for `Robomimic Lift` and `Can` and all `Gym` tasks, we utilize the diffusion policy checkpoints provided by Ren et al. [21] for π_{dp} (in particular, we use the identical checkpoints as they utilize for their online finetuning experiments). For `Robomimic Square` and `Transport` we train new diffusion policies with a larger number of denoising steps than those provided by Ren et al. [21] (100 vs 20) which we found to increase the performance of π_{dp} somewhat. To train these policies, we use the diffusion policy training code provided by Ren et al. [21], and keep all hyperparameters fixed other than the number of denoising steps. For all methods we utilize DDIM sampling, and note the number of denoising steps used at inference time in the tables below. As suggested in Ren et al. [21], during training, we experiment with injecting additional noise into the DDIM denoising process to help induce additional exploration. For each baseline method, we run variants with and without this additional noise injection, and include results for the best-performing one. We utilize the default rewards for each task (in particular, for all `Robomimic` tasks the reward is a 0-1 success reward).

We utilize DSRL-NA for all environments but `Robomimic Transport` where we use DSRL-SAC. For DSRL, in all experiments we run 4 environments in parallel. We update the actor and critic after every (parallel) environment step, taking the number of gradient steps at each update as stated below (we take the same number of steps on both the actor and critic). For updating the latent-noise critic $Q^{\mathcal{W}}$ in DSRL-NA, we denote the number of gradient steps per update as "$Q^{\mathcal{W}}$ update steps" in the following. For DSRL-SAC we use the implementation of SAC given by [102], and for DSRL-NA we use an entropy penalty identical to that used for SAC, otherwise running DSRL-NA as stated in Algorithm 1. For each environment, we first roll out π_{dp} with $\boldsymbol{w} \sim \mathcal{N}(0, I)$ for some number of steps (denoted as "Initial rollouts" in the following) and place the data collected in the replay buffer, before starting the actor and critic online training updates. The policies given by Ren

Table 4: **Hyperparameters for DSRL `Robomimic` experiments.**

Hyperparameter	`Lift`	`Can`	`Square`	`Transport`
Action chunk size	4	4	4	8
Hidden size	2048	2048	2048	128
Gradient steps per update	30	20	20	20
$Q^{\mathcal{W}}$ update steps	10	10	10	-
Discount factor	0.99	0.99	0.999	0.99
Action magnitude ($b_{\mathcal{W}}$)	1.5	1.5	1.5	1
Initial steps	24000	24000	32000	320000
π_{dp} train denoising steps	20	20	100	100
π_{dp} inference denoising steps	8	8	8	100

Table 5: **Hyperparameters for DSRL `OpenAI Gym` experiments.**

Hyperparameter	`Hopper-v2`	`Walker2D-v2`	`HalfCheetah-v2`
Action chunk size	4	4	4
Hidden size	2048	2048	1024
Gradient steps per update	20	20	20
$Q^{\mathcal{W}}$ update steps	10	10	10
Discount factor	0.99	0.99	0.99
Action magnitude ($b_{\mathcal{W}}$)	1.5	2.5	1.5
Initial steps	32000	32000	32000
π_{dp} train denoising steps	20	20	20
π_{dp} inference denoising steps	5	5	5

et al. [21] all utilize action chunking ("Action chunk size" in the following). To handle RL with action chunking, we treat the entire chunk as a single action and ignore the observations from within the chunk, treating the entire chunk as a single step in the environment (however, the number of timesteps stated in all results gives the total number of timesteps in the original environment).

In all plots, error bars denote 1 standard error. All experiments in this section were run on an NVIDIA A5000 GPU.

Table 3: **Common DSRL hyperparameters for online experiments.**

Hyperparameter	**Value**
Learning rate	0.0003
Batch size	256
Activation	Tanh
Target entropy	0
Target update rate (τ)	0.005
Number of actor and critic layers	3
Number of critics	2
Number of environments	4

D.2 Details of Offline Experiments (Section 5.2)

In the offline setting, we utilize the flow policy implementation given by Park et al. [23], and train each flow policy for 1M steps. For each environment we run on the default task given in `OGBench` [90], and specify the dataset used in the task name given in Table 1. As noted in the main text, all baseline values are taken directly from [23]. While DSRL was run with the same number of gradient steps and same network sizes as all baseline methods, we note that we utilize 10 critics for DSRL

Table 6: **Common DSRL hyperparameters for offline experiments.**

Hyperparameter	Value
Learning rate	0.0003
Batch size	256
Activation	GELU
Target update rate (τ)	0.005
Hidden size	512
Number of actor and critic layers	4
Number of critics	10
Flow steps	10
Clipped double Q-learning	False

(with mean aggregation), while all baselines were run with only 2 critics each, and thus the total number of parameters DSRL utilizes is larger than that used by the baselines.

For hyperparameters, we mostly follow the original choices by Park et al. [23], and present the full list of hyperparameters used in our offline experiments in Table 6. Also, following Park et al. [23], we individually tune the degree of conservatism (i.e., the action magnitude $b_{\mathcal{W}}$) for each task, and report the per-task hyperparameters in Table 7. In Table 1, "$\pm$" denotes 1 standard deviation. All experiments in this section were run on an NVIDIA A5000 GPU.

Table 7: **Per-task DSRL hyperparameters for offline experiments.**

Task	Action Magnitude ($b_{\mathcal{W}}$)	Discount Factor
`antmaze-large-navigate-singletask`	1.25	0.99
`antmaze-giant-navigate-singletask`	1.25	0.995
`humanoidmaze-medium-navigate-singletask`	0.5	0.995
`humanoidmaze-large-navigate-singletask`	0.75	0.995
`antsoccer-arena-navigate-singletask`	0.75	0.995
`cube-single-play-singletask`	0.5	0.99
`cube-double-play-singletask`	1.5	0.99
`scene-play-singletask`	0.75	0.99
`puzzle-3x3-play-singletask`	0.5	0.99
`puzzle-4x4-play-singletask`	0.5	0.99

D.3 Details of Offline-to-Online Experiments (Section B)

For DSRL, we utilize identical hyperparameters as are used for `Robomimic Can` and `Square` in Section 5.1. For RLPD and CAL-QL, we plot the numbers given in [21].

D.4 Details of Real-World Experiments (Section 5.3)

D.4.1 Single-Task Diffusion Policy

In the Franka single-task experiments, for π_{dp} we trained a diffusion policy using the CNN architecture proposed by [7] on a self-collected dataset. Please see Table 8 for the hyperparameters used for π_{dp}. For DSRL, we begin by running π_{dp} for 10 episodes sampling $\boldsymbol{w} \sim \mathcal{N}(0, I)$, and initialize the replay buffer with the data collected from these rollouts. For RLPD and RLPD with intervention, every batch is sampled with half taken from the 10 human demonstrations used to train π_{dp}, and half taken from additional data collected online, as is outlined in [94]. For RLPD and RLPD with intervention, we use the hyperparameters used by Luo et al. [96], which were found to work well on a similar real-world robot policy learning task. For DSRL, we use the hyperparameters given in Table 3 (with the exception of "Number of environments", which is 1 in the real-world setting) and

Table 8: **Hyperparameters for π_{dp} in Franka experiments.**

Hyperparameter	Value
Batch size	256
Learning rate	0.0003
Training steps	3500
Warmup steps	0
Action chunk size	1
Train denoising steps	100
Inference denoising steps	8
Image encoder	ResNet-18
Hidden size	1024
Number of Heads	1
Number of Layers	3
Feedforward dimension	512

Table 8. For both DSRL and RLPD we update the actor and critic at every environment step, for DSRL taking the number of gradient steps given in Table 8 per update.

In the cube pick-and-place task, we define an episode as successful if the cube is dropped into the bowl. We utilize a 0-1 success reward for each task. See Figure 18 for additional visualizations of our setups.

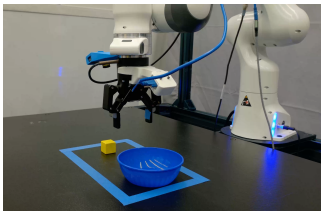


Figure 17: Single-task Franka setup.

D.4.2 Multi-Task Diffusion Policy

In the WidowX experiments, for π_{dp} we trained a diffusion policy using the DiT architecture proposed by [19] on the Bridge v2 dataset. Please see Table 9 for the hyperparameters used for π_{dp}. We trained π_{dp} with image goal conditioning, and in deployment would condition it on a goal image of the desired final position.

For both DSRL and RLPD, we begin by running π_{dp} for 20 episodes sampling $\boldsymbol{w} \sim \mathcal{N}(0, I)$, and initialize the replay buffer with the data collected from these rollouts. For RLPD, every batch is sampled with half taken from the 20 initial rollouts of π_{dp}, and half taken from additional data collected online, as is outlined in [94]. For RLPD, we use the hyperparameters used by Luo et al. [96], which were found to work well on a similar real-world robot policy learning task. For DSRL, we use the hyperparameters given in Table 3 (with the exception of "Number of environments", which is 1 in the real-world setting) and Table 10. As input features for both DSRL and RLPD, we provide the proprioceptive state, and also the image features learned by π_{dp}'s pretrained ResNet encoder. For both DSRL and RLPD we update the actor and critic every other environment step, for DSRL taking the number of gradient steps given in Table 10 per update.

For the pick-and-place task, we define an episode as successful if the mushroom is at least half on the cloth, and the WidowX has released the mushroom. For the drawer closing task we define an episode as successful if the drawer is successfully shut. For the block stacking task, we define an episode as successful if the yellow block is on the blue blocks and the WidowX has released the yellow block.

Table 9: **Hyperparameters for DiT π_{dp} in WidowX experiments.**

Hyperparameter	Value
Batch size	2048
Learning rate	0.0003
Training steps	100000
LR scheduler	cosine
Warmup steps	2000
Action chunk size	1
Train denoising steps	100
Inference denoising steps	8
Image encoder	ResNet-34
Hidden size	256
Number of Heads	1
Number of Layers	3
Feedforward dimension	512

Table 10: **Hyperparameters for DSRL WidowX experiments.**

Hyperparameter	`Pick-and-Place`	`Drawer Closing`	`Block Stacking`
Hidden size	1024	2048	1024
Gradient steps per update	30	20	20
Discount factor	0.97	0.97	0.97
Action magnitude ($b_{\mathcal{W}}$)	2	2	1.5

We utilize a 0-1 success reward for each task. See Figure 18 for additional visualizations of our setups.

For DSRL training, we utilize an NVIDIA GeForce RTX 4090 GPU.

D.5 Details of π_0 Experiments (Section 5.4)

Tasks. In the π_0 experiments, we study two simulation tasks: `Libero` [97] and `AlohaTransferCube` [98], as shown in Figure 19. The `Libero` benchmark consists of 130 tasks which are divided into separate suites—`Libero-{Spatial, Object, Goal, 10, 90}`. We study the task "pick up the cream cheese and put it in the tray" from the `Libero-90` suite as our target task for online fine-tuning. The `AlohaTransferCube` task requires the bimanual robots to pick up the cube with one hand and transfer it to the other. Both tasks use a sparse reward scheme: the agent receives –1 at every time step until it succeeds, at which point it receives 0.

Base policy. We use the public weights from π_0 for both tasks. Specifically, we use the checkpoint at `s3://openpi-assets/checkpoints/pi0_libero` for the `Libero` task, and `s3://openpi-assets/checkpoints/pi0_aloha_sim` for the `AlohaTransferCube` task. Both

(a) Pick-and-place task. (b) Drawer closing task. (c) Block stacking task.

Figure 18: Setups for multi-task WidowX diffusion policy experiments.

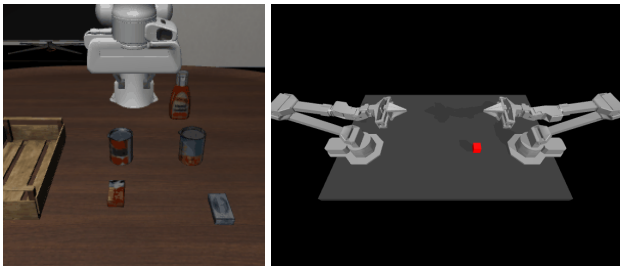
checkpoints were obtained by fine-tuning the π_0 flow-matching base policy, which was pre-trained on over 10k hours of robot data. The `Libero` checkpoint was fine-tuned for 30k steps on the `Libero-`{`Spatial, Object, Goal, 10`} suites, containing a total of 1,693 demonstrations on 40 tasks. Note that our target task is from the `Libero-90` suite, which is not included in the dataset. This allows us to study how to rapidly adapt the base policy to master a new, unseen task. The `Aloha` checkpoint was fine-tuned for 20k steps using 50 demonstrations on the same task.

Training details. We use the DSRL-SAC variant of our method for these experiments. Since both tasks receive visual observations, we train the actor and critic with vision encoders composed of four convolutional layers. The π_0 policy is trained in a noise and action space with a chunk size of 50, where each noise and action is 32-dimensional, resulting in a total dimensionality of 1,600. Naively training the actor $\pi^{\mathcal{W}}(\boldsymbol{s})$ and critic $Q^{\mathcal{W}}(\boldsymbol{s}, \boldsymbol{w})$ is challenging due to this high dimensionality.

Here, we propose a simple yet effective approach to train DSRL-SAC on large action chunks. To avoid training the actor and critic on the whole noise space $\mathcal{W} = (\mathbb{R}^d)^C = \left\{ \boldsymbol{w} = (w_1, \ldots, w_C) \mid w_i \in \mathbb{R}^d \right\}$, where C is the action chunk size and d is the per-step noise dimension, we instead train a single-step actor $\pi^{\mathcal{W}}_{\text{single}}(\boldsymbol{s})$ and critic $Q^{\mathcal{W}}_{\text{single}}(\boldsymbol{s}, \boldsymbol{w}_{\text{single}})$ where $\mathcal{W}_{\text{single}} = \mathbb{R}^d$. At inference time, we simply sample $w_{\text{single}} \sim \pi^{\mathcal{W}}_{\text{single}}$ and repeat it across the action chunk axis $\boldsymbol{w} = \{(w_1, \ldots, w_C) \mid w_i = w_{\text{single}}\}$, and use it to query the π_0 policy. We find that this makes SAC training more efficient and stable, while the repeated-noise formulation remains expressive enough to steer π_0 to produce effective actions. This shows that the noise space of pre-trained π_0 policy is a well-structured latent space, highlighting another advantage of running RL in this latent-noise action space. We provide the hyperparameters in Table 11.

Real-world experiments. For the real-world π_0 experiments, we utilize the pubic π_0-DROID checkpoint at `s3://openpi-assets/checkpoints/pi0_droid`. We adopt the same method to handle large action chunks as is described above, though note that this π_0-DROID variant we used only has an action chunk of size 10. Please see Figure 19 for visualizations of our setup. We input the proprioceptive state, the final token's last hidden feature from π_0's VLM backbone (a 2,048-dimensional vector), and visual features into the noise policy. The visual features are extracted by training a shallow convolutional encoder on the concatenated images from the left, right, and wrist cameras, following the same procedure described above. For text commands to π_0, we use "turn on the toaster" and "put the spoon on the plate".

We run π_0 inference on a remote policy server, but run DSRL training locally on a NVIDIA GeForce RTX 3070 GPU with 8GB of VRAM. We note that this is a significantly smaller amount of compute than is typically required for π_0 finetuning—as stated in the π_0 repository, LoRA of π_0 requires at least 22.5GB of VRAM, and full finetuning requires 70GB.



(a) `Libero` (b) `AlohaTransferCube`

Figure 19: Simulation tasks for π_0.

Table 11: **Hyperparameters for DSRL π_0 experiments.**

Hyperparameter	`Libero`	`AlohaTransferCube`	Real-world
Batch size	256	256	256
Hidden size	128	128	1024
Number of layers	3	3	3
Actor learning rate	0.0001	0.0001	0.0001
Critic learning rate	0.0003	0.0003	0.0003
Discount factor	0.999	0.999	0.99
Target entropy	$-d/2$	$-d/2$	0
Target update rate (τ)	0.005	0.005	0.005
Image observation size	64×64	64×64	128×128
CNN features	$(32, 32, 32, 32)$	$(32, 32, 32, 32)$	$(32, 32, 32, 32)$
Gradient steps per update	20	20	30
Action magnitude ($b_{\mathcal{W}}$)	1.0	2.0	2.5
Action chunk size	50	50	10
Number of actions to execute	20	50	10
Number of critics	10	10	2
Flow steps	10	10	10
Clipped double Q-learning	False	False	False

(a) Toaster task. (b) Spoon task.

Figure 20: π_0 real-world task setups.