Beyond Action Residuals: Real-World Robot Policy Steering via Bottleneck Latent Reinforcement Learning

Dongjie Yu*[1,2], Kun Lei*[2,3], Zhennan Jiang[4], Jia Pan†[1], and Huazhe Xu†[2,5]

***Abstract*—Pretrained imitation policies have become a strong foundation for robot manipulation, but they often require online improvement to overcome execution errors, limited dataset coverage, and deployment mismatch. A central question is therefore how reinforcement learning (RL) should adapt policies after offline pretraining. Existing lightweight methods commonly apply residual corrections directly in action space, but this often leads to noisy and poorly structured exploration. In this work, we propose Z-Perturbation Reinforcement Learning (ZPRL), an approach that steers pretrained policies through a compact bottleneck latent rather than through policy weights or output actions. During offline training, we augment the policy with a plug-and-play variational information bottleneck (VIB) module to extract a task-relevant latent interface from observation embeddings. During online finetuning, the base policy is frozen and RL learns only a residual perturbation on this latent, whose decoded representation conditions the frozen action generator. We instantiate ZPRL on flow-matching policies and evaluate it on eight simulation tasks and four real-world tasks. Across diverse manipulation settings, ZPRL improves both sample efficiency and final performance over strong post-training baselines. In the real world, ZPRL improves the average success rate on four tasks by 33.7% over imitation base policies while producing smoother exploration behaviors than an action residual counterpart. These results suggest that a compact, task-aligned bottleneck latent provides an effective interface for online RL adaptation. More videos can be found at https://manutdmoon.github.io/ZPRL/.**

***Index Terms*—Robot manipulation, reinforcement learning, imitation learning, policy adaptation.**

I. Introduction

Imitation learning (IL) on offline datasets has become an increasingly popular approach to building robot manipulation policies given demonstrations [1]. Recent policy classes, including transformers [2], [3] and generative sensorimotor policies such as variational auto-encoders (VAE) [4] or diffusion and flow models [5]–[9], can reproduce complex behaviors from data with remarkable fidelity. However, strong offline pretraining does not eliminate the need for online improvement: in real-world deployment, policies may still fail to finish tasks due to execution error, insufficient task coverage by demonstrations, etc. [10]. As a result, learning from interaction [11], [12], especially reinforcement learning (RL), remains an appealing mechanism for post-training robot policies, rather than relying exclusively on collecting more human data [10], [13]–[16].

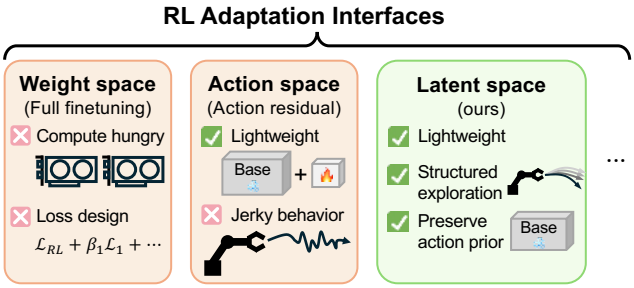

Fig. 1. **Different interfaces for RL adaptation of pretrained robot policies.** Full finetuning in weight space is expressive but computationally heavy and often tied to policy-specific loss designs. Residual adaptation in action space is lightweight, but exploration can be jerky and inefficient. ZPRL instead steers a compact bottleneck latent, providing a lightweight yet structured interface for online adaptation.

A key question lies in how to apply RL to modern imitation-learned policies. Full policy finetuning is increasingly costly as policy size grows [17], [18], and for generative policies it is often entangled with model-specific optimization designs [13], [14], [19], [20]. To avoid these difficulties, recent lightweight adaptation methods freeze the base policy and learn only a corrective policy [20]–[23]. This line of work is attractive because it preserves pretrained ability while reducing the burden of online adaptation. However, when the correction is applied directly in action space, exploration remains low-level: the policy is encouraged to modify motor commands rather than the underlying action pattern [24]. In robot manipulation, such action-space residuals can easily lead to oscillatory or erratic behavior, making exploration both inefficient and potentially unsafe. Our intuition is that meaningful behaviors rarely emerge from unstructured oscillation; what is needed is not merely different actions, but different actions that remain consistent in style with the pretrained policy [12].

We illustrate the design space of RL adaptation interfaces in Fig. 1, which suggests that the key issue is not only *how much* of the policy should be adapted, but also *where* RL should intervene. A useful intervention space should be: (1) compact for efficient online steering, compared to full finetuning in weight space; and (2) structured to keep exploration near valid behaviors, compared to residual corrections in action space. Recent work has shown the promise of latent-space RL for diffusion policies [24], and has also shown that compact latent representations can support on-manifold exploration for bootstrap IL [12]. Therefore, a natural question emerges from these findings: instead of correcting actions directly, *can RL steer a policy through a different but more efficient interface*

* Equal contribution.
† Corresponding authors.
[1] School of Computing and Data Science, The University of Hong Kong, Hong Kong SAR. {djyu@connect., panj@}hku.hk.
[2] Shanghai Qizhi Institute, Shanghai, China.
[3] Shanghai Jiao Tong University, Shanghai, China. leikun980116@gmail.com
[4] Institute of Automation, Chinese Academy of Sciences, Beijing, China. jiangzhennan2024@ia.ac.cn
[5] Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China. huazhe_xu@mail.tsinghua.edu.cn

that better captures the structure of pretrained behaviors?

In this work, we instantiate this idea as Z-Perturbation Reinforcement Learning (ZPRL). Following [12], [25], ZPRL injects a plug-and-play variational information bottleneck (VIB) module that extracts a compact latent representation of task-relevant features during offline training. During online finetuning, the base policy is frozen, and RL learns only a residual perturbation on this bottleneck latent. Perturbed latents are then passed through the VIB decoder and serve as conditions for the action head in the base policy to produce actions. Compared with action-space residuals [20]–[23], this design moves exploration from low-level motor commands to structured latent steering. Compared with latent-space methods defined over diffusion noise [24], it uses a compact task-relevant bottleneck as the steering interface, making the exploration space smaller and more explicitly aligned with tasks.

We evaluate ZPRL with both simulated manipulation benchmarks and real-world tasks, showing that it improves sample efficiency and final performance over baselines. After few hours of online interaction, ZPRL improves the average success rate by 33.7% over base policies. Furthermore, under similar action magnitudes, ZPRL produces smoother and more consistent behaviors than an action residual approach, with lower end-effector velocity and acceleration, indicating that it steers the base policy in a more structured manner rather than injecting high-frequency jitter. These results demonstrate that bottleneck latent steering provides an effective RL interface for post-training robot policies.

II. Related Work

A. Imitation Learning in Robot Manipulation

Robot manipulation with IL has evolved from low-dimensional skill representations, such as dynamical and probabilistic movement primitives [26]–[28], to large-scale visuomotor behavior modeling with expressive neural architectures [4], [7], [18], [29]–[32]. Benefiting from the rapid growth of robot datasets [33]–[37] and compute resources, modern IL methods can build increasingly capable pretrained policies that perform a wide range of household and industrial manipulation tasks [1], [2]. Despite their impressive performance, these policies remain fundamentally constrained by the coverage and quality of offline data. In realistic deployments, task failures, execution errors, and distribution shifts often require further improvement beyond offline pretraining [38], [39]. This motivates methods that can adapt or steer pretrained policies through online interaction.

B. Reinforcement Learning for Adapting Pretrained Policies

Reinforcement learning provides a natural framework for improving pretrained policies beyond the limits of offline datasets [40], [41]. A common approach is to finetune the entire policy end-to-end during online adaptation [10], [13], [14], [16], [42]–[46]. From this perspective, adaptation occurs in the *weight space* of the policy. While such methods can be effective for smaller models or policy classes with dedicated optimization procedures, full online finetuning becomes increasingly expensive for modern large-scale visuomotor and vision-language-action (VLA) policies, and often requires substantial systems and algorithmic support [13].

A complementary line of work improves pretrained policies without updating the entire model. Residual RL methods perform adaptation in the *action space* by learning corrective actions on top of classical controllers [47], [48] or frozen imitation-learned policies [20]–[23], [49]. These methods are attractive because they preserve pretrained ability and provide a lightweight interface for online improvement. However, when the correction is applied directly in action space, exploration remains low-level, and can become inefficient in high-dimensional or chunked action spaces [50]. In robot manipulation, such action-space perturbations may also induce jerky and potentially unsafe behaviors. In addition, useful designs in residual RL [49] can also be leveraged in our work because we are injecting residuals in a latent space.

More recent works have moved the RL intervention from action space to internal latent spaces, such as initial diffusion noise [24], [51]. These results highlight that the *steering interface* itself is a critical design choice for RL post-training. However, noise steering still operates in a space whose dimensionality is equal to that of robot actions, which can limit efficiency as the action horizon or robot degrees of freedom increase. In this work, we instead study a compact bottleneck latent as the steering interface, drawing inspiration from recent work on structured on-manifold exploration [12]. Our method combines lightweight adaptation with a task-aligned low-dimensional latent space, enabling smooth behaviors and efficient online improvement.

III. Preliminaries and Problem Formulation

A. Robot Manipulation as an Observation-Conditioned Decision Process

We consider robot manipulation in an episodic decision process induced by an underlying Markov Decision Process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, \rho_0, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, respectively. In our setting, actions correspond to desired end-effector poses. Since the policy does not access full environmental states, it instead acts on observations $\boldsymbol{o}_t \in \mathcal{O}$, which may include vision and robot proprioception. At the beginning of each episode, the manipulation environment is initialized to $\boldsymbol{s}_0 \sim \rho_0$, producing an initial observation $\boldsymbol{o}_0$. At each timestep t, the robot policy chooses an action $\boldsymbol{a}_t \sim \pi(\cdot|\boldsymbol{o}_t)$, and then the environment transitions to the next state $\boldsymbol{s}_{t+1} \sim T(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t)$ and emits the next observation $\boldsymbol{o}_{t+1}$. The policy receives a scalar reward $r_t = R(\boldsymbol{s}_t, \boldsymbol{a}_t)$. Unless otherwise specified, we use sparse binary rewards, i.e., $r_t = 1$ upon task success (and the episode is terminated) and 0 otherwise. During RL post-training, the goal is to learn a policy maximizing the expected discounted return (i.e., the Q-function), which we re-write with respect to the observation-conditioned policy for simplicity, $Q^\pi(\boldsymbol{o}, \boldsymbol{a}) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | \boldsymbol{o}_0 = \boldsymbol{o}, \boldsymbol{a}_0 = \boldsymbol{a}\right]$. In robot manipulation, this objective corresponds to solving tasks with both high success rate and high efficiency.

B. Flow-Matching Base Policies From Imitation

Although the proposed adaptation framework is not restricted to a particular policy parameterization, we instantiate it with flow-matching (FM) policies [6] throughout this work. We choose FM models because they retain the expressive action generation capabilities of diffusion-based policies [5], [7], [52], while being simpler to implement and requiring only a small number of iterative steps at inference time, enabling high-frequency policy execution. We assume that the base policy consists of an observation encoder $\mathcal{E}$ and a conditional flow model. Given an observation $\boldsymbol{o}$, the encoder produces a latent conditioning vector $\boldsymbol{c} = \mathcal{E}(\boldsymbol{o})$, which is then used by the flow model to generate actions. This factorization is common in modern policies [1], [7], [16], where perception first extracts a representation and action generation is performed conditioned on that representation.

Given an offline dataset $\mathcal{D}_{\text{off}} = \{\tau_i\}_{i=1}^{N}$, where each trajectory $\tau_i = \{(\boldsymbol{o}_t, \boldsymbol{a}_t)\}_{t=1}^{T_i}$ contains observation–action pairs, the FM policy is trained by learning a velocity field v_ϕ for an ordinary differential equation (ODE) that transports a noise sample toward the target action. Specifically, let $\boldsymbol{a}^0 = \boldsymbol{a}$ denote the clean action and $\boldsymbol{a}^1 = \boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ denote a Gaussian noise sample. For an interpolation variable $k \sim \text{Unif}[0,1]$, we define the intermediate point $\boldsymbol{a}^k = (1-k)\boldsymbol{a}^0 + k\boldsymbol{a}^1$. The imitation learning objective is then

$$\mathcal{L}_{\text{IL}}(\phi) = \mathbb{E}_{\substack{(\boldsymbol{o},\boldsymbol{a})\sim\mathcal{D}_{\text{off}} \\ k\sim\text{Unif}[0,1] \\ \boldsymbol{w}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}} \left[\left\| v_\phi(\boldsymbol{a}^k, k, \boldsymbol{c}) - (\boldsymbol{w} - \boldsymbol{a}) \right\|_2^2 \right], \quad (1)$$

where $\boldsymbol{c} = \mathcal{E}_\phi(\boldsymbol{o})$. The encoder $\mathcal{E}_\phi$ and the velocity model v_ϕ are optimized jointly during imitation learning.

During deployment, actions are generated by numerically integrating the learned flow from noise to action using a reversed discrete schedule $1 = k_M > \cdots > k_m > \cdots > k_0 = 0$. Starting from $\boldsymbol{a}^1 = \boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, the action is obtained by

$$\begin{aligned} \boldsymbol{a}^{k_m} &= \boldsymbol{a}^{k_{m+1}} - (k_{m+1} - k_m) v_\phi(\boldsymbol{a}^{k_{m+1}}, k_{m+1}, \boldsymbol{c}), \\ & m = M-1, \ldots, 0. \end{aligned} \quad (2)$$

The final sample $\boldsymbol{a}^0$ is taken as the generated action $\boldsymbol{a}_{\text{off}}$. We denote the full policy by $\pi_{\text{base}}(\cdot|\mathcal{E}(\boldsymbol{o}))$. Note that, throughout this paper, actions refer to a multi-step chunk denoted by $\boldsymbol{a}$ and we denote a single-step action by a.

C. RL Post-training Setting

Given a pretrained policy π_{base}, our goal is to further improve its performance on manipulation tasks through online interaction and RL. RL post-training can intervene at different interfaces of the base policy. Full finetuning updates the policy parameters directly, i.e., adaptation happens in weight space. Action-space residual RL instead freezes π_{base} and learns an online policy π_{on} that perturbs the generated action, $\boldsymbol{a} \sim \pi_{\text{base}}(\cdot|\boldsymbol{c}) + \lambda\, \pi_{\text{on}}(\cdot|\boldsymbol{o})$, where λ controls the perturbation scale. A more recent method steers pretrained policies through diffusion noise [24]. In this work, we focus on a latent steering setting and study how the encoded observation representation $\boldsymbol{c} = \mathcal{E}(\boldsymbol{o})$ can serve as an efficient interface for adapting the frozen base policy.

IV. Steering Policies with Z-Perturbation Reinforcement Learning

We present Z-Perturbation Reinforcement Learning (ZPRL), a lightweight post-training framework for adapting pretrained manipulation policies through online RL. Our key idea is to intervene on an internal task-relevant interface of the base policy, rather than on policy weights or output actions. Concretely, we first augment the base policy with a plug-and-play variational information bottleneck (VIB) module to obtain a compact bottleneck latent from the observation embedding, following prior work on structured representations for iterative imitation [12]. We then learn an online residual policy that perturbs this latent during post-training, while keeping the entire base policy frozen. The perturbed latent is then passed through the VIB decoder and used as the condition of the flow model, enabling efficient policy improvement with smoother and more structured actions than action residuals. Fig. 2 illustrates the overall two-stage pipeline, including offline bottlenecked IL and online residual perturbation on the latent.

A. Bottleneck Task Latent for Policy Conditioning

A central challenge in post-training manipulation policies is to find an efficient interface for behavior steering. Direct exploration in action space becomes increasingly difficult as the action horizon grows, since the dimensionality of the action chunk scales linearly with time. In such spaces, adding Gaussian noise typically produces high-frequency jitter rather than meaningful behavioral diversity. A more suitable interface should therefore preserve action-relevant structure while discarding unnecessary details in the observation or its embedding. Following Deep VIB [25] and SOE [12], we adopt a compact bottleneck latent on top of the observation embedding. This latent is intended to retain only the information necessary for action prediction, while filtering out irrelevant variation. Thus, we formalize the information bottleneck objective

$$\min \; -I(\mathbf{z}; \mathbf{a}) + \beta I(\mathbf{z}; \mathbf{c}), \quad (3)$$

where I denotes mutual information between random variables, $\boldsymbol{c} = \mathcal{E}(\boldsymbol{o})$ is the observation embedding produced by the base encoder, and β controls the trade-off between informativeness and compactness.

To optimize this objective in practice, we instantiate it with a variational encoder-decoder parameterization following prior work [12], [25]. This yields the standard variational information bottleneck (VIB) surrogate objective

$$\begin{aligned} \mathcal{L}_{\text{VIB}}(\varphi) = \mathbb{E}_{(\mathbf{o},\boldsymbol{a})\sim\mathcal{D}} \Big[& -\mathbb{E}_{\boldsymbol{z}\sim p_\varphi(\cdot|\boldsymbol{c})} \log q(\boldsymbol{a}|g_\varphi(\boldsymbol{z})) \\ & + \beta\, \text{KL}(p_\varphi(\mathbf{z}|\boldsymbol{c}) \| r(\mathbf{z})) \Big], \end{aligned} \quad (4)$$

where the VIB encoder $p_\varphi(\cdot|\boldsymbol{c})$ maps the observation embedding to a latent posterior, and the VIB decoder $g_\varphi(\boldsymbol{z})$ reconstructs a feature for downstream action generation. For computational tractability, we parameterize the posterior as a diagonal Gaussian with mean and variance predicted by the VIB encoder. The latent prior is chosen as a standard Gaussian $r(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

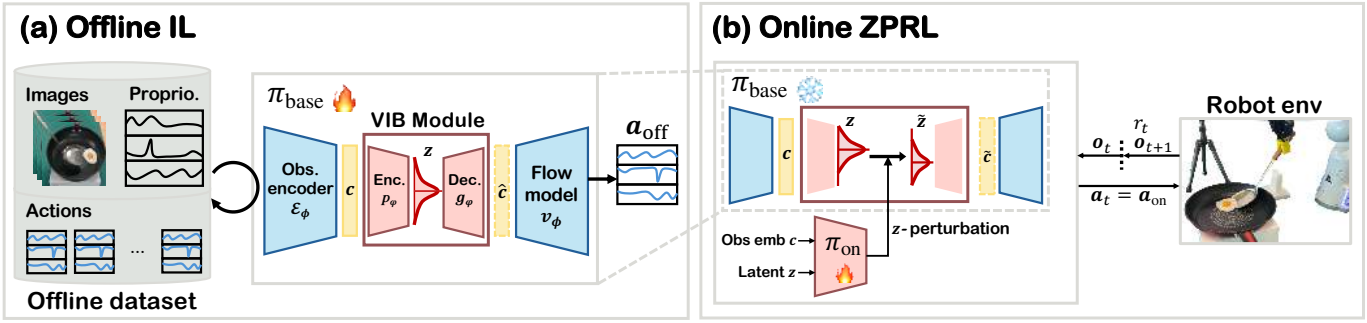

Fig. 2. **Two-stage training pipeline of ZPRL.** (a) Offline, a flow-based manipulation policy is pretrained with a VIB bottleneck over the task-conditioning embedding. (b) Online, the pretrained backbone is frozen and a latent residual policy predicts $\Delta \boldsymbol{z}$ to perturb the bottleneck, $\tilde{\boldsymbol{z}} = \boldsymbol{z} + \lambda \Delta \boldsymbol{z}$, thereby steering the generated action through the frozen VIB decoder and flow policy.

In our flow-policy instantiation, the first term in Equation (4) is realized by the same flow-matching objective used for behavior cloning, with the reconstructed feature $\hat{\boldsymbol{c}} = g_\varphi(\boldsymbol{z})$ replacing the original observation embedding $\boldsymbol{c}$ as the conditioning input. Therefore, the VIB loss becomes

$$\mathcal{L}_{\text{VIB}}(\varphi) = \mathbb{E}_{\substack{(\boldsymbol{o},\boldsymbol{a}),\\ k,\boldsymbol{w},\boldsymbol{z}}} \Big[\big\| v_\phi(\boldsymbol{a}^k, k, \hat{\boldsymbol{c}}) - (\boldsymbol{w} - \boldsymbol{a}) \big\|_2^2 + \beta\, \mathrm{KL}(p_\varphi(\mathbf{z}|\boldsymbol{c}) \,\|\, r(\mathbf{z})) \Big], \tag{5}$$

where $\boldsymbol{c} = \mathcal{E}_\phi(\boldsymbol{o})$ and $\hat{\boldsymbol{c}} = g_\varphi(\boldsymbol{z})$. The overall offline loss is

$$\mathcal{L}_{\text{off}}(\phi, \varphi) = \mathcal{L}_{\text{IL}}(\phi) + \mathcal{L}_{\text{VIB}}(\varphi). \tag{6}$$

As in [12], we treat the bottleneck as a plug-in auxiliary path: gradients from $\mathcal{L}_{\text{VIB}}$ are blocked from updating the base encoder $\mathcal{E}_\phi$ and flow generator v_ϕ, so the original imitation path remains unaffected. This preserves the performance of the base policy while learning a compact latent interface.

The role of this bottleneck in our method is different from that in SOE [12]. SOE uses stochastic sampling in the bottleneck space with a specified variance to generate diverse on-manifold actions for rollout, where successful trajectories are retained to re-train the base policy. Here, instead, we argue that the learned latent $\boldsymbol{z}$ contains the steering potential for RL post-training: rather than resampling actions by enlarging posterior variance, an online policy will learn to perturb $\boldsymbol{z}$ to steer the frozen base action generator toward higher-return behaviors.

B. RL Perturbation on Bottleneck Latents

Rather than overwriting the bottleneck latent learned offline, we adapt the pretrained policy by applying a residual perturbation to it. Specifically, given the current bottleneck representation $\boldsymbol{z}$, the online policy predicts a latent correction $\Delta \boldsymbol{z}$ and forms a perturbed latent

$$\tilde{\boldsymbol{z}} = \boldsymbol{z} + \lambda \Delta \boldsymbol{z}, \qquad \Delta \boldsymbol{z} \sim \pi_{\text{on}}(\cdot|\boldsymbol{c}, \boldsymbol{z}), \tag{7}$$

where π_{on} is the online RL policy and λ controls the perturbation magnitude. The perturbed latent is then decoded into a new conditioning embedding $\tilde{\boldsymbol{c}} = g_\varphi(\tilde{\boldsymbol{z}})$, which replaces the original condition used by the flow policy. The final action is therefore generated as $\boldsymbol{a}_{\text{on}} \sim \pi_{\text{base}}(\cdot|\tilde{\boldsymbol{c}})$, so that online RL steers the base policy by modifying its internal task-conditioning variable before action generation. The perturbed control loop is illustrated in Fig. 2(b).

This design relies on the expressiveness of the pretrained policy. Offline training already equips the base policy with a strong prior over task-relevant behaviors, so online adaptation does not need to synthesize actions from scratch. Instead, it only needs to correct the latent toward one that yields higher-return behavior because observation-action pairs learned by the base policy are not always optimal. Since the condition is compressed by the VIB bottleneck, irrelevant details have been filtered out and the latent mainly retains task-relevant information. Thus, perturbing $\boldsymbol{z}$ changes the generated action in a structured manner through the frozen decoder and flow generator, allowing RL to search for better behaviors while preserving the pretrained action prior.

During online learning, rewards and values should be associated with the *resulting perturbed latent* rather than the residual $\Delta \boldsymbol{z}$ alone. We therefore define

$$\hat{r}(\boldsymbol{c}, \tilde{\boldsymbol{z}}) := r(\boldsymbol{s}, \boldsymbol{a}_{\text{on}}), \qquad \hat{Q}(\boldsymbol{c}, \tilde{\boldsymbol{z}}) := Q(\boldsymbol{s}, \boldsymbol{a}_{\text{on}}), \tag{8}$$

where $\boldsymbol{a}_{\text{on}} \sim \pi_{\text{base}}(\cdot|g_\varphi(\tilde{\boldsymbol{z}}))$. The critic is defined on $\tilde{\boldsymbol{z}}$ rather than on $\Delta \boldsymbol{z}$ because the same residual can induce different action distributions when added to different $\boldsymbol{z}$, due to the stochasticity of $p_\varphi(\boldsymbol{z}|\boldsymbol{c})$. Therefore, $\Delta \boldsymbol{z}$ alone is insufficient to determine the downstream action and cannot be assigned a well-defined value without its latent base. In our implementation, the actor takes $(\boldsymbol{c}, \boldsymbol{z})$ as input to predict $\Delta \boldsymbol{z}$, while the critic evaluates the resulting pair $(\boldsymbol{c}, \tilde{\boldsymbol{z}})$.

Compared with action residuals or diffusion-noise steering, the bottleneck latent provides a more efficient control interface in two aspects. First, it preserves the ability to influence the whole action chunk while operating in a much lower-dimensional space: in our setting, an action chunk may have up to 100 dimensions, whereas the bottleneck latent typically has only 16 or 32 dimensions. This substantially reduces the difficulty of online exploration and optimization. Second, the perturbation acts on the policy condition rather than directly on the generated action, so the final action is still produced by the pretrained generator and remains structured by the offline data distribution [53]. In this way, online RL steers the action generation process instead of bypassing it, which can lead to safer and more stable actions in the environment.

C. Online RL Objective and Design Choices

In principle, ZPRL can use various RL algorithms; we instantiate it with soft-actor-critic (SAC) [54]. To simplify

notation, we drop the subscript on and denote the online perturbation policy by π_θ. During online adaptation, only the residual actor π_θ and the latent critic Q_ψ are updated, while the pretrained base policy remains frozen.

Given a latent state $(\boldsymbol{c}, \boldsymbol{z})$, the actor predicts a stochastic residual $\Delta \boldsymbol{z} \sim \pi_\theta(\cdot \mid \boldsymbol{c}, \boldsymbol{z})$, which forms the perturbed latent $\tilde{\boldsymbol{z}} = \boldsymbol{z} + \lambda \Delta \boldsymbol{z}$. The actor is optimized to maximize returns while maintaining sufficient exploration:

$$\mathcal{L}_\pi(\theta) = \mathbb{E}_{\substack{(\boldsymbol{c},\boldsymbol{z})\sim\mathcal{D}_{\mathrm{on}} \\ \Delta\boldsymbol{z}\sim\pi_\theta(\cdot|\boldsymbol{c},\boldsymbol{z})}} \Big[\alpha \log \pi_\theta(\Delta \boldsymbol{z} \mid \boldsymbol{c}, \boldsymbol{z}) - Q_\psi(\boldsymbol{c}, \tilde{\boldsymbol{z}})\Big], \tag{9}$$

where α is the temperature coefficient and is adjusted automatically. The critic is trained with a temporal-difference objective,

$$\mathcal{L}_Q(\psi) = \mathbb{E}_{\substack{(\boldsymbol{c},\boldsymbol{z},r,\boldsymbol{c}',\boldsymbol{z}')\sim\mathcal{D}_{\mathrm{on}} \\ \Delta\boldsymbol{z}'\sim\pi_\theta(\cdot|\boldsymbol{c}',\boldsymbol{z}')}} \Big[\big(Q_\psi(\boldsymbol{c}, \tilde{\boldsymbol{z}}) - y\big)^2\Big], \tag{10}$$

with target

$$y = r + \gamma\, \bar{Q}_{\bar{\psi}}(\boldsymbol{c}', \tilde{\boldsymbol{z}}'), \qquad \tilde{\boldsymbol{z}}' = \boldsymbol{z}' + \lambda \Delta \boldsymbol{z}', \tag{11}$$

where $\bar{Q}_{\bar{\psi}}$ denotes the target critic updated by exponential moving average of Q_ψ. Additional implementation details are provided in the Appendix. Here we use a variant wihout entropy term in Q function to stablize training.

Choosing the perturbation scale λ. The perturbation scale λ controls the strength of online steering in the bottleneck latent space. A small λ has little effect on the latent and adaptation remains weak; if it is too large, the perturbed latent may move too far from the pretrained latent distribution and become harder for the VIB decoder and action prior to handle reliably. In practice, we track the magnitude of the original latent $\boldsymbol{z}$ and the perturbation $\Delta \boldsymbol{z}$ to determine a fixed λ for each task. We study its effects in Section V-B and provide practical guidelines and task-specific values in the Appendix.

Update-to-data ratio. A high update-to-data (UTD) ratio can improve performance measured against environment steps, but it does not necessarily improve wall-clock efficiency. In our setting, increasing the number of gradient updates per action chunk quickly shifts the computational bottleneck from robot interaction to policy updates, especially when multiple Q functions are used for stability [55]. Since our goal is efficient online adaptation in real time rather than only step efficiency, we do not use an excessively high UTD ratio (such as 10 or 20) in experiments. Instead, we use $\mathrm{UTD} = 1$ in simulations and 2 or 5 in real-world tasks to balance sample efficiency and wall-clock throughput.

Why not an action-space critic. One alternative is to train an additional critic $Q^{\mathrm{a}}(\boldsymbol{o}, \boldsymbol{a})$ directly in the action space and supervise the latent critic through the action produced by the base policy, similar to the noise aliasing discussed in [24]. However, this design would require iterative action denoising during critic optimization. For diffusion-based policies, even a small number of denoising or integration steps (such as two) introduce noticeable overhead, which hurts wall-clock efficiency during online training. We therefore optimize the critic directly in the latent perturbation space and avoid an additional action-space critic.

V. Experiments

We evaluate ZPRL on eight tasks across three simulation benchmarks and compare it against several RL finetuning baselines in Section V-A. We then investigate how the bottleneck latent inherited from the pretrained base policy influences online finetuning in Section V-B. Next, we show that ZPRL produces smoother and more structured robot behavior than action-residual methods in Section V-C. Finally, we demonstrate real-world ZPRL on four robotic tasks in Section V-D and conclude with a discussion of its limitations and possible extensions in Section V-E.

A. Online RL Finetuning in Simulation

We first compare ZPRL against recent methods for RL adaptation of IL base policies. We consider eight tasks from three benchmarks: `can`, `square`, and `transport` from `Robomimic` [34]; `door`, `hammer`, and `pen` from `Adroit` [56]; and `box-close` and `push-wall` from `Metaworld` [57]. We focus on vision-based manipulation, where observations include both images and proprioception. For each task, all base policies are trained on the same offline dataset: for `Robomimic`, 100 trajectories from the official mixed-quality multi-human (MH) dataset released by [34]; for `Adroit` and `Metaworld`, 100 trajectories generated by a medium-expert policy from [58]. We compare with four baselines: DPPO [13] and ReinFlow [14], which finetune the full diffusion or flow policies with on-policy optimization; Policy-Decorator (Po-Dec) [20], a representative action-residual method with progressive exploration and scaled residuals for stable learning; and DSRL [24], which adapts diffusion policies by steering the noise $\boldsymbol{w}$. Additional details are provided in the Appendix.

As shown in Fig. 3, ZPRL consistently reaches strong final performance across all eight tasks and is among the most effective methods in terms of online adaptation speed. This holds for both parallel-jaw manipulation (`Robomimic`, `Metaworld`) and dexterous hands (`Adroit`), suggesting that bottleneck latent perturbation provides a broadly effective interface for improving pretrained policies with RL. In particular, ZPRL achieves high asymptotic returns while remaining competitive in sample efficiency on most tasks. One notable exception is the two `Metaworld` tasks, where Po-Dec learns faster than ZPRL in the early stage of training. We attribute this gap to the relatively small action space in these tasks. Specifically, we set the length of an action chunk to 2 in `Metaworld`, since longer chunks degrade the base policy, and each action step has only four dimensions (3D translation and gripper). The resulting action-chunk dimension is only 8, which is already small enough for standard action-space RL to be effective. In contrast, ZPRL uses a bottleneck latent of dimension 16, so the advantage of latent-space perturbation is less pronounced in this regime. Nevertheless, ZPRL still reaches higher returns in the later stage of finetuning.

B. How the Bottleneck Latent Interface Shapes Online RL

To understand whether and how the bottleneck latent serves as an effective interface for online finetuning, we perform four

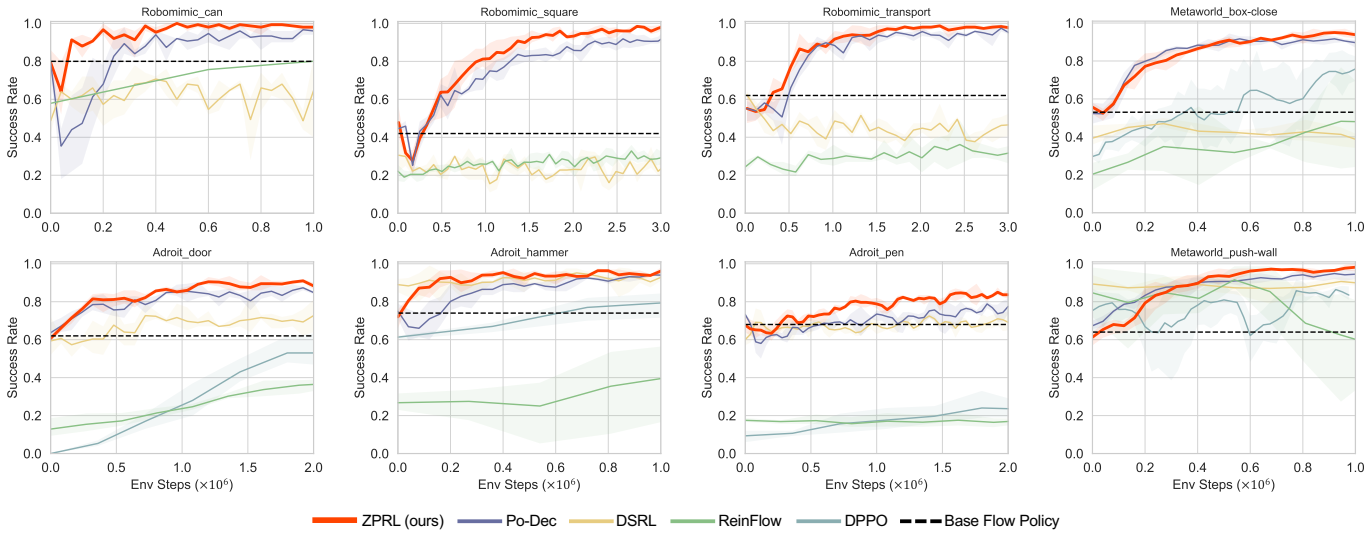

Fig. 3. **Simulation results across three benchmarks.** Success rate versus online environment steps during finetuning. Curves are averaged over 3 random seeds, with evaluation on 50 random initial layouts; shaded regions indicate the 95% interval across seeds.

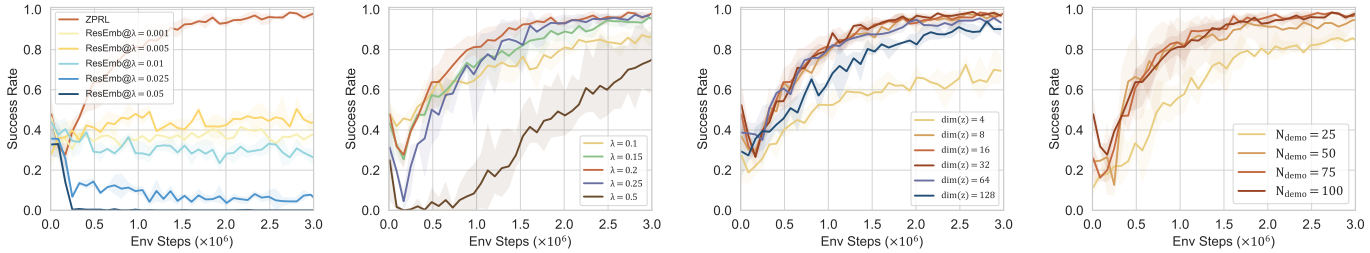

Fig. 4. **ZPRL with different latent interface settings on `square` in `Robomimic`.** We study (from left to right): (a) direct perturbation on the observation embedding; (b) the perturbation scale λ; (c) the dimension of $\boldsymbol{z}$; and (d) the number of trajectories in the offline dataset. Each variant is averaged over 3 random seeds and the shaded region is the 95% confidence interval.

ablations on the `square` task in `Robomimic`. We study: (a) the type of steering interface, where we compare ZPRL against directly learning residuals on the observation embedding without VIB compression; (b) the perturbation scale λ, which controls how far RL can move around the pretrained latent; (c) the bottleneck dimension $\dim(\boldsymbol{z})$, which determines how compact the steering space is; and (d) the number of offline demonstrations used to train the base policy, which affects how well the latent manifold is supported by data. Together, these ablations reveal that the effectiveness of ZPRL depends not only on the magnitude of online perturbation, but also on whether RL operates on a compact, structured, and data-supported latent interface.

Bottleneck latent versus direct steering on observation embeddings. We first compare ZPRL with ResEmb, which directly learns residuals on the high-dimensional observation embedding without VIB compression. ResEmb performs consistently worse and becomes more unstable as residual scale grows, suggesting that effective online adaptation does not arise from steering arbitrary intermediate features. A likely reason is that the raw observation embedding still contains task-irrelevant information, which makes RL exploration harder and less focused. In contrast, the VIB bottleneck compresses the representation into a more task-relevant latent, allowing RL to explore in a more compact space and steer action generation more efficiently.

Table I. Summary metrics of online finetuning under different latent ablations. Final success rates (SRs) are averaged over the last 5 data points. N/A means the variant cannot reach the threshold with all seeds.

Setting	Steps ($\times 10^6$) to SR=0.9 ↓	Final SR ↑
$\lambda = 0.10$	2.40	0.86
$\lambda = 0.15$	1.54	0.95
$\lambda = 0.20$	1.16	0.98
$\lambda = 0.25$	1.15	0.96
$\lambda = 0.50$	N/A	0.74
$\dim(\boldsymbol{z}) = 4$	N/A	0.68
$\dim(\boldsymbol{z}) = 8$	1.23	0.98
$\dim(\boldsymbol{z}) = 16$	1.16	0.98
$\dim(\boldsymbol{z}) = 32$	1.08	0.99
Po-Dec ($\dim(\boldsymbol{a}) = 40$)	1.68	0.92
$\dim(\boldsymbol{z}) = 64$	1.29	0.94
$\dim(\boldsymbol{z}) = 128$	1.90	0.91
$N_{\text{demo}} = 25$	2.54	0.86
$N_{\text{demo}} = 50$	1.21	0.96
$N_{\text{demo}} = 75$	1.06	0.98
$N_{\text{demo}} = 100$	1.16	0.98

Perturbation scale λ controls the exploration–stability trade-off. As shown in the second plot of Fig. 4, the perturbation scale λ affects both the speed of online improvement and the final success rate (SR). A larger λ gives the RL policy stronger ability to steer the pretrained policy, but it also makes the randomly initialized $\Delta\boldsymbol{z}$ more likely to push $\tilde{\boldsymbol{z}}$ away from the local neighborhood of the pretrained latent. This weakens the support provided by the frozen action prior

and can substantially destabilize finetuning. In the extreme case of $\lambda = 0.5$, the policy hardly reaches SR$= 0.9$ and only achieves a final SR of 0.74. In contrast, a smaller λ keeps the perturbation more local and leads to more stable optimization, but also restricts how quickly and how strongly RL can modify behavior. This is reflected in the slower learning curves of $\lambda = 0.10$ and 0.15, which require 2.40M and 1.54M steps to reach SR$= 0.9$, respectively. Empirically, $\lambda = 0.20$ provides the best balance between adaptability and stability: it reaches SR$= 0.9$ in 1.16M steps, nearly matching $\lambda = 0.25$ (1.15M), while achieving the highest final SR of 0.98. These results support our central design principle that online RL should steer the pretrained latent locally through residual perturbations, rather than overwrite the latent interface with overly large corrections.

The gain is not merely from dimensionality reduction. We vary the bottleneck dimension from 4 to 128, while adjusting λ according to the magnitude of $\boldsymbol{z}$. As shown in Fig. 4 (mid-right) and Table I, ZPRL is robust to a moderate range of bottleneck sizes: $\dim(\boldsymbol{z}) = 8, 16, 32$ all achieve strong online performance with similar steps to reach SR$= 0.9$ and similar final SRs. The best result is obtained at $\dim(\boldsymbol{z}) = 32$. Importantly, the gain of ZPRL cannot be explained simply by using a lower-dimensional control variable than the action chunk. Even when $\dim(\boldsymbol{z}) = 64$, which already exceeds the action dimension of Po-Dec ($\dim(\boldsymbol{a}) = 40$), ZPRL still reaches SR$= 0.9$ faster (1.29M vs. 1.68M steps) and achieves a higher final SR (0.94 vs. 0.92). This suggests that what matters is not dimension reduction alone, but whether RL operates in a task-relevant space. As long as perturbations remain local, the frozen VIB decoder and action generator can still map corrected latents to meaningful actions. At the same time, the two extremes both hurt performance. When $\dim(\boldsymbol{z}) = 4$, the bottleneck is too restrictive and likely discards useful task information, leading to poor online learning. When $\dim(\boldsymbol{z}) = 128$, the steering space becomes less compact and harder to optimize efficiently, reducing both learning speed and final performance. Overall, these results suggest that a useful steering interface should be compact, but not so restrictive that it removes information needed for success. A similar phenomenon can also be observed in [59].

The amount of offline data determines how much structure RL can exploit. Finally, we reduce the number of demonstrations used to train the base policy from 100 to 75, 50, and 25 trajectories. The right plot of Fig. 4 shows a clear degradation as the offline dataset becomes smaller, although ZPRL remains effective even with only 25 demonstrations. This trend is consistent with the role of the pretrained latent as a data-supported action prior: given fewer demonstrations, the base policy is a weaker controller and learns a latent manifold with narrower coverage of task-relevant behaviors, leaving RL with a less reliable interface to steer. In other words, reducing offline data hurts both stages of the method: it weakens the offline policy itself and reduces the quality of the latent space on which online RL operates. Table I further quantifies the degradation under reduced offline data: fewer demonstrations lead to more steps to reach SR=0.9 and lower final performance.

Takeaway. These ablations show that the bottleneck latent shapes ZPRL through several properties: locality of intervention, compactness of the control space, coverage of the pretrained latent manifold, and the form of the steering interface itself. In particular, the comparison with direct residual steering on the observation embedding shows that the gain does not come from perturbing an arbitrary intermediate feature. Instead, ZPRL benefits from steering a compressed bottleneck latent that is semantically organized, locally controllable, and supported by offline data, while preserving the frozen action prior.

C. Smooth and Structured Action Generation

We next examine an appealing byproduct of ZPRL: it preserves smooth and structured behavior even during online exploration. We still take the `square` task as an example and compare Po-Dec and ZPRL checkpoints obtained after different numbers of online interaction steps (from 0 to 2.4M). For each checkpoint, we roll out the policy from a fixed initial layout and measure the smoothness of the desired end-effector position implied by the predicted actions. Specifically, we compute two finite-difference metrics, the translational velocity and acceleration,

$$\mathrm{Vel}_{\mathrm{EE}} = \left\| (\boldsymbol{a}^{\mathrm{p}}_{t+1} - \boldsymbol{a}^{\mathrm{p}}_{t}) / \,\mathrm{d}t \right\|_2 ,$$
$$\mathrm{Acc}_{\mathrm{EE}} = \left\| (\boldsymbol{a}^{\mathrm{p}}_{t+1} + \boldsymbol{a}^{\mathrm{p}}_{t-1} - 2\boldsymbol{a}^{\mathrm{p}}_{t}) / \mathrm{d}t^2 \right\|_2 ,$$

where $\boldsymbol{a}^{\mathrm{p}}_t$ denotes the positional components of the policy output at timestep t, and $\mathrm{d}t$ is the simulation interval (0.05 s in `Robomimic`). Lower values indicate smoother commanded motion.

Table II. Evolution of smoothness metrics during online RL, reported as [Po-Dec | ZPRL]. Lower values indicate smoother behavior.

Env Steps	$\mathrm{Vel}_{\mathrm{EE}}$ (m/s)	$\mathrm{Acc}_{\mathrm{EE}}$ (m/s^2)
0	0.14 | 0.15	3.43 | 3.74
0.8M	0.33 | 0.22	10.42 | 6.35
1.6M	0.32 | 0.21	9.78 | 6.03
2.4M	0.31 | 0.22	9.60 | 5.79

Table II shows that, once online finetuning starts, ZPRL consistently yields lower velocity and acceleration than Po-Dec. At 2.4M steps, ZPRL reduces $\mathrm{Vel}_{\mathrm{EE}}$ from 0.31 to 0.22 (about 29%) and $\mathrm{Acc}_{\mathrm{EE}}$ from 9.60 to 5.79 (about 39%). The same trend already appears at 0.8M and remains stable throughout training. This result suggests that action-space residual exploration introduces persistent high-frequency oscillations, whereas ZPRL perturbs the bottleneck latent and still relies on the frozen flow model to produce actions, leading to smoother and more structured policy behavior. Figure 5 provides a qualitative view of this difference. In the representative rollout, Po-Dec shows visibly stronger oscillation in the desired y-axis position after online adaptation, whereas ZPRL continues to steer the trajectory away from the pretrained behavior in a structured and regular manner without introducing comparable jitter.

Overall, these results support our claim that bottleneck latent perturbation preserves the structural prior of the pretrained

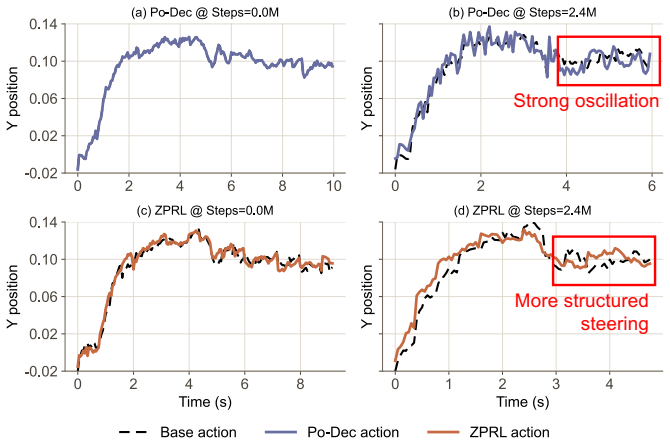

Fig. 5. **Representative rollout trajectories on `Robomimic square`.** Desired y-axis position produced by policies at different stages of online training. Although both methods start from similarly jerky randomly initialized RL policies, Po-Dec exhibits increasingly strong oscillations after online adaptation, while ZPRL preserves smoother and more structured steering throughout training.

base policy better than action-space residuals. In practice, this can reduce the need for additional low-pass filtering or handcrafted smoothness regularization during training and deployment.

D. ZPRL in the Real World

We further evaluate ZPRL on four challenging real-world manipulation tasks: `Place Orange`, `Flip Egg`, `Open Box`, and `Insert Bills` (Fig. 6). These tasks cover a diverse range of physical skills, including single-arm pick-and-place, highly dynamic contact-rich manipulation, coordinated bimanual interaction, and deformable-object manipulation (DOM). In `Place Orange`, the robot must grasp one half of a real orange randomly placed on a tray (29 cm $\times$ 20 cm), rotate its wrist, and place the orange half onto the filter of a juicer. This task requires reliable visual localization, accurate grasping, and precise pose alignment at placement. In `Flip Egg`, the robot starts with a spatula grasped by a 3D-printed gripper and must insert it underneath a sunny-side-up egg model, flip the egg with a high end-effector (EE) acceleration of up to 6 $\mathrm{m/s^2}$, and drag it back to the center of the pan. This task is highly dynamic and sensitive to insertion depth and contact timing. In `Open Box`, two robot arms must coordinate to open the latches on both sides of a box, after which the right arm lifts and releases the lid. This task involves bimanual coordination, contact-rich interaction, and temporal synchronization between the two manipulators. In `Insert Bills`, two arms must cooperatively pick up four banknotes from the table, open a wallet, and insert the bills into the narrow inner pocket. This task requires bimanual coordination, DOM, and contact-rich insertion under partial occlusion. Together, these tasks form a real-world benchmark that spans increasing levels of difficulty in dynamics, contact, and coordination, allowing us to assess whether the advantages of ZPRL remain beyond simulation.

1) Training Base Policies: For each task, we first collect human demonstrations with a Virtual Reality (VR) headset and use them to train the base flow policy. We adopt the same model architecture as in simulation, with slightly increased model capacity for real-world observations; detailed hyperparameters are provided in the Appendix. The amount of offline data and the corresponding collection time are summarized in Table III.

Table III. Amount of training data and collection time for each real-world task, reported as [offline | online]. The number of online (o, a) pairs is not directly comparable to the offline count, because online action chunks are added to the replay buffer end-to-end, rather than as overlapping windows in the offline dataset.

Task	# of trajs.	# of (o, a)	Time (h)
`Place Orange`	100 | 370	29.1k | 6.6k	1.0 | 3.5
`Flip Egg`	150 | 1350	20.2k | 10.5k	2.0 | 8.5
`Open Box`	82 | 330	23.5k | 5.5k	1.0 | 4.0
`Insert Bills`	100 | 949	76.8k | 30.3k	2.5 | 12.5

The resulting base policies already achieve meaningful SRs, providing a reasonable starting point for online finetuning. However, they still exhibit several failure modes in real-world execution, as illustrated in Fig. 7. In `Place Orange`, failures come from inaccurate grasping and collisions with the juicer during placement. In `Flip Egg`, shallow insertion may fail to lift the egg, while overly aggressive motion may throw the egg out of the pan. In `Open Box`, the policy may miss either the left or the right latch, causing the opening to fail. These remaining errors indicate that imitation alone does not fully resolve the fine-grained control and contact uncertainties in the real world, leaving clear room for improvement through online RL. In `Insert Bills`, the bills may stop halfway during insertion due to partial occlusion, or bend and collide with the wallet edge.

2) Real-World Online RL: For `Place Orange`, `Flip Egg`, and `Open Box`, we finetune both Po-Dec and ZPRL from the same offline base checkpoint under the same interaction budget. For the more time-consuming `Insert Bills`, we only run ZPRL, and use it as an additional real-world stress test for evaluating whether the proposed method remains deployable in a more complex bimanual DOM setting. At the beginning of each episode, the task object is randomly initialized within a predefined region of the workspace: the tray in `Place Orange`; half of the pan in `Flip Egg`; a 35 cm $\times$ 20 cm rectangular area with an additional rotation range within $\pm 10^\circ$ in `Open Box`. For `Insert Bills`, the banknotes are placed below the right arm with loose alignment and a lateral displacement within ± 3 cm, while the wallet is placed in the upper-left region of the table with its horizontal position randomized within ± 5 cm and its orientation randomized within $\pm 10^\circ$. The policy takes image observations together with robot joint angles as input, and outputs a chunk of desired EE poses for the next few timesteps to low-level controllers. A human supervisor monitors the rollout, resets the workspace between episodes, and provides the final sparse reward. Specifically, we use $r_t = 1$ if and only if the task is successfully completed, and $r_t = 0$ otherwise. Each episode terminates upon one of three conditions: (i) task success; (ii) an irrecoverable failure; or (iii) reaching a predefined maximum horizon. For implementation simplicity, we use a synchronous interaction-and-update loop rather than an asynchronous actor–learner architecture because the bottleneck of time cost is

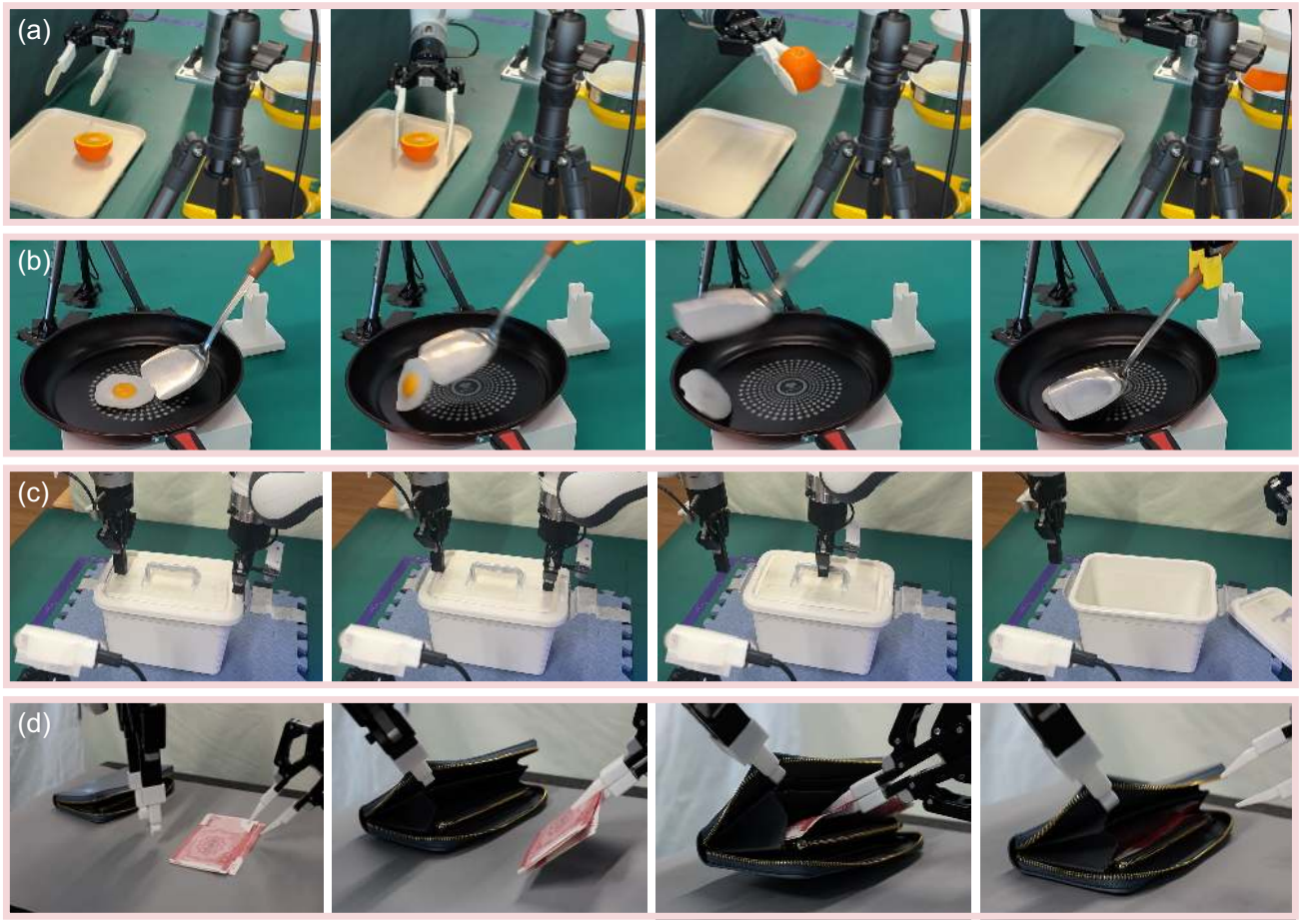

Fig. 6. **Rollout trajectories for four real-world tasks.** Each row shows temporally ordered, subsampled snapshots from one rollout. From top to bottom: (a) `Place Orange`, (b) `Flip Egg`, (c) `Open Box`, and (d) `Insert Bills`.

interaction, rather than updating, in real-world RL. During data collection, the policy is updated with UTD $= 5$ for `Insert Bills` due to its complexity and UTD $= 2$ for other tasks.

3) Main Results: We report the success rate (SR) and the average episode length by evaluating the checkpoints with 40 randomly initialized trajectories at fixed intervals during finetuning, plotted against the number of executed controller steps. The learning curves are shown in Fig. 8.

On the three tasks where both methods are evaluated (`Place Orange`, `Flip Egg`, and `Open Box`), ZPRL reaches high SR faster than Po-Dec and achieves higher final SR, with larger gains on the more challenging `Flip Egg` (by roughly 12.5%) and `Open Box` (by about 7.5%). At the same time, ZPRL also reduces episode length by about 6% on both `Flip Egg` and `Open Box` compared with Po-Dec, indicating better online learning efficiency in real-world finetuning. These results suggest that the pretrained bottleneck latent provides a more effective RL interface across diverse real-world scenarios. One exception is the longer episode of ZPRL in `Place Orange`. We attribute this to the higher control frequency (30 Hz, versus 20 Hz in `Open Box` and 10 Hz in `Flip Egg`) and the fact that cautious end-effector alignment quickly adds environment steps, often yielding more reliable execution at the cost of a longer trajectory.

On the additional `Insert Bills`, where only ZPRL is trained online due to the high hardware cost, ZPRL substantially improves the base policy from a rather low SR (20%) to a final SR of 77.5%. This result suggests that the proposed latent-steering interface can also be deployed in complex tasks involving fine-grained manipulation, deformable objects, and bimanual coordination, and can improve itself through interaction.

4) Robustness of ZPRL Policies: We further evaluate the robustness of final ZPRL policies under disturbances that are not seen during online training. We design task-specific perturbations, including human intervention after the episode starts, object replacement, and out-of-distribution (OOD) initial layouts. These test cases are illustrated in Fig. 9. We roll out the policy 10 times for each case, and summarize the zero-shot SRs in Table IV.

Overall, ZPRL achieves an average SR of 69% across all disturbed settings, indicating that the online finetuned policies retain non-trivial robustness beyond the nominal training distribution. In particular, the policies remain reasonably robust to direct human disturbances, with SRs of 0.7 on `Place Orange`, 0.8 on `Flip Egg`, and 0.6 on `Open Box`. They also generalize well to moderate appearance changes: the `Place Orange` policy achieves 1.0 SR on the visually different plastic orange model, while the `Flip Egg` policy achieves 0.8 and 0.9 SR on eggs with different shape and color, respectively. For `Open Box`, the policy still achieves 0.7 SR when the box is initialized with an additional positional (2 cm)

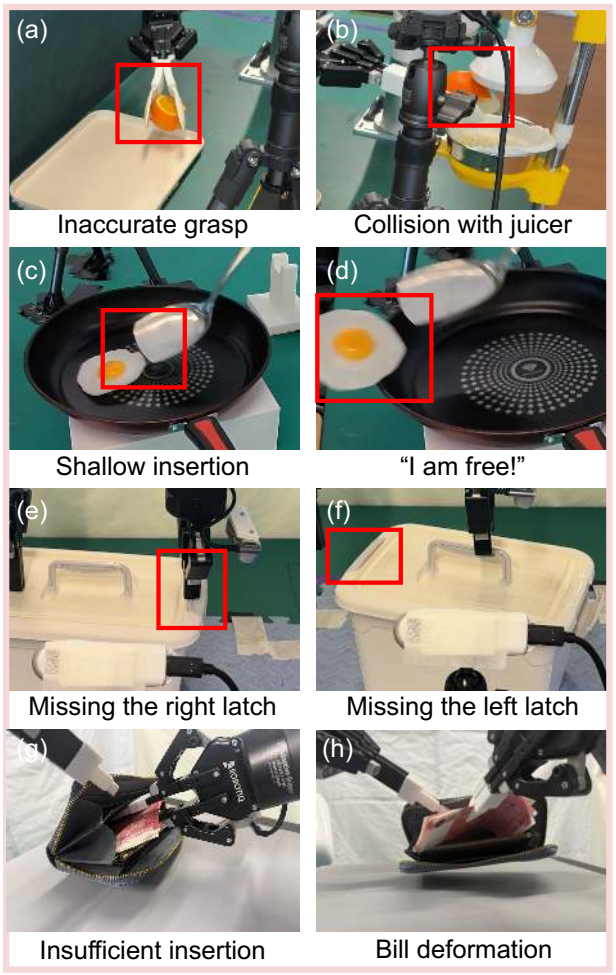

Fig. 7. **Common failure modes in the real world.** In `Place Orange`: (a) inaccurate grasping and (b) collision with the juicer. In `Flip Egg`: (c) shallow insertion that fails to flip the egg and (d) overly high end-effector velocity causing the egg to fly out of the pan. In `Open Box`: (e) missing the right or (f) the left latch. In `Insert Bills`: (g) insertion stops halfway due to partial occlusion and (h) bills bend or collide with the wallet.

Table IV. Success rate (SR) under human disturbances, novel objects, and OOD initial layouts. All policies are evaluated zero-shot after online finetuning.

Task	Test Case	SR
`Place Orange`	Human disturbance	0.7
	Model orange	1.0
`Flip Egg`	Human disturbance	0.8
	Different shape	0.8
	Different color	0.9
	Real egg	0.5
`Open Box`	Human disturbance	0.6
	OOD position	0.7
`Insert Bills`	Visual distractors	0.4
	Human disturbance	0.5
Average		0.69

or rotational (10°) offset, suggesting tolerance to mild layout shifts.

A challenging case is the *Real egg* setting in `Flip Egg`, where SR drops to 0.5. Besides color or shape changes, replacing the rigid model egg with a much softer real egg changes the contact mechanics of insertion and flipping. In this case, a strategy that only inserts the spatula by a few

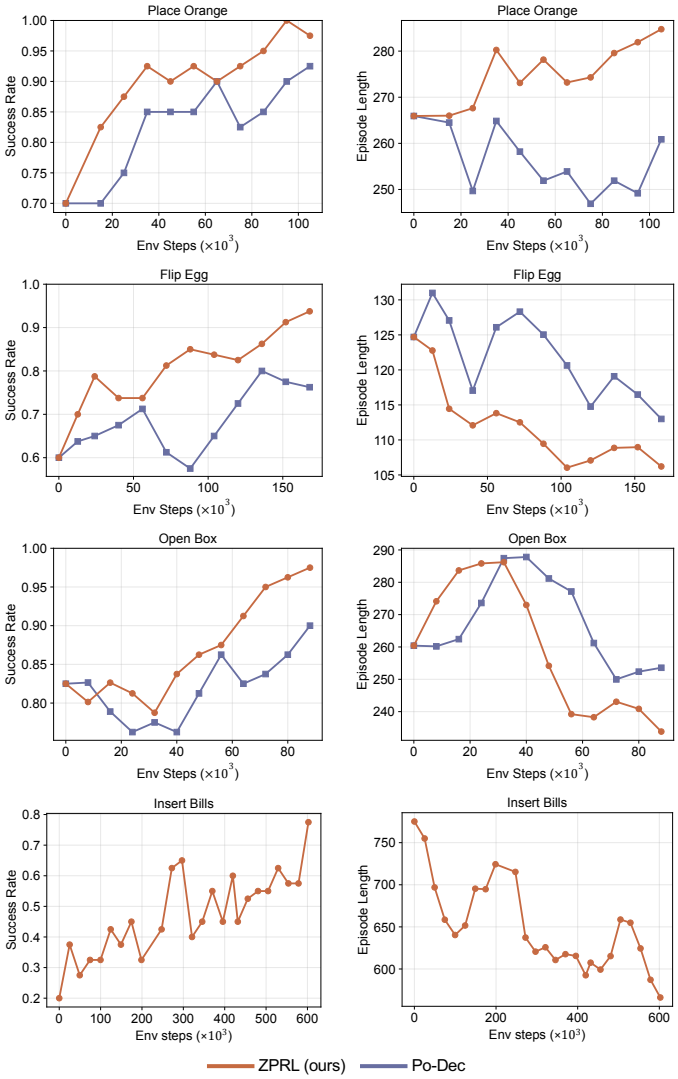

Fig. 8. **Real-world online RL learning curves on four tasks.** Success rate (SR, left) and average episode length (right) are plotted against environment steps. For `Insert Bills`, only ZPRL is trained online due to the substantially higher hardware and time cost.

centimeters, which is often sufficient for the model egg, is no longer reliable.

The policy shows lower robustness on disturbed `Insert Bills` because this task is highly sensitive to both visual and physical perturbations. The thin and deformable paper bills can easily bend or become misaligned after disturbance, which makes fine-grained insertion and recovery difficult. Since such recovery behaviors are not well covered in the offline dataset, SRs drop substantially under these conditions. Therefore, the robustness of ZPRL should be understood as robustness to moderate disturbances around the training manifold, rather than invariance to arbitrarily large physical or dynamical shifts.

5) Smoothness of ZPRL Actions: We next examine the smoothness of online exploration in the real world. Starting from the same initial state, we record representative trajectories of Po-Dec and ZPRL at middle-stage checkpoints in `Place Orange` when enabling exploration. We project the EE positions onto the image plane. The resulting trajectories are shown in Fig. 10, where darker dots indicate later positions in time.

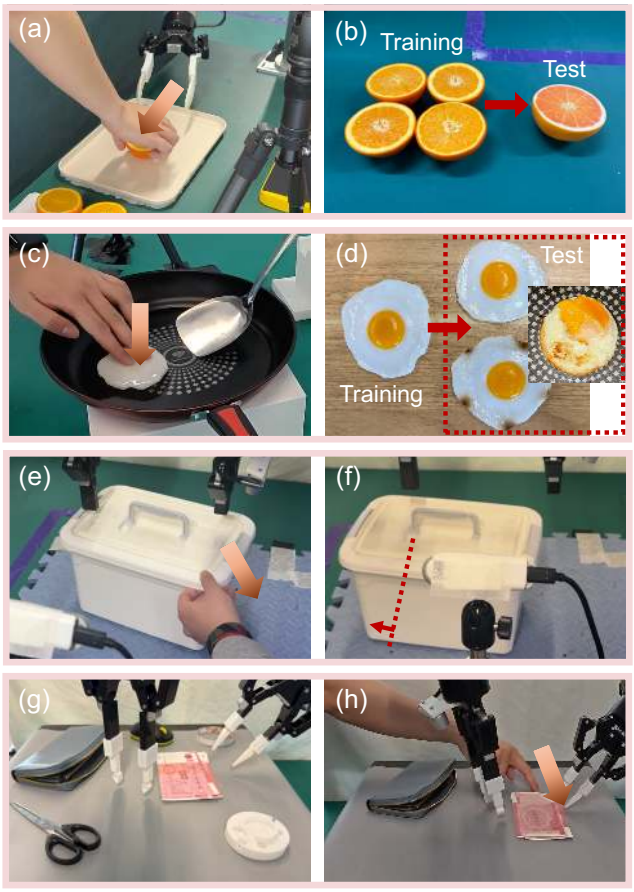

Fig. 9. **Robustness test cases for evaluating ZPRL under different disturbances.** In (a), (c), (e), and (g), a human perturbs the object after the episode begins. In (b) and (d), the training object is replaced with a novel one. In (f), the initial object pose is perturbed with additional positional or rotational offsets. In (h), several distractors are placed on the workspace.

ZPRL produces substantially more coherent and smooth motion than Po-Dec. Its EE trajectory follows a relatively consistent path, while Po-Dec exhibits frequent oscillations. This difference is especially important in real-world control, where unstable motion can easily induce contact failures or object slippage. To make Po-Dec deployable on hardware, we additionally apply a temporal filter that averages the desired poses over the most recent three steps to reduce jerkiness. Even with this stabilization, Po-Dec still explores less smoothly than ZPRL.

This difference aligns with one motivation of our method. ZPRL perturbs a bottleneck latent and then decodes it through a frozen model, so exploratory actions remain constrained to behaviorally meaningful directions. In contrast, directly adding residuals in action space more easily produces high-frequency and less structured motion. We believe this is one key reason why Po-Dec is compromised on the highly dynamic `Flip Egg`, where success depends on both rapid motion and precise control of contact.

E. Limitations and Future Directions

Despite the asymptotic SR and learning efficiency of ZPRL, the current framework still has several limitations that lead us to promising directions for future work.

Fig. 10. **Representative trajectories of Po-Dec (left) and ZPRL (right) from the same initial state.** Dots denote recorded EE positions projected onto the image frame. Darker dots indicate later timesteps along the trajectory.

Dependence on the Base Policy. Because ZPRL steers a frozen pretrained policy rather than fully finetuning all model parameters, its performance is bounded by the support of the base policy. In many cases, online steering can substantially improve runtime metrics such as SR by making the generated action chunks more suitable for the current state distribution. However, this improvement may come more from *recomposing* or reweighting behaviors already encoded in the base policy than from *creating* new behaviors. As a result, when solving rare corner cases requires skills that are poorly represented or entirely absent in the offline dataset, ZPRL may have limited room for improvement, such as flattening bent bills in the inner pocket in `Insert Bills`. This dependency on the base policy is also one reason why the method does not reach perfect success on all tasks. A practical way to alleviate this issue may be to enlarge the offline dataset and rerun the offline-to-online pipeline.

When to Train the VIB Module. In this work, the VIB encoder and decoder are trained jointly with the base policy during offline pretraining. Although the VIB branch does not interfere with the original policy optimization objective, this design still assumes access to the offline training stage. An open question is whether a bottleneck latent interface can be attached to an already trained policy checkpoint in a post-hoc manner. If this is possible, ZPRL could be applied to a much wider range of publicly available pretrained robot policies without repeating large-scale offline training, substantially reducing the cost of adoption.

Extension to Broader Policy Architectures. Our current formulation is built on an observation-encoder and action-decoder structure, where the task-relevant latent can be naturally extracted and perturbed before action generation. While this design is common in robot policies, advanced models increasingly rely on more entangled architectures, such as multi-layer cross-attention between visual tokens and action queries [2]. In such settings, task-relevant information may be distributed across multiple hidden states rather than concentrated in a single embedding. It remains unclear which representation should be selected as the steering interface, and whether a compact bottleneck latent would still be expressive enough to adapt downstream action generation. Exploring how to generalize the steering principle to these more complex architectures is an important direction. A concurrent work suggests one possible way by extracting a task-relevant token from a sequence of backbone embeddings using an additional transformer encoder-decoder module [60]. However, the authors train another actor with regularization instead of steering the base policy.

Table V. Hyperparameters of the online RL policy. Values are reported as [simulation | real world] when they differ.

Hyperparameters	Value
Batch size	256
Actor learning rate	1e-4
Critic learning rate	3e-4
Discount factor γ	0.99 (transport 0.997, `Insert Bills` 0.998)
Optimizer	Adam
UTD	1 | 2 (5 for `Insert Bills`)
Target entropy	$-d/2$
Target update rate (τ)	0.005
Initial temperature	0.01
Hidden size	256
# of layers	4
Activation	GELU
# of critics	2 | 5 (10 for `Insert Bills`)
Actor parameterization	tanh(Gaussian)
Actor logstd max	2
Actor logstd min	-10

Beyond Reinforcement Learning. Finally, although this work focuses on online RL, the idea of policy steering through a bottleneck latent may extend beyond RL. Prior work on iterative IL with VIB has shown that collecting additional data over multiple rounds can improve policy quality [12], but such methods typically retrain or finetune the entire policy after each data collection round. It is worth studying whether latent-space steering can serve as a lighter-weight alternative when the offline dataset is already large and each new round contributes only a relatively small amount of additional data. In that regime, steering may offer a more targeted way to adapt to long-tail or failure-prone cases, rather than repeatedly mixing all collected samples into a single growing dataset.

VI. Conclusion

We presented Z-Perturbation Reinforcement Learning (ZPRL), a lightweight post-training framework that adapts pretrained robot manipulation policies through online RL by steering a compact bottleneck latent rather than policy weights or output actions. By learning a task-relevant latent interface during offline training and perturbing it with RL while freezing the pretrained action prior, ZPRL provides a structured and effective control space for online adaptation. Across simulation and real-world tasks where baselines are available, ZPRL consistently improves sample efficiency and final performance over strong post-training baselines, while producing smoother and more coherent exploratory behaviors, especially in dynamic and contact-rich settings. These results suggest that the choice of steering interface is a key factor in RL post-training, and that a compact, semantically organized latent offers a practical middle ground between full finetuning and direct action-space correction.

APPENDIX

Additional Experimental Details

Model Architectures and Training.

Our implementation is built on top of Diffusion Policy [7], but replaces the diffusion objective with the flow-matching objective in Equation (1) and adds the online RL pipeline. The policy architectures in simulation and in the real world share the same overall design. The observation representation consists of two parts: (1) an image embedding from each camera at each observation step, with embedding dimension 256, produced by a ResNet-18 encoder trained from scratch followed by a linear projection; and (2) a raw proprioceptive vector, namely end-effector poses in simulation and joint angles in the real world.

The flow policy backbone is a 1D U-Net with convolutions applied along the action-chunk dimension. The only architecture difference between simulation and real-world experiments is the channel width of the 1D U-Net, which is 128-256-512 in simulation and 256-384-512 in the real world. The added VIB module consists of two multi-layer perceptrons

Table VI. Task-specific hyperparameters for ZPRL in simulation. Here $\dim(\boldsymbol{z})$ is the bottleneck latent dimension, λ is the residual perturbation scale, $\dim(\boldsymbol{q})$ is the proprioceptive dimension, and $\dim(a)$ is the action dimension per step.

Hyperparameters	Can	Square	Transport	Box-close
$\dim(\boldsymbol{z})$	16	16	32	16
λ	0.25	0.2	0.1	1.5
Image size	84×84	84×84	84×84	84×84
Crop size	76×76	76×76	76×76	80×80
# of cameras	2	2	4	1
$\dim(\boldsymbol{q})$	9	9	18	9
Obs chunk size	1	1	1	1
$\dim(a)$	10	10	20	4
Action chunk size	4	4	5	2
Action repeat	1	1	1	1
# of warm-up chunks	1000	2000	4000	10000
Hyperparameters	**Door**	**Hammer**	**Pen**	**Push-wall**
$\dim(\boldsymbol{z})$	16	16	16	16
λ	0.75	0.5	0.4	1.5
Image size	84×84	84×84	84×84	84×84
Crop size	80×80	80×80	80×80	80×80
# of cameras	1	1	1	1
$\dim(\boldsymbol{q})$	24	24	24	9
Obs chunk size	1	1	1	1
$\dim(a)$	28	26	24	4
Action chunk size	2	2	2	2
Action repeat	2	2	2	1
# of warm-up chunks	4000	4000	2000	10000

Table VII. Task-specific hyperparameters for ZPRL in the real world.

Hyperparameters	`Place Orange`	`Flip Egg`
$\dim(\boldsymbol{z})$	32	32
λ	0.2	0.2
Image size	110×280	128×128
Crop size	106×276	124×124
# of cameras	1	1
$\dim(\boldsymbol{q})$	7	7
Obs chunk size	1	3
$\dim(a)$	7	6
Action chunk size	16	16
# of warm-up chunks	300	300
Env Frequency (Hz)	30	10
Hyperparameters	**`Open Box`**	**`Insert Bills`**
$\dim(\boldsymbol{z})$	32	32
λ	0.175	0.15
Image size	128×128	120×160
Crop size	124×124	116×156
# of cameras	3	3
$\dim(\boldsymbol{q})$	16	16
Obs chunk size	1	2
$\dim(a)$	16	16
Action chunk size	16	24
# of warm-up chunks	500	500
Env Frequency (Hz)	20	20

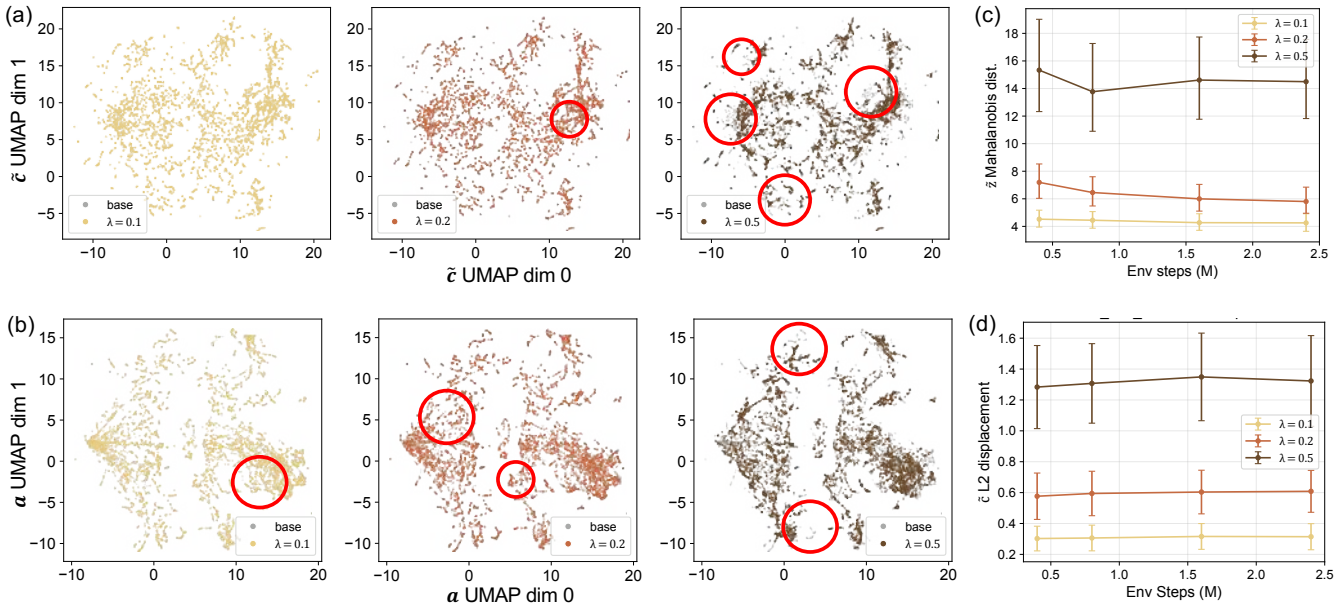

Fig. 11. **What ZPRL changes during online finetuning.** (a) UMAP projections of the decoded observation embedding $\tilde{\boldsymbol{c}}$ and (b) the generated action $\boldsymbol{a}$ on `square` at 0.4M environment steps, comparing samples from the base policy and ZPRL policies under different perturbation scales λ. The SRs for each checkpoint are 0.51 ($\lambda = 0.1$), 0.56 ($\lambda = 0.2$), 0.0 ($\lambda = 0.5$), respectively. Red circles highlight representative regions with clear mismatch after steering. (c) Mean Mahalanobis distance of perturbed latents $\tilde{\boldsymbol{z}}$ to the offline latent distribution and (d) mean L2 displacement between decoded embeddings with and without perturbation. Larger λ induces stronger latent shifts and larger decoded-feature changes.

(MLPs), namely the encoder and decoder, each with four layers of width 256 and GELU activations. During offline training, we discretize the flow schedule into 100 steps, i.e., $1, 0.99, 0.98, \ldots, 0.01$. During inference, we use a 2-step sampling schedule ($1 \rightarrow 0.01 \rightarrow 0$) in both simulation and real-world experiments, as we find it sufficient to preserve strong base policy performance while enabling high-frequency control. Offline pretraining lasts for 1000 epochs with batch size 128 or 256 depending on the dataset size.

The online policy π_{on} uses the same architecture in both simulation and real-world experiments. Its hyperparameters are summarized in Table V. One exception is the `Adroit` tasks, where we add layer normalization to the critic networks to stabilize training under dense rewards. Tables VI and VII summarize the task-specific settings for the simulation and real-world experiments, respectively.

Practical guideline for choosing λ. A practical question is how large the latent perturbation should be. Intuitively, λ should be chosen to ensure that the online residual has substantial influence on the bottleneck to steer the base policy. If λ is too large, however, the perturbed latent may move outside the local support of the pretrained latent distribution, making the decoded feature less reliable and degrading the frozen action prior, as shown in the next section. In practice, we find it useful to choose λ according to the ratio between the magnitude of perturbation $\Delta \boldsymbol{z}$ and the original $\boldsymbol{z}$. A convenient starting point is to set λ such that $\mathrm{RMS}(\lambda \Delta \boldsymbol{z})$ is roughly 10%–20% of $\mathrm{RMS}(\boldsymbol{z})$ (RMS, root-mean-square), and then adjust it according to early online learning behavior: increasing λ when learning is stable but slow, and decreasing it when performance degrades quickly. Final task-specific values are listed in Tables VI and VII.

Real-world platform. The real-world experiments are conducted on an xArm-6 (in `Place Orange`) and a Franka Emika Panda (in `Flip Egg`) robot arm(s) with third-view RGB cameras (wrist-mounted cameras on both arms in `Open Box`and `Insert Bills`) and Robotiq 2F-85 grippers. Policies output desired end-effector pose commands (and gripper width if applicable) executed by a low-level interpolation controller. The `Flip Egg` uses a 3D-printed spatula holder, and `Place Orange` uses two 3D-printed rubber tong holders.

What Is ZPRL Doing?

To better understand what ZPRL is doing, and especially how the perturbation scale λ affects training, we visualize how online RL changes the policy's intermediate representations and outputs on the `square` task. We use UMAP (Uniform Manifold Approximation and Projection [61]) to project the decoded observation embedding $\tilde{\boldsymbol{c}}$ and the generated action $\boldsymbol{a}$, before and after online finetuning, to a 2D plane. We focus on checkpoints at 0.4M environment steps, where similar patterns also appear at other stages of training.

As shown in Fig. 11(a) and (b), ZPRL appears to mainly *remap* state–action associations rather than invent entirely new actions. After adding the perturbation $\lambda \Delta \boldsymbol{z}$ to the base latent, the RL policy changes the decoded feature that conditions the frozen action generator, so that the sampled action becomes better aligned with reward objective given current state instead of simply replaying behavior from the offline dataset [53]. In the UMAP visualizations, the point clouds of $\tilde{\boldsymbol{c}}$ and $\boldsymbol{a}$ after finetuning still largely overlap with those of the base policy, but their local density and pairing structure are reorganized; representative mismatched regions are highlighted by red circles. A larger λ leads to a stronger reorganization.

However, this steering must remain local to be effective. To quantify this, we track both the out-of-distribution (OOD)

degree of the perturbed latent $\tilde{\boldsymbol{z}}$ and the displacement of the decoded feature caused by the perturbation. Specifically, we fit a Gaussian distribution to 10,000 latent samples $\boldsymbol{z}$ from the offline `square` dataset using a Ledoit–Wolf covariance estimator [62], and compute the Mahalanobis distance [63] of perturbed latents $\tilde{\boldsymbol{z}}$ produced by the online policy. We also compute the L2 distance between the decoded feature with perturbation, $\tilde{\boldsymbol{c}}_{\text{on}}$, and that without perturbation, $\tilde{\boldsymbol{c}}_{\text{off}}$. All computations are implemented with `sklearn` [64].

As shown in Fig. 11(c) and (d), increasing λ consistently increases both the OOD score of $\tilde{\boldsymbol{z}}$ and the decoded-feature displacement. When λ becomes too large, the frozen VIB decoder is forced to extrapolate to latents far outside its training support. The resulting decoded feature then deviates substantially from the original one, making the downstream action generator produce less meaningful actions for the current state. In our experiments, keeping the average L2 displacement between $\tilde{\boldsymbol{c}}_{\text{on}}$ and $\tilde{\boldsymbol{c}}_{\text{off}}$ on the offline dataset below about 0.8 works well in practice. A useful starting point is to choose λ such that $\mathrm{RMS}(\lambda\Delta\boldsymbol{z}) \approx 0.1\,\mathrm{RMS}(\boldsymbol{z})$, then increase it if learning is stable but slow, or decrease it if performance degrades rapidly.

Different β During Offline Training

We also ablate the KL weight β used in the VIB objective (Equation (5)) during offline training. Starting from base policies trained with different β values, we run the same online RL finetuning procedure on `square` with a fixed perturbation scale $\lambda = 0.2$. As shown in Fig. 12, the resulting learning curves are highly similar across $\beta = 10^{-3}, 10^{-4}, 10^{-5}$, with no clear difference in either convergence speed or final success rate. This suggests that, within a reasonable range, the offline KL regularization strength has limited impact on the effectiveness of ZPRL during online finetuning. In the main experiments, we therefore use $\beta = 10^{-4}$ as a default setting.

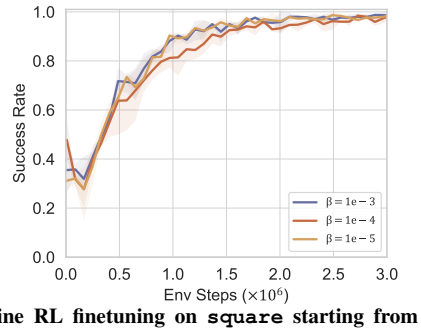

Fig. 12. **Online RL finetuning on `square` starting from base policies trained with different KL weights β.** All variants use the same online setting with $\lambda = 0.2$. The three learning curves are highly similar, indicating limited sensitivity to β within this range.

REFERENCES

[1] T. L. Team, J. Barreiros, A. Beaulieu *et al.*, "A careful examination of large behavior models for multitask dexterous manipulation," 2025. [Online]. Available: https://arxiv.org/abs/2507.05331

[2] K. Black, N. Brown, D. Driess *et al.*, "π_0: A Vision-Language-Action Flow Model for General Robot Control," in *Proc. Robot. Sci. Syst.*, LosAngeles, CA, USA, June 2025.

[3] M. J. Kim, C. Finn, and P. Liang, "Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success," in *Proc. Robot. Sci. Syst.*, LosAngeles, CA, USA, June 2025.

[4] T. Z. Zhao, V. Kumar, S. Levine *et al.*, "Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware," in *Proc. Robot. Sci. Syst.*, Daegu, Republic of Korea, July 2023.

[5] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Proc. Adv. Neural Inf. Process. Syst.*, H. Larochelle, M. Ranzato, R. Hadsell *et al.*, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 6840–6851.

[6] X. Liu, C. Gong, and Q. Liu, "Flow straight and fast: Learning to generate and transfer data with rectified flow," in *Proc. Int. Conf. Learn. Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=XVjTT1nw5z

[7] C. Chi, S. Feng, Y. Du *et al.*, "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion," in *Proc. Robot. Sci. Syst.*, Daegu, Republic of Korea, July 2023.

[8] S. Park, Q. Li, and S. Levine, "Flow q-learning," in *Proc. Int. Conf. Mach. Learn.*, ser. Proceedings of Machine Learning Research, A. Singh, M. Fazel, D. Hsu *et al.*, Eds., vol. 267. PMLR, 13–19 Jul 2025, pp. 48 104–48 127. [Online]. Available: https://proceedings.mlr.press/v267/park25f.html

[9] Y. Lu, Y. Tian, Z. Yuan *et al.*, "H^{3}dp: Triply-hierarchical diffusion policy for visuomotor learning," in *Proc. Int. Conf. Learn. Representations*, 2026. [Online]. Available: https://openreview.net/forum?id=Q1CP0iAmOb

[10] J. Luo, C. Xu, J. Wu *et al.*, "Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning," *Sci. Robot.*, 2025.

[11] Y. Jin, J. Lv, W. Yu *et al.*, "Sime: Enhancing policy self-improvement with modal-level exploration," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2025, pp. 9792–9799.

[12] Y. Jin, J. Lv, H. Xue *et al.*, "Soe: Sample-efficient robot policy self-improvement via on-manifold exploration," 2025. [Online]. Available: https://arxiv.org/abs/2509.19292

[13] A. Z. Ren, J. Lidard, L. L. Ankile *et al.*, "Diffusion policy policy optimization," in *Proc. Int. Conf. Learn. Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=mEpqHvbD2h

[14] T. Zhang, C. Yu, S. Su *et al.*, "Reinflow: Fine-tuning flow matching policy with online reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2025. [Online]. Available: https://openreview.net/forum?id=ACagRwCCqu

[15] Z. Yuan, T. Wei, L. Gu *et al.*, "Hermes: Human-to-robot embodied learning from multi-source motion data for mobile dexterous manipulation," 2025. [Online]. Available: https://arxiv.org/abs/2508.20085

[16] K. Lei, H. Li, D. Yu *et al.*, "Rl-100: Performant robotic manipulation with real-world reinforcement learning," 2025. [Online]. Available: https://arxiv.org/abs/2510.14830

[17] K. Chen, Z. Liu, T. Zhang *et al.*, "π_{RL}: Online rl fine-tuning for flow-based vision-language-action models," 2026. [Online]. Available: https://arxiv.org/abs/2510.25889

[18] P. Intelligence, A. Amin, R. Aniceto *et al.*, "$\pi^*_{0.6}$: a vla that learns from experience," 2025. [Online]. Available: https://arxiv.org/abs/2511.14759

[19] N. M. Shafiullah, Z. Cui, A. A. Altanzaya *et al.*, "Behavior Transformers: Cloning k modes with one stone," in *Proc. Adv. Neural Inf. Process. Syst.*, S. Koyejo, S. Mohamed, A. Agarwal *et al.*, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 22 955–22 968.

[20] X. Yuan, T. Mu, S. Tao *et al.*, "Policy decorator: Model-agnostic online refinement for large policy model," in *Proc. Int. Conf. Learn. Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=e5jGTEiJMT

[21] L. Ankile, A. Simeonov, I. Shenfeld *et al.*, "From imitation to refinement - residual rl for precise assembly," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2025, pp. 01–08.

[22] L. Ankile, Z. Jiang, R. Duan *et al.*, "Residual off-policy rl for finetuning behavior cloning policies," 2025. [Online]. Available: https://arxiv.org/abs/2509.19301

[23] T. Johannink, S. Bahl, A. Nair *et al.*, "Residual reinforcement learning for robot control," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 6023–6029.

[24] A. Wagenmaker, Y. Zhang, M. Nakamoto *et al.*, "Steering Your Diffusion Policy with Latent Space Reinforcement Learning," in *Proc. Conf. Robot Learn.* PMLR, 2025, pp. 258–282.

[25] A. A. Alemi, I. Fischer, J. V. Dillon *et al.*, "Deep Variational Information Bottleneck," in *Proc. Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=HyxQzBceg

[26] A. J. Ijspeert, J. Nakanishi, H. Hoffmann *et al.*, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Comput.*, vol. 25, no. 2, pp. 328–373, 02 2013. [Online]. Available: https://doi.org/10.1162/NECO_a_00393

[27] A. Paraschos, C. Daniel, J. R. Peters *et al.*, "Probabilistic movement primitives," in *Proc. Adv. Neural Inf. Process. Syst.*, C. Burges, L. Bottou, M. Welling *et al.*, Eds., vol. 26. Curran Associates, Inc., 2013. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2013/file/e53a0a2978c28872a4505bdb51db06dc-Paper.pdf

[28] C. Chu and H. Xu, "Da-mmp: Learning coordinated and accurate throwing with dynamics-aware motion manifold primitives," 2026. [Online]. Available: https://arxiv.org/abs/2509.23721

[29] A. Brohan, N. Brown, J. Carbajal *et al.*, "Rt-1: Robotics transformer for real-world control at scale," 2022. [Online]. Available: https://arxiv.org/abs/2212.06817

[30] Y. Ze, G. Zhang, K. Zhang *et al.*, "3D Diffusion Policy: Generalizable Visuomotor Policy Learning via Simple 3D Representations," in *Proc. Robot. Sci. Syst.*, Delft, Netherlands, July 2024.

[31] M. Janner, Y. Du, J. Tenenbaum *et al.*, "Planning with diffusion for flexible behavior synthesis," in *Proc. Int. Conf. Mach. Learn.* PMLR, 2022, pp. 9902–9915.

[32] Y. Tian, S. Cheng, T. Wei *et al.*, "Vitas: Visual tactile soft fusion contrastive learning for visuomotor learning," 2026. [Online]. Available: https://arxiv.org/abs/2602.11643

[33] Y. Zhu, J. Wong, A. Mandlekar *et al.*, "robosuite: A modular simulation framework and benchmark for robot learning," 2020. [Online]. Available: https://arxiv.org/abs/2009.12293

[34] A. Mandlekar, D. Xu, J. Wong *et al.*, "What matters in learning from offline human demonstrations for robot manipulation," in *Proc. Conf. Robot Learn.*, ser. Proceedings of Machine Learning Research, A. Faust, D. Hsu, and G. Neumann, Eds., vol. 164. PMLR, 08–11 Nov 2022, pp. 1678–1690. [Online]. Available: https://proceedings.mlr.press/v164/mandlekar22a.html

[35] A. Khazatsky, K. Pertsch, S. Nair *et al.*, "DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset," in *Proc. Robot. Sci. Syst.*, Delft, Netherlands, July 2024.

[36] Q. Vuong, S. Levine, H. R. Walke *et al.*, "Open x-embodiment: Robotic learning datasets and RT-x models," in *Towards Generalist Robots: Learning Paradigms for Scalable Skill Acquisition @ CoRL2023*, 2023. [Online]. Available: https://openreview.net/forum?id=zraBtFgxT0

[37] C. Li, R. Zhang, J. Wong *et al.*, "Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation," in *Proc. Conf. Robot Learn.* PMLR, 2023, pp. 80–93.

[38] M. Du and S. Song, "Dynaguide: Steering diffusion polices with active dynamic guidance," in *Proc. Adv. Neural Inf. Process. Syst.*, 2025. [Online]. Available: https://openreview.net/forum?id=XOw7Yf8qN3

[39] S. Yang, Y. Zhang, H. He *et al.*, "Steering vision-language-action models as anti-exploration: A test-time scaling approach," 2025. [Online]. Available: https://arxiv.org/abs/2512.02834

[40] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[41] S. Levine, A. Kumar, G. Tucker *et al.*, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020. [Online]. Available: https://arxiv.org/abs/2005.01643

[42] K. Lei, Z. He, C. Lu *et al.*, "Uni-o4: Unifying online and offline deep reinforcement learning with multi-step on-policy optimization," in *Proc. Int. Conf. Learn. Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=tbFBh3LMKi

[43] H. Li, K. Lei, S. Zang *et al.*, "Failure-Aware RL: Reliable Offline-to-Online Reinforcement Learning with Self-Recovery for Real-World Manipulation," *arXiv e-prints*, p. arXiv:2601.07821, Jan. 2026.

[44] I. Uchendu, T. Xiao, Y. Lu *et al.*, "Jump-start reinforcement learning," in *Proc. Int. Conf. Mach. Learn.* PMLR, 2023, pp. 34 556–34 583.

[45] Z. Zhou, A. Peng, Q. Li *et al.*, "Efficient online reinforcement learning fine-tuning need not retain offline data," in *Proc. Int. Conf. Learn. Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=HN0CYZbAPw

[46] P. J. Ball, L. Smith, I. Kostrikov *et al.*, "Efficient online reinforcement learning with offline data," in *Proc. Int. Conf. Mach. Learn.* PMLR, 2023, pp. 1577–1594.

[47] T. Silver, K. Allen, J. Tenenbaum *et al.*, "Residual policy learning," 2019. [Online]. Available: https://arxiv.org/abs/1812.06298

[48] T. Davchev, K. S. Luck, M. Burke *et al.*, "Residual Learning From Demonstration: Adapting DMPs for Contact-Rich Manipulation," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 4488–4495, 2022.

[49] Z. Sun and S. Song, "From prior to pro: Efficient skill mastery via distribution contractive rl finetuning," 2026. [Online]. Available: https://arxiv.org/abs/2603.10263

[50] Q. Li, Z. Zhou, and S. Levine, "Reinforcement learning with action chunking," in *Proc. Adv. Neural Inf. Process. Syst.*, 2025. [Online]. Available: https://openreview.net/forum?id=XUks1Y96NR

[51] D. Ki, J. Oh, S.-W. Shim *et al.*, "Prior-guided diffusion planning for offline reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2025. [Online]. Available: https://openreview.net/forum?id=lC4WKmTScD

[52] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," in *Proc. Int. Conf. Learn. Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=St1giarCHLP

[53] C. He, X. Liu, G. M. S. Camps *et al.*, "Demystifying robot diffusion policies: Action memorization and a simple lookup table alternative," in *Proc. Int. Conf. Learn. Representations*, 2026. [Online]. Available: https://openreview.net/forum?id=PL0tJOfm7I

[54] T. Haarnoja, A. Zhou, K. Hartikainen *et al.*, "Soft actor-critic algorithms and applications," 2019. [Online]. Available: https://arxiv.org/abs/1812.05905

[55] X. Chen, C. Wang, Z. Zhou *et al.*, "Randomized ensembled double q-learning: Learning fast without a model," in *Proc. Int. Conf. Learn. Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=AY8zfZm0tDd

[56] V. Kumar, "Manipulators and manipulation in high dimensional spaces," Ph.D. dissertation, University of Washington, Seattle, 2016. [Online]. Available: https://digital.lib.washington.edu/researchworks/handle/1773/38104

[57] R. McLean, E. Chatzaroulas, L. McCutcheon *et al.*, "Meta-world+: An improved, standardized, RL benchmark," in *Proc. Adv. Neural Inf. Process. Syst.*, 2025. [Online]. Available: https://openreview.net/forum?id=1de3azE606

[58] G. Xu, R. Zheng, Y. Liang *et al.*, "Drm: Mastering visual reinforcement learning through dormant ratio minimization," in *Proc. Int. Conf. Learn. Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=MSe8YFbhUE

[59] K. Pertsch, Y. Lee, and J. Lim, "Accelerating reinforcement learning with learned skill priors," in *Proc. Conf. Robot Learn.*, ser. Proceedings of Machine Learning Research, J. Kober, F. Ramos, and C. Tomlin, Eds., vol. 155. PMLR, 16–18 Nov 2021, pp. 188–204. [Online]. Available: https://proceedings.mlr.press/v155/pertsch21a.html

[60] C. Xu, J. T. Springenberg, M. Equi *et al.*, "RL Token: Bootstrapping Online RL with Vision-Language-Action Models," 2026. [Online]. Available: https://www.pi.website/research/rlt

[61] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2020. [Online]. Available: https://arxiv.org/abs/1802.03426

[62] O. Ledoit and M. Wolf, "A well-conditioned estimator for large-dimensional covariance matrices," *J. Multivar. Anal.*, vol. 88, no. 2, pp. 365–411, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0047259X03000964

[63] P. C. Mahalanobis, "On the generalised distance in statistics," in *Proc. Natl. Inst. Sci. India*, vol. 12, 1936, pp. 49–55.

[64] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.