

A Additional Experiments and Results

A.1 Ablation: Hyperparameter Choice

To understand how the hyperparameters in LPB affect the final task success rate, we run a set of experiments on the *Transport* task. We focus on the two hyperparameters: the guidance scale η and the number of denoising steps that receive gradient guidance, k_{guide} . As shown in Figure 1 left, when η is very small, the improvement over the base policy is minimal because the guidance signal is too weak. As η increases, the improvement becomes more pronounced; however, once η reaches 0.3, performance drops slightly, indicating an overshooting effect caused by overly strong guidance. Figure 1 right shows that LPB is generally robust to the choice of k_{guide} : even when gradients are applied only during the last 35 denoising steps, LPB still outperforms the base policy, demonstrating that the dynamics model can provide meaningful guidance even when the noisy action samples are far from valid expert actions. Across all tasks, we find that injecting gradient guidance during the final 10 denoising steps is sufficient for a substantial performance boost.

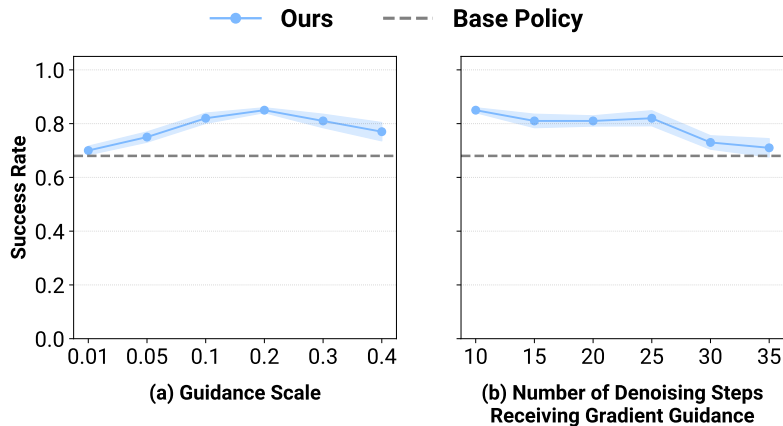


Figure 1: (a) Performance under varying guidance scale. (b) Performance under varying numbers of denoising steps that receive gradient guidance.

A.2 Ablation: Data Source for Dynamics Model Training

We evaluate different data sources for training the dynamics model on the *Tool-Hang* and *Transport* task. LPB leverages policy rollout data as an inexpensive and scalable way to improve the dynamics model’s generalization, but alternative sources are also available. We compare against two such baselines: (1) **Expert Demonstration with Noise Injection**. Gaussian noise is injected into expert demonstrations with a probability of 0.3 to generate diverse, perturbed trajectories. (2) **Epsilon-Greedy Exploration**. A base policy is first trained on limited expert data, then rolled out from randomized initial configurations with Gaussian noise added to the output action at a probability of 0.3 for exploration. As shown in Table 1, LPB, which relies on policy rollouts, achieves the highest success rate. We also note that rollout collection does not require manual tuning of noise magnitudes or injection probabilities, which simplifies the data collection procedure in practice. Moreover, it naturally captures a broad spectrum of data, spanning early exploratory trajectories to those generated by a converged policy.

Table 1: Success Rate with different data sources for dynamics model training

	Policy Rollout	Noisy Demos	Epsilon-Greedy
Transport	0.85 ± 0.009	0.73 ± 0.025	0.71 ± 0.024
Tool-Hang	0.39 ± 0.009	0.30 ± 0.031	0.30 ± 0.028

27 A.3 Ablation: Action Optimization

28 We ablate the action optimization strategies used in LPB on the *Tool-Hang* and *Transport* tasks. LPB
 29 adopts a classifier guidance-style approach for test-time action refinement. We compare this with two
 30 alternatives: (1) **LPB-MPC**, which uses Model Predictive Control with stochastic action sampling,
 31 and (2) **LPB-GD**, which directly optimizes actions via gradient descent. As shown in Table 2, LPB
 32 achieves the highest success rate. LPB-MPC also improves over the base policy, but not as much
 33 as LPB - likely because it relies on sampling, which can fail to capture rare but optimal actions if
 34 they are underrepresented in the action distribution. By contrast, LPB nudges each sampled action
 35 toward expert-like directions using a differentiable guidance signal, enabling exploration that remains
 36 on-manifold while still biasing toward high-value corrections. LPB-GD performs the worst, as its
 37 unconstrained optimization can produce actions that are out-of-distribution for both the base policy
 38 and the dynamics model.

Table 2: Success Rate for different action optimization strategies

	LPB (Ours)	LPB-MPC	LPB-GD
Tool-Hang	0.39 ± 0.009	0.32 ± 0.028	0.26 ± 0.028
Transport	0.85 ± 0.009	0.80 ± 0.043	0.73 ± 0.025

39 A.4 Ablation: Choice of Latent Representation

40 To study how the choice of latent space affects deviation detection and recovery toward expert
 41 states, we evaluate different latent representations on the *Square* and *Transport* tasks. We compare
 42 LPB, which uses the base policy’s encoder, against two alternatives: (1) **DINOv2 Encoder** and
 43 (2) **Encoder Trained with Reconstruction**. For (1), we use a pretrained DINOv2 model as the
 44 observation encoder for the dynamics model; it outputs 256 patch embeddings of dimension 384. This
 45 representation preserves fine-grained visual details, but may include task-irrelevant information, as it
 46 is not directly trained with task supervision. For (2), we pretrain a ResNet-18 as an autoencoder using
 47 a reconstruction loss and then freeze it to serve as the encoder for dynamics model training. As shown
 48 in Table 3, both the base policy’s encoder and DINOv2 improve over the base policy alone, with the
 49 base policy’s encoder outperforming DINOv2. This may be because DINOv2 encoder, trained in
 50 a task-agnostic manner, may focus on irrelevant features, while the base policy’s encoder, trained
 51 via behavior cloning loss, is more aligned with task-relevant information. The reconstruction-based
 52 encoder performs the worst, suggesting that the reconstruction objective alone does not produce a
 53 latent space suitable for effective action optimization.

Table 3: Success Rate with different latent space

	Base Policy’s Encoder	DINOv2	Encoder Trained with Reconstruction
Square	0.65 ± 0.019	0.61 ± 0.025	0.47 ± 0.034
Transport	0.85 ± 0.009	0.79 ± 0.025	0.68 ± 0.043

54 B Implementation Details

55 B.1 Implementation Details of Latent Policy Barrier

56 **Base Policy.** We adopt Diffusion Policy as the base policy in LPB due to its strong capability in
 57 modeling complex robot action distributions. It uses a ResNet-18 as the image encoder and a U-Net as
 58 the noise prediction network, with FiLM layers to condition the denoising process on both observation
 59 features and the current diffusion timestep. For all tasks, the base policy is trained using 20% of the
 60 available expert demonstrations. Task-specific and shared hyperparameters are provided in Table 4
 61 and Table 5, respectively.

62 **Dynamics Model.** Once base-policy training passes an initial warm-up of t_0 epochs, during which the
 63 policy remains highly exploratory, we begin saving checkpoints at fixed intervals. Concretely, every
 64 Δt epochs we store ckpt _{$t_0+n\Delta t$} and roll it out for N complete episodes from randomly initialized

65 configurations. All transitions, whether successful or not, are retained to train the dynamics model.
 66 This procedure continues until the final epoch t_{final} . The values of t_0 , Δt , t_{final} , N , and the resulting
 67 total number of rollout trajectories for each simulated task are summarized in Table 6.

68 The dynamics predictor f_ϕ is implemented as a Vision Transformer (ViT). It receives the concatenation
 69 of the latent observation token, the proprioception state token, and an action token, and predicts the
 70 next latent observation and proprioception state tokens. Training hyperparameters for f_ϕ are provided
 71 in Table 7.

Table 4: Simulation task-dependent hyperparameters for base diffusion policy training.

Env Name	T_a	T_p	#Demo	Training Epochs
Push-T	8	16	41	500
Square	8	16	40	600
Tool-Hang	15	32	40	300
Transport	15	32	40	300

Table 5: Shared hyperparameters for base diffusion policy training.

Name	Value
T_o	2
Image Size	140
Crop Size	128
Batch Size	64
Learning Rate	1×10^{-4}
Diffusion Step	100

Table 6: Rollout trajectories collection schedule for simulation tasks

Env Name	t_0	Δt	t_{final}	N	Total
Push-T	150	40	470	30	270
Square	70	50	470	30	270
Tool-Hang	70	50	270	30	150
Transport	70	50	270	30	150

Table 7: Hyperparameters for dynamics model training.

Name	Value
History Length	1
Depth	6
Heads	16
MLP Dim	2048
Dropout	0.1
Batch Size	64
Learning Rate	5×10^{-4}
Training Epoch	100

72 **Test-time Optimization.** At test time, we denoise actions with a DDPM scheduler. For each timestep
 73 t we first compute the latent OOD score; if the score is below the threshold τ , we run standard
 74 denoising and execute the resulting action A_t^0 . If the score exceeds τ , we apply latent steering during
 75 the final 10 denoising steps. Specifically, the current observation and the intermediate noisy action
 76 samples at denoising timestep k , A_t^k , are fed to the dynamics model, which predicts the future latent
 77 state z_{t+h} . We then measure the Euclidean distance between z_{t+h} and its nearest expert state in latent
 78 space; this distance is back-propagated through the dynamics model, and the resulting gradient with
 79 respect to the noisy action samples provides the guidance signal. The OOD threshold τ is chosen
 80 empirically by rolling out the final policy checkpoint, while the guidance scale η is selected via a grid
 81 search. Both η and τ for each task are listed in Table 8.

Table 8: Hyperparameters for test-time optimization

	η	τ
Push-T	0.05	3.2
Square	0.05	5
Tool-Hang	0.05	1.4
Transport	0.2	2.8

82 **Compute Resources.** All simulated experiments are run on a single NVIDIA L40S GPU (46 GB
 83 VRAM). Base policies require roughly 24–48 h to converge, while the dynamics models converge
 84 within 24 h. For each simulated task, we evaluate the final three checkpoints, each spaced 10
 85 training epochs apart; the results reported in Table 1 in the main paper are averages over those three
 86 checkpoints. In the real-robot setting, the base policy is pretrained and thus incurs no additional

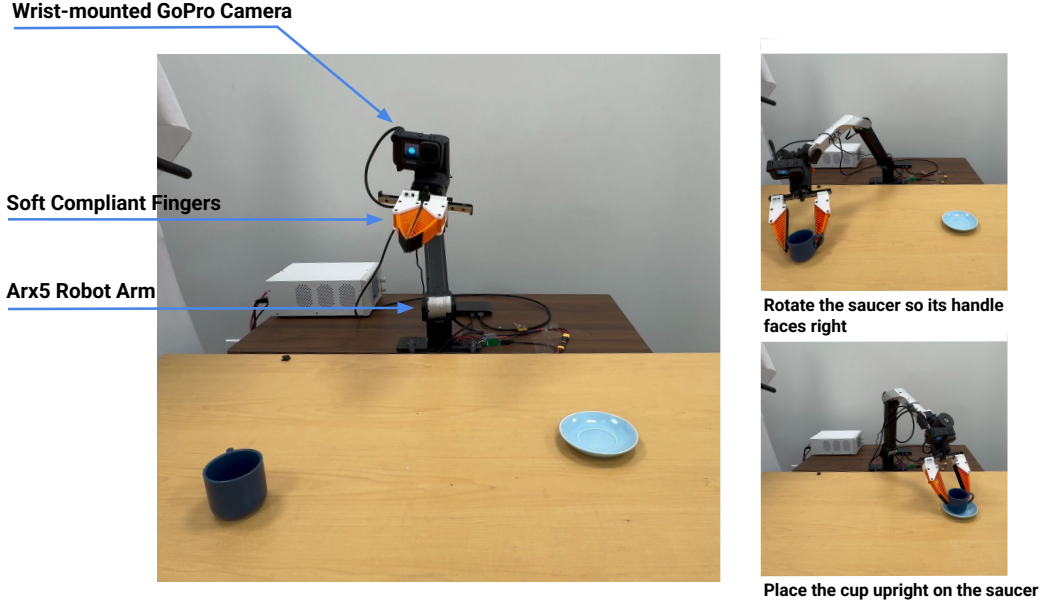


Figure 2: Real robot experiment setup.

87 training cost. The dynamics model is trained in parallel on six NVIDIA L40S GPUs and converges in
 88 approximately 36 h.

89 B.2 Implementation Details of Baselines

90 **Filtered BC.** This baseline uses the same rollout data collected for LPB. We discard trajectories that
 91 do not meet the task-specific success criteria and merge the remaining successful rollouts with the
 92 original expert demonstrations. The diffusion policy is then retrained from scratch on this augmented
 93 dataset.

94 **CQL.** We adopt the CQL implementation provided in the Robomimic codebase without modification.

95 **CCIL.** CCIL was originally designed for tasks with low-dimensional state inputs. To extend it to
 96 high-dimensional image observations, we cache latent representations produced by the base policy’s
 97 encoder. Specifically, we first train a base diffusion policy π_θ on the limited expert demonstrations,
 98 identical to the setup used for LPB. We then encode every image observation in the expert dataset
 99 with π_θ ’s encoder and store the resulting latent transition pairs. CCIL’s dynamics model is trained on
 100 these low-dimensional latents, after which CCIL’s corrective-label generation procedure is applied
 101 to augment the expert data. Finally, we fine-tune π_θ on the augmented latent dataset. We use the
 102 authors’ original implementation for all CCIL components.

103 B.3 Real Robot Experiment Setup

104 **Setup.** We use a 6-DoF ARX5 robot arm with 3D-printed soft compliant fingers and a wrist-mounted
 105 GoPro camera, mirroring the sensor configuration of the original training data (Figure 2). Low-
 106 level controller is provided by the Universal Manipulation Interface (UMI) stack ported to the
 107 ARX5 platform <https://github.com/real-stanford/umi-arx>. The cup arrangement task consists of three
 108 sequential stages: (i) rotate the cup so its handle points right, (ii) grasp and lift the cup, and (iii) place
 109 it upright on a saucer.

110 **Pre-trained Base Policy.** We start from the publicly released diffusion policy ck-
 111 eckpoint in the Universal Manipulation Interface repository https://github.com/real-stanford/universal_manipulation_interface. The base policy was trained on 1447 in-the-wild
 112 demonstrations of the cup arrangement task collected in diverse environments. The test environment
 113 is not included in the training data. At each timestep, the policy receives a single RGB observation
 114

115 plus the past two proprioceptive states and predicts a 16-step action sequence; the first 12 actions are
116 executed each control cycle.

117 **Data Collection.** To collect training data for the dynamics model we roll out the pre-trained policy
118 from deliberately out-of-distribution initial poses, gathering 80 additional trajectories. Because
119 intermediate policy checkpoints are unavailable, we augment the dataset with human play data
120 recorded via the handheld UMI device. These play trajectories are intended to broaden state–action
121 coverage rather than solve the task, for example, by sweeping the gripper through random motions to
122 capture wide-angle views.

123 **Evaluation Protocol.** We evaluate two initial-pose regimes: (i) *in-distribution initial poses* and (ii)
124 *out-of-distribution initial poses*. For each group we conduct 20 trials with both the base policy and
125 LPB, using identical initial poses for fair comparison. During deployment we denoise actions with a
126 DDIM scheduler (16 diffusion steps) and apply gradient guidance during the final five steps.

127 C Broader Impact

128 Our work contributes to improving the robustness of visuomotor policies in robotic manipulation
129 settings by introducing a test-time optimization framework that leverages learned dynamics and
130 latent-space guidance. This has the potential to reduce failure rates in deployment, especially in
131 out-of-distribution scenarios, which is critical for the safe and reliable operation of robots in human-
132 centered environments. However, as with any data-driven system, care must be taken to ensure
133 that failure modes are thoroughly evaluated before real-world deployment. Future extensions could
134 investigate formal guarantees for test-time optimization mechanisms.