POLICY-AGNOSTIC RL: RL FINE-TUNING OF ANY POLICY CLASS AND BACKBONE

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent successes in imitation learning have shown how critical it is to use expressive and multimodal policies. What would it take to replicate this success of better policy models in Reinforcement Learning (RL)? RL training of the best-performing policy models is challenging, as most deep RL machinery is co-developed with a specific policy class and backbone, resulting in poor performance when this synergy breaks. For e.g., SAC utilizes a policy gradient reparameterization for Gaussian policies, but this is unstable for diffusion policies (Wang et al.) and intractable for categorical policies. In this paper, we develop an approach called **policy-agnostic RL** (***PA-RL***) that can effectively train multiple policy classes with varying architectures and sizes, with offline RL and online fine-tuning methods. We build on the idea that supervised learning can replace the policy improvement step in RL, as long as it is applied on "optimized" actions. Concretely, we replace the policy improvement operator in RL with a supervised learning loss to imitate actions that maximize the critic value predictions, while staying close to the support of the data. Due to the universal nature of supervised learning, ***PA-RL*** is applicable to any policy model readily. Empirically, ***PA-RL*** enables fine-tuning continuous action diffusion and categorical autoregressive policies, entirely via actor-critic RL. ***PA-RL*** attains state-of-the-art results in simulation, and makes it possible, for the first time, to efficiently RL fine-tune OpenVLA (Kim et al., 2024), a 7B-parameter robot policy, directly on a real robot.

1 INTRODUCTION

Recent successes in learning decision-making policies in a number of domains come largely from the use of expressive models combined with large-scale imitation learning (Zitkovich et al., 2023; Chi et al., 2023; Kim et al., 2024; Chen et al., 2023), an approach that has been tried and tested in other areas of machine learning. However, training a policy once and freezing it is not good enough for many real-world deployment scenarios, where some adaptation is needed. For example, a robot must adapt its behavior as the surrounding environment or task changes; a language-model web navigation agent must improve its behavior with the evolution of websites on the Internet (Bai et al., 2024). The hallmark of an adaptation process is in its use of autonomous, non-expert data. In these use cases, imitation alone done once or applied repeatedly is not enough to guarantee the most efficient learning.

Reinforcement learning (RL) provides a flexible framework for adaptation and fine-tuning. In principle, off-the-shelf offline or offline-to-online RL algorithms (e.g., actor-critic algorithms) could be used to fine-tune any policy. However, most existing deep RL algorithms entangle the choice of algorithm design and loss functions with the choice of the policy class. For example, SAC (Haarnoja et al., 2018a), the base learner for many offline and online fine-tuning algorithms (Kumar et al., 2020; Nakamoto et al., 2024b), uses reparameterization which is stable for Gaussian (or Tanh-Gaussian) policies, but becomes unstable and time consuming for diffusion policies (Wang et al.). This can be problematic to the extent that weaker policy training techniques, e.g. critic-based re-ranking (Hansen-Estruch et al., 2023; Nakamoto et al., 2024a) on top of imitation policies outperform policy gradient (Wang et al.), even though this re-ranking approach is theoretically weaker and empirically worse with Gaussian policies (Fujimoto et al., 2019; Ghasemipour et al., 2021). Likewise, to extend CQL (Kumar et al., 2020) to token-based action distributions, Chebotar et al. (2023) had to make many modifications to the loss function. Overall, this means that adapting the best policy training methodologies from one policy model to another can be challenging, leading practitioners to choose a weaker algorithm or spend cycles modifying other components.

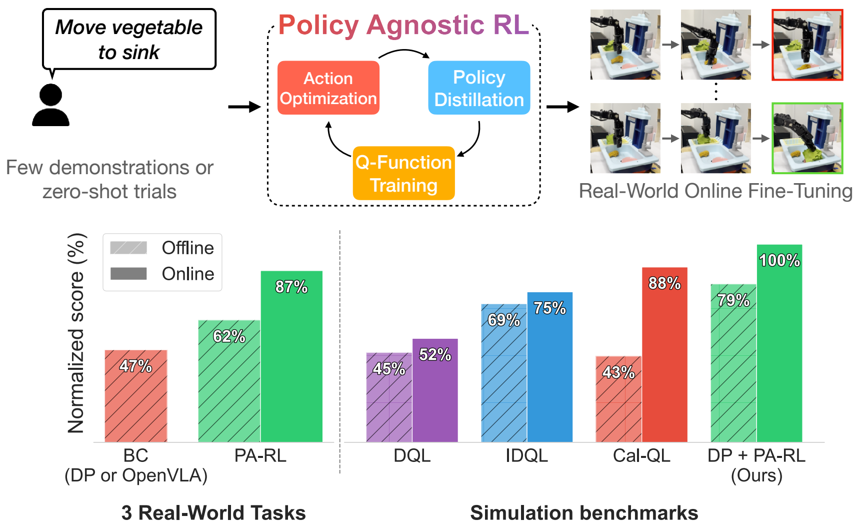


Figure 1: ***Policy-agnostic RL*** **(PA-RL*)*** is an approach to train any policy class in offline and online RL fine-tuning regimes. ***PA-RL*** effectively improves both diffusion policies and large generalist policies in real-world robotic manipulation tasks.

In this paper, we tackle this challenge by developing an offline RL and online fine-tuning approach, which we call policy-agnostic RL (***PA-RL***), to effectively fine-tune any policy class or backbone. Instead of directly training the policy parameters, ***PA-RL*** breaks down policy improvement into two steps. First, ***PA-RL*** directly optimizes *actions* (instead of policy parameters). Doing so decouples policy improvement from training the parameteric policy, which can now be done by maximizing the likelihood of "optimized" actions via supervised learning, a universal procedure applicable to most policy models. Concretely, to obtain these optimized actions, we first sample from the base policy several times to get multiple action candidates, and then take gradient steps with respect to the value function to improve those actions in the direction of maximizing Q-values. Ultimately we pick the action that attains the highest Q-value and distill it into the policy via supervised learning. These optimized action samples also replace the samples from the policy for training the value function. Note that while prior work does use supervised losses for policy training, our main contribution is to show that a single approach of this sort can effectively train multiple policy classes.

We empirically evaluate ***PA-RL*** in a number of domains, including simulated and real-world robotic manipulation tasks, with Gaussian, diffusion, and autoregressive categorical policies based on transformer backbones, in offline RL and online RL fine-tuning settings. Our results show that ***PA-RL*** achieves state-of-the-art performance, outperforming the next-best approach by 13% in aggregate over various domains. ***PA-RL*** produces the largest gains on long-horizon tasks that present multimodal offline data distributions (e.g. CALVIN (Mees et al., 2022) in our experiments), where a more expressive policy class beyond standard tanh-Gaussian is necessary for performance but has been challenging to use thus far. Most notably, ***PA-RL*** improves diffusion policies on two manipulation tasks by ***80-100% within only 1-2 hours*** of online RL fine-tuning on a *real* WidowX robot, and improves OpenVLA (Kim et al., 2024), a 7B parameter robot VLA foundation model, by ***75%*** after 1 hour of zero-shot trials and 40 minutes of online RL fine-tuning on the real robot. We also provide a recommended workflow for setting knobs in ***PA-RL*** according to the dataset and task structure, making it easy to use our approach. To our knowledge, our results are the first to fine-tune diffusion policies (Chi et al., 2023) (both in simulation and in the real world), autoregressive categorical transformer policies (in simulation), and a generalist robotic VLA policy, all via a single actor-critic RL method in the real world.

2 Related Work

Recent work (Park et al., 2024) shows that policy learning can be a big bottleneck in RL, especially when learning from offline data. A natural implication is to use expressive policy models, but RL algorithms are often tailored to a specific policy model, which might not work well in other settings. For e.g., while it is beneficial to reparameterize the policy gradient for Gaussian policies (Lillicrap et al., 2015; Haarnoja et al., 2018a; Fujimoto et al., 2018), doing so for diffusion policies (Wang et al.) or flows (Mazoure et al., 2020) can be quite unstable. Wang et al. for example requires using BC regularization and performs offline checkpoint selection against a BC loss. When learning with sub-optimal data, this seems problematic. As a result, Hansen-Estruch et al. (2023) resort to Q-function re-ranking on top of a frozen behavior policy, resulting in a less powerful policy improvement operator (e.g., compare EMaQ (Ghasemipour et al., 2021), which uses a similar reranking-based policy improvement operator to TD3+BC (Fujimoto & Gu, 2021) that performs better). The challenge with categorical policies is of a similar nature (Chebotar et al., 2023).

Motivated by these findings, we build a method for fine-tuning several policy models and evaluate it on diffusion and autoregressive categorical policies. Prior work that fine-tunes diffusion policies include: DPPO (Ren et al., 2024), which uses a two-layer diffusion-specific policy gradient loss, whereas our approach is applicable outside of diffusion policies; IDQL (Hansen-Estruch et al., 2023), which only utilizes action re-ranking akin to global optimization in ***PA-RL***, but does not distill it into the policy iteratively and hence results in poor fine-tuning performance; DIPO (Yang et al., 2023) and DDiffPG (Li et al., 2024), which only utilize the "action gradient" akin to local optimization in ***PA-RL***, but do not use offline data, and DQL (Wang et al.), which is unsatable as it naïvely runs reparameterized gradient. Psenka et al. learn diffusion policies via score matching, which Ren et al. (2024) find to be quite unstable. We also study how to fine-tune autoregressive categorical transformer policies in simulation. To our knowledge, we are the first to fine-tune all such policy models with one approach.

Our method ***PA-RL*** is related to prior approaches that pose "RL as supervised learning", and use weighted or filtered negative log likelihood (NLL) losses for training (Peng et al., 2019; Peters et al., 2010; Peters & Schaal, 2007; Oh et al., 2018; Abdolmaleki et al., 2018). There is a subtle but crucial difference: while these prior works largely use dataset or buffer actions for training, ***PA-RL*** samples *new* actions from the policy, optimizes them against the critic, and then trains the policy via NLL on these actions. This allows ***PA-RL*** to make aggressive updates, thus avoiding the "slowness" associated with weighted regression (Tajwar et al.; Kostrikov et al.; Park et al., 2024), while also inheriting its simplicity.

Action optimization from ***PA-RL*** also resembles prior work that uses CEM optimization to obtain actions from a Q-function in the online RL setting (Kalashnikov et al., 2018; Simmons-Edler et al., 2019), and supervised learning to improve a policy based on the obtained actions (Neumann et al.; Shao et al., 2022). Unlike ***PA-RL***, these methods do not make use of offline RL pre-training to train the proposal distribution, which we find to be important for attaining the best performance since the critic can produce erroneous values outside the support of the dataset seen so far (see Figures 15 and 16). We compare directly to representative methods in this line of work, and find that ***PA-RL*** outperforms CEM optimization in our results.

3 Problem Setup and Preliminaries

We operate in the standard RL setting, where we want to find the optimal policy in an MDP, $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s'|s,a)$ and $r(s,a)$ are the dynamics and reward functions, and $\gamma \in (0,1)$ is the discount factor. The optimal policy $\pi^* : \mathcal{S} \mapsto \mathcal{A}$ maximizes the discounted sum of rewards. The Q-function of a policy π is defined as $Q^\pi(s,a) = \mathbb{E}_\pi[\sum_t \gamma^t r(s_t,a_t)|s_0 = s, a_0 = a]$. We use Q^π_θ to denote the estimate of the Q-function of a policy π as obtained via a neural network with parameters θ.

Problem setting. We study two problem settings: **(a)** fully offline (Levine et al., 2020) and **(b)** offline-to-online fine-tuning (Nakamoto et al., 2024b). In **(a)**, we are given access to an offline dataset of experience, $\mathcal{D}_{\text{off}} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$, collected by a behavior policy, π_β, and want to learn a policy that attains best performance using this dataset. In **(b)**, we want to optimize the policy learned offline, π_{off}, using autonomously collected interaction data in $\mathcal{M}$, using as few online samples as possible.

Policy classes and parameterizations. We consider fine-tuning two types of policy classes: diffusion policies that produce continuous-valued actions, and autoregressive policies that produce categorical action tokens. Diffusion policies use a conditional DDPM (Ho et al., 2020)) to represent the distribution over actions conditioned on the state. A DDPM trains a diffusion step-dependant (t) denoising model, $\varepsilon_\phi(a,t|s)$. We detail the training objective and inference process in Appendix C. Note that while the DDPM loss in Eq. C.1 is not identical to a negative log likelihood loss, it is typically derived from a lower-bound approximation to it. This loss function is standard for training diffusion models. An autoregressive policy represents $\pi_\phi(a|s)$ as a product of conditional categorical distributions over each action dimension. We use a transformer architecture, along with uniform discretization into 128 bins per action dimension for simulation, and the OpenVLA (Kim et al., 2024) tokenizer for real-world experiments. We show the training objective in Appendix C.3.

Offline RL and online fine-tuning methods. The approach we build only affects policy optimization and inference, and retains the same training procedure for the critic as the base algorithm. We focus on two classes of methods: **(1)** methods that decouple critic updates from actor updates (e.g., IQL (Kostrikov et al.)), and **(2)** algorithms that sample from the actor to train the critic (e.g., Cal-QL (Nakamoto et al., 2024b)). Briefly, Cal-QL trains the Q-function to reduce temporal-difference (TD) error, with an additional regularizer that penalizes the learned Q-values on out-of-distribution (OOD) actions as long as Q-values are higher than $V^\mu(s)$, the values of a reference policy. Computing the Cal-QL loss (Equation C.4) requires sampling actions from the learned policy $\pi_\phi(\cdot|s)$, which is now an expressive policy class. In contrast, IQL trains the Q-function to regress to a higher expectile of the value function (Equation C.5), without querying any new actions from the learned policy.

4 *PA-RL*: Training Multiple Policy Classes with Value Functions

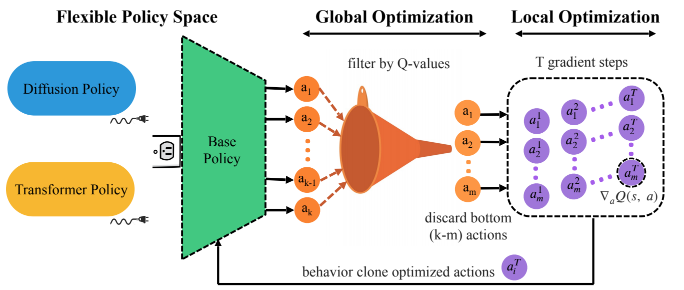


Figure 2: ***An overview of*** **PA-RL.** Instead of directly passing critic gradients through the policy parameters, ***PA-RL*** first "optimizes" actions via critic re-ranking and gradient ascent. Then, it trains the policy to mimic the most optimized action.

Our approach aims to stably and efficiently fine-tune multiple policy classes with RL, regardless of scale, class and output type. A prevalent approach for effective policy improvement is to use methods derived from off-policy actor-critic. This typically involves alternating between fitting an action-value Q-function and updating the policy parameters in the direction of larger predicted Q-values. Value learning normally treats the policy as a black-box that provides actions for the Bellman update. Policy improvement, on the other hand, requires optimizing the value function with respect to the policy parameters. For example, several algorithms estimate the gradient $\nabla_\phi Q(s, \pi_\phi(s))$ with respect to the parameters of the policy ϕ for policy improvement. Unfortunately, estimating this gradient can be tricky for the more "modern" policy models. For diffusion policies, propagating the policy gradient through the denoising chain can be unstable, often requiring extensive hyperparameter tuning per environment (Wang et al.) or truncating the gradient propagation after a subset of denoising steps (Ren et al., 2024). Similarly, for auto-regressive policies that operate on discrete action tokens, we must use a high-variance REINFORCE (Williams, 1992) policy gradient estimator.

Can we devise a simple yet universal approach to policy improvement? Our key insight to address this question is to train the policy using a loss function that is universally applicable to most deep learning machinery. One such loss is the supervised learning loss, e.g. negative log-likelihood (NLL). ***Our method (Fig. 2) builds on the idea*** that policy improvement can be performed via supervised learning, as long as the loss is applied on *optimized* actions. Hence, we can decompose the policy improvement step in two stages: **(1)** directly optimizing action samples produced by the policy, and **(2)** training the policy to imitate these "optimized" actions. This decomposition avoids computing $\nabla_\phi Q(s, \pi_\phi(s))$, or estimating a high-variance policy gradient. Since policy improvement is decoupled from policy training, we refer to this approach as ***"policy-agnostic RL"*** or ***PA-RL*** in short. We would expect this approach to inherit appealing attributes of scaling, reliability, and easy tuning of supervised learning losses. We now discuss our approach in more detail.

4.1 Stage I: Action Optimization

Given a state s, a policy $\pi_\phi(\cdot|s)$, and the current snapshot of the Q-function $Q_\theta(s, a)$, the objective in this stage is to obtain an action sample that optimizes the Q-function while not deviating too far from the support of seen actions at state s. We use $\pi_\phi(\cdot|s)$ as an initializer for the action optimization procedure. In the offline setting, doing so allows us to stay close to the support of seen actions, which is critical for attaining good performance and preventing value overestimation. During online RL fine-tuning of the offline initialization, this enables us to still leverage priors learned by the offline policy while adapting it to maximize returns on the task.

To produce an optimized action, we utilize a combination of different types of *action optimization* procedures. First, we consider ***global*** optimization[1] that samples multiple actions from the pre-trained policy, followed by discarding all but the top few actions with the highest Q-values under the trained critic for computational efficiency. Formally, let $\mathcal{A}_{\pi_\phi,k}(s) := \{a_0, a_1, \cdots, a_{k-1}\} \sim \pi_\phi(\cdot|s)$ denote k actions sampled from the current policy. Let $\widetilde{\mathcal{A}}_{\pi_\phi,k}(s) := \{a[0], a[1], \cdots, a[k-1]\}$ denote the set $\mathcal{A}_{\pi_\phi,k}(s)$ with actions put in order of their ranking obtained from the Q-function, i.e., $Q_\theta(s, a[i]) \geq Q_\theta(s, a[j])$, for $i \leq j$. Then, global optimization retains the following subset:

$$\widetilde{\mathcal{A}}_{\pi_\phi,m}(s) = \{a[0], a[1], \cdots, a[m-1]\}, \;\; m \leq k. \qquad \textbf{(global optimization)} \qquad (4.1)$$

Given this subset of the top m actions at a state s, we now ***locally*** improve each action, by performing gradient steps on the action in the direction of the gradient of the Q-function, directly on the action itself, without changing the policy parameters at all. This sort of a fine-grained local optimization is complementary to the fairly coarse global optimization procedure above as it perturbs the action to another one in its vicinity. Formally, given an action sample $a[i]$, we run T steps of gradient ascent starting from $a^0[i] := a[i]$ to obtain the locally optimal action, $a^T[i]$ as shown below.

$$\text{for } j = 0, \cdots, T-1, \;\; a^{j+1}[i] = a^j[i] + \alpha \nabla_a Q_\theta(s, a)\big|_{a=a^j[i]}, \qquad \textbf{(local optimization)}, \qquad (4.2)$$

where α is an appropriate learning rate that we choose for optimization. Applying both of these steps enables action optimization to leverage their complementary benefits while avoiding failure modes of either approach (e.g., not being fine-grained enough vs. being trapped in a local minimum). Let us denote the action set obtained by running local optimization on $\widetilde{\mathcal{A}}_{\pi_\phi,m}(s)$ as $\widetilde{\mathcal{A}}^T_{\pi_\phi,m}(s)$. A pseudocode for action optimization is in Algorithm 5, Appendix B.1.

4.2 Stage II: Policy Training via Supervised Learning

The second stage of ***PA-RL*** distills optimized actions into the learned policy model. Importantly, this distillation is performed via supervised learning losses that most deep learning models are designed to work with. While the most straightforward option is to simply take the action with the highest Q-value from the set $\widetilde{\mathcal{A}}^T_{\pi,m}(s)$ and maximize its likelihood under the learned policy $\pi_\phi(\cdot|s)$, another alternative is to distill all action samples from $\widetilde{\mathcal{A}}^T_{\pi_\phi,m}(s)$, but weight the contributions of different actions using the Q-value. We prescribe a simple strategy to choose between these methods (Appendix B.1). To accomplish this, we define a categorical policy distribution over the optimized action samples:

$$\pi^{\text{Opt}}_\phi(a|s, m) := \mathbb{I}\Big[a \in \widetilde{\mathcal{A}}^T_{\pi_\phi,m}(s)\Big] \cdot \frac{e^{Q_\theta(s,a)}}{\sum_{a'} e^{Q_\theta(s,a')}}, \qquad (4.3)$$

and train the policy $\pi_\phi(\cdot|\cdot)$ to match this distribution. To do so, we annotate all states in the offline dataset (including the replay buffer in online fine-tuning) with an action sample from $\pi^{\text{Opt}}_\phi(a|s, m)$, and maximize the likelihood of these actions under the policy, following best practices for supervised learning on this policy class. Formally, we denote this dataset of optimized actions as:

$$\mathcal{D}^{\text{Opt}}_{(\phi,\theta,m)} = \left\{ \left(s_i, \tilde{a}^{\text{Opt}}_i\right), \;\; \tilde{a}^{\text{Opt}}_i \sim \pi^{\text{Opt}}_\phi(a|s_i, m) \right\}_{i=1}^N.$$

For instance, if the policy π_ϕ is parameterized as a diffusion model, we follow the DDPM (Ho et al., 2020) behavior cloning (BC) objective, and train the policy to model the dataset $\mathcal{D}^{\text{Opt}}_{(\phi,\theta,m)}$. An exact loss function is shown in Appendix C.2. By using this loss instead of the reparameterized Q-function gradient, we avoid ever backpropagating through the denoising chain, and instead supervise every step of the chain independently. For auto-regressive transformer policies, we use cross-entropy loss objective for training.

Finally, we would like to note that while prior work does explore supervised learning losses for training policies (Peng et al., 2019; Abdolmaleki et al., 2018; Oh et al., 2018), the crucial differences

[1] "global" because it coarsely directs optimization to the vicinity of seen actions, not "global optimum".

between ***PA-RL*** and these prior techniques stem from the fact that action samples are drawn from the *current* policy, instead of a previous policy or a behavioral policy (Peng et al., 2019), which enables actions to deviate farther from the data manifold. That said, since these action particles are still drawn from the *current* snapshot of the learned policy, we are also able to ensure that global and local optimization do not move the actions too far away from the data manifold which can be problematic in offline RL settings. We show in our experiments that they have a substantial impact on performance and efficiency of RL training. Concretely, we outperform methods that use advantage-weighted regression (AWR) for policy extraction since ***PA-RL*** deviates farther away from the data manifold as well as CEM-based policy extraction (Kalashnikov et al., 2018; Simmons-Edler et al., 2019) which falls prey to out-of-distribution actions since it finds action particles that maximize the critic in any region of the action space.

4.3 Putting it All Together: Final ***PA-RL*** Algorithm

PA-RL can replace the policy improvement step in multiple offline RL and online fine-tuning algorithms. Hence, we instantiate ***PA-RL*** using two prominent methods in this space: Cal-QL (Nakamoto et al., 2024b) and IQL (Kostrikov et al.). ***PA-RL*** only modifies the policy improvement step and inference of each of these methods, while keeping the critic training as is. Since IQL training does not utilize policy backups, using ***PA-RL*** in conjunction with IQL is straightforward: simply replace the advantage-weighted regression (AWR) update with the above supervised learning update (e.g., Eq. C.2 for diffusion policies). For Cal-QL, where the policy $\pi_\phi(\cdot|s)$ is also used to generate action samples to perform the TD backup, we utilize the optimized action set $\widetilde{\mathcal{A}}^T_{\pi_\phi,m}$ for the Bellman backup. Formally, this means that instead of computing Bellman targets using an updated π_ϕ, we simply compute targets using the optimized policy $\pi_\phi^{\text{Opt}}(\cdot|\cdot, m)$ (Eq. 4.3) for Cal-QL. Pseudocode for the algorithm is shown in Alg. 6, Appendix B.1.

Implementation details. We initialize the base policy π_ϕ with BC. During offline pre-training, we keep the weights of the policy fixed, and during online fine-tuning we use the distillation scheme from Stage II. We provide a detailed list of hyperparameters and best practices for ***PA-RL*** in Appendix B.1. We run ***PA-RL*** with both state-based and image-based environments, where we utilize best design practices for the critic (Kumar et al., 2023). We also find that additionally including the action a appearing at a given state in the dataset into action optimization can sometimes be helpful. Finally, since gradient ascent is not guaranteed to improve the Q-value for a larger than ideal step size, we only execute a local update if it increases the Q-value. Code is available here.

5 Experimental Evaluation

The goal of our experiments is to understand the efficacy of ***PA-RL*** in fine-tuning policies of various parameterizations, classes, and types. Hence, we evaluate ***PA-RL*** and several prior approaches, in a number of benchmark domains that require learning policies from static offline data and then fine-tune them with limited online interaction in the MDP (offline-to-online fine-tuning (Nair et al., 2020)). We also study the hybrid RL problem setting (i.e., online RL with offline data in the replay buffer (Song et al., 2023; Ball et al., 2023)). We will also present results validating the efficacy of ***PA-RL*** on three real-robot manipulation tasks. Finally, we perform ablation experiments to understand the utility of different components of ***PA-RL***.

5.1 Results: Simulated Benchmarks

We first compare Cal-QL+***PA-RL*** with prior methods in the D4RL (Fu et al., 2020) and CALVIN (Mees et al., 2022) benchmarks. Since we report performance in both the offline RL and offline-to-online RL settings, we apply ***PA-RL*** on top of Cal-QL (Nakamoto et al., 2024b) and IQL (Kostrikov et al.), two common offline RL and offline-to-online fine-tuning algorithms, although most of our results use Cal-QL. We first demonstrate the efficacy of ***PA-RL*** in training diffusion policies and compare it to methods that train diffusion policies. Specifically, we compare ***PA-RL*** to: **(1)** Implicit Diffusion Q-Learning (IDQL, Hansen-Estruch et al. (2023)), which extends IQL to use diffusion policies via critic-based reranking; **(2)** Diffusion Policy Policy Optimization (DPPO, Ren et al. (2024)), which fine-tunes diffusion policies learned via imitation learning using PPO (Schulman et al., 2017); and **(3)** Diffusion Q-Learning (DQL, Wang et al.), which trains diffusion policies via a reparameterized policy gradient estimator from SAC (Haarnoja et al., 2018b).

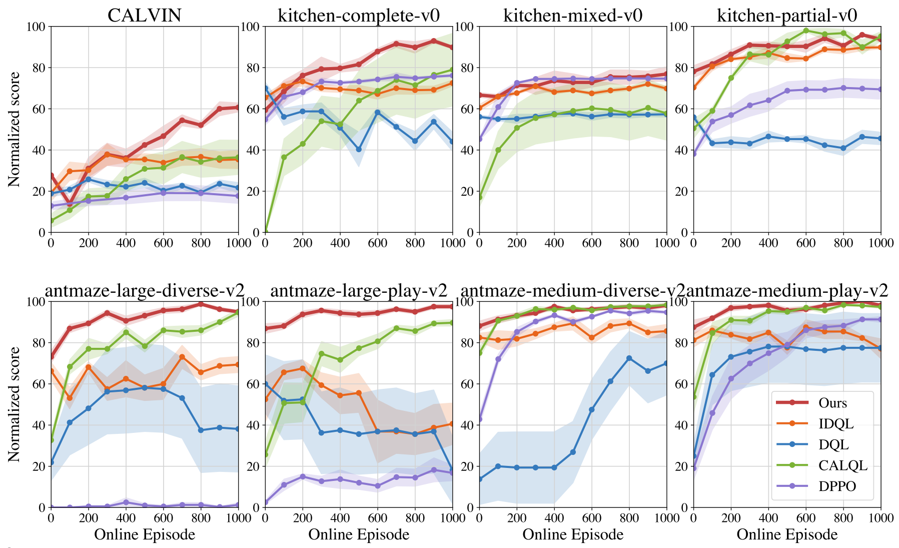


Figure 3: ***Learning curves of online fine-tuning*** with various methods. Observe that ***PA-RL*** + Cal-QL (red) largely always dominates or attains similar performance to the next best method. Other methods for fine-tuning diffusion policies (IDQL (Hansen-Estruch et al., 2023), DQL (Wang et al.), DPPO (Ren et al., 2024)) are a bit unstable, and perform substantially worse. Since DPPO is substantially more data inefficient, we plot it with different x-axis units: for kitchen each unit is 500 episodes (axis goes from 0 to 500k), for antmaze each unit is 100 episodes (axis goes from 0 to 100k) and for calvin each unit is 10 episodes (axis goes until 10k).

Domains and tasks. We study: **(1)** AntMaze tasks (Fu et al., 2020) that require controlling a quadruped ant to reach a goal location in four different maze layouts, with sparse binary rewards; **(2)** FrankaKitchen tasks (Gupta et al., 2020), which require solving a sequence of four manipulation tasks in a kitchen environment; and **(3)** the CALVIN benchmark (Mees et al., 2022) (D $\rightarrow$ D, with distractor objects), which requires solving a sequence of four manipulation tasks in a tabletop environment. The FrankaKitchen and CALVIN tasks require chaining different skills into a coherent long episode. That said, the CALVIN task is substantially harder than FrankaKitchen since policies must be learned directly from pixels and with offline play data generated via human teleoperation. Due to the diversity of offline data, we believe CALVIN should stress test the ability of any approach to effectively utilize the multimodal nature of diffusion policies. More details are in Appendix A.

Domain / Task	IDQL	DQL	DPPO	Cal-QL	***PA-RL*** (Ours)
CALVIN	19 → 35	19 → 22	13 → 18	6 → 36	**28** → **61**
Kitchen (-v0)					
complete	65 → 72	**70** → 44	55 → 76	19 → 57	59 → **90**
mixed	60 → 70	56 → 57	45 → 75	37 → 72	**67** → **77**
partial	70 → 90	56 → 46	38 → 69	59 → 84	**78** → **94**
Antmaze (-v2)					
large-diverse	66 → 69	22 → 38	0 → 1	33 → **95**	**73** → **95**
large-play	53 → 41	60 → 18	2 → 17	26 → 90	**87** → **98**
med-diverse	83 → 86	14 → 70	43 → 95	75 → **98**	**88** → **98**
med-play	81 → 77	25 → 78	19 → 91	54 → **97**	**88** → **98**
Aggregate	497 → 540	322 → 373	215 → 442	309 → 629	**568** → **711**

Table 1: ***Offline-to-online RL fine-tuning on benchmarks***. ***PA-RL*** + Cal-QL outperforms every other approach in aggregate, both in terms of the offline performance (left of →) and performance after 1k episodes of fine-tuning (right of →).

PA-RL *significantly improves learning efficiency and asymptotic performance of Cal-QL with diffusion policies.* We show results in Table 1 and learning curves in Figure 3. First, note that ***PA-RL*** attains higher offline performance than other methods that use diffusion policies, and also standard Cal-QL with a tanh-Gaussian policy. Fine-tuning ***PA-RL*** from the offline RL policy also leads to the best fine-tuned performance in aggregate. Concretely, the fine-tuning performance of ***PA-RL*** is 13% higher than the next best method, with a **69% improvement** on the hardest task CALVIN (where we must learn to control policies from images). This perhaps hints at the efficacy of ***PA-RL*** in effectively leveraging the increased capacity and expressive power of diffusion policies. Diving deeper, the learning curves in Figure 3 reveal a much stronger trend: the performance of ***PA-RL*** largely stays above the performance of all other methods throughout training. We also evaluate ***PA-RL*** in conjunction with IQL on the FrankaKitchen tasks in Table 2, and observe that ***PA-RL*** + IQL also outperforms standard IQL. Hence, ***PA-RL*** is broadly effective.

PA-RL *with hybrid RL.* Next, we run ***PA-RL*** on top of RL with prior data (RLPD Ball et al. (2023)), a method that incorporates offline data into online RL training, but without offline RL pre-training. In this case, we replace the standard Gaussian policy from RLPD with a diffusion policy and keep the critic randomly initialized. As shown in Table 2 (left), ***PA-RL*** is able to improve upon the imitation-learning performance of the diffusion policy after 200 episodes to substantially better performance values than when a Gaussian policy is used. This further corroborates the efficacy of ***PA-RL*** in efficiently leveraging the policy expressivity.

PA-RL **+ Cal-QL with autoregressive categorical policies.** Our next result shows that ***PA-RL*** is also effective in training transformer-based policies that model the distribution over actions autoregressively, using categorical distributions. This policy discretizes each dimension of the action independently into a set of 128 bins and trains an autoregressive model over the resulting action tokens. Observe in Table 2 (right) that ***PA-RL*** is also able to effectively improve autoregressive categorical policies with Cal-QL, and gets 26% better performance than using Gaussian policies on average across the three tasks considered. This establishes ***PA-RL***'s efficacy in fine-tuning policies of multiple classes.

Task	Gaussian RLPD @ 200	Diffusion *PA-RL* + RLPD @ 200	Gaussian IQL @ 1k	Diffusion *PA-RL* + IQL @ 1k	Gaussian Cal-QL @ 1k	Autoregressive *PA-RL* + Cal-QL @ 1k
partial	0 → 18	**58** → **73**	40 → 60	**62** → **75**	**59** → 84	33 → **95**
mixed	0 → 14	**58** → **58**	48 → 48	**69** → **73**	37 → 72	**42** → **84**
complete	0 → 34	**70** → **81**	57 → 50	**63** → **88**	**19** → 57	8 → **90**

Table 2: ***Running*** **PA-RL** ***with different policy classes and algorithms.*** In the hybrid RL setting, ***PA-RL*** **+ RLPD** is able to effectively improve a pre-trained diffusion policy without pre-training the critic. ***PA-RL*** **+ IQL** attains a similar performance on the FrankaKitchen domain as IDQL, proving our method can work with different objectives for the critic. **Autoregressive** ***PA-RL*** improves an autoregressive categorical policy by 224%.

5.2 Results: RL Fine-Tuning of Real Robot Policies

We now show that ***PA-RL*** *can* enable fine-tuning policies on a real robot, resulting in substantial success rate improvements of the pre-trained policy initialization within just 40 min to 2 hrs (i.e., 10-70 episodes) of real-world autonomous interaction. To our knowledge, *this is one of the first results to fine-tune diffusion policies and generalist policies on a real robot with value-based actor-critic RL.*

Real robot and task setup. We study three manipulation tasks (Figures 8, 1, and 9) on a WidowX-250 arm with six degrees of freedom and a single third-person mounted RGB camera. Our setup is inspired by Ebert et al. (2022); Walke et al. (2023). The policy controls the end-effector pose at a frequency of 5 Hz for a diffusion policy and 3 Hz for larger autoregressive policies. The tasks are as follows: **(a)** *"cup to drying rack"*, which requires grasping a plastic cup and placing it on the drying rack across the sink; **(b)** *"pot to sink"*, which requires picking and moving a toy pot from the drying rack to the sink; and **(c)** *"vegetable to sink"*, which requires grasping a toy cabbage and placing it on a plate in the sink. For tasks **(a)** and **(c)** the sink contains distractor objects, and for all tasks the position and rotation of the target object are randomized. We collect 10 teleoperated human demonstrations for task **(a)** and 20 for task **(b)** to pre-train a diffusion policy and the critic via Cal-QL + ***PA-RL*** that we then fine-tune online. For task **(b)**, we consider a "distribution shift" fine-tuning scenario, where the demonstrations show no distractors, but fine-tuning is supposed to be done with distractor objects. While seemingly benign, this sort of difference between pre-training and fine-tuning setups is still challenging as it leads to poor fine-tuning performance in many prior work that has attempted to run some form of real-robot RL (Kumar et al., 2023). For task **(c)** we consider improving performance of a generalist pre-trained policy (OpenVLA (Kim et al., 2024)), without any demonstrations. While OpenVLA has seen this environment in its dataset, the specific task is new, and the camera angle and background are not guaranteed to be the same. We collect 50 rollout episodes by zero-shot prompting OpenVLA with the instruction "put the vegetable on the plate", and use them to pre-train the critic with Cal-QL + ***PA-RL***. In each case, we fine-tune with a sparse reward function that is based on the detected positions of the target objects and the gripper state. After every robot trial, we perform a manual reset and randomization of the position and orientation of the object.

Task	DDPM (offline)	Iterated BC (online)	Cal-QL + ***PA-RL*** (offline → online)
(a) Cup to Rack	50%	50%	55% → **90%**
(b) Pot to Sink (w/ dist. shift)	50%	-	80% → **100%**

Table 3: ***Real-robot fine-tuning of diffusion policies.*** ***PA-RL*** improves the performance of a pre-trained diffusion policy on two tasks.

Results for fine-tuning diffusion policies. When running ***PA-RL*** with a diffusion policy on the real robot, we found it important to collect 20 warm-up episodes from the pre-trained offline RL policy before updating it. We also compare our approach to a filtered BC for autonomous improvement, based on Zhou et al. (but without goal conditioning) for task **(a)**. We omit this comparison for task **(b)** since the pre-trained diffusion policy did not produce any success under distribution shift for seeding iterative filtered BC. We also found the diffusion policy to be brittle on task **(b)**, and we are reporting only the best result it was able to get.

We observed significant performance improvement in both tasks when fine-tuning with ***PA-RL***, resulting in a 75-100% higher success rate within 40-110 minutes (see Table 3). We noticed a performance drop during the first 50 episodes in the *"cup to drying rack"* task, which was consistent with our findings in the CALVIN task and many other works studying online fine-tuning (Nakamoto et al., 2024b). We hypothesize that our expressive policy enables the robot to quickly recover and improve within the next 20 episodes.

Results for fine-tuning OpenVLA with *PA-RL*. Next we fine-tune OpenVLA, a 7B parameter generalist policy. To make finetuning such a large policy with real-world RL feasible we had to make some technical modifications, which are discussed in Appendix B.1. After 1 hour of zero-shot trials (where base OpenVLA obtained 40% success rate) and 40 minutes of online RL fine-tuning, the resulting fine-tuned OpenVLA policy obtained 70% success rate. We observe that the base OpenVLA policy often grasps the wrong object if the gripper is close to the distractor object. After fine-tuning, this error mode is significantly reduced. The fine-tuned policy often grasped the target object securely, whereas base OpenVLA sometimes let the object fall (see website for evaluation rollouts).

5.3 Ablation Studies and Controlled Experiments

Finally, we present experiments to understand the importance of each component of ***PA-RL***: **(1)** when are global optimization (Eq. 4.1) and local optimization (Eq. 4.2) important for policy improvement? **(2)** is using a pre-trained policy for action optimization initialization necessary, or would a strong optimizer (e.g., CEM) from a random initialization suffice?, and **(3)** is sampling actions from the current policy as opposed to a random initialization important for Stage II of ***PA-RL*** (addressed in App. E.6)?

Task	***PA-RL*** no global opt.	***PA-RL*** no local opt.	***PA-RL***
antmaze-large	$0 \rightarrow 0$	$74 \rightarrow 95$	$73 \rightarrow 93$
CALVIN	$215 \rightarrow 389$	$201 \rightarrow 357$	$234 \rightarrow 455$

Table 4: **The importance of global and local optimization.** We compare the performance of ***PA-RL*** + Cal-QL with and without global optimization as measured by average return obtained. Note that not using both local and global optimization leads to worse performance.

(1): Effect of global and local optimization. On the two tasks we study (antmaze-large-diverse and CALVIN), we make a number of interesting observations (see Table 4, and Figures 11 and 12 for number of gradient steps and base policy samples ablations). First, we find that both local and global optimization are critical for performance in some environment: on antmaze-large-diverse, global optimization is critical, but local optimization is not as important. On CALVIN, on the other hand, both components are important. This tells us that global optimization is important in general, but local optimization is perhaps only useful when we have a somewhat narrow dataset (e.g., action coverage on CALVIN is narrow; while action coverage on antmaze is quite high). Thus, *we recommend the general workflow* of always deploying global optimization when running ***PA-RL*** and strongly using local optimization when the dataset action distributions are somewhat narrow.

(2): CEM ablations. To study whether using a pre-trained policy to initialize action optimization close to the seen actions distribution is important, we replace the action optimization procedure in ***PA-RL*** for a CEM optimizer, which optimizes actions considering the whole action space. We observe (Figure 14) that CEM without a pre-trained policy initialization leads to both a very poor offline initialization and reduced sample efficiency during fine-tuning. Initializing CEM with either a fixed pre-trained policy or using the same policy distillation approach ***PA-RL*** uses improves the offline performance, but still shows poor fine-tuning performance. Additional CEM details and experiments are shown in App. E.5.

6 Discussion and Conclusion

In this paper, we developed ***PA-RL***, a method to train and fine-tune policies of various classes and backbones via offline RL and online fine-tuning methods. We showed state-of-the-art results across a number of simulation tasks and on real-robot tasks. Despite promising results, ***PA-RL*** still has some limitations. Most importantly, ***PA-RL*** requires sampling multiple actions from the policy, which is computationally expensive for large foundation policies. Understanding interplay between global and local optimization, and more generally, developing ways to use test-time compute for robotic policies better are promising directions towards mitigating this. Identifying how expressive policies help Q-function training is also an interesting future work.

7 Reproducibility Statement

In order to foster reproducibility of our work, we make our code available in this anonymous repo. Additionally, we outline implementation details in Appendix B.1 and Section 4. We have also provided more information about our experiments and settings in Appendix A and B.1 along with a listing of our hyperparameters.

References

A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations (ICLR)*, 2018.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *arXiv preprint arXiv:2406.11896*, 2024.

Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pp. 1577–1594. PMLR, 2023.

Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, et al. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *Conference on Robot Learning*, pp. 3909–3928. PMLR, 2023.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, pp. 02783649241273668, 2023.

Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. Bridge data: Boosting generalization of robotic skills with cross-domain datasets. *Robotics: Science and Systems*, 2022.

Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taiga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl. In *Forty-first International Conference on Machine Learning*.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, pp. 1587–1596, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pp. 2052–2062. PMLR, 2019.

Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Shane Gu. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. In *International Conference on Machine Learning*, pp. 3682–3691. PMLR, 2021.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning*, pp. 1025–1037. PMLR, 2020.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *arXiv*, 2018a. URL https://arxiv.org/pdf/1801.01290.pdf.

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

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL `http://arxiv.org/pdf/2106.09685`. cite arxiv:2106.09685Comment: Draft V2 includes better baselines, experiments on GLUE, and more on adapter latency.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *CoRL*, 2018.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Aviral Kumar, Anikait Singh, Frederik Ebert, Yanlai Yang, Chelsea Finn, and Sergey Levine. Pre-training for robots: Offline rl enables learning new tasks from a handful of trials. *RSS 2023; arXiv:2210.05178*, 2023.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Zechu Li, Rickmer Krohn, Tao Chen, Anurag Ajay, Pulkit Agrawal, and Georgia Chalvatzaki. Learning multimodal behaviors from scratch with diffusion policy gradient. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL `https://openreview.net/forum?id=vU1SiBb57j`.

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

Bogdan Mazoure, Thang Doan, Audrey Durand, Joelle Pineau, and R Devon Hjelm. Leveraging exploration in off-policy algorithms via normalizing flows. In *Conference on Robot Learning*, pp. 430–444. PMLR, 2020.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):7327–7334, 2022.

Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Mitsuhiko Nakamoto, Oier Mees, Aviral Kumar, and Sergey Levine. Steering your generalists: Improving robotic foundation models via value guidance. *Conference on Robot Learning (CoRL)*, 2024a.

Mitsuhiko Nakamoto, Simon Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. *Advances in Neural Information Processing Systems*, 36, 2024b.

Samuel Neumann, Sungsu Lim, Ajin George Joseph, Yangchen Pan, Adam White, and Martha White. Greedy actor-critic: A new conditional cross-entropy method for policy improvement. In *The Eleventh International Conference on Learning Representations*.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *International conference on machine learning*, pp. 3878–3887. PMLR, 2018.

Seohong Park, Kevin Frans, Sergey Levine, and Aviral Kumar. Is value learning really the main bottleneck in offline rl? *arXiv preprint arXiv:2406.09329*, 2024.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

J. Peters and S. Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *International Conference on Machine Learning (ICML)*, 2007.

Jan Peters, Katharina Mülling, and Yasemin Altün. Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pp. 1607–1612. AAAI Press, 2010.

Michael Psenka, Alejandro Escontrela, Pieter Abbeel, and Yi Ma. Learning a diffusion model policy from rewards via q-score matching. In *Forty-first International Conference on Machine Learning*.

Allen Z Ren, Justin Lidard, Lars L Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. *arXiv preprint arXiv:2409.00588*, 2024.

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

Lin Shao, Yifan You, Mengyuan Yan, Shenli Yuan, Qingyun Sun, and Jeannette Bohg. Grac: Self-guided and self-regularized actor-critic. In *Conference on Robot Learning*, pp. 267–276. PMLR, 2022.

Lucy Xiaoyang Shi, Joseph J Lim, and Youngwoon Lee. Skill-based model-based reinforcement learning. In *Conference on Robot Learning*, pp. 2262–2272. PMLR, 2023.

Riley Simmons-Edler, Ben Eisner, Eric Mitchell, Sebastian Seung, and Daniel Lee. Q-learning for continuous actions with cross-entropy guided policies. *arXiv preprint arXiv:1903.10605*, 2019.

Yuda Song, Yifei Zhou, Ayush Sekhari, Drew Bagnell, Akshay Krishnamurthy, and Wen Sun. Hybrid RL: Using both offline and online data can make RL efficient. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=yyBis80iUuU.

Fahim Tajwar, Anikait Singh, Archit Sharma, Rafael Rafailov, Jeff Schneider, Tengyang Xie, Stefano Ermon, Chelsea Finn, and Aviral Kumar. Preference fine-tuning of llms should leverage suboptimal, on-policy data. In *Forty-first International Conference on Machine Learning*.

Homer Walke, Kevin Black, Abraham Lee, Moo Jin Kim, Max Du, Chongyi Zheng, Tony Zhao, Philippe Hansen-Estruch, Quan Vuong, Andre He, Vivek Myers, Kuan Fang, Chelsea Finn, and Sergey Levine. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning (CoRL)*, 2023.

Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

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

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. In *International Conference on Learning Representations*.

Zhiyuan Zhou, Pranav Atreya, Abraham Lee, Homer Rich Walke, Oier Mees, and Sergey Levine. Autonomous improvement of instruction following skills via foundation models. In *8th Annual Conference on Robot Learning*.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183. PMLR, 2023.

Appendices

A ENVIRONMENT DETAILS

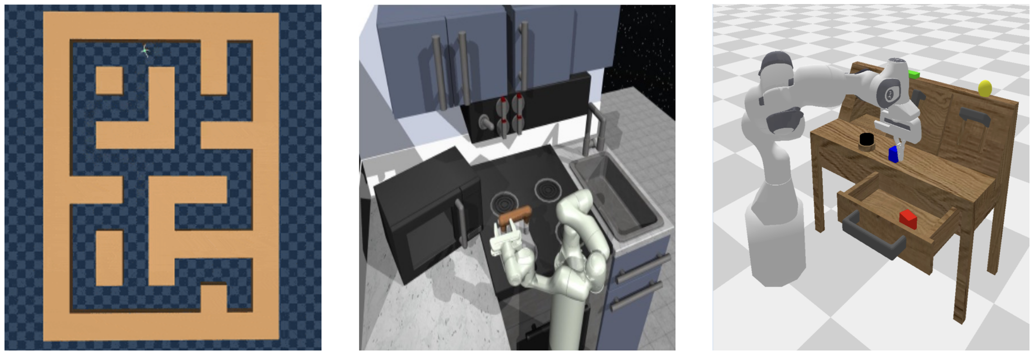

(a) Ant Maze Environment (b) Franka Kitchen Environment (c) Calvin Environment

Figure 4: ***Simulation Environments***

D4RL AntMaze: We test methods across two maze sizes (medium and large) and two dataset types (play and diverse). The diverse and large datasets differ in the starting locations and goal locations of trajectories. The diverse dataset consists of trajectories with random initial and goal locations, whereas play contains a set of specific hand-picked locations. The offline datasets for this benchmark have high coverage over states and actions.

D4RL FrankaKitchen: The FrankaKitchen benchmark contains three tele-operated datasets: kitchen-complete, which contains trajectories that fully solve all sub-tasks, but is 37 times smaller than the other datasets; kitchen-partial, where there are both trajectories that fully solve all sub-tasks, and undirected data that performs unrelated behaviors; and kitchen-mixed, where no trajectory solves all tasks, requiring exploration from the agent.

Calvin: We use the task setup introduced by Shi et al. (2023), in which the robot arm needs to complete four tasks (OpenDrawer, TurnonLightbulb, MoveSliderLeft, and TurnonLED), with the distinction that we only use image observations (i.e., the agent does not have access to proprioception nor object states). To ensure Markovian rewards, we make the reward function equal to the number of completed sub-tasks at each time-step (i.e., the agent only gets reward +4 if all sub-tasks are completed). The evaluation score for a trajectory is the maximum number of sub-tasks completed simultaneously at any single point in the trajectory.

Results for all environments and experiments are averaged over 5 random seeds and 32 evaluations per seed at each evaluation time-step (Figure 3). Scores are scaled from [0, 4] to [0, 100]. Shaded regions in the plots are standard errors over random seeds.

B Experiment Details

B.1 Details and hyperparameters for ***PA-RL***

[H]

Figure 5: Action Optimization $\pi^{\text{opt}}_{(\phi,\theta)}$

[1] base policy π_ϕ, Q-function Q_θ Sample actions from π to obtain $\mathcal{A}_{\pi_\phi,k}(s)$. Run global optimization for every state s to retain top m actions, $\widetilde{\mathcal{A}}_{\pi_\phi,m}(s)$ a in $\widetilde{\mathcal{A}}_{\pi_\phi,m}(s) \cup \{a_{\text{data}}(s)\}$ i in $\{1, \ldots, \text{T}\}$ $a^{(i)} \leftarrow a^{(i-1)} + \alpha \nabla_a Q_\theta(s, a^{(i-1)})$ $Q_\theta(s, a^{(i)}) \leq Q_\theta(s, a^{(i-1)})$ $a^{(i)} \leftarrow a^{(i-1)}$ **Break return** $\pi^{\text{opt}}_{(\phi,\theta)}$ computed via Equation 4.3

[H]

Figure 6: Cal-QL + ***PA-RL***

[1] BC loss $\mathcal{L}_{\text{policy}}$, e.g. $\mathcal{L}^{\text{ddpm}}_{\text{policy}}$ Pre-train policy π_ϕ via BC Initialize Q-function Q_θ step t in {1, ..., M} Train Q-function using Eq. C.4, but use optimized actions for TD targets

$$\theta_t = \theta_{t-1} - \eta_Q \nabla_\theta \mathcal{L}^{\text{Cal-QL}}_Q(\theta; \phi)$$

Distill optimized actions to policy

$$\phi_t = \phi_{t-1} + \eta_\pi \nabla_\phi \mathcal{L}_{\text{policy}}(\phi; \theta)$$

Collect new online rollouts: $a_t \sim \pi^{\text{opt}}_{(\phi,\theta)}; s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$

Action optimization hyperparameters: For all experiments shown in the paper except for ablations, the number of actions sampled from the base policy is **32**, which are filtered down to the top **ten**, and then propagated through the Q-function for **ten** gradient steps with gradient step size of **3e-4**. While we find that these values are robust to all the tested settings, these choices might require changes according to the characteristics of the available dataset and action space. For example, larger action spaces (such as bimanual manipulation) might require larger gradient step sizes or close-to-optimal datasets might perform well with significantly fewer action samples and gradient steps.

OpenVLA fine-tuning implementation details: Implementation wise, we had to make some modifications to make it feasible to fine-tune such a large policy autonomously with real-robot RL. First, we discuss some important design decisions for the offline RL stage that trains only a critic. In this phase, we implemented a cache of actions to store actions OpenVLA would take in each state by sampling 16 actions from this generalist policy. This cache enables offline RL critic training in Cal-QL with an OpenVLA policy to still run at similar speeds as a much smaller policy because actions in this cache can be reused for TD backups in the offline RL phase. When coupled with the action optimization phase from ***PA-RL***, using optimized action particles in the TD backup still allows for policy improvement, even though the OpenVLA policy is not updated in this offline RL phase due to computational cost associated with it. **During online fine-tuning,** we now update the parameters of the generalist OpenVLA policy. Concretely, we distill optimized actions into OpenVLA via LoRA (Hu et al., 2021) fine-tuning with rank=32 to speed up training. After policy distillation epochs, we recompute the cache of OpenVLA actions with 12 distributed processes, and use these cached actions for critic training. Aside from using half the number of samples from the base policy due to memory constraints and reduced learning rate for stability, all hyperparameters are the same as in the experiments above.

Distributional critic: When any of the random seeds in a domain showed instability in the critic pre-training (i.e. had exploding Q-values) we switched the critic from an MLP that predicts the continuous action value to a distributional critic and trained with the HL-Gauss loss (Farebrother et al.) instead. Specifically, we switched to a distributional critic for the $\mathrm{AntMaze}$ and $\mathrm{FrankaKitchen}$ domains, and we trained with MSE on Calvin and the real robot experiments.

Sampling vs argmax for action candidate selection: For environments in which CQL/Cal-QL used the max-backup version of Q-target calculation (namely, all 4 $\mathrm{AntMaze}$ environments), we find that taking the argmax of π^{Opt}_ϕ during inference yielded slightly faster convergence than sampling from the considered actions. During **policy distillation**, to decide whether to imitate only the argmax of π^{Opt}_ϕ or whether to imitate all samples, we keep track of the variance of action candidate Q-values during pre-training. If the variance is too small, we find that training only with the argmax performs better. Otherwise, training with samples from the categorical distribution yields slightly better results.

Environment	Policy Training Argmax Action	Policy Training Softmax
kitchen-partial-v2	89.375	95.3125
kitchen-complete-v2	90.3125	94.53125
kitchen-mixed-v2	67.96875	75.15625
CALVIN	60.6771	46.5625

Table 5: ***Comparison between doing policy distillation with samples from π_ϕ^{Opt} and only the argmax.***

Environment	STD of Action Candidate Q-values
kitchen-partial-v2	1.56
kitchen-complete-v2	2.66
kitchen-mixed-v2	11.54
CALVIN	0.02

Table 6: ***Standard deviation of the Q-values of action candidates ($\widetilde{\mathcal{A}}^T_{\pi,m}$) during pre-training.***

Details for image-based domains: Following Yarats et al. we augment image observations with random shift augmentations of 4 pixels. To mitigate the failure case in which the Q-values for different actions on the same state collapse to the same value, we use the Q-function architecture introduced by Kumar et al. (2023). At every layer of the critic MLP, we concatenate the action vector to the inputs, so that the network places more importance to the actions.

Base policy hyperparameters: We use the same Diffusion Policy architecture and training hyperparameters as IDQL (Hansen-Estruch et al., 2023). In particular, we use batch size 1024, T=5 diffusion steps, cosine beta schedule, the LN_Resnet architecture with hidden dimension size = 256 and n = 3 blocks. We pre-train the diffusion policy with learning rate decay but with a constant learning rate during fine-tuning. For image-based domains (CALVIN and real robot) we use a ResNet 18 encoder trained from scratch. For the auto-regressive transformer policy, we discretize each action dimension into 128 bins, and do not use discretization for the state observations. We use a transformer architecture with 4 layers, 256 hidden size, 8 heads, and learning rate 3e-5.

Reward scale and bias: To maintain consistency of hyperparameters across all domains, we bias all rewards from the offline dataset and replay buffer such that the maximum possible timestep reward is zero, and other possible rewards are negative. In particular, we use bias = -1 for AntMaze and real robot, and -4 for FrankaKitchen and CALVIN.

Cal-QL hyperparameters: We carry over most hyper-parameter choices from Cal-QL: critic architecture and learning rate, discount, mixing ratio.

Table of hyperparameters:

Critic LR	3e-4
Discount γ	0.99
Critic batch size	256
Base policy batch size	1024 (Diffusion Policies), 256 (Transformers)
CQL α	0.005 (AntMaze & Kitchen), 0.01 (CALVIN & Real robot)
Mixing ratio	0.25 (Kitchen), 0.5 (Rest)
Optimizer (critic and base policy)	Adam (Kingma & Ba, 2015)
Critic pre-training grad steps	1e6 (AntMaze), Rest: 5e5
Base policy grad steps	Diffusion policies: 3e6 Transformers: 2e6
Base policy distillation learning rate	1e-5 (kitchen), 5e-5 (Rest)
Critic hidden layer sizes	[256, 256, 256, 256] (AntMaze), [512, 512, 512] (Rest)

B.2 Details and hyperparameters for baselines

IDQL We use the IDQL-Imp version of IDQL, in which the Q-function, the value function, and the diffusion policy are fine-tuned with new experiences. We use the same network architectures as ***PA-RL***. For the IQL τ expectile, we use 0.9 for $\mathrm{AntMaze}$ and 0.7 for everything else. We remark that results for IDQL are not entirely comparable to their paper because Hansen-Estruch et al. (2023) used the "-v0" antmaze datasets from D4RL, but Fu et al. (2020) deprecated the "-v0" datasets in favor of "-v2" due to a bug associated with termination flags in -v0 datasets.

DQL We extensively tuned DQL for fine-tuning in the absence of any official fine-tuning results. For the main η RL weight hyperparameter, we performed an environment-specific hyperparameter search at the pre-training phase, selected the one that performed best, and then kept η fixed for fine-tuning. For $\mathrm{AntMaze}$ tasks we tried $\eta = \{0.05, 0.5, 1, 3, 3.5, 5, 7, 9, 11, 13, 15\}$. We chose $\eta = 11$ for large-diverse, $\eta = 15$ for large-play, $\eta = 9$ for medium-diverse, and $\eta = 7$ for medium-play. For $\mathrm{FrankaKitchen}$ tasks we tried $\eta = \{0.005, 0.01, 0.05, 0.1\}$. For partial, complete, and mixed, we chose $\eta = 0.005$. For CALVIN we tried $\eta = \{0.01, 0.1, 1, 5, 10, 15\}$. We picked $\eta = 0.01$. For offline checkpoint selection, we follow the original methodology of selecting the checkpoint with second lowest DDPM loss, saving checkpoints every 50k gradient steps.

Cal-QL Since we branch off our hyperparameter choices from Cal-QL, this baseline shares most of ***PA-RL***'s hyperparameters. We used (256, 256) hidden sizes for the policy architecture for every environment.

DPPO We train a diffusion-based PPO policy based on a DPPM model pretrained on an offline dataset in each simulated task. For the state-based tasks $\mathrm{AntMaze}$ and $\mathrm{FrankaKitchen}$, we train DPPO-MLP with 40 parallelized environments and an action chunking size of 6 for $\mathrm{AntMaze}$ and 8 for $\mathrm{FrankaKitchen}$. For the pixel-based task CALVIN, we train DPPO-ViT-MLP with 50 parallelized environments and an action chunking size of 4.

RLPD For Table 2, we train a gaussian policy from scratch with UTD ratio of 10 (same as with Diffusion ***PA-RL*** + RLPD), critic ensemble size ten, and critic ensemble subsample size of two.

C Training objective definitions

Training objectives for the base policy: For Diffusion Policies we use the Denoising Diffusion Probabilistic Model (DDPM, Ho et al. (2020)) training objective. The loss function for the step-dependent (t) denoising model, $\varepsilon_\phi(a, t|s)$, is given by:

$$\mathcal{L}^{\text{ddpm}}(\phi) = \mathop{\mathbb{E}}_{\substack{t\sim\mathcal{U}(1,K)\\ \epsilon\sim\mathcal{N}(0,I)\\ (s,a)\sim\mathcal{D}}} \left[\|\epsilon - \epsilon_\phi(\sqrt{\bar{\alpha}_i}a + \sqrt{1-\bar{\alpha}_i}\epsilon, s, t)\|\right] \tag{C.1}$$

where, given a fixed variance schedule $\beta_1, \ldots, \beta_K$ for the forward diffusion process, α_t is defined as $1 - \beta_t$, and $\bar{\alpha}_t$ as $\prod_{s=1}^{K} \alpha_s$. The inference process to obtain final "denoised" actions is as follows: we start with a random sample $a_K \sim \mathcal{N}(0, I)$, and iteratively denoise the sample such that $a_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(a_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon_\phi(a_t, s, t)\right) + \sqrt{\beta_t}\boldsymbol{z}$, where $\boldsymbol{z} \sim \mathcal{N}(0, I)$ if $t > 1$ and 0 otherwise, for K total denoising steps.

When performing policy distillation with a DDPM model, we modify the loss function to imitate optimized actions $\mathcal{D}^{\text{Opt}}_{(\phi,\theta,m)}$ as follows:

$$\mathcal{L}^{\text{ddpm}}_{\text{policy}}(\phi;\theta) = \mathop{\mathbb{E}}_{\substack{t\sim\mathcal{U}(1,T),\epsilon\sim\mathcal{N}(0,I),\\ (s,a)\sim\mathcal{D}^{\text{Opt}}_{(\phi,\theta,m)}}} \left[\|\epsilon - \epsilon_\phi(\sqrt{\bar{\alpha}_i}a + \sqrt{1-\bar{\alpha}_i}\epsilon, s, t)\|\right]. \tag{C.2}$$

For autoregressive policy training, we use the following negative log likelihood loss of tokenized actions:

$$\pi_\phi(a|s) = \Pi_{i=1}^{d-1}\pi_\phi(\text{tokenize}(a_i)|s, a_{0:i-1}). \tag{C.3}$$

Training objectives for the critic: We instantiate ***PA-RL*** on top of two Offline RL and online fine-tuning algorithms, Cal-QL (Nakamoto et al., 2024b) and IQL (Kostrikov et al.). The Cal-QL critic training objective is given by:

$$\mathcal{L}_Q^{\text{Cal-QL}}(\theta;\phi) = \alpha\Big(\mathop{\mathbb{E}}_{\substack{s,a^{(\mathcal{D})}\sim\mathcal{D}\\ a^{(\pi)}\sim\pi_\phi(\cdot|s)}}\Big[\max(Q_\theta(s,a^{(\pi)}), V^\mu(s)) - Q_\theta(s,a^{(\mathcal{D})})\Big]\Big) + \frac{1}{2}\mathop{\mathbb{E}}_{s,a,s'\sim\mathcal{D}}\Big[(Q_\theta(s,a) - \mathcal{B}^\pi\bar{Q}(s,a))^2\Big]\,. \tag{C.4}$$

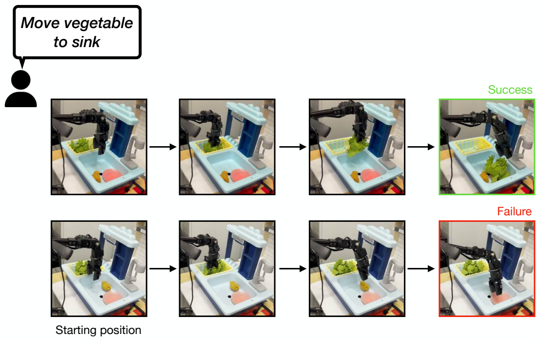
Where Q_θ is the learned critic, $\bar{Q}$ is the delayed target Q-function, and $\mathcal{B}^\pi\bar{Q}(s,a)$ is the backup operator: $\mathcal{B}^\pi\bar{Q}(s,a) = r(s,a) + \gamma\mathbb{E}_{a'\sim\pi_\phi(a'|s')}[\bar{Q}(s',a')]$.

The IQL critic training objective is given by:

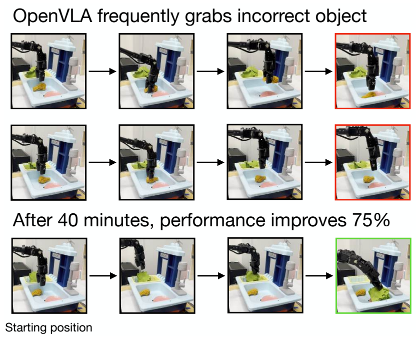
$$\mathcal{L}_V^{\text{IQL}}(\psi) = \mathop{\mathbb{E}}_{(s,a)\sim\mathcal{D}}\Big[L_2^\tau(Q_{\hat{\theta}}(s,a) - V_\psi(s))\Big]$$
$$\mathcal{L}_Q^{\text{IQL}}(\theta) = \mathop{\mathbb{E}}_{(s,a,s')\sim\mathcal{D}}\Big[(r(s,a) + \gamma V_\psi(s') - Q_\theta(s,a))^2\Big] \tag{C.5}$$

Where $L_2^\tau(u) = |\tau - \mathbb{1}(u < 0)|u^2$ is the expectile loss, and $\hat{\theta}$ are the target parameters for the Q-function.

D Filmstrips for Real-World Fine-Tuning of OpenVLA with ***PA-RL***



(a) Zero-shot language-based trials with OpenVLA


OpenVLA frequently grabs incorrect object
After 40 minutes, performance improves 75%
Starting position

(b) Online fine-tuning with ***PA-RL***

Figure 7: **Filmstrips of the manipulation task we fine-tune OpenVLA on.** (Left) the new task, “vegetable to sink”, requires identifying the vegetable from the distractor (a fried chicken wing), grasping it, and placing it on the pink plate. We collect 50 trials by zero-shot prompting OpenVLA to solve the task. 40% of the trials are successful. (Right) we deploy ***PA-RL*** to improve OpenVLA for this task, interacting on the real-robot. We observe that OpenVLA frequently grasps the distractor object instead of the vegetable. After 40 minutes of wall clock time, we evaluate the resulting fine-tuned policy. OpenVLA + ***PA-RL*** attained a 70% success rate.

E Additional Figures

E.1 Real Robot Fine-Tuning on Task (a)

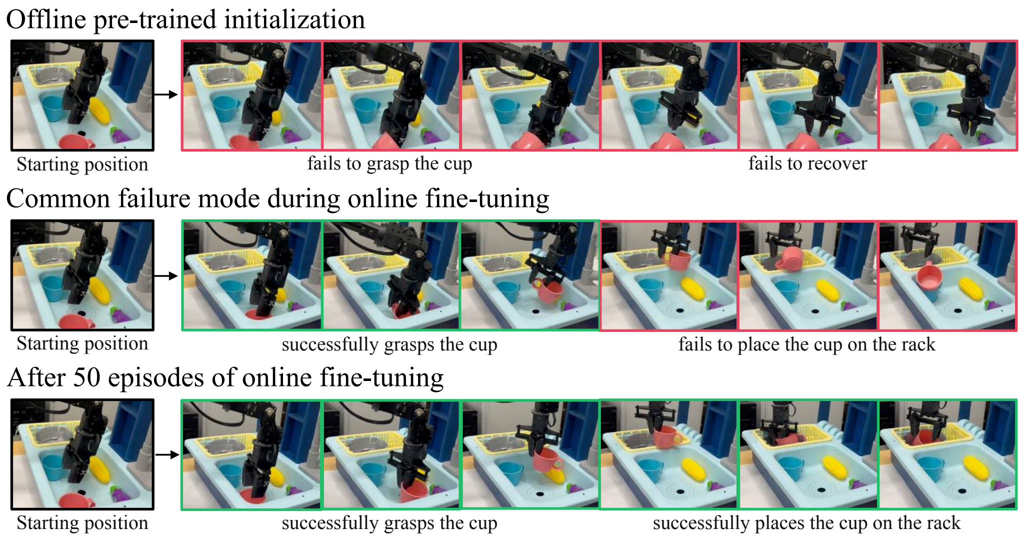

Offline pre-trained initialization
Starting position
fails to grasp the cup
fails to recover
Common failure mode during online fine-tuning
Starting position
successfully grasps the cup
fails to place the cup on the rack
After 50 episodes of online fine-tuning
Starting position
successfully grasps the cup
successfully places the cup on the rack

Figure 8: **Evolution of learned behaviors during online fine-tuning of diffusion policies with *PA-RL* on task (a).** The offline BC policy (in red) fails to even grasp the cup. During intermediate online interaction episodes (in yellow), it successfully grasps the cup, but fails to place it on the rack. After 50 episodes (in green), it learns to successfully grasp the cup and place it on the rack.

E.2 Real Robot Fine-tuning on task (b)

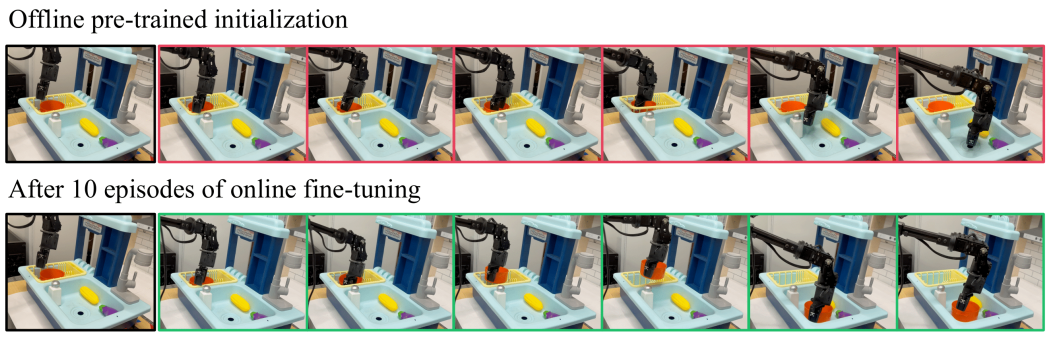

Offline pre-trained initialization
After 10 episodes of online fine-tuning

Figure 9: **Evolution of learned behaviors during autonomous online finetuning of *PA-RL* on task (b) on a difficult pot placement**. The offline initialization (in red) fails to grasp the pot, and gets stuck when attempting to move it to the sink. After only 10 online fine-tuning episodes (in green), ***PA-RL*** learns to successfully complete the task.

E.3 Learning Curves for Auto-Regressive Transformers and IQL with ***PA-RL***

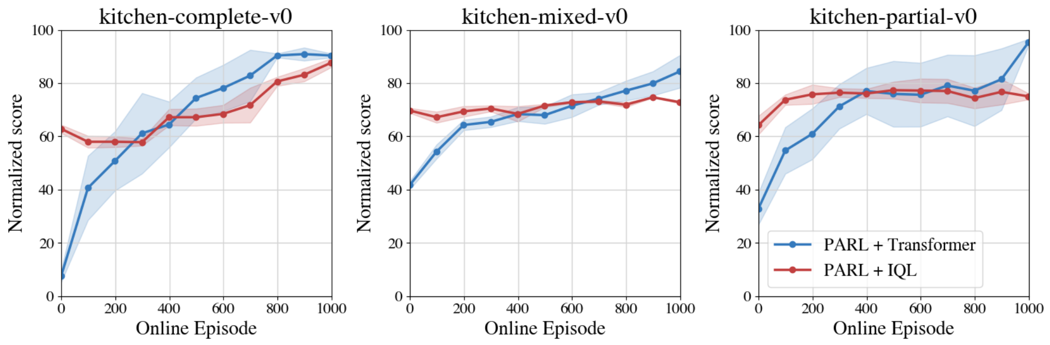


Figure 10: **Learning Curves for auto-regressive transformer based policies with *PA-RL* and Cal-QL, and diffusion policies with *PA-RL* and IQL.**

E.4 Local and Global Optimization Ablation Experiments

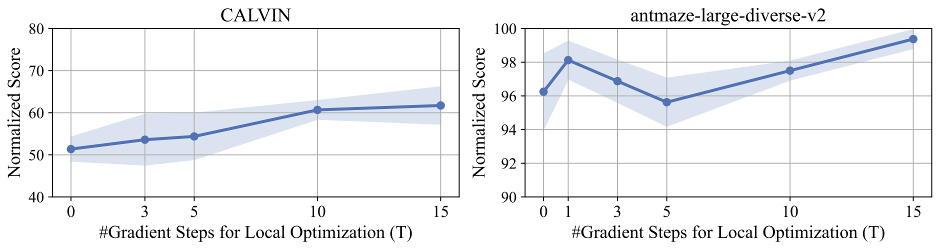


Figure 11: **Ablation for the number of gradient steps for local optimization (T).** We plot the evaluation performance for ***PA-RL*** + Diffusion Policy at the end of a fine-tuning budget of 1k episodes on CALVIN (left) and antmaze-large-diverse-v2 (right), taking different numbers of gradient steps during the Local Optimization procedure. We chose to analyze the effect of local optimization on these two tasks because they sit on opposite sides of the data coverage spectrum: CALVIN features relatively little coverage over actions, since the provided dataset is "play data", while antmaze-large-diverse-v2 provides high-coverage over actions (as measured by delta x, delta y, which is more relevant to the task). (Left) CALVIN benefits significantly from increased number of gradient steps, getting up to 20% increase in final performance compared to taking no gradient steps. (Right) antmaze-large-diverse-v2 already reaches 96% success rate without taking any gradient steps (i.e., without the local optimization step). We hypothesize that because of the high-coverage, using global optimization with a large-enough number of samples from the base policy already recovers good actions.

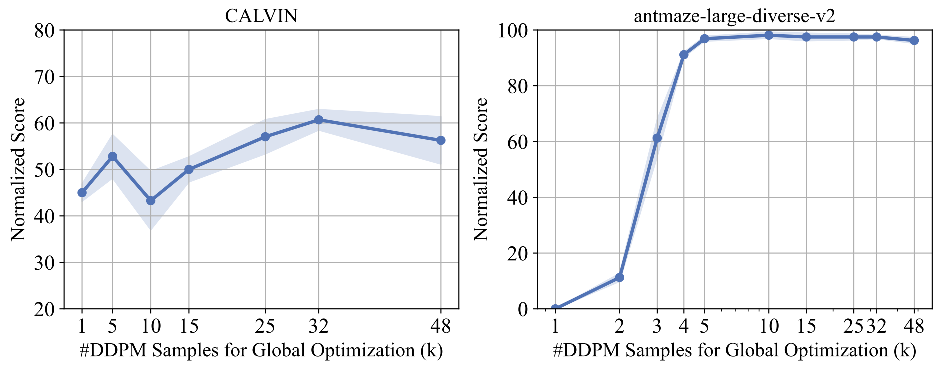


Figure 12: **Ablation for the number of samples from the base policy (k).** We plot the evaluation performance for ***PA-RL*** + Diffusion Policy at the end of a fine-tuning budget of 1k episodes on CALVIN (left) and antmaze-large-diverse-v2 (right), sampling different number of actions from the base policy to generate action candidates both for policy distillation and during inference. (Left) CALVIN benefits significantly from increased number of samples from the base policy, attaining 33% higher normalized score when taking 32 samples (the default value used for ***PA-RL***) from the policy compared to only 1 sample. (Right) antmaze-large-diverse-v2 exhibits a sharp decrease in final performance when taking fewer than 5 samples from the base policy.

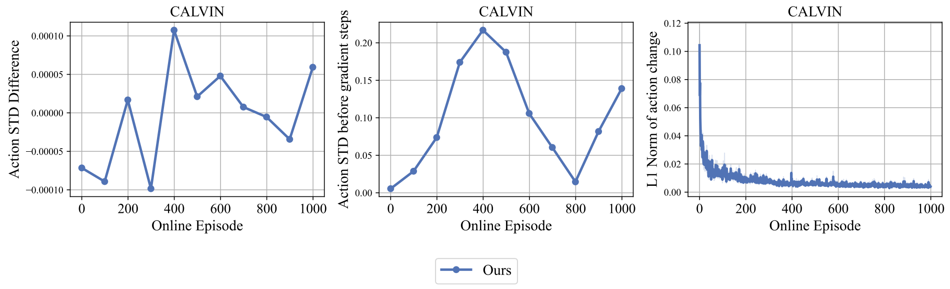


Figure 13: **Analysis of the effects of local optimization.** To test whether local optimization results in duplicated action samples, we plot the difference between the standard deviation of action samples before and after taking gradient steps (left) during evaluation episodes on the CALVIN task throughout fine-tuning. The difference in standard deviations is extremely low throughout training. Further, to ensure action samples were not largely duplicates to begin with, and to put the value scale into perspective, we plot the raw standard deviation of action samples before taking gradient steps (center). Standard deviation of actions changes by less than 0.1% on average during training. Thus, local optimization does not lead to action sample duplication. (Right) we plot the L1-Norm of the change in actions by the local optimization procedure (i.e. the L1 norm of the difference in actions before and after the gradient steps). The biggest direct effect on actions happens in the beginning of fine-tuning, and it quickly decays throughout online training. Note that because of policy distillation, action changes from the local optimization step are compounding (i.e., the actions before applying the gradient steps have already been optimized in past iterations). This might explain the decay in action changes from local optimization.

E.5 CEM OPTIMIZER COMPARISONS

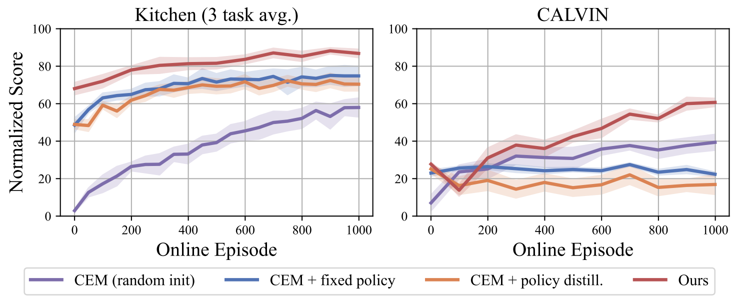


Figure 14: **Action optimization vs. CEM**. We replace ***PA-RL***'s method for sampling actions during both inference and Cal-QL critic training with 3 different CEM flavors: random initialization, and using the same BC Diffusion Policy used in ***PA-RL***, either with fixed weights or with the same policy distillation as ***PA-RL***.

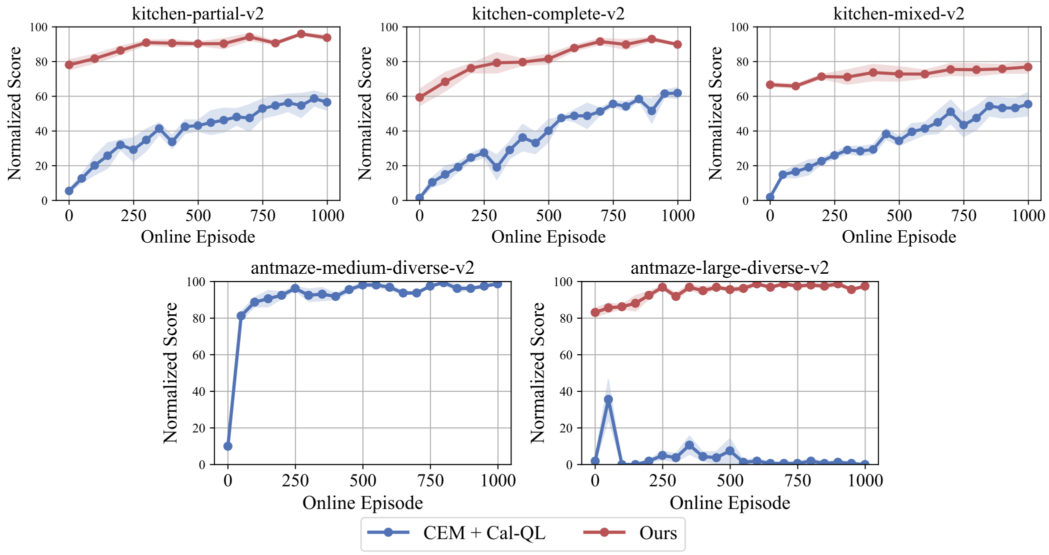


Figure 15: ***Comparison with CEM optimizer.*** Instead of using the action optimization procedure detailed in Section 4, any time the Cal-QL algorithm queries the policy we perform a Cross-Entropy Method optimization process to obtain actions. We use the same CEM hyper-parameters as Simmons-Edler et al. (2019), and maintain the Cal-QL hyper-parameters and architectures as ***PA-RL***. For all tested environments, the performance after pre-training (i.e. at step 0, before taking any online steps) is at or close to 0, and performance improves over the course of fine-tuning, but remaining well below PA-RL with a diffusion policy.

(2): CEM optimizer. The action optimization procedure in ***PA-RL*** seeks to find actions that maximize predicted Q-values within a limited budget for both global and local optimization. This implicitly constrains the action optimization procedure to not deviate substantially far away from the current policy, which is initialized via pre-training on offline data. To understand whether using a pre-trained policy is helpful, or whether simply maximizing Q-values is enough, we replace action optimization with a more powerful optimization procedure, cross-entropy method (CEM), which iteratively refines actions by keeping the top few according to the learned critic, but starts from random actions. Figure 15 shows a comparison in the Antmaze and Kitchen domains. CEM initialized from scratch performs very poorly after pre-training for all tested tasks and reaches significantly lower asymptotic performance. Figure 16 shows that this is because CEM finds actions where the Q-function greatly overestimates values. This implies that not deviating too far from the data is also important for ***PA-RL***.

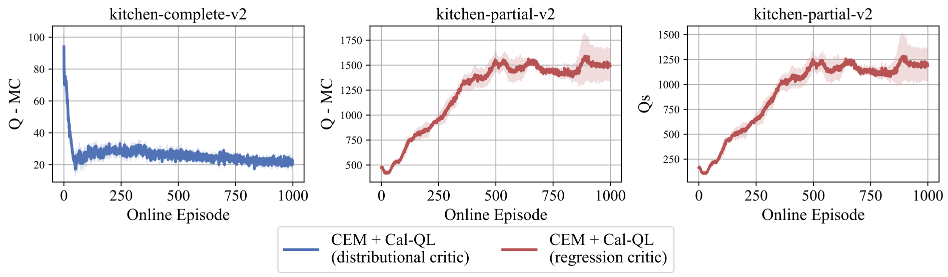


Figure 16: **CEM exploits Q-function over-optimism.** (Left) We plot the difference between predicted Q-values of CEM actions, and the Monte-Carlo discounted returns that those actions actually got, on kitchen-complete-v2, a task whose dataset contains optimal actions. The critic is trained in the same manner as in Figure 15. We observe that at the beginning of fine-tuning, predicted Q-values are much higher than the MC returns, even much higher than the predicted Q-values further into training, when task performance is much higher (see Figure 15). This points to the fact that the CEM optimizer is able to find actions that maximize the Q-function, but are not actually good. (Center) We repeat the same experiment but with a regression-trained critic instead of a distributional critic trained with HL-Gauss. The distributional critic bounds the predicted values by design, which limits over-estimation. By training a Cal-QL critic without a fixed value range (on kitchen-partial-v2), we see much larger over-estimation of Q-values. Predicted Q-values become large positive numbers (right), where rewards are always non-positive.

Previous works have also considered initializing the CEM optimizer using a parameterized policy instead of starting from scratch as we did in Figure 15 (Shao et al., 2022; Neumann et al.). We compare ***PA-RL*** against two versions of CEM + Cal-QL in Figure 17: initializing the CEM procedure with a fixed diffusion policy pre-trained with imitation learning (using the same checkpoint used to start ***PA-RL***); and, similarly to ***PA-RL***, performing policy distillation after every online episode with "optimized actions", where these action targets come from CEM instead of our Action Optimization procedure from Section 4.1. On kitchen-complete-v2 and CALVIN, tasks which exhibit lower coverage of the action space and highly multimodal datasets, ***PA-RL*** still significantly outperforms CEM. We believe that the data composition on these domains is hurting CEM performance, as CEM can *average* the different modes of behavior. Concretely, a CEM iteration consists of selecting a new action distribution using the mean and variance of the highest ranked under the critic (i.e., CEM assumes a Gaussian distribution). If the pre-trained policy is multimodal, this averaging operation can lose multimodality, and result in a new action "lying in the middle" of two modes of the pre-trained policy, but which itself is less likely and out-of-distribution under the pre-trained policy.

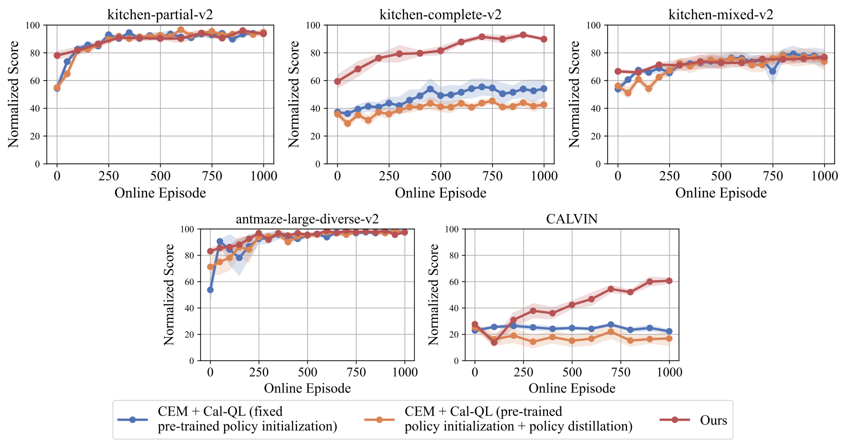


Figure 17: **Comparison with CEM optimizer with a pre-trained policy initialization.** We compare to using a CEM optimization procedure where the initial population of actions comes from the same pre-trained policy used for ***PA-RL***. ***PA-RL*** results in 42% better offline-only performance across tested domains. In antmaze-large-diverse-v2, kitchen-partial-v2, and kitchen-mixed-v2, CEM quickly catches up and ends with very similar asymptotic performance. In kitchen-mixed-v2 and CALVIN ***PA-RL*** significantly outperforms CEM, with 66% and 172% better performance respectively. kitchen-complete-v2 and CALVIN have lower coverage of actions in their datasets, and CALVIN has highly multi-modal data. We hypothesize these dataset characteristics, which are highly common in real-world robotics datasets, are hurting CEM performance, since CEM can average the different modes of behavior, resulting in OOD actions. Further, CEM lacks an equivalent of the local optimization step to direct exploration towards actions the critic rates highly.

E.6 Self-Imitation Learning comparison

(3): Effect of using on-policy actions over dataset actions for policy distillation. ***PA-RL*** is similar to self-imitation learning (Oh et al., 2018) or advantage-weighted regression (AWR) (Peng et al., 2019) if we only optimize weighted log likelihoods on one action sample from a stale policy (e.g., the behavior policy of the offline dataset or the replay buffer), turn off local optimization, and the policy distillation loss is weighed by positive action advantages (or exponentiated advantages). We compare against this approach on antmaze-large-diverse-v2 in Figure 18. We generically refer to this approach as self-imitation learning (SIL) and note that it fails to achieve any positive performance on this task. As shown in Figure 12, taking multiple samples from the base policy is critical for antmaze-large-diverse-v2, which could explain the poor performance of methods that only sample one action from a stale policy for learning, despite using advantage weights.

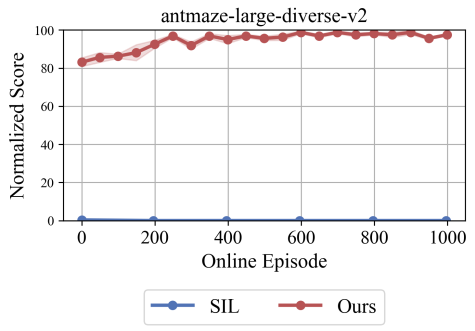


Figure 18: **Comparison with training on dataset actions** on antmaze-large-diverse-v2. For fairness, critic pre-training and fine-tuning are done in the same manner as ***PA-RL***.

E.7 Comparison with computing actions for Bellman Backup with the base policy

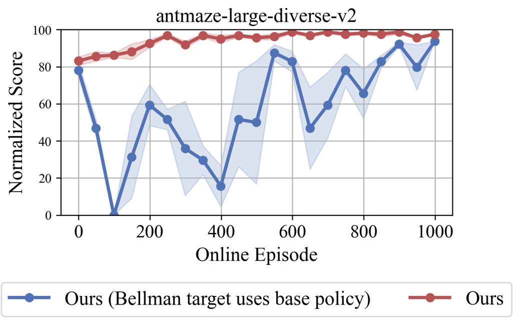


Figure 19: **Ablation for the choice of using the optimized action for Bellman backups.** To ablate the choice of computing targets using the optimized policy $\pi^{\text{Opt}}_{(\phi,\theta)}(\cdot|\cdot, m)$, we compare it against directly sampling from the base policy π_ϕ, and test it on antmaze-large-diverse-v2 fine-tuning. Both methods start from the same pre-trained critic checkpoints. Using the base policy for Bellman targets makes fine-tuning much more unstable, with a sharp drop in performance in the beginning, but ultimately obtains similar performance.

E.8 Learning curves for Gaussian Policies with PA-RL

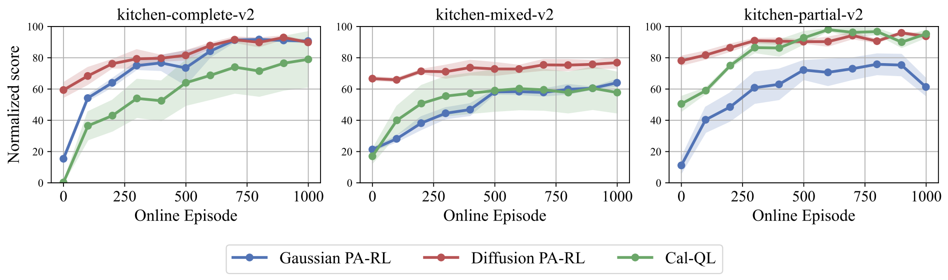


Figure 20: **Learning curves for gaussian policies with *PA-RL*, compared with Diffusion Policies with *PA-RL* and the standard Cal-QL with gaussian policies.** As with other experiments, we first train the base gaussian policy with BC on each dataset, and then do critic pre-training, followed by online RL fine-tuning. The only hyper-parameter we change for gaussian policies is the distillation learning rate, setting it to 3e-4. We observe Gaussian ***PA-RL*** performs competitively with the standard Cal-QL on kitchen tasks.

F Training time discussion

PA-RL optimizes actions using the procedure described in Section 4 any time an action from the policy is needed. We discuss how this affects the App of our method at different stages.

Critic training. In principle, action optimization should increase memory and computation requirements to critic training, but it also enables using an action cache to compute ahead of time, even in a distributed manner, when sufficient numbers of actions from the base policy are available. To make sure that this cache is not stale and to ensure that the critic models the optimal / on-policy value function, the actions cache is updated after every epoch of policy training via supervised learning. When sampling from the base policy is more than T times more expensive than taking T gradient steps of the critic (as is the case with OpenVLA or with diffusion policies with a large number of denoising steps), ***PA-RL*** can be significantly more efficient than alternatives that do not do caching.

Policy distillation. Compared to standard offline RL and online fine-tuning objectives, the supervised learning objective ***PA-RL*** can be significantly more efficient than policy improvement through reparameterization. For example, for a diffusion policy, backpropagating critic gradients through the diffusion chain uses a larger memory footprint than the DDPM objective ***PA-RL*** uses, by a factor equal to the number of denoising steps.

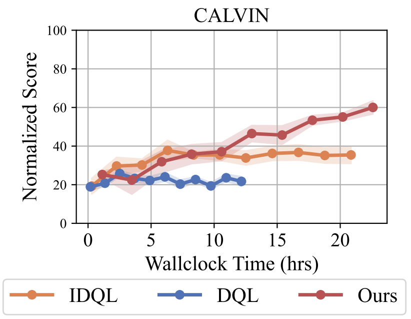


Figure 21: **Performance on CALVIN task as a function of wall clock time for *PA-RL*, IDQL, and DQL.** All three methods ran on the same compute instence type (TPU v4), were implemented in the same codebase. Observe that ***PA-RL*** improves at a similar rate per unit amount of wall-clock time as IDQL, but is able to improve far beyond to a better performance value. DQL largely remains flat as a function of more unit wall-clock time put into training.

Inference. During inference, ***PA-RL*** can optionally also apply action optimization by querying the base policy multiple times to sample an action. This can significantly increase the memory requirements of our method. That said, we do note that the number of samples from the base policy during inference can be much smaller than during training, as we do with OpenVLA (see Appendix D). ***PA-RL*** additionally requires taking multiple gradient steps of the critic with respect to the actions. We note that depending on the architecture used, this can be much cheaper than doing multiple full forward passes through the Q-function. For example, for image-based domains, the bulk of the computation happens for image encoding, which does not depend on the action. Therefore, the gradient steps will ignore that part of the network. There is also room for improvement for future work to investigate reducing the number of gradient steps further into training (as Figure 13 right suggests local optimization might have diminishing effects as fine-tuning progresses).