Latent Policy Steering through One-Step Flow Policies

Hokyun Im[1] Andrey Kolobov[2] Jianlong Fu[2] Youngwoon Lee[1]
[1]Department of Artificial Intelligence, Yonsei University [2] Microsoft Research

***Abstract*—Offline reinforcement learning (RL) should be ideal for robotics, allowing learning from dataset without risky exploration. Yet, offline RL's performance often hinges on a brittle trade-off between (1) return maximization, which can push policies outside dataset support, and (2) behavioral constraints, which typically require sensitive hyperparameter tuning. Latent steering offers a structural way to stay within dataset support during RL but, in order to approximate action values, existing offline adaptations commonly rely on latent-space critics learned via indirect distillation, which can lose information and hinder convergence. We propose Latent Policy Steering (LPS), which enables high-fidelity latent policy improvement by backpropagating original-action-space Q-gradients through a differentiable one-step MeanFlow policy to update a latent-action-space actor. By eliminating proxy latent critics, LPS allows an original-action-space critic to guide end-to-end latent-space optimization, while the one-step MeanFlow policy serves as a behavior-constrained generative prior. This decoupling yields a robust method that works out-of-the-box with minimal tuning. Across OGBench and real-world robotic tasks, LPS achieves state-of-the-art performance and consistently outperforms behavioral cloning and strong latent steering baselines.**

I. Introduction

Offline reinforcement learning (RL) promises to enable robots to acquire complex behaviors from large-scale, pre-collected datasets without costly and dangerous real-world interaction. Despite recent progress in offline RL [26, 25, 18, 14] and impressive results in simulation [17], reliably transferring these methods to real-world robotics remains challenging.

Most state-of-the-art offline RL algorithms follow the TD3+BC [7] paradigm and its generative variants with more expressive regularization [18, 14]. These approaches aim to maximize return while constraining the learned policy to dataset support by adding a regularization term, weighted by a hyperparameter α. In practice, this formulation introduces a delicate trade-off: weak regularization leads to out-of-distribution actions and extrapolation error, while excessive regularization reduces offline RL to behavioral cloning. The best α is highly sensitive to reward scale, dataset diversity, and model capacity, making extensive hyperparameter sweeps feasible in simulation but prohibitively expensive and risky with real-world robots. This sensitivity limits the practicality and scalability of offline RL in real-world deployment.

This raises a fundamental question: can we enforce behavioral constraints safely and effectively *without* relying on sensitive hyperparameter tuning? Prior work explores structural constraints via latent action models, such as VAEs [30], or by learning skill priors [20]. However, these methods often still require task-specific tuning or additional online interaction. In this work, we draw inspiration from recent advances in online fine-tuning of robot policies. Instead of directly updating the parameters of a generative base policy [26, 25] or distilling it into a simplified one-step actor [18], methods such as DSRL [24] improve behavior by *steering* the generation process through its latent variables. Optimizing the latent input w.r.t. a critic while keeping the pre-trained generative model fixed naturally confines the resulting policy to the data manifold, providing a form of structural regularization without an explicit regularization weight.

However, adapting this latent steering paradigm to the fully offline setting poses a key challenge. Offline datasets provide supervision for an action-space critic $Q(s, a)$, but not for a value function defined over the *latent space*. DSRL addresses this mismatch with noise aliasing, distilling action-space values into an approximate latent-space critic. This additional distillation step can be lossy and may fail to capture the high-

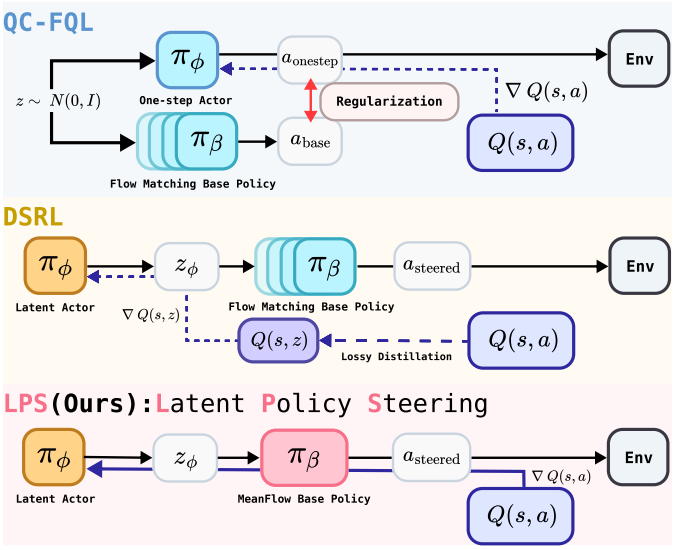

Fig. 1: **Comparison of policy extraction paradigms.** *(Top)* **QC-FQL** constrains the policy via an explicit regularizer, creating a trade-off between reward maximization and behavioral regularization. *(Middle)* **DSRL** resolves this trade-off via latent steering, but requires learning a latent-space critic $Q(s, z)$ via distillation in the offline RL setting. *(Bottom)* **LPS (Ours)** achieves robust, tuning-free optimization by backpropagating action-space critic gradients $\nabla_a Q(s, a)$ through a differentiable one-step generative policy.

frequency details of the true value landscape, limiting the quality of offline policy improvement. As a result, such methods are often used primarily as initializations for subsequent online fine-tuning rather than as standalone offline RL solutions.

To overcome these limitations, we introduce **Latent Policy Steering (LPS)**, a framework that combines the safety of latent steering with direct value-based improvement. LPS leverages **MeanFlow** [8], a differentiable one-step generative model as our base policy, enabling efficient and stable gradient flow from the action space back to the latent space. Unlike DSRL, LPS *directly* optimizes the latent actor using gradients from an action-space critic, bypassing the need for proxy latent critics while preserving the tuning-free structural constraints imposed by the generative prior (Figure 1). This decoupling allows the agent to focus on policy improvement without tuning an explicit behavioral regularization weight, resulting in an out-of-the-box method that consistently matches or surpasses behavioral cloning (BC). We evaluate LPS on standard offline RL benchmarks, confirming its robust, tuning-free nature, and demonstrate strong real-world performance on robotic manipulation tasks, where it reliably improves beyond BC.

Our contributions are:

- We identify two practical bottlenecks for real-world offline RL: the sensitivity induced by explicit behavior regularization and the approximation error induced by indirect latent distillation (e.g., noise aliasing).
- We propose **Latent Policy Steering (LPS)**, which structurally decouples behavioral constraints from reward maximization by enabling **direct** latent policy improvement via backpropagation through a differentiable one-step generative model.
- We demonstrate that LPS achieves state-of-the-art performance on OGBench and exhibits superior practicality on real-world robotic manipulation, consistently outperforming behavioral cloning without task-specific tuning.

II. Related Work

A. Generative Behavior Constraints in Offline RL

TD3+BC [7] is a widely used baseline for offline RL, but standard parametric actor can struggle to model multimodal action distributions common in robotics data. Because of this, recent methods incorporate expressive generative behavior models, including diffusion policies [26**?**] and flow-based models [18], often combined with *action chunking* [29, 4, 14] to capture long-horizon structure. However, many of these methods rely on an explicit trade-off between policy improvement and behavior regularization, controlled by a sensitive hyperparameter, which can be difficult to tune reliably on real robots.

An alternative direction is to adjust this trade-off at inference time. For example, CFGRL [6] applies classifier-free guidance (CFG) [9] to an optimality-conditioned generative policy, enabling controllable interpolation between behavioral adherence and task performance even in large VLA models [10]. In contrast, our approach performs *direct* latent policy improvement using action-space Q-gradients propagated through a differentiable generative policy, preserving the structural constraints of the behavior model while maximizing expected policy return.

B. Reinforcement Learning in Latent Action Spaces

Latent action models address distributional shift in offline RL by restricting optimization to a learned manifold. Approaches such as PLAS [30] and LAPO [2] use variational autoencoders (VAEs) [13] to construct a compact latent space capturing dataset support, then optimize a policy over latents rather than the unconstrained action space to reduce extrapolation error. Related work in hierarchical and skill-based RL [1, 20] similarly leverages latent variables to represent temporally extended behaviors.

This paradigm has recently been extended to expressive generative behavior models. DSRL [24] steers a frozen diffusion policy by optimizing latent inputs with respect to a critic, effectively combining structural constraints with value-driven improvement. In the offline setting, its noise aliasing variant DSRL-NA introduces an additional distillation step to approximate a latent-space critic from an action-space critic. While effective in some settings, this extra approximation can limit purely-offline performance. Our method removes the need for latent critic distillation by backpropagating gradients from an action-space critic through a differentiable one-step generative policy to update the latent actor directly.

C. One-Step Generative Models for Robot Learning

To reduce the cost of iterative denoising, recent work has pivoted towards accelerating sampling via distillation and rectification, including progressive distillation [21], consistency models [27], rectified flow [16] and shortcut models [5]. These techniques have been actively adopted in robot learning to enable fast action sampling and fine-tuning. For instance, Flow Q-Learning (FQL) [18] distills a generative behavior model into a one-step policy for deterministic policy extraction, and other approaches uses consistency-style objectives for fast inference [28] and efficient online fine-tuning [3].

MeanFlow [8] provides a differentiable one-step generative formulation that has recently been applied to robotics and RL. MeanFlowQL [25] integrates a MeanFlow behavior policy into the TD3+BC framework, and MP1 [22] leverages MeanFlow for fast 3D manipulation. We build on this line of work by using MeanFlow as a differentiable mapping from latents to actions, enabling direct latent policy steering via gradients from an action-space critic.

III. Preliminaries

A. Reinforcement Learning with Action Chunking

Action chunking is often crucial in real-world robotics, offering both temporal coherence and improved handling of multi-modality. Following prior work, we adopt Q-Chunking (QC), introduced as part of QC-FQL offline RL algorithm [14], to train action-chunked critics for our method and all baselines.

Rather than predicting a single action a_t at each timestep, an action-chunked policy produces a length-h action sequence

$a_{t:t+h} = (a_t, a_{t+1}, ..., a_{t+h-1})$ via $\pi_\phi(a_{t:t+h} \mid s_t)$, and the corresponding chunked critic is defined as $Q_\theta(s_t, a_{t:t+h})$. This formulation enables temporally coherent behavior generation in an open-loop manner. More importantly, unlike standard n-step returns that introduce off-policy bias, QC supports h-step bootstrapped value backups using in-dataset rewards:

$$\mathcal{L}_Q = \mathbb{E}_{\mathcal{D}}\Big[Q_\theta(s_t, a_{t:t+h}) - \big(r_{t:t+h} + \gamma^h Q_{\bar{\theta}}(s_{t+h}, a_{t+h:t+2h})\big)^2\Big], \tag{1}$$

where $r_{t:t+h} = \sum_{i=0}^{h-1} \gamma^i r_{t+i}$, $a_{t+h:t+2h} \sim \pi_\phi(\cdot \mid s_{t+h})$, and $\bar{\theta}$ is a target network.

To mitigate distribution shift, QC-FQL constrains the learned policy to remain close to the offline behavior distribution. In QC-FQL, this is implemented via a squared 2-Wasserstein upper bound between the learned policy and a behavior policy. Concretely, the policy is parameterized as a one-step flow model $\pi_\phi(s, z)$, and optimized to maximize Q-value while staying close to a flow-based behavior policy $\pi_\beta(s, z)$:

$$\mathcal{L}_{\text{QC-FQL}} = \underbrace{-\mathbb{E}[Q(s, \pi_\phi(s, z))]}_{\text{Extraction}} + \alpha \cdot \underbrace{\mathbb{E}[(\pi_\phi(s, z) - \pi_\beta(s, z))^2]}_{\text{Regularization}}. \tag{2}$$

This objective encourages actions that are both high-value and close to consistent behaviors in the dataset.

B. MeanFlow for One-step Generative Modeling

In this work, we employ **MeanFlow** [8] as our base generative policy. MeanFlow models average velocity (equivalently, displacement) along a probability path, enabling one-step sampling without an auxiliary loss and iterative denoising. Let z_t denote an intermediate state on the probability path connecting the data distribution (at $t = 0$) to a prior latent distribution (at $t = 1$). Standard flow matching models the instantaneous velocity field $v(z_t, t)$, whereas MeanFlow models the *average velocity* $u(z_t, r, t)$ between two time steps $r < t$:

$$u(z_t, r, t) \triangleq \frac{1}{t - r} \int_r^t v(z_\tau, \tau) d\tau. \tag{3}$$

Differentiating Eq. (3) with respect to t yields the *MeanFlow Identity*, which relates the learnable average velocity to the instantaneous velocity:

$$\underbrace{u(z_t, r, t)}_{\text{average vel.}} = \underbrace{v(z_t, t)}_{\text{instant. vel.}} - (t - r) \underbrace{\frac{d}{dt} u(z_t, r, t)}_{\text{time derivative}}. \tag{4}$$

MeanFlow trains a parameterized network u_β to satisfy Eq. (4) by regressing to a target constructed from the right-hand side:

$$\mathcal{L}_{\text{MF}} = \mathbb{E}_{t, z_t}\left[\|u_\beta(z_t, r, t) - \text{sg}(u_{\text{tgt}})\|_2^2\right], \tag{5}$$

where $\text{sg}(\cdot)$ denotes stop-gradient and $u_{\text{tgt}} = v(z_t, t) - (t - r)\,(v(z_t, t)\partial_z u_\beta + \partial_t u_\beta)$. After training, one-step sampling maps a latent z (at $t = 1$) to a data sample (at $t = 0$), an action chunk $\hat{a}$ in our case, via a simple **one-step ODE**:

$$\hat{a} = z - u_\beta(z, 0, 1). \tag{6}$$

This provides a differentiable one-step generative policy that we later exploit for end-to-end gradient propagation.

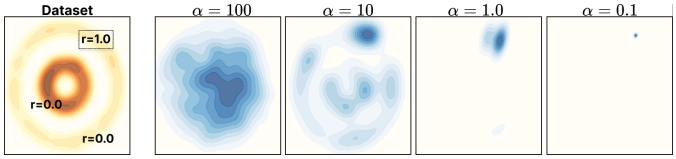

Fig. 2: **Sensitivity to the regularization weight α in FQL.** Learned policy densities on a 2D toy task with reward concentrated in the top-right corner reveal a pattern: large α yields overly conservative policies, while small α encourages out-of-support actions.

C. Limitations of Prior Work

Behavior regularization is sensitive to α. The way behavior-regularized offline RL methods balance value maximization and behavioral adherence – using a weighting hyperparameter α in Eq. (2) – can be fragile even in simple settings. Figure 2 provides an example: large α yields overly conservative policies, while small α encourages out-of-support actions. In general, appropriate α can vary substantially with reward scale and task characteristics. As a result, methods that rely on it often require task-specific hyperparameter sweeps, which is impractical in real-world deployments.

Distilled latent critics can provide poor gradients. Latent steering method (e.g., DSRL) optimize latents by relying on a value function defined in the latent space. In the offline setting, this is typically obtained by distilling the action-space critic through the frozen decoder, i.e., $\min_\phi \mathbb{E}\left[|Q_\phi(s, z) - Q_\theta(\pi_\beta(s, z))|^2\right]$. However, matching values does not guarantee that the latent gradients used for improvement are accurate. As illustrated in Figure 3, even when Q_ϕ approximates values reasonably, its gradient direction $\nabla_z Q_\phi(s, z)$ can deviate substantially from the gradient of the action-space critic $\nabla_z Q_\theta(s, \pi_\beta(z))$, particularly near sharp boundaries of the data manifold. Such gradient mismatch can lead to suboptimal latent updates and degrade purely-offline performance.

IV. Latent Policy Steering (LPS)

We propose **Latent Policy Steering (LPS)**, which addresses both of the above limitations. First, LPS avoids explicit

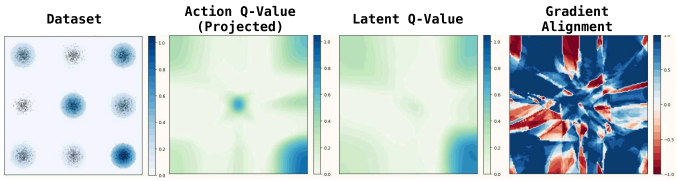

Fig. 3: **Comparing action space Q-value and distilled latent-space Q-value.** Left to right: (1) dataset distribution with reward intensity; (2) action-space Q-value $Q_\phi(s, a)$ projected into the latent space; (3) learned latent Q-value $Q_\phi(s, z)$; (4) cosine similarity between the gradients in (2) and (3).

behavior-regularization trade-off by *separating* reward maximization and distributional constraints: a fixed generative behavior policy defines the support, while a latent actor performs value-driven steering (**resolving α-sensitivity**). Second, LPS eliminates proxy latent critics by *directly* backpropagating action-space critic gradients through a differentiable generative base policy to update the latent actor (**avoiding the inaccurate latent critic**). We instantiate LPS using three key components: a differentiable one-step base policy (Section IV-A), a spherical latent geometry (Section IV-B), and a direct latent optimization objective (Section IV-C).

A. Differentiable Base Policy via MeanFlow

The first component is the base policy $\pi_\beta : \mathcal{Z} \times \mathcal{S} \to \mathcal{A}$, which defines the "safe manifold" or the support of the dataset. While DSRL treats the base policy as a black box, LPS treats it as a *differentiable mapping*. This allows us to backpropagate gradients from the action-space critic to the latent actor through π_β directly.

However, a practical obstacle is that standard diffusion or flow-matching policies typically require iterative sampling, making end-to-end backpropagation expensive and unstable. We therefore employ **MeanFlow** for the base policy, which enables efficient one-step deterministic generation.

Noise-to-action reformulation. In the original MeanFlow formulation, samples are produced by applying a learned displacement to latent noise. Early in training, errors in the displacement filed can amplify output variance, which in turn destabilizes the critic gradients used for steering. Following recent practice [15, 25], we use a **noise-to-action reformulation** in which π_β directly predicts the denoised action (or action chunk) rather than the displacement. Concretely, we write the implied mean velocity u_β and its time derivative as residual quantities:

$$u_\beta(z_t, r, t) = z_t - \pi_\beta(z_t, r, t), \quad \frac{\mathrm{d}u_\beta}{\mathrm{d}t} = v - \frac{\mathrm{d}\pi_\beta}{\mathrm{d}t}. \tag{7}$$

Substituting Eq. (7) into the MeanFlow training objective Eq. (5) yields a numerically more stable training procedure by grounding the training in the action space.

B. Spherical Latent Geometry

Given the base policy (mapping) π_β, we next define the latent space $\mathcal{Z}_{\text{sphere}}$ where the latent actor operates. A known failure mode with unconstrained Gaussian latents is the *"norm explosion"* problem. Because the latent actor is optimized to increase value without explicit bounds, it may increase $|z|$ to query latents that are atypical under the base policy prior, leading to out-of-distribution decoding and unstable learning.

To address this, we leverage the *concentration of measure* property of high-dimensional Gaussians: for $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, most probability mass concentrates on a thin shell of radius $\sqrt{d}$ [23]. This suggests treating the "typical set" of the base policy as naturally spherical. Therefore, we synchronize the support of the base policy and latent actor's output $l_\phi(s)$ by constraining both to the hypersphere S^{d-1} with radius $\sqrt{d}$:

$$\text{Base Policy Latent:} \quad z \sim \sqrt{d} \cdot \frac{\epsilon}{\|\epsilon\|_2}, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d), \tag{8a}$$

$$\text{Latent Actor Output:} \quad z_\phi = \pi_\phi(s) = \sqrt{d} \cdot \frac{l_\phi(s)}{\|l_\phi(s)\|_2}. \tag{8b}$$

By training the base policy using latents sampled from Eq. (8a) and constraining the latent actor via Eq. (8b), LPS ensures that latent actor's queries always remain within the valid coverage of the base policy while maintaining well-conditioned gradients.

Algorithm 1: Latent Policy Steering (LPS)

Initialize: base policy $\pi_\beta(s, z)$ (MeanFlow), latent actor $\pi_\phi(s)$, critic $Q_\theta(s, a)$, action chunk size h
while *not converged* **do**
  Sample batch $\mathcal{B} = \{(s_t, a_{t:t+h}, r_{t:t+h}, s_{t+h})\} \sim \mathcal{D}$
  ▷ 1. Update base policy (MeanFlow behavior prior) π_β
  Sample $z \sim \mathrm{Unif}(\mathcal{Z}_{\text{sphere}})$ ▷ Eq. (8a)
  Update β to minimize $\mathcal{L}_{\text{MF}}$ ▷ Eq. (5), Eq. (7)
  ▷ 2. Update latent actor π_ϕ
  $z_\phi \leftarrow \pi_\phi(s_t)$ ▷ Eq. (8b)
  $a_{\text{pred}} \leftarrow \pi_\beta(s_t, z_\phi)$
  Update ϕ to minimize $\mathcal{L}_{\text{LPS}}$ ▷ Eq. (9)
  ▷ 3. Update critic Q_θ (Q-Chunking)
  Sample $z \sim \mathrm{Unif}(\mathcal{Z}_{\text{sphere}})$ ▷ Eq. (8a)
  $z'_\phi \leftarrow \pi_\phi(s_{t+h})$
  $a'_{\text{pred}} \leftarrow \pi_\beta(s_{t+h}, z'_\phi)$
  Update θ to minimize $\mathcal{L}_Q$ ▷ Eq. (1)

C. Direct Latent Policy Steering

Finally, we learn a latent actor $\pi_\phi : \mathcal{S} \to \mathcal{Z}$ that steers the base policy toward high-value actions. Since the base policy π_β is differentiable, we can optimize π_ϕ directly using an action-space critic Q_θ:

$$\mathcal{L}_{\text{LPS}} = -\mathbb{E}_{s\sim\mathcal{D}}\left[Q_\theta(s, \pi_\beta(s, \pi_\phi(s)))\right]. \tag{9}$$

Gradients of Eq. (9) propagate through π_β via the chain rule, yielding low-variance latent updates without introducing proxy $Q(s, z)$.

The overall training objective of LPS sums the base-policy loss (reformulated MeanFlow), the latent steering loss, and the critic loss:

$$\mathcal{L}_{\text{Total}} = \mathcal{L}_{\text{MF}} + \mathcal{L}_{\text{LPS}} + \mathcal{L}_Q. \tag{10}$$

Notably, LPS does not require an explicit behavior-regularization coefficient α: behavioral constraints are enforced structurally by the fixed generative prior, whlie policy improvement is performed in the safe, synchronized latent space by maximizing the action-space critic. The full procedure is summarized in Algorithm 1.

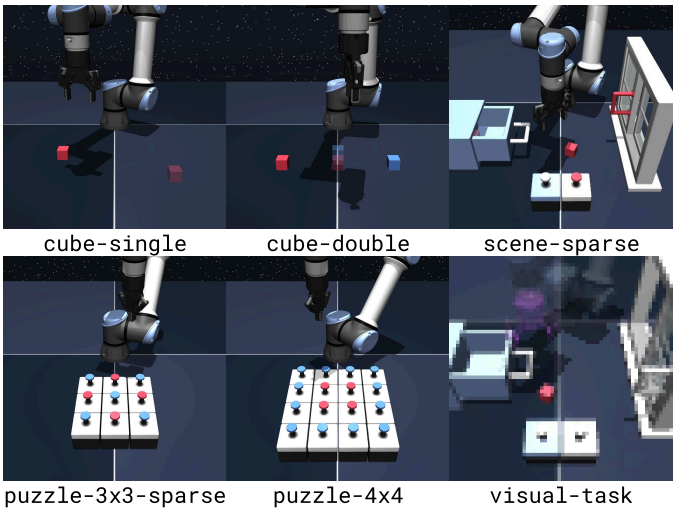

Fig. 4: **OGBench Manipulation Tasks.**

V. Simulation Experiments

A. Experimental Setup

In the simulation experiments, we evaluate on **(1)** five *state-based* manipulation tasks from OGBench [17]: `cube-single`, `cube-double`, `scene-sparse`, `puzzle-3x3-sparse`, and `puzzle-4x4`. Each task includes five variants corresponding to different goal configurations. We additionally consider **(2)** *pixel-based* settings using the first task from each corresponding visual benchmark split [18] (denoted `visual-task`). These environments provide a rigorous testbed for isolating the effect of policy extraction under a shared value-learning algorithm.

To focus on policy extraction mechanisms, we compare methods that all use Q-Chunking (QC) [14] for value learning, but differ in how they represent the base policy and how they perform policy improvement:

- **LPS** (Ours): latent policy steering with a MeanFlow base policy, trained via direct backpropagation of action-space Q-gradients.
- **QC-FQL** and **QC-MFQL**: action-space policy extraction via behavior distillation. QC-MFQL matches QC-FQL, but replaces the base policy with MeanFlow.
- **DSRL**: latent steering with a latent-space critic. For a fair comparison, we re-implement DSRL-NA using flow matching and jointly train the base policy, critic, and noise-aliasing (NA) components under the same QC value learning.
- **CFGRL**: inference-time steering via classifier-free guidance (CFG) applied to an optimality-conditioned generative policy [6].

Across all tasks, we use chunk length $h = 5$ and train each method for $1M$ gradient steps with batch size 256. We use a 4-layer MLP with hidden size 512 for the base policies and 256 for the critics. For the latent actors in **LPS** and **DSRL**, we use a 2-layer MLP with hidden size 256. We carefully tune α for **QC-FQL** and **QC-MFQL**. For **CFGRL**, we use the best-reported CFG strength w from [6]. Following common practice, we normalize the critic loss to have unit norm.

B. Experimental Results

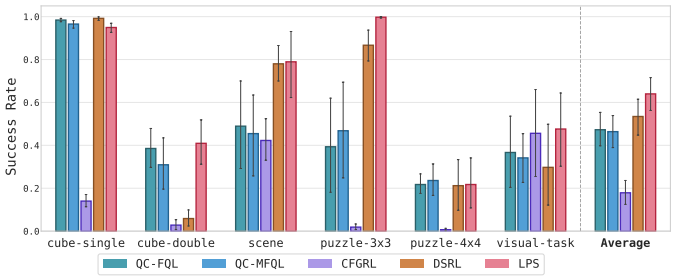

Fig. 5: **Performance on OGBench.** We evaluate the success rates across tasks. Bars report the mean success rate over 3 seeds, and error bars indicate the 95% confidence interval estimated using bootstrap resampling with 1K iterations.

Figure 5 reports success rates on OGBench. Although all methods share the same QC value-learning mechanism, performance varies significantly with the policy extraction strategy. **LPS** consistently outperforms the one-step distillation baselines (**QC-FQL** and **QC-MFQL**).

DSRL exhibits higher variance across tasks and performs poorly on the challenging `cube-double` domain, highlighting the limitations of relying on a distilled latent-space critic in the offline setting. Despite being an out-of-the-box solution, **CFGRL** underperforms explicit policy extraction methods, suggesting that inference-time guidance alone provides weaker and less precise improvement signals than direct critic-based optimization.

C. Sensitivity to the Regularization Weight α

To evaluate robustness to behavior-regularization tuning, we sweep α on three representative tasks used during hyperparameter tuning, as shown in Figure 6. For both **LPS** and **DSRL**, which do not inherently include α, we weighted the base policy loss by α to align with the experimental setting. As expected, **QC-MFQL**, representing action-space regularization methods, exhibits a sharp performance peak at a specific α, with success

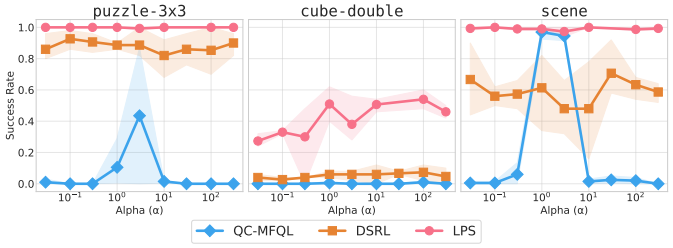

Fig. 6: **Sensitivity to α.** We report the success rates of **QC-MFQL**, **DSRL**, and **LPS** **(Ours)** across varying α (swept from 0.01 to 300) on representative default tasks used for hyperparameter tuning. Solid lines denote the mean success rate, and shaded regions show 95% confidence interval.

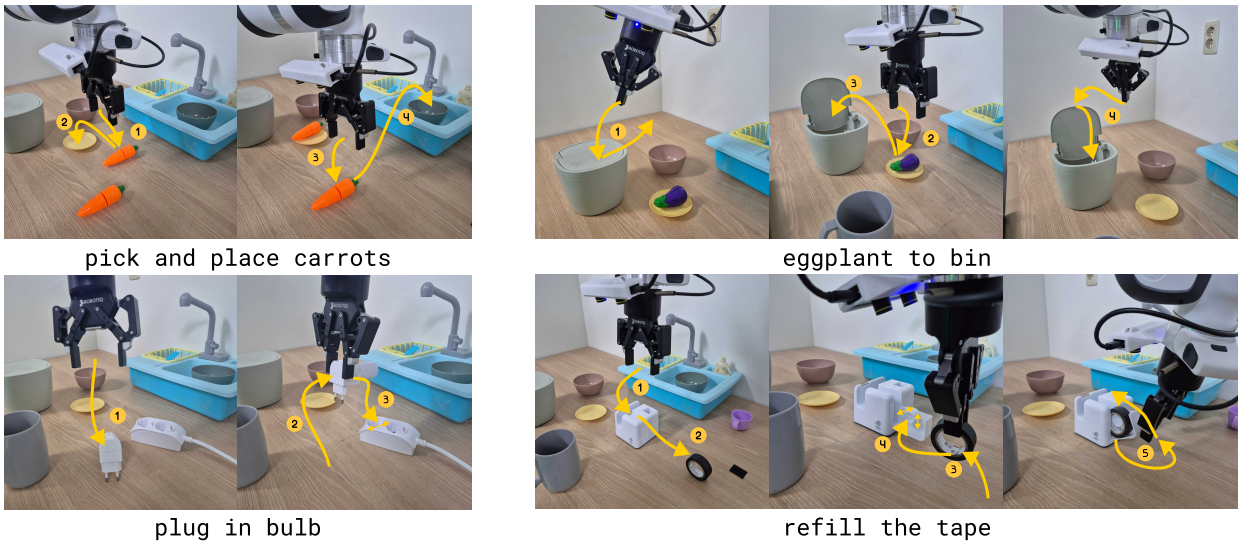

Fig. 7: **Overview of real-world tasks.** Our real-world benchmark spans four manipulation tasks, ranging from simple pick-and-place to precision insertion and trajectory stitching. We collect 50 human teleoperated demonstrations per task.

rates dropping rapidly when α deviates from the task-specific optimum. In stark contrast, **LPS** remains stable across a wide range of α, consistent with our design goal of decoupling policy improvement from explicit behavior-regularization weights.

Meanwhile, **DSRL** also exhibits strong robustness to α, supporting the intuition that latent-space optimization can be robust to behavior-regularization tuning. However, it consistently underperforms **LPS**, suggesting that robustness to α alone is not sufficient. Accurate policy extraction in the offline setting benefits from directly optimizing with action-space critic gradients rather than relying on a distilled latent-space critic.

VI. Real-World Experiments

Our goal in this section is to verify whether LPS functions as a practical, out-of-the-box solution readily applicable to real-world robotic tasks.

A. Experimental Setup

Our simulation experiments suggest that LPS provides a robust offline RL algorithm under a shared value-learning backbone. We now evaluate whether these gains transfer to real robots. We conduct experiments on the DROID platform [11] across four tasks and collect 50 demonstrations per task (Figure 7). These tasks require high precision, closed-loop correction, and trajectory stitching–regimes where standard BC often struggles.

We compare against **DSRL** as a representative latent-steering baseline in offline setting. For a fair comparison, we train its base policy, critic, and noise-aliasing components jointly under the same training pipeline. We also include **Flow-BC** and **MF-BC**, which correspond to the underlying generative base policies trained with BC only (i.e., without RL).

We use action chunking with $h = 5$ for all tasks. For the base policy, we adopt a Diffusion Transformer (DiT) [19] resulting $114M$ parameters and train it for 10K gradient steps with batch size 256. We use a semi-sparse reward, where an agent receives -1 per time step and 0 upon success, with a discount factor $\gamma = 0.99$. During evaluation, an episode is terminated if the agent fails the task or exceeds the maximum horizon of 500 steps.

B. Performance Comparison

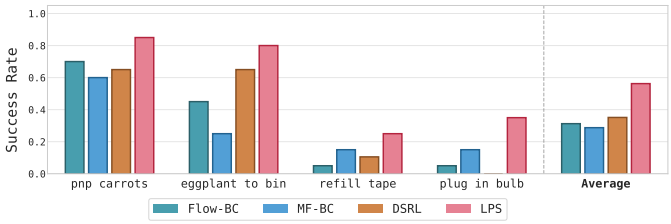

Fig. 8: **Success rates on real-world tasks.** We report the success rates measured over 20 evaluation trials for each task. Our method (**LPS**) consistently outperforms BC-based baselines and prior latent-steering methods (DSRL), demonstrating superior robustness or our method in the real world.

Figure 8 summarizes the real-world performance. Across all tasks, **LPS** achieves the highest success rates and the best average performance, outperforming both behavioral cloning and prior latent-steering methods. These results indicate that directly steering a behavior policy using action-space critic gradients yields practical improvements on real robots.

Limitations of DSRL. While **DSRL** improves over BC on relatively simpler tasks, e.g., `pnp carrots` and `eggplant to bin`, but it struggles on more challenging, precision-critical tasks. In particular, on the `plug in bulb` task, **DSRL** achieves 0% success and performs worse than the base policies, suggesting that relying on a distilled latent-space critic can be fragile in purely offline deployment for challenging tasks.

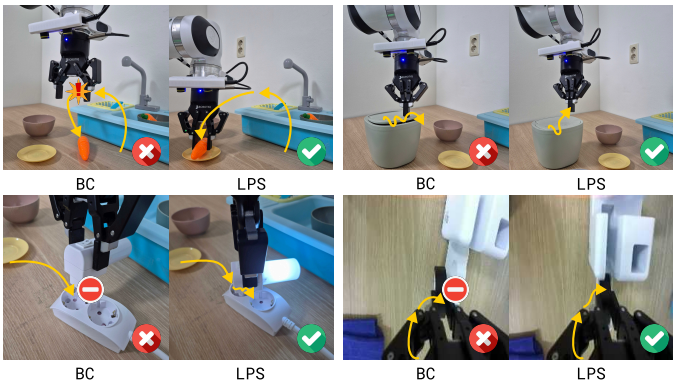

Fig. 9: **Qualitative failure modes and corrections.** Teleoperation artifacts can induce failures such as premature release *(top-left)*, repetitive loops *(top-right)*, and freezing during alignment *(bottom)*. LPS reduces these failures by selecting higher-value actions at critical decision points.

C. When BC Fails and How LPS Improves?

Our dataset consists solely of successful trajectories, which are often suboptimal due to human teleoperation artifacts like hesitation, micro-corrections, jittery motions, and pauses. These artifacts inherently limit the asymptotic performance of pure BC methods (**Flow-BC** and **MF-BC**). We qualitatively analyzed the resulting policy behaviors to identify specific failure modes caused by these limitations, as illustrated in Figure 9. For instance, BC baselines frequently suffer from premature release due to hesitation (`pnp carrots`), repetitive motion loops (`eggplant to bin`), and freezing during precision alignment (`plug in bulb`, `refill tape`). In contrast, **LPS** effectively mitigates these issues by steering the latent policy toward high-value regions, enabling the agent to execute decisive actions where BC baselines would otherwise stall or oscillate. While LPS does not eliminate all failures, it substantially reduces their frequency, yielding more reliable real-world policy deployment than BC.

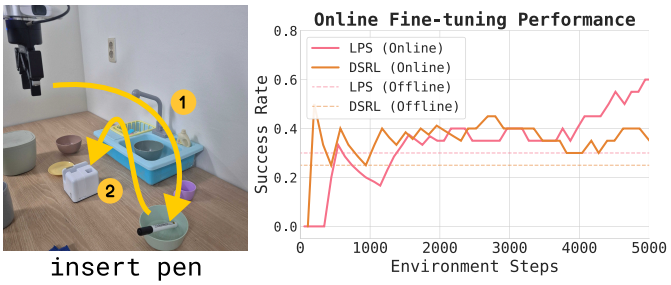

Fig. 10: **Online fine-tuning results.** We evaluate online fine-tuning performance on the `insert pen` task *(left)*. The learning curves *(right)* show that **LPS** efficiently improves upon its offline initialization via online interaction, surpassing **DSRL** within 5K environment steps.

D. LPS can improve via online interaction

To demonstrate the extensibility of our framework, we investigated whether **LPS** can be effectively applied to online fine-tuning. We conducted experiments on the `insert pen` task, initializing with offline training for 10K steps on a limited dataset of 20 teleoperated demonstrations. We then fine-tuned over 5K environment steps. To ensure efficient learning we adopted a balanced sampling strategy: each mini-batch consisted of 64 samples from the online replay buffer and 64 samples from the offline dataset. We performed 200 gradient updates between data collection rollouts, totaling 49 and 42 rollouts for **LPS** and **DSRL**, respectively. As illustrated in Figure 10, **LPS** demonstrates rapid adaptation, surpassing both its offline baseline and **DSRL** within limited steps. This highlights the sample efficiency of our approach in leveraging online feedback.

E. Computational Efficiency of LPS

We also investigate computational efficiency, which is critical for real-world deployment (Figure 11). Both **DSRL** and **LPS** use more VRAM than BC baselines due to the additional critic and latent actor, yet their memory footprints are comparable to each other. However, in terms of training speed, **LPS** is notably faster. **DSRL** incurs high computational overhead from iterative sampling and noise-aliasing updates, whereas **LPS** benefits from one-step generation and direct backpropagation through the differentiable base policy, avoiding latent-critic distillation.

At inference time, multi-step flow matching policies can introduce substantial latency. In contrast, **LPS** utilizes MeanFlow's one-step generation, achieving inference speeds comparable to **MF-BC** while delivering significantly higher success rates. Overall, **LPS** provides an attractive practical trade-off: improved performance with efficient training and fast inference.

VII. Ablation Study

We conduct comprehensive ablation studies to validate the key design choices in **LPS**. We focus on three components: (1) the latent-space geometry, (2) the choice of one-step generative backbone, and (3) the noise-to-action reformulation used to train MeanFlow. We follow the same evaluation protocol as in the main simulation experiments and report mean performance averaged across all state-based OGBench manipulation tasks.

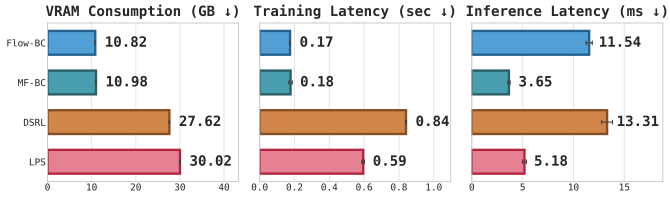

Fig. 11: **Computational efficiency.** We report VRAM usage and speed in training and inference. Benchmarks are measured on an NVIDIA L40S GPU with an Intel Xeon Gold 5320 CPU.

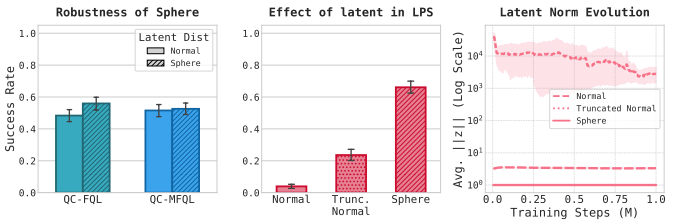

Fig. 12: **Impact of latent-space geometry.** We compare success rates of baseline methods *(left)* and LPS *(middle)* across different latent geometries. *(Right)* Latent norm during training: the spherical constraint prevents norm growth.

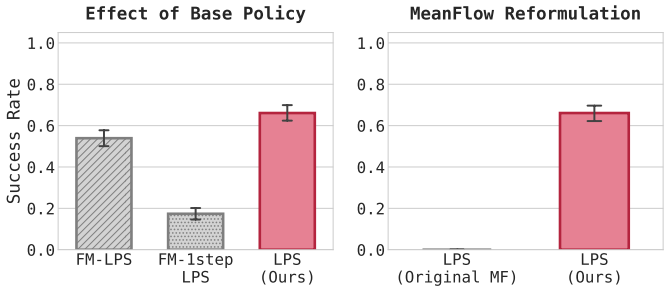

Fig. 13: **Generative backbone and reformulation ablations.** *(Left)* Comparing MeanFlow (LPS) to flow-matching variants: FM-LPS (10-step sampling) and FM-1step-LPS (one-step ODE). *(Right)* LPS trained with original MeanFlow objective vs. the noise-to-action reformulation.

A. *Effect of Latent-Space Geometry*

We first study whether the spherical latent space retains expressivity. As shown in Figure 12 (left), replacing the standard normal prior with a spherical prior (`sphere`) does not degrade baseline performance, suggesting that the sphere latent retains sufficient representational capacity.

In contrast, for LPS, the choice of the latent geometry is critical. We compare our method against a standard normal and a truncated normal distribution (bounded to $[-2, 2]$ and apply $2 * \tanh$ to latent actor). Both alternatives significantly reduce performance (Figure 12, middle). The training dynamics in Figure 12 (Right) help explain why: without a spherical constraint, latent optimization tends to increase $|z|$ (norm growth), pushing the actor into atypical regions of the base policy. With a truncated prior, the actor often saturates near the boundary where gradients diminish. Contrasting both the base policy and actor to the same hyperspherical typical set avoids these failure modes and yields stable optimization.

B. *Effect of One-Step Generation Backbone*

Next, we evaluate the role of MeanFlow's one-step generation by comparing LPS against two flow-matching (FM) variants: **FM-LPS**, which uses 10 step denoising, and **FM-1step-LPS**, which forces a single Euler step at inference and during backpropagation. As shown in Figure 13 (Left), **FM-1step-LPS** performs worst, consistent with the fact that standard FM vector fields incur large approximation errors under single-step integration. **FM-LPS** improves substantially but still underperforms our method, indicating that backpropagation through multi-step generation (i.e., BPTT through sampling trajectory) introduces additional instability and overhead that degrades learning. Overall, these results support MeanFlow as a practical backbone for LPS: it enables high-fidelity one-step generation and stable end-to-end gradients without requiring multi-step integration.

C. *Effect of MeanFlow Noise-to-Action Reformulation*

Finally, we ablate the noise-to-action reformulation used to train MeanFlow in our setting. Figure 13 (Right) shows that training LPS with the original MeanFlow parameterization leads to unstable learning and poor downstream control, whereas the reformulated objective consistently yields strong performance. This highlights the importance of predicting denoised actions (equivalently, start-end displacement) rather than raw velocity fields.

VIII. Conclusion

In this work, we highlighted a practical bottleneck in offline RL for robotics: explicit behavior regularization can create a sensitive trade-off that often requires costly hyperparameter tuning. Latent steering offers a structural alternative, but existing offline adaptation (e.g., DSRL) typically rely on distilling an action-space critic into a latent-space critic, which can introduce approximation errors and limit purely-offline performance. To address this, we proposed **Latent Policy Steering (LPS)**, which enables direct latent policy improvement by backpropagating action-space critic gradients through a differentiable one-step base policy, together with a synchronized spherical latent geometry. Across simulation benchmarks and real-world robotic tasks, LPS consistently improves over behavioral cloning and strong latent steering baselines, providing a practical out-of-the-box approach with minimal tuning.

Limitations. LPS is ultimately bounded by the quality and coverage of the base policy. If the base policy fails to capture important modes in the data, latent steering cannot recover them. In addition, the spherical constraint is intentionally conservative. While it stabilizes optimization and keeps latent queries within the safe, typical set of the behaviors, it may restrict extrapolation to behaviors far beyond the demonstration distribution.

Future work. Promising directions include scaling LPS to large *Vision-Language-Action (VLA)* models for general-purpose robot manipulation, and exploiting the temporal structure within action chunks by using structured latent representations rather than treating chunks as flat vectors.

Acknowledgments

References

[1] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. OPAL: offline primitive discovery for accelerating offline reinforcement learning. In *International Conference on Learning Representations*. OpenReview.net, 2021. URL https://openreview.net/forum?id=V69LGwJ0lIN.

[2] Xi Chen, Ali Ghadirzadeh, Tianhe Yu, Jianhao Wang, Alex Yuan Gao, Wenzhe Li, Liang Bin, Chelsea Finn, and Chongjie Zhang. LAPO: Latent-variable advantage-weighted policy optimization for offline reinforcement learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 36902–36913. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/efb2072a358cefb75886a315a6fcf880-Paper-Conference.pdf.

[3] Yuhui Chen, Shuai Tian, Shugao Liu, Yingting Zhou, Haoran Li, and Dongbin Zhao. ConRFT: A reinforced fine-tuning method for VLA models via consistency policy. In *Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.019.

[4] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu, editors, *Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026. URL https://doi.org/10.15607/RSS.2023.XIX.026.

[5] Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. In *International Conference on Learning Representations*, 2025.

[6] Kevin Frans, Seohong Park, Pieter Abbeel, and Sergey Levine. Diffusion guidance is a controllable policy improvement operator, 2025. URL https://arxiv.org/abs/2505.23458.

[7] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, pages 20132–20145, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/a8166da05c5a094f7dc03724b41886e5-Abstract.html.

[8] Zhengyang Geng, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In *Advances in Neural Information Processing Systems*, 2025.

[9] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[10] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachy Groom, Hunter Hancock, Karol Hausman, Gashon Hussein, Brian Ichter, Szymon Jakubczak, Rowan Jen, Tim Jones, Ben Katz, Liyiming Ke, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Yao Lu, Vishnu Mano, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Charvi Sharma, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Alex Swerdlow, James Tanner, Marcel Torne, Quan Vuong, Anna Walling, Haohuan Wang, Blake Williams, Sukwon Yoo, Lili Yu, Ury Zhilinsky, and Zhiyuan Zhou. $\pi^*_{0.6}$: a VLA that learns from experience, 2025. URL https://arxiv.org/abs/2511.14759.

[11] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R. Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Donovon Jackson, Charlotte Le, Yunshuang Li, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J. Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. DROID: A large-scale in-the-wild robot manipulation dataset. In Dana Kulic, Gentiane Venture, Kostas E. Bekris, and Enrique Coronado, editors, *Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.120. URL https://doi.org/10.15607/RSS.2024.XX.120.

[12] Changyeon Kim, Haeone Lee, Younggyo Seo, Kimin Lee, and Yuke Zhu. Deas: Detached value learning with action sequence for scalable offline rl. In *International Conference on Learning Representations*, 2026.

[13] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations*, 2014. URL http://arxiv.org/abs/1312.6114.

[14] Qiyang Li, Zhiyuan Zhou, and Sergey Levine. Reinforcement learning with action chunking. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=XUks1Y96NR.

[15] Tianhong Li and Kaiming He. Back to basics: Let denoising generative models denoise, 2026. URL https://arxiv.org/abs/2511.13720.

[16] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*. OpenReview.net, 2023. URL https://openreview.net/forum?id=XVjTT1nw5z.

[17] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned RL. In *International Conference on Learning Representations*. OpenReview.net, 2025. URL https://openreview.net/forum?id=M992mjgKzI.

[18] Seohong Park, Qiyang Li, and Sergey Levine. Flow q-learning. In *International Conference on Machine Learning*. OpenReview.net, 2025. URL https://openreview.net/forum?id=KVf2SFL1pi.

[19] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pages 4172–4182. IEEE, 2023. doi: 10.1109/ICCV51070.2023.00387. URL https://doi.org/10.1109/ICCV51070.2023.00387.

[20] Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. Accelerating reinforcement learning with learned skill priors. In Jens Kober, Fabio Ramos, and Claire J. Tomlin, editors, *Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 188–204. PMLR, 2020. URL https://proceedings.mlr.press/v155/pertsch21a.html.

[21] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*. OpenReview.net, 2022. URL https://openreview.net/forum?id=TIdIXIpzhoI.

[22] Juyi Sheng, Ziyi Wang, Peiming Li, and Mengyuan Liu. MP1: Mean flow tames policy learning in 1-step for robotic manipulation. In *Association for the Advancement of Artificial Intelligence*, 2026.

[23] Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge university press, 2018.

[24] Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. *Conference on Robot Learning*, 2025.

[25] Zeyuan Wang, Da Li, Yulin Chen, Ye Shi, Liang Bai, Tianyuan Yu, and Yanwei Fu. One-step generative policies with Q-learning: A reformulation of meanflow, 2025. URL https://arxiv.org/abs/2511.13035.

[26] Zhendong Wang, Jonathan J. Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. In *International Conference on Learning Representations*. OpenReview.net, 2023. URL https://openreview.net/forum?id=AHvFDPi-FA.

[27] Ling Yang, Zixiang Zhang, Zhilong Zhang, Xingchao Liu, Minkai Xu, Wentao Zhang, Chenlin Meng, Stefano Ermon, and Bin Cui. Consistency flow matching: Defining straight flows with velocity consistency. *CoRR*, abs/2407.02398, 2024. doi: 10.48550/ARXIV.2407.02398. URL https://doi.org/10.48550/arXiv.2407.02398.

[28] Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. Flowpolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation. In Toby Walsh, Julie Shah, and Zico Kolter, editors, *Association for the Advancement of Artificial Intelligence*, pages 14754–14762. AAAI Press, 2025. doi: 10.1609/AAAI.V39I14.33617. URL https://doi.org/10.1609/aaai.v39i14.33617.

[29] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu, editors, *Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.016. URL https://doi.org/10.15607/RSS.2023.XIX.016.

[30] Wenxuan Zhou, Sujay Bajracharya, and David Held. PLAS: latent action space for offline reinforcement learning. In Jens Kober, Fabio Ramos, and Claire J. Tomlin, editors, *Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 1719–1735. PMLR, 2020. URL https://proceedings.mlr.press/v155/zhou21b.html.

APPENDIX A
OFFLINE-TO-ONLINE REINFORCEMENT LEARNING

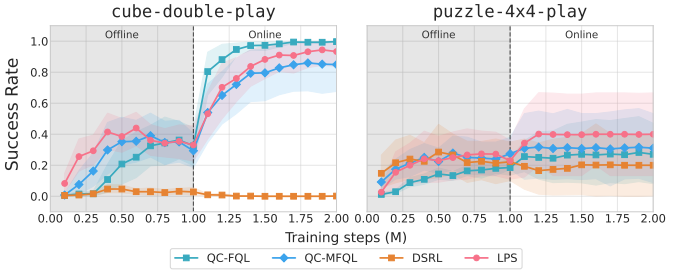

Fig. 14: **Offline-to-online learning curves of LPS and baselines on OGBench tasks.**

To evaluate the adaptability and sample efficiency of LPS in a semi-offline setting, we conducted extra offline-to-online fine-tuning experiments on OGBench [17]. We first pre-trained the agents using the static offline dataset for 1M gradient steps. Subsequently, the agents were deployed into the online environment to interact and collect new experiences for an additional 1M steps. This setup mimics a realistic deployment scenario where an agent is initialized with a prior policy derived from offline data and then refined via online interaction, consistent with the evaluation protocol of Q-Chunking [14]. Figure 14 illustrates the learning curves on `cube-double-play` and `puzzle-4x4-play` tasks. The vertical dashed line at 1M steps marks the transition from offline pre-training to online fine-tuning.

Upon switching to online interaction, LPS retains the performance level acquired during offline training without significant degradation. In the subsequent online phase (1M to 2M steps), the agent effectively leverages new interactions to further refine its policy. For instance, in the `cube-double-play` task, LPS demonstrates a steady improvement in success rate, reaching near-perfect performance, whereas baseline methods such as DSRL [24] struggle to adapt or remain at near-zero performance.

Interestingly, we observe that QC-FQL [14] achieves the highest asymptotic performance in `cube-double-play` but fails to exhibit similar dominance in `puzzle-4x4-play`. Given that the primary distinction between QC-FQL and QC-MFQL lies in their underlying base modeling methodologies, investigating the factors contributing to this task-dependent discrepancy remains an intriguing direction for future work. Despite these variations among baselines, LPS maintains stable competitiveness across tasks. Although the primary focus of this work is to establish a robust framework for immediate offline deployment, these findings indicate that LPS also serves as a reliable initialization for subsequent fine-tuning.

APPENDIX B
DETAILED EXPERIMENTAL RESULTS

OGBench. We report the learning curves of reported results at Figure 5 averaged on 3 differet seeds described in Figure 15. The detailed numerical results are presented in

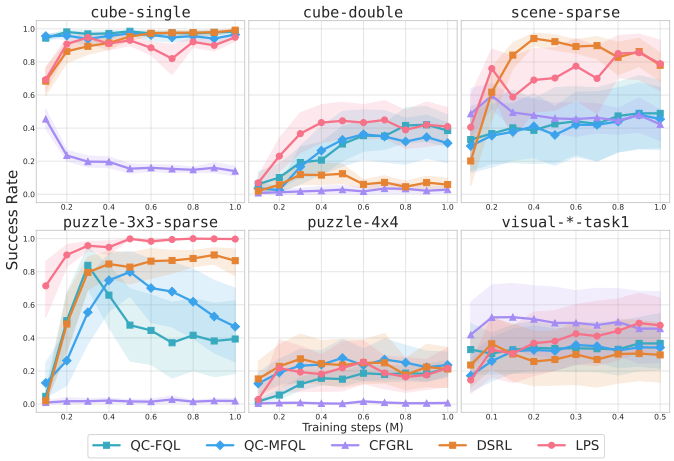

Fig. 15: **Learning curves of LPS and baselines on OGBench tasks.**

Table II. We applied two different latent space distribution (`Normal`, `Sphere`) for QC-FQL and QC-MFQL for ablation. Specifically, QC-FQL and QC-MFQL suffer from unstable training trajectories and fail to match the performance of latent actor-based methods like DSRL and LPS. Although DSRL shows strong results, it remains inferior to LPS, reaffirming the efficacy of the policy extraction mechanism employed in LPS. We also included BC baselines for comparison, and as evidenced by the results, they failed completely.

Real world. We report the success rates over 20 rollouts per task, appeared in Table I. While LPS demonstrates superior performance over BC as an offline RL algorithm, it is not entirely free from limitations. It occasionally exhibits failure modes similar to those of BC, indicating room for improvement in areas such as policy and value learning.

APPENDIX C
EXPERIMENTAL DETAILS

A. *Detailed explanation on each domain*

OGBench. We adapt OGBench for standard reward-maximizing offline RL by employing its single-task, reward-based variants. Specifically, we focus on the manipulation domain, which comprises five environments: `cube-single`, `cube-double`, `scene`, `puzzle-3x3`, and `puzzle-4x4`. Each environment consists of five distinct single-task configurations, yielding a total of 25 state-based tasks. Additionally, we evaluate five visual manipulation tasks ($64 \times 64 \times 3$ pixel observations) corresponding to the first configuration of each environment. The default reward is defined as $-n$ per step and

TABLE I: **Success rates on real-world tasks.**

Task	BC-FM	BC-MF	DSRL	**LPS**
`eggplant to bin`	9/20 (45%)	5/20 (25%)	13/20 (65%)	**16/20 (80%)**
`pnp carrots`	14/20 (70%)	12/20 (60%)	13/20 (65%)	**17/20 (85%)**
`plug in bulb`	1/20 (5%)	3/20 (15%)	0/20 (0%)	**7/20 (35%)**
`refill tape`	1/20 (5%)	3/20 (15%)	2/20 (10%)	**5/20 (25%)**
Average	25/80 (31.2%)	23/80 (28.7%)	28/80 (35.0%)	**45/80 (56.2%)**

TABLE II: **Success rates on OGBench tasks.** Each cell represents mean ± SEM for both normal and sphere latent distributions across 3 different seeds.

Task (`normal` / `sphere`)	BC-FM	BC-MF	QC-FQL	QC-MFQL	CFGRL	DSRL	**LPS**
`cube-single-play-singletask`	9 ± 1 / 9 ± 1	10 ± 2 / 9 ± 1	98 ± 0 / 97 ± 1	97 ± 1 / 93 ± 2	14 ± 2 / -	0 ± 0 / 99 ± 0	0 ± 0 / 95 ± 1
`cube-double-play-singletask`	1 ± 1 / 1 ± 1	2 ± 1 / 2 ± 1	39 ± 5 / 36 ± 6	31 ± 6 / 40 ± 8	3 ± 1 / -	0 ± 0 / 6 ± 2	0 ± 0 / 41 ± 6
`scene-play-sparse-singletask`	5 ± 2 / 4 ± 1	3 ± 1 / 3 ± 1	49 ± 11 / 87 ± 6	45 ± 10 / 42 ± 11	42 ± 5 / -	0 ± 0 / 78 ± 5	0 ± 0 / 79 ± 8
`puzzle-3x3-play-sparse-singletask`	2 ± 1 / 1 ± 1	1 ± 1 / 1 ± 1	39 ± 12 / 39 ± 12	47 ± 12 / 42 ± 10	2 ± 1 / -	0 ± 0 / 87 ± 4	14 ± 7 / 100 ± 0
`puzzle-4x4-play-singletask`	0 ± 0 / 0 ± 0	0 ± 0 / 0 ± 0	22 ± 2 / 22 ± 5	24 ± 4 / 30 ± 6	1 ± 0 / -	0 ± 0 / 21 ± 6	3 ± 1 / 22 ± 6
`visual-*-task1`	-	-	37 ± 9 / -	34 ± 6 / -	46 ± 11 / -	- / 30 ± 10	- / 48 ± 9

0 upon success, where n represents the number of remaining sub-goals (e.g., unlit bulbs in `puzzle` or unmatched cubes in `cube`). However, for `scene` and `puzzle-3x3`, we adopt a *sparse* reward structure (-1 per step, 0 upon success). We empirically found that the sub-goal-based dense rewards in these environments were not consistently aligned with the final objective, whereas the sparse setting led to superior performance.

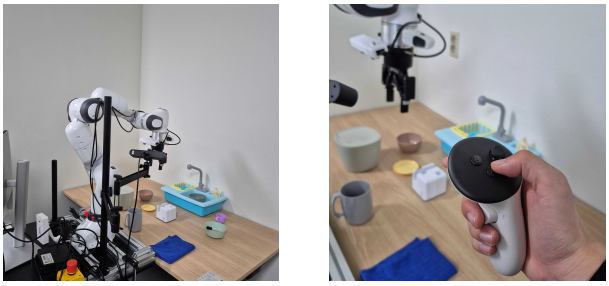

Fig. 16: **Real-world setup overview.**

Real-world Franka. In our real-world experiments, we strictly adhered to the DROID [11] hardware configuration. The setup comprises a Franka Research 3 robot and two ZED2 cameras, providing both third-person and wrist views ($224 \times 224 \times 3$), as shown in Figure 16 (Left). We employed delta end-effector control combined with a binary gripper, resulting in a 7-dimensional action space (6 dimensions for end-effector velocity and 1 for gripper actuation). For proprioceptive information, we utilize the end-effector pose and gripper state. We collected 50 human-teleoperated demonstrations using a Meta Quest 3 headset, following the DROID data collection system Figure 16 (Right). The collected data is stored in HDF5 format. We load all the files to the memory and concatenate it, allowing us to utilize a data loading pipeline consistent with OGBench.

B. Evaluation protocol

OGBench. We run 3 seeds on each OGBench task. All plots use 95% confidence interval with stratified sampling (1000 samples). The success rate is computed by running the policy in the environment for 50 episodes and record the number of times that the policy succeeds at solving the task (and divide it by 50).

Real-world Franka. For all methods, we run 20 episodes on each task and calculated the success rate. We terminate each rollout when the robot stopped moving, or reach 500 environment step.

APPENDIX D
IMPLEMENTATION DETAILS

A. Computational resources

We use NVIDIA RTX-3090 GPU to run all our OGBench experiments, L40S for training real world policies, and RTX-5090 for inference and online fine-tuning.

B. Baseline Implementation

LPS and all baseline methods are built upon the **Q-chunking** codebase. We adopted the dataloading pipeline from DEAS [12]. This version is simpler because it filters and samples valid data directly within the dataloader, eliminating the need for a validity mask.

QC-MFQL. We adapted the QC-FQL framework by modifying the base policy learning component to utilize a Mean Flow objective and employing one-step ODE sampling. We use a *normal* prior distribution for training both the base policy and the one-step distillation policy.

DSRL. We employ the DSRL-NA variant, replacing the one-step policy distillation of QC-FQL with latent-space critic distillation and latent actor optimization. While the original DSRL incorporates an entropy term for exploration, we omit this term to align with the standard actor-critic formulation used in other offline RL baselines. Regarding the latent space structure, we replace the original $\tanh$-bounded actor with a spherical distribution (*sphere*). We adopted this spherical structure as we found it to be more efficient, a finding consistent with our observations for LPS.

CFGRL [6]. We strictly follow the original implementation of CFGRL and use the classifier-free guidance strength w reported in the original paper.

For the DiT [19] architecture, we leveraged the JAX implementation from MeanFlowQL [25]. However, we modified the embedding strategy to assign a unique embedding to each action rather than a single embedding for the entire action chunk, thereby better leveraging the structural capabilities of DiT.

C. Training hyperparameter

We report the common training parameters for both LPS and the baselines, along with the baseline-specific parameters for OGBench, in Table III, Table IV, and Table V. We extensively tuned α over the set $\{0.01, 0.03, 0.1, 0.3, 1.0, 3.0, 10.0, 30.0, 100.0, 300.0\}$. Note

that the parameter α was tuned individually for each model and latent structure configuration for ablation, except `visual-task` as we did not include in ablation on latent structure. Additionally, we provide the common training parameters used for the real-world experiments in Table VI.

TABLE III: **Common parameters for OGBench experiments.**

Parameter	Value
Batch size (M)	256
Discount factor (γ)	0.99
Optimizer	Adam
Learning rate	3×10^{-4}
Learning rate scheduler	constant
Target network update rate (τ)	5×10^{-3}
Critic ensemble size (K)	2
UTD Ratio	1
Number of flow steps (T)	10 (Flow matching), 1 (MeanFlow)
Number of training steps	10^6
Actor network	MLP
Actor network width	512
Actor network depth	4
Critic network	MLP
Critic network width	256
Critic network depth	4
Latent actor network	MLP
Latent actor network width	256
Latent actor network depth	2
Image encoder (`visual-task`)	*impala small*

TABLE IV: **Behavior regularization coefficient (α).**

Environments (`normal` / `sphere`)	QC-FQL	QC-MFQL
`cube-single-*`	30.0 / 10.0	30.0 / 10.0
`cube-double-*`	3.0 / 3.0	3.0 / 3.0
`scene-sparse-*`	1.0 / 3.0	1.0 / 1.0
`puzzle-3x3-sparse-*`	3.0 / 3.0	3.0 / 3.0
`puzzle-4x4-*`	10.0 / 3.0	3.0 / 3.0
`visual-cube-single-task1`	10.0 / -	30.0 / -
`visual-cube-double-task1`	1.0 / -	3.0 / -
`visual-scene-sparse-task1`	30.0 / -	3.0 / -
`visual-puzzle-3x3-sparse-task1`	0.1 / -	1.0/ -
`visual-puzzle-4x4-task1`	1.0 / -	1.0 / -

TABLE V: **CFG strength (w).**

Environments	CFGRL
`cube-single-*`	1.25
`cube-double-*`	2.00
`scene-sparse-*`	3.00
`puzzle-3x3-sparse-*`	1.50
`puzzle-4x4-*`	1.25
`visual-cube-single-task1`	1.25
`visual-cube-double-task1`	2.00
`visual-scene-sparse-task1`	3.00
`visual-puzzle-3x3-sparse-task1`	1.50
`visual-puzzle-4x4-task1`	1.25

APPENDIX E
LPSD: LATENT POLICY STEERING VIA DISTILLATION

To further demonstrate the efficacy of latent-space optimization, we introduce a variant of our framework termed **Latent Policy Steering via Distillation (LPSD)**. While our main method, LPS, focuses on tuning-free optimization via geometric constraints, LPSD incorporates explicit regularization to enable high-fidelity policy extraction via a stochastic latent actor.

TABLE VI: **Common hyperparameters for Real-world experiments.**

Parameter	Value
Batch size (M)	256
Discount factor (γ)	0.99
Optimizer	Adam
Learning rate	3×10^{-4}
Learning rate scheduler	cosine
Target network update rate (τ)	5×10^{-3}
Critic ensemble size (K)	2
UTD Ratio	1
Number of flow steps (T)	10 (Flow matching), 1 (MeanFlow)
Number of training steps	10^4
Actor network	DiT
Actor network hidden dim	384
Actor network depth	12
Actor num heads	6
Critic network	MLP
Critic network width	512
Critic network depth	4
Latent actor network	MLP
Latent actor network width	512
Latent actor network depth	2
Image encoder	*impala*

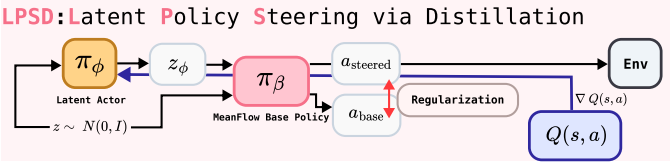

Fig. 17: **Overview of LPSD.**

Unlike the deterministic latent actor in LPS, the latent actor in LPSD, denoted as $\pi_\phi(s, z)$, takes Gaussian noise z as input. It is trained to maximize the Q-value of the generated action while minimizing the divergence between its output and the action generated by the fixed base policy. The objective function is defined as:

$$\mathcal{L}_{\text{LPSD}} = \mathcal{L}_{\text{LPS}} + \alpha \cdot \mathbb{E}_{z \sim \mathcal{Z}} \left[\left\| \pi_\beta(s, \pi_\phi(s, z)) - \pi_\beta(s, z) \right\|^2 \right], \tag{11}$$

where π_β denotes the MeanFlow base policy. This formulation is mathematically equivalent to the objective of QC-FQL, but with a key distinction: instead of training a raw one-step policy from scratch, LPSD performs distillation *within the latent space* of the generative model. The overview and the pseudo algorithm is shown Figure 17 and Eq. (11).

Due to this explicit regularization, LPSD relaxes the reliance on the spherical latent geometry used in LPS. Empirically, we found that LPSD achieves state-of-the-art results in simulation benchmarks, highlighting the superiority of latent-space extraction over direct action-space distillation. However, since this approach reintroduces the need for hyperparameter tuning (i.e., α), it serves primarily as an ablation study demonstrating the potential of latent-space optimization rather than as our primary tuning-free solution. We present this perspective to encourage more effective investigation within offline RL on simulations,

Algorithm 2: Latent Policy Steering via Distillation (LPSD)

Initialize: MeanFlow base policy $\pi_\beta(s, z)$, Latent actor $\pi_\phi(s, z)$, Critic $Q_\theta(s, a)$, Action chunk size h
while *not converged* **do**
  Sample batch $\mathcal{B} = \{(s_t, a_{t:t+h}, r_{t:t+h}, s_{t+h})\} \sim \mathcal{D}$
  // 1. Train MeanFlow Base Policy π_β
  Sample $z \sim N(0, I_d)$
  Update β to minimize $\mathcal{L}_{\text{MF}}$ // Eq. (5), Eq. (7)
  // 2. Train Latent Actor π_ϕ
  Sample $z \sim N(0, I_d)$
  // Forward pass through latent actor (with noise)
  $z_\phi \leftarrow \pi_\phi(s_t, z)$
  $a_{\text{pred}} \leftarrow \pi_\beta(s_t, z_\phi)$
  $a_{\text{base}} \leftarrow \pi_\beta(s_t, z)$
  Update ϕ to minimize $\mathcal{L}_{\text{LPSD}}$ // Eq. (11)
  // 3. Train Critic Q_θ
  Sample $z \sim N(0, I_d)$
  $z'_\phi \leftarrow \pi_\phi(s_{t+h}, z)$
  $a'_{\text{pred}} \leftarrow \pi_\beta(s_{t+h}, z'_\phi)$
  Update θ to minimize $\mathcal{L}_Q$ // Eq. (1)

in the expectation that such progress will eventually translate back to the real world.

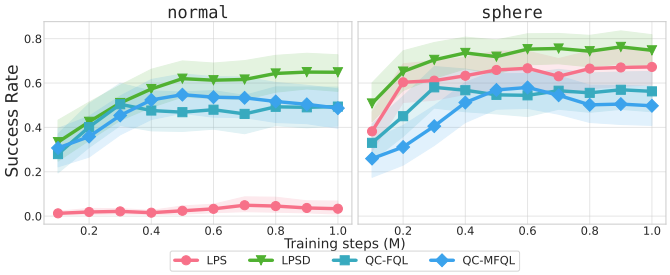

Fig. 18: **Performance comparison of LPSD against baselines.** The plot illustrates the aggregated learning curves averaged across five manipulation tasks in OGBench. Solid lines represent the mean performance, and shaded regions indicate the 95% confidence interval.

The comparison between LPSD and baselines on both normal and sphere latent structure on 5 OGBench tasks is shown in Figure 18. Although **LPSD** requires tuning like other baselines, we observed that it demonstrates significantly superior performance compared to them. In particular, unlike LPS, LPSD employs explicit regularization and demonstrates superior performance regardless of the latent distribution. Consequently, we envision this framework as a superior policy extraction mechanism capable of superseding the QC-FQL paradigm, thereby enhancing various ongoing offline RL methodologies. We use same training hyperparameters as Table III, and reported tune α at Table VII.

TABLE VII: **Behavior regularization coefficient (α).**

Environments (`normal` / `sphere`)	LPSD
`cube-single-*`	3.0 / 3.0
`cube-double-*`	0.3 / 0.3
`scene-sparse-*`	0.3 / 0.1
`puzzle-3x3-sparse-*`	1.0 / 0.3
`puzzle-4x4-*`	1.0 / 1.0