APEX: Adaptive Policy Execution for Precise Manipulation

Mengfei Zhao[1,*], **Chenxi Jiang**[1,*], **Tuo An**[1], **Jindou Jia**[1,†,‡], **Jianfei Yang**[1,†]

[1]MARS Lab, Nanyang Technological University
[*]Equal contribution, [†]Corresponding authors, [‡]Project leader

Modern imitation learning methods, including visuomotor and Vision-Language-Action (VLA) policies, typically output high-level action references that are executed by low-level controllers. However, the absence of higher-order reference signals, together with the policy's lack of awareness of the underlying low-level control dynamics during training, inevitably induces an execution gap. As a result, realized actions deviate systematically from policy-commanded ones, with a critical impact on precision-sensitive manipulation. Prior work either modifies the policy architecture or the low-level controller, both requiring intrusive changes to the pretrained policy or packaged controller. This raises a natural question: when the policy and controller are both treated as inaccessible black boxes, can we bridge the execution gap? We propose **A**daptive **P**olicy **EX**ecution (**APEX**), a plug-and-play framework inserted between the policy and the controller that reconstructs a dynamically feasible reference from policy outputs and adapts at test-time according to low-level state feedback, with a provable convergence guarantee. Extensive empirical studies show that APEX reduces controller-induced tracking error by 41.2% on demonstration replay and improves manipulation success by 4.8–25.8 percentage points across four visuomotor and VLA policy classes.

Project site: https://murphyzhao04.github.io/apex-project-page/
Correspondence: jindou.jia@ntu.edu.sg, jianfei.yang@ntu.edu.sg

1 Introduction

Imitation learning, which trains robot policies from demonstration trajectories (Pomerleau, 1988; Argall et al., 2009), has become a cornerstone of modern robot learning, powering recent advances in visuomotor policies (Zhao et al., 2023; Chi et al., 2025; Jia et al., 2026b,a) and Vision-Language-Action (VLA) models (Kim et al., 2025; Octo Model Team et al., 2024; Black et al., 2024, 2025a). Such schemes typically produce joint targets (Zhao et al., 2023) or end-effector (Chi et al., 2025; Open X-Embodiment Collaboration et al., 2024; Kim et al., 2025) trajectories (Open X-Embodiment Collaboration et al., 2024), which are subsequently executed by a low-level controller, such as a Proportional-Derivative (PD) scheme. However, inadequate high-level outputs (e.g., discontinuities and missing higher-order references) and imperfect low-level controllers (e.g., unmodeled dynamics and parameter uncertainties) make precise execution elusive in real-world deployment. We refer to this discrepancy between policy-level command and robot-native execution as the *test-time execution gap*. The issue becomes particularly critical in precision-sensitive manipulation tasks, where even small tracking errors can disrupt contact timing or cause misalignment during grasping and insertion, ultimately leading to task failure.

Existing approaches attempt to mitigate this problem from several directions. One line of work augments policy outputs with richer trajectory-level representations, for example, through spline-based parameterizations or explicit velocity augmentations that provide analytical velocity profiles for low-level tracking (Yang et al., 2026; Hechtl et al., 2026; Bai et al., 2026). However, these methods are inapplicable when the underlying policy architecture and training pipeline cannot be modified. Another line learns residual corrections on top of a base policy or controller (Silver et al., 2018; Johannink et al., 2019). Although effective, these methods require additional data collection and a separate training phase. More recently, test-time refinement methods have been proposed to smooth or refine policy outputs without retraining (Zhao et al., 2023; Black et al., 2025b; Liu et al., 2025; Son and Park, 2025; Sendai et al., 2025). These approaches mainly address policy-side

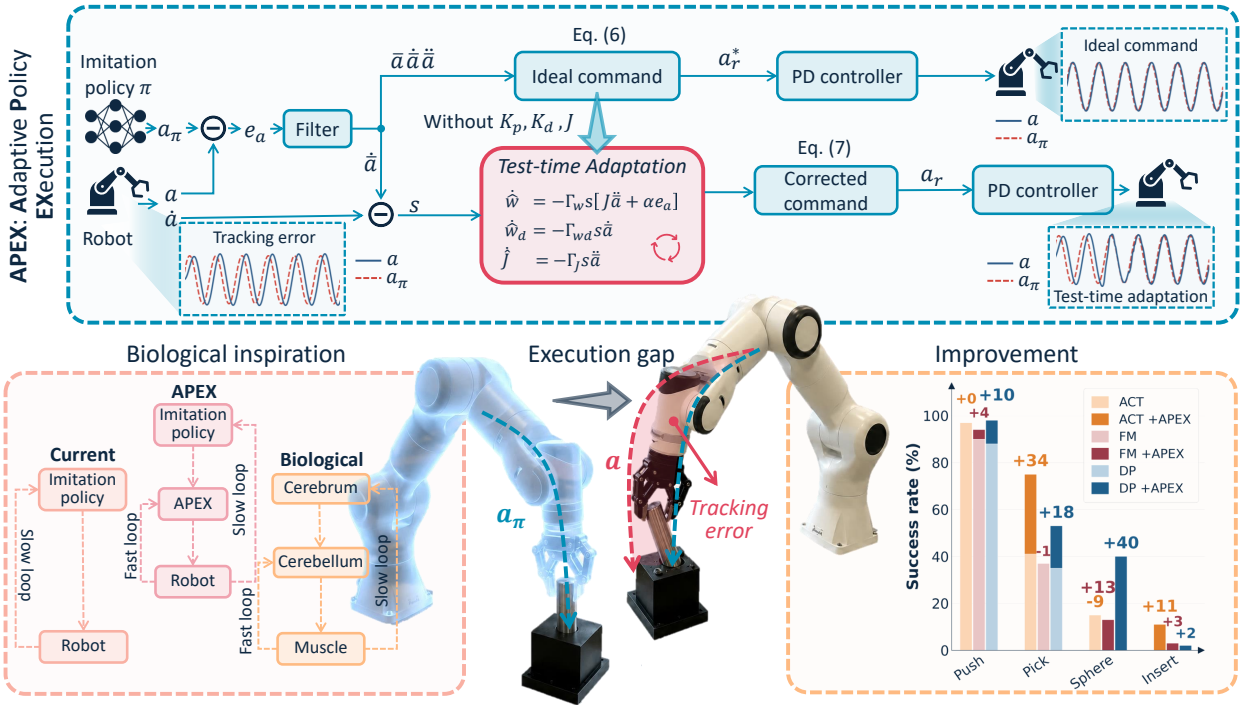

Figure 1 Overview of APEX. Bottom center: At deployment, the command is issued by a learned policy, and the motion actually realized by the PD controller can diverge from the commanded reference. **Bottom left:** Motivated by biological motor control, where fast sensorimotor feedback loops correct small deviations on the fly rather than regenerating the time-consuming high-level plan, APEX inserts an *online correction layer* between the policy and controller, leaving both untouched. **Top:** APEX first reconstructs a dynamically feasible reference from the policy output through an adaptive passive filter, supplying the higher-order signals the controller requires. Second, it augments this reference with test-time adaptive terms whose coefficients are updated online from the real-time tracking error, compensating for unknown system dynamics and controller parameters that are typically inaccessible on commercial platforms. **Bottom right:** Across four policy classes, APEX reduces tracking error and improves the success rate.

artifacts, such as latency and temporal discontinuities, but have not considered the tracking errors inherent to the low-level controller. This raises a natural question: can the execution gap be bridged without modifying either the policy or the controller?

A useful perspective comes from biological motor control. When humans perform fine manipulation, e.g., threading a needle or placing a cup on a saucer, the high-level plan is rarely regenerated during movement, as doing so would be time-consuming. Instead, the nervous system continuously adjusts the ongoing movement through fast sensorimotor feedback loops that correct small deviations on the fly (Aoki et al., 2019; Markov et al., 2021; Jia et al., 2025b). This separation between what to do (slow, policy-level) and how to execute it accurately (fast, execution-level) suggests that the test-time execution gap is more naturally closed by an online correction layer acting on the reference stream, rather than by modifying the policy and the controller themselves.

Building on this principle, we propose the **A**daptive **P**olicy **EX**ecution (**APEX**) framework, which implements such correction in two stages. First, APEX reconstructs a smooth, dynamically feasible reference from policy outputs through an adaptive passive filter (Wang et al., 2020), supplying the higher-order signals that low-level controllers require. Second, it augments this reference with test-time adaptive terms that compensate for unknown system dynamics and controller parameters, with coefficients updated online from the real-time tracking error. Notably, APEX is agnostic to the specific policy and controller class, also admitting provable convergence guarantees.

We evaluate APEX in three settings. First, we remove the policy and replay demonstration trajectories through the low-level controller, isolating the execution gap from policy performance. APEX reduces 41.2% of the tracking error. Second, we pair APEX with four policy classes, including ACT (Zhao et al., 2023),

diffusion policy (Chi et al., 2025), flow matching (Lipman et al., 2023), and the VLA $\pi_{0.5}$ (Black et al., 2025a), across multiple manipulation tasks and controller configurations, covering both deterministic and generative paradigms. APEX improves the success rate by 4.8–25.8 percentage points (pp), with clear gains on PegInsertion, a task with only 3 mm radial clearance. Finally, we deploy APEX on a UR5 for precision-sensitive tasks. The most pronounced gain occurs on the contact-rich PegInsertion task, where APEX improves the flow matching policy's success rate from 20% to 35%.

2 Related Work

We discuss three lines of work most closely related to the test-time execution gap.

Modifying the action representation. One line of work modifies how policies generate actions. The action space itself, joint angles, end-effector poses, or variable-impedance commands, shapes learning behavior (Martín-Martín et al., 2019; Aljalbout et al., 2024). Various methods enrich the output with B-spline trajectories or velocity augmentations to supply higher-order tracking signals (Yang et al., 2026; Hechtl et al., 2026; Bai et al., 2026), or output torques directly (Sood et al., 2024). All require retraining, which is prohibitive for large pretrained VLAs with action spaces fixed by collected datasets (Open X-Embodiment Collaboration et al., 2024). APEX leaves the policy untouched and corrects execution downstream.

Learning residual corrections. Residual learning keeps a base policy or controller fixed and learns an additive correction (Silver et al., 2018; Johannink et al., 2019). Recent extensions train the residual via differentiable simulation for legged locomotion (Luo et al., 2025), learn residual dynamics online within model predictive control for contact-rich manipulation (Huang et al., 2024), or fit a delta action model to bridge sim-to-real gaps for humanoids (He et al., 2025). All require additional training or interactive data collection. APEX requires no additional data collection. It updates existing unknown parameters at test-time from the real-time tracking error.

Refining policy outputs at test time. Another line refines policy outputs at test time without retraining, mostly targeting action-chunking policies like temporal ensembling (Zhao et al., 2023), bidirectional decoding (Liu et al., 2025), real-time chunking via inpainting (Black et al., 2025b), jerk-minimizing post-optimization (Son and Park, 2025), and observation-conditioned asynchronous correction (Sendai et al., 2025). These methods reshape the action stream to address policy-side artifacts such as latency and stale predictions, implicitly assuming the resulting reference is tracked faithfully on hardware. We show this assumption breaks down in Sec. 5.1. Even a replayed expert reference produces systematic tracking error under a commercial off-the-shelf controller. In contrast, APEX explicitly accounts for execution-side errors.

3 Problem Formulation

Modern robot policies are often trained by imitation learning from demonstration trajectories (Pomerleau, 1988; Argall et al., 2009). Given a demonstration dataset $\mathcal{D} = \{(o_i, a_i)\}_{i=1}^{N}$ of observation-action pairs with N samples, a policy π is trained as a mapping

$$\pi : o \mapsto a_\pi, \tag{1}$$

where o denotes the current sensor observation and a_π represents a policy-level action such as joint-position targets (Zhao et al., 2023) or end-effector pose/delta-pose targets (Chi et al., 2025; Open X-Embodiment Collaboration et al., 2024; Kim et al., 2025). At deployment, the policy produces a time-varying action reference $a_\pi(t)$, which is executed by a low-level controller. Such a controller is often inaccessible to users, particularly on commercial platforms such as UR and xArm.

Let $a(t)$ denote the actual action executed by the controller. The low-level dynamics follow the general *Lagrange* equation with inertia matrix M, Coriolis matrix C, gravity vector G, and control torque τ. Most policy-based architectures adopt a low-level PD controller τ_c with gains K_p and K_d to track a_π. Then we can formalize them as

$$M(a)\ddot{a} + C(a, \dot{a})\dot{a} + G(a) = \tau, \quad \tau_c = K_p(a_\pi - a) - K_d\dot{a}. \tag{2}$$

Define the tracking error $e_a = a_\pi - a$. In the absence of desired velocity and acceleration signals of $a_\pi(t)$, as well as the model knowledge, *Lyapunov*'s theorem implies that the above PD controller inevitably incurs noticeable tracking errors (Slotine and Li, 1991). Moreover, the PD controller is often inaccessible to users, especially on commercial platforms such as UR and xArm.

Our objective: In this work, our goal is to construct a *test-time correction module* that refines the policy output a_π into an adjusted command a_r, such that feeding a_r into the low-level PD controller yields an accurate tracking of the original a_π, i.e., $e_a \to 0$. Crucially, to ensure practicality, it is expected to require no modification to either the high-level policy or the low-level controller.

4 Adaptive Policy Execution

As shown in Fig. 1, APEX is a test-time execution adapter that transforms the intended action from the high-level policy into an adjusted command for the low-level execution controller. In this section, we first present APEX under *known* execution dynamics to illustrate its core principle, and then extend the formulation to the *unknown*-dynamics setting by introducing a test-time adaptation algorithm, for which we provide a stability analysis.

4.1 Correction with Known Execution Dynamics

We begin with an idealized setting in which the low-level execution dynamics are known. Without loss of generality, we adopt the *joint* coordinate as the action space, so that $a_\pi(t)$ denotes the policy-intended joint angle. With the fixed PD controller of the platform, the closed-loop dynamics of each joint take the form

$$J\ddot{a} = K_p(a_r - a) - K_d\dot{a} + d, \tag{3}$$

where J denotes the inertia of the joint motor and $d(t)$ lumps all unmodeled effects, including joint coupling and other nonlinear terms. Compared with the full *Lagrangian* model in Eq. (2), Eq. (3) treats each joint as a perturbed single-input channel.

Considering that the final goal is to ensure that the realized motion follows $a_\pi(t)$ exactly, substituting $a = a_\pi$ into Eq. (3) and solving for a_r would yield

$$a_r = a_\pi + \frac{J}{K_p}\,\ddot{a}_\pi + \frac{K_d}{K_p}\,\dot{a}_\pi - \frac{d}{K_p}. \tag{4}$$

This expression reveals that faithfully performing a_π requires acceleration- and velocity-dependent offsets in the command, which current policies do not provide (Zhao et al., 2023; Chi et al., 2025; Black et al., 2025a). Eq. (4) is therefore not directly applicable.

Although finite differencing is a practical remedy in implementation, we adopt an adaptive passive filter (Wang et al., 2020) here for theoretical completeness. We construct an auxiliary smooth reference $\bar{a}$ together with its derivatives, generated by the adaptive passive filter driven by the tracking error $e_a = a_\pi - a$

$$\dot{y} = -K_1 y - K_1 e_a, \qquad \dot{\bar{a}} = -K_2 y, \tag{5}$$

where y is an internal filter state and $K_1, K_2 > 0$ are filter gains. The trajectory $\bar{a}$ is obtained by integrating $\dot{\bar{a}}$. Intuitively, whenever $e_a \neq 0$ the filter drives $\bar{a}$ toward a_π in a smooth, bandwidth-limited manner (Wang et al., 2020). At steady state ($e_a \to 0$), $\bar{a}$ coincides with a_π, and Eq. (5) simultaneously yields the well-defined $\dot{\bar{a}}$ and $\ddot{\bar{a}} = K_1 K_2(y + e_a)$ that Eq. (4) requires.

Replacing a_π and its derivatives in Eq. (4) with the filtered counterparts and adding a feedback term to anchor $\bar{a}$ to a_π via a, we obtain the *ideal command*

$$a_r^\star = \bar{a} + \frac{J}{K_p}\,\dddot{\bar{a}} + \frac{K_d}{K_p}\,\ddot{\bar{a}} + \frac{\alpha}{K_p}\,e_a, \tag{6}$$

with a gain $\alpha > 0$. The last term steers the realized motion toward a_π and stabilizes the closed loop (convergence is established later). Note the term $d(t)$ is dropped from the design, as obtaining its analytical form is typically nontrivial. We instead adopt the standard assumption in adaptive control (Slotine and Li, 1991) that $d(t)$ is bounded, i.e., $|d(t)| \le \bar{d}$ for some unknown $\bar{d} > 0$.

4.2 Correction under Unknown Execution Dynamics

Building on Eq. (6), a direct implementation would require knowledge of the parameters J, K_p, and K_d, which are typically inaccessible on closed-source commercial arms. APEX instead introduces a test-time adaptive algorithm (Wang et al., 2020) that learns these unknown parameters directly from execution feedback. Before proceeding, we re-parameterize Eq. (6) as

$$a_r = \bar{a} + w\big[J\,\ddot{\bar{a}} + \alpha\, e_a\big] + w_d\,\dot{\bar{a}}, \tag{7}$$

where w plays the role of $1/K_p$ and w_d corresponds to K_d/K_p. These scalar weights together with J are *unknown a priori*. APEX performs a test-time adaptive learning on these coefficients

$$\dot{\hat{w}} = -\Gamma_w\, s\big[\hat{J}\,\ddot{\bar{a}} + \alpha\, e_a\big], \qquad \dot{\hat{w}}_d = -\Gamma_{wd}\, s\,\dot{\bar{a}}, \qquad \dot{\hat{J}} = -\Gamma_J\, s\,\ddot{\bar{a}}, \tag{8}$$

where $\Gamma_w, \Gamma_{wd}, \Gamma_J > 0$ are learning rates, $\hat{(\cdot)}$ denotes the estimation of signal $(\cdot)$, and $s = \dot{a} - \dot{\bar{a}}$. Here, $\dot{a}$ is the action velocity measured from robot state feedback. For joint-space coordinates, this corresponds to the measured joint velocity. For task-space coordinates, it can be obtained from end-effector velocity feedback or computed from joint velocity through the robot *Jacobian*. Importantly, APEX does not differentiate the policy output a_π. The reference velocity $\dot{\bar{a}}$ is generated internally by the reconstruction filter in Eq. (5).

Eqs. (5), (7), and (8) together specify the APEX framework. We now establish that the resulting closed-loop system, the joint dynamics (3) interconnected with the filter (5), correction (7), and adaptation laws (8), is stable and achieves asymptotic tracking.

Theorem 1. *Consider the execution model (3) with unknown parameters $J, K_p, K_d > 0$ and a bounded disturbance satisfying $|d(t)| \le \bar{d}$, driven by the corrected command (7) with the filtered reference (5) and adaptation law (8). Assume that the policy action a_π is fixed for the nominal analysis. For any positive gains $K_1, K_2, \alpha, \Gamma_w, \Gamma_{wd}, \Gamma_J > 0$ and any finite initial condition, all closed-loop signals remain bounded, and tracking error e_a converges to a residual set, i.e.,*

$$\limsup_{t\to\infty} |s(t)| \;\le\; c_1\,\bar{d}, \qquad \limsup_{t\to\infty} |e_a(t)| \;\le\; c_2\,\bar{d}, \tag{9}$$

for constants $c_1, c_2 > 0$ depending on the design gains.

Proof. See Appendix A for complete proof details. □

We emphasize that, although several assumptions are introduced for theoretical completeness, our empirical results below show that APEX achieves substantial gains even in regimes where such idealized conditions are unlikely to hold in reality.

5 Experiments

Our central hypothesis is that the test-time execution gap arises not only from high-level policy prediction errors but also from suboptimal execution of the low-level controller. To verify, we design the experiments around three progressively broader questions: (1) How much do low-level controllers contribute to test-time execution failures? (2) Does APEX effectively bridge the joint policy–controller execution gap? (3) If so, does this benefit generalize across policies, action representations, simulators, and platforms? We describe the experimental setups in Appendix B.

5.1 Imprecise controller alone can cause significant failures

We first ask to what extent low-level controller execution errors degrade overall task performance. To answer this question, we consider an idealized setting: instead of using actions predicted by a learned policy, we directly replay expert demonstration trajectories as action references. These references represent best-case policy outputs. If execution failures are mainly due to inaccurate policy predictions, then this idealized setting should produce reliable rollouts.

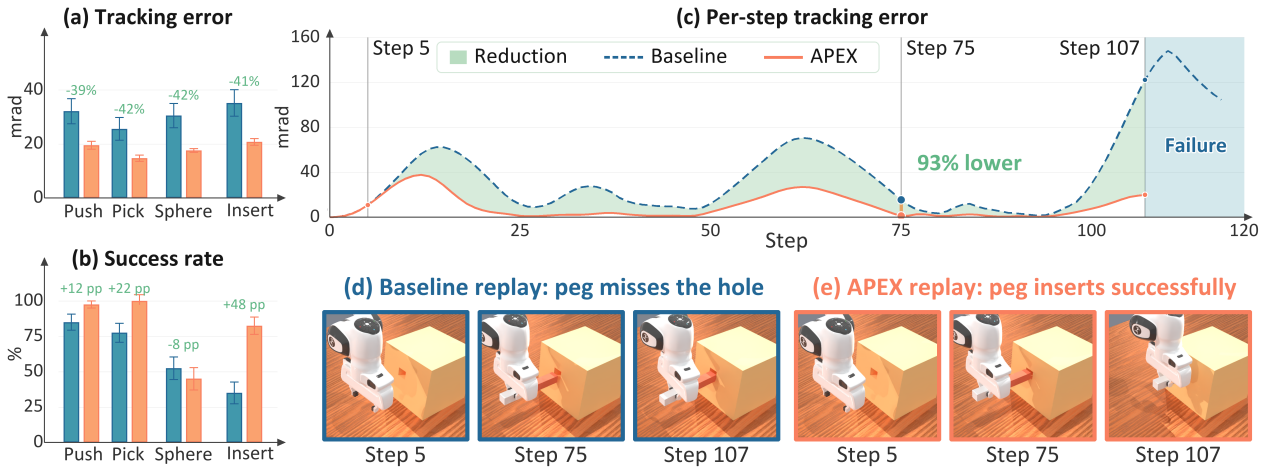

Figure 2 Expert-replay benchmark in simulation. **(a)** Under a vanilla PD controller, APEX reduces tracking error across all four tasks, leading to improved success rates in **(b)**. **(c)** On PegInsertion, APEX reduces the per-step tracking error throughout the trajectory, with up to a 93% reduction near the insertion phase, enabling the successful insertion shown in **(d,e)**.

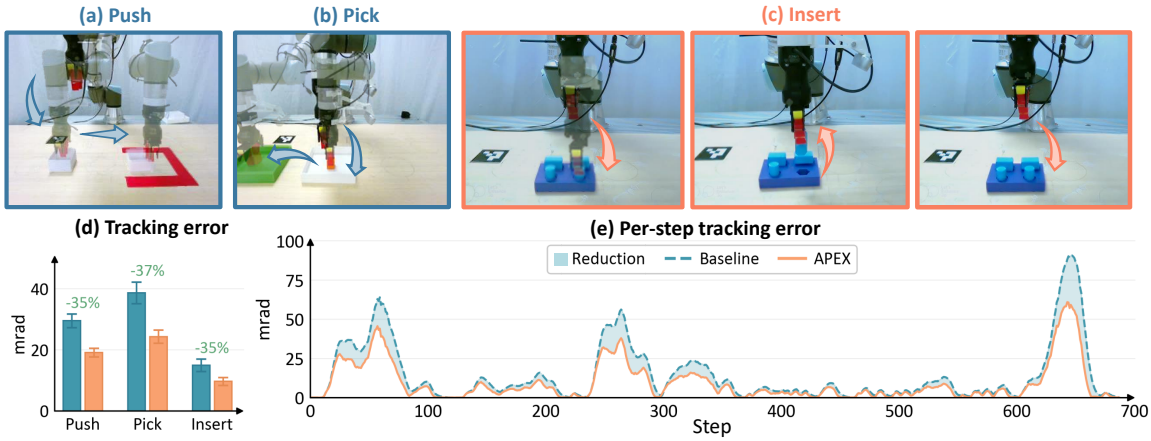

Figure 3 Real-world deployment on a UR5. APEX is evaluated on three precision-sensitive manipulation tasks executed on UR5: **(a)** Push, **(b)** Pick, and **(c)** Insert, shown here as the recorded execution sequences. **(d)** Across all three tasks, APEX reduces the tracking error under the robot's native position servo. **(e)** On PegInsertion, APEX lowers the per-step tracking error.

Fig. 2 shows that this is not the case. In the following experiments, **Baseline** denotes uncorrected execution, where the reference command is sent directly to the low-level execution stack without APEX. Under mismatched low-level execution parameters, the realized motion still deviates substantially from the expert reference. As shown in Fig. 2(a), baseline replay produces tracking errors of roughly 25–35 mrad across the four tasks. These deviations are large enough to affect task outcomes. Fig. 2(b) shows that replay success varies widely across tasks and drops sharply on precision-sensitive settings such as PegInsertion. Even with expert-level action references, an inaccurate low-level controller can fail to execute the intended motion accurately.

Without changing the replayed reference, APEX reduces the mean tracking error from 30.84 mrad to 18.15 mrad across all replay settings, a 41.2% reduction. Replay success also increases from 62.5% to 81.3%. The PegInsertion tracking error in Fig. 2(c) shows APEX keeps the realized motion closer to the expert reference near insertion. The visual rollouts in Fig. 2(d,e) show the corresponding task outcome, where the baseline misses the hole while the APEX-corrected replay inserts successfully. The same effect holds on a physical UR5, as presented in Fig. 3. Across three real-world precision tasks, APEX reduces per-step tracking error and turns failed insertions into successful ones, confirming that the simulation findings generalize to hardware.

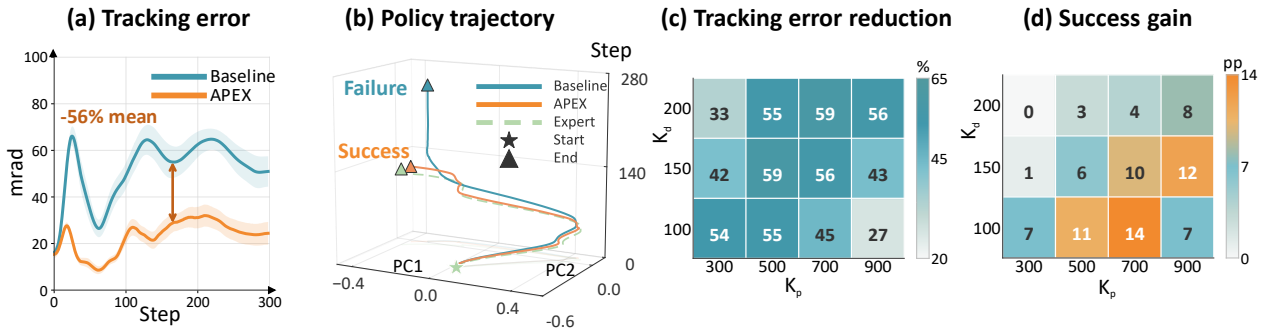

Figure 4 Deployment with ACT policy on PegInsertion. **(a)** Action tracking error. **(b)** PCA visualization of the joint-action trajectory, where PC1 and PC2 denote the first two principal components of joint positions, and steps are shown on the vertical axis. **(c)** Tracking-error reduction over the $K_p \times K_d$ grid. **(d)** Success-rate gain over the same grid.

Table 1 Cross-policy success comparison under the same fixed controller mismatch. Δ reports the success-rate gain. All values are success rates in %, and Δ is in percentage points (pp).

	ACT			FM			DP		
Task	**Baseline**	**APEX**	**Δ**	**Baseline**	**APEX**	**Δ**	**Baseline**	**APEX**	**Δ**
PushCube-v1	97	97	+0	90	94	+4	88	98	+10
PickCube-v1	41	75	+34	37	36	-1	35	53	+18
PlaceSphere-v1	15	6	-9	0	13	+13	0	40	+40
PegInsertionSide-v1	0	11	+11	0	3	+3	0	2	+2

5.2 APEX reliably bridges the policy–controller gap

In practice, no policy can output actions perfectly aligned with the demonstrations, which compounds the execution gap. We next evaluate whether APEX can reduce this action execution gap when the action reference is generated by a learned policy. Specifically, we use the ACT (Zhao et al., 2023) as learned policy. In this setting, the tracking error is measured between the policy-intended action reference and the executed action. The results are provided in Fig. 4. APEX substantially reduces tracking error from 51.55 mrad to 25.95 mrad across all ACT rollouts. Fig. 4(a) and Fig. 4(b) illustrate that the realized trajectory under APEX remains closer to the policy reference over time, whereas direct execution accumulates larger deviations.

We then examine whether this execution-level correction is robust across different low-level controller settings. Fig. 4(c) reports the tracking-error reduction on PegInsertion over the evaluated $K_p \times K_d$ settings. APEX reduces the policy-reference tracking error in every shown setting, indicating that the adapter is not tuned to a single PD configuration but remains effective across a range of inaccurate test-time controllers. The reduction in tracking error naturally leads to the success-rate gain, as shown in Fig. 4(d). The gains are larger under more inaccurate control settings. Moreover, across all four ACT tasks and 48 controller settings, mean success increases from 27.4% to 40.6%. APEX improves 40 settings, leaves 5 unchanged, and degrades only 3. The full per-task grids are provided in Appendix F.

5.3 APEX generalizes across policies and platforms

We next examine whether this benefit is tied to that particular policy-controller setting or whether it remains observable across different policy families, action representations, and benchmark platforms. We first evaluate ACT (Zhao et al., 2023), Diffusion Policy (DP) (Chi et al., 2025), and Flow Matching (FM) (Lipman et al., 2023) on ManiSkill under the same controller. As shown in Tab. 1, APEX improves performance across all three policy classes, with average gains of +9.0 pp on ACT, +4.8 pp on FM, and +17.5 pp on DP over the four tasks. These results indicate that the benefit of execution-side correction is not specific to ACT.

We further test the same idea on a different benchmark platform, LIBERO Spatial, using the $\pi_{0.5}$ (Black et al., 2025a). In this setting, we change the simulator backend, task distribution, and action representation

Table 2 Real-robot success rate (%) on a UR5 low-level controller. Δ reports the success-rate change in pp.

	FM			DP		
Task	**Baseline**	**APEX**	**Δ**	**Baseline**	**APEX**	**Δ**
PushBox	55	45	-10	75	80	+5
PickCube	55	65	+10	40	45	+5
PegInsertion	20	35	+15	30	25	-5

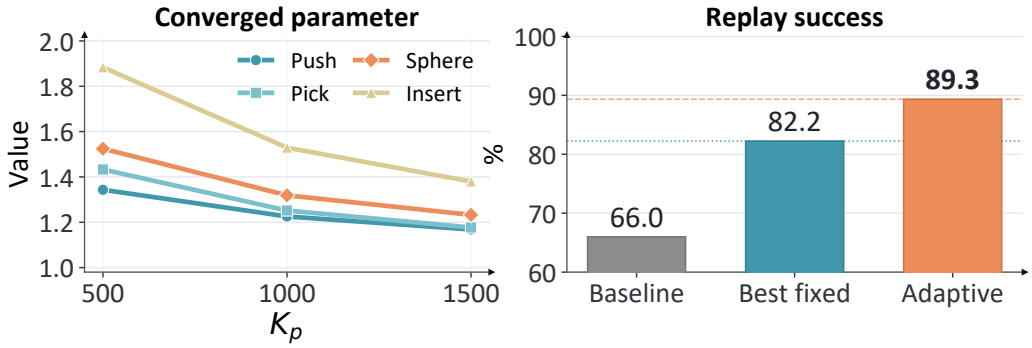

Figure 5 Adaptive-update ablation. Left: Convergence behavior under varying control gains. *Value* denotes the effective multiplier on the position-error correction, corresponding to $1 + w\alpha$ in Eq. (6). 1 means no extra correction. It decreases with K_p, showing that APEX adapts a stronger correction when tracking is degraded. **Right:** Success rates for direct replay, the best fixed-parameter APEX, and adaptive APEX.

simultaneously. Unlike the ManiSkill policies, $\pi_{0.5}$ outputs end-effector delta actions, which are mapped to joint-space commands through inverse kinematics before execution correction is applied. Across the evaluated LIBERO tasks, APEX reduces the mean end-effector tracking error from 8.0 mm to 4.9 mm and improves the mean success rate from 72.1% to 97.9%. This result suggests that the observed benefit is not restricted to joint-position policies. Full results are provided in Appendix G.

Finally, we validate APEX on a UR5 across three real-robot tasks, where we collect demonstrations and train both DP and FM policies with end-effector delta action outputs to assess transfer beyond simulation. As shown in Tab. 2, APEX improves the mean success rate for both policy classes, with the largest gain on the most precision-sensitive task, PegInsertion, where it raises the FM success rate from 20% to 35% (+15 pp).

5.4 Ablation Study

This ablation asks whether the adaptive update in APEX is doing useful work or merely acting as a fixed offset. Further ablations on the adaptation gains and simulator-side joint dynamics parameters, including friction loss and armature, are deferred to the Appendix E.

We first perform a grid search over fixed correction coefficients with online adaptation disabled throughout execution (Appendix D, Tab. S2). The best fixed-correction setting achieves a mean success rate of 82.2% across the swept configurations. We compare this fixed-parameter variant against adaptive APEX in Fig. 5. The adaptive variant reaches 89.3%, surpassing both the uncorrected baseline and the best fixed-parameter variant. From an adaptive-control perspective, this finding is consistent with a classical insight in adaptive control. Control-oriented adaptation driven by closed-loop tracking error can outperform regression-oriented identification (Richards et al., 2023).

6 Limitations

One limitation of APEX is that it involves several adaptation gains, which currently require manual tuning. We plan to address this by integrating self-tuning mechanisms that adjust these gains online based on the

observed tracking behavior. In addition, the disturbance term $d(t)$ in Eq. (3) is omitted in the current design, leading to a bounded convergence. In principle, $d(t)$ could be estimated online via a disturbance observer (Jia et al., 2025a) to reduce conservatism. We leave such an extension to future work.

7 Conclusion

We presented **APEX**, a test-time correction layer that bridges the *execution gap* between high-level learned policies and low-level inaccessible controllers without modifying either. APEX reconstructs higher-order signals via a dynamic filter and refines the reference through an online adaptive law with stability guarantees. Empirically, APEX raises replay success from 62.5% to 81.3% and yields 4.8–25.8 pp improvements across four policy classes, including ACT, flow matching, diffusion policy, and $\pi_{0.5}$. These results suggest that even a well-trained policy can fail simply because its action references are not faithfully executed, and that a lightweight correction layer at the policy-controller interface can recover much of this lost performance.

References

Elie Aljalbout, Felix Frank, Maximilian Karl, and Patrick van der Smagt. On the role of the action space in robot manipulation learning and sim-to-real transfer. *IEEE Robotics and Automation Letters*, 9(6):5895–5902, 2024.

Stephanie K. Aoki, Gabriele Lillacci, Ankit Gupta, Armin Baumschlager, David Schweingruber, and Mustafa Khammash. A universal biomolecular integral feedback controller for robust perfect adaptation. *Nature*, 570(7762): 533–537, 2019.

Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

Jiaqi Bai, Jindou Jia, Yuxuan Hu, Gen Li, Xiangyu Chen, Tuo An, Kuangji Zuo, and Jianfei Yang. FLASH: Efficient visuomotor policy via sparse sampling. *arXiv preprint arXiv:2605.15492*, 2026.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Proceedings of the 9th Conference on Robot Learning*, pages 17–40. PMLR, 2025a.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *Proceedings of Advances in Neural Information Processing Systems*, 2025b.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Tairan He, Jiawei Gao, Wenli Xiao, Yuanhang Zhang, Zi Wang, Jiashun Wang, Zhengyi Luo, Guanqi He, Nikhil Sobanbab, Chaoyi Pan, Zeji Yi, Guannan Qu, Kris Kitani, Jessica Hodgins, Linxi Fan, Yuke Zhu, Changliu Liu, and Guanya Shi. ASAP: Aligning simulation and real-world physics for learning agile humanoid whole-body skills. In *Proceedings of Robotics: Science and Systems*, 2025.

Johannes Hechtl, Philipp Schmitt, Georg von Wichert, and Wolfram Burgard. Enabling dynamic tracking in vision-language-action models via time-discrete and time-continuous velocity feedforward. *arXiv preprint arXiv:2603.16218*, 2026.

Wei-Cheng Huang, Alp Aydinoglu, Wanxin Jin, and Michael Posa. Adaptive contact-implicit model predictive control with online residual learning. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 5822–5828, 2024.

Jindou Jia, Kexin Guo, Yuyang Wang, Sicheng Zhou, Jiayi Zhang, Yuhang Liu, Xiang Yu, Yang Shi, and Lei Guo. FORESEER: Recognize and utilize uncertainties by integrating data-based learning and symbolic feedback. *The International Journal of Robotics Research*, 2025a.

Jindou Jia, Zihan Yang, Meng Wang, Kexin Guo, Jianfei Yang, Xiang Yu, and Lei Guo. Feedback favors the generalization of neural ODEs. In *Proceedings of International Conference on Learning Representations*, 2025b.

Jindou Jia, Tuo An, Yuxuan Hu, Gen Li, Jingliang Li, Bohan Hou, Xiangyu Chen, Jiaqi Bai, Bofan Lyu, and Jianfei Yang. MARS policy: Multimodality only when it matters. *arXiv preprint arXiv:2605.29766*, 2026a.

Jindou Jia, Gen Li, Xiangyu Chen, Tuo An, Yuxuan Hu, Jingliang Li, Xinying Guo, and Jianfei Yang. Action-to-action flow matching. *arXiv preprint arXiv:2602.07322*, 2026b.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2019.

Hassan K. Khalil. *Nonlinear systems*. Prentice Hall, 3rd edition, 2002.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of the 8th Conference on Robot Learning*, pages 2679–2713. PMLR, 2025.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *Proceedings of International Conference on Learning Representations*, 2023.

Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Max Du, and Chelsea Finn. Bidirectional decoding: Improving action chunking via guided test-time sampling. In *Proceedings of International Conference on Learning Representations*, 2025.

Jing Yuan Luo, Yunlong Song, Victor Klemm, Fan Shi, Davide Scaramuzza, and Marco Hutter. Residual policy learning for perceptive quadruped control using differentiable simulation. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2025.

Daniil A. Markov, Luigi Petrucco, Andreas M. Kist, and Ruben Portugues. A cerebellar internal model calibrates a feedback controller involved in sensorimotor control. *Nature Communications*, 12(1):6694, 2021.

Roberto Martín-Martín, Michelle A. Lee, Rachel Gardner, Silvio Savarese, Jeannette Bohg, and Animesh Garg. Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

Open X-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, et al. Open X-embodiment: Robotic learning datasets and RT-X models. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 6892–6903, 2024.

Dean A. Pomerleau. ALVINN: An autonomous land vehicle in a neural network. In *Proceedings of Advances in Neural Information Processing Systems*, pages 305–313, 1988.

Spencer M Richards, Navid Azizan, Jean-Jacques Slotine, and Marco Pavone. Control-oriented meta-learning. *The International Journal of Robotics Research*, 42(10):777–797, 2023.

Kohei Sendai, Maxime Alvarez, Tatsuya Matsushima, Yutaka Matsuo, and Yusuke Iwasawa. Leave no observation behind: Real-time correction for VLA action chunks. *arXiv preprint arXiv:2509.23224*, 2025.

Tom Silver, Kelsey R. Allen, Joshua B. Tenenbaum, and Leslie Pack Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

Jean-Jacques E. Slotine and Weiping Li. *Applied nonlinear control*. Prentice Hall, 1991.

Dongwoo Son and Suhan Park. LiPo: A lightweight post-optimization framework for smoothing action chunks generated by learned policies. *International Journal of Control, Automation and Systems*, 23(11):3284–3292, 2025.

Shivam Sood, Ge Sun, Peizhuo Li, and Guillaume Sartoretti. DecAP: Decaying action priors for accelerated imitation learning of torque-based legged locomotion policies. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2809–2815, 2024.

Hanlei Wang, Wei Ren, Chien Chern Cheah, Yongchun Xie, and Shangke Lyu. Dynamic modularity approach to adaptive control of robotic systems with closed architecture. *IEEE Transactions on Automatic Control*, 65(6): 2760–2767, 2020.

Fan Yang, Peiguang Jing, Kaihua Qu, Ningyuan Zhao, and Yuting Su. ABPolicy: Asynchronous B-Spline flow policy for real-time and smooth robotic manipulation. *arXiv preprint arXiv:2602.23901*, 2026.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems*, 2023.

Appendices

Within this supplementary material, we elaborate on the following aspects:

A Stability Analysis of Theorem 1

This appendix proves Theorem 1 from Sec. 4.2. The proof first establishes asymptotic convergence under the nominal setting ($d \equiv 0$), then extends to the bounded-disturbance case via *Young*'s inequality.

A.1 Closed-Loop Error Dynamics

We analyze a single execution channel. The scalar nominal model ($d \equiv 0$) is

$$J\ddot{a} = K_p(a_r - a) - K_d\dot{a}, \tag{10}$$

where a is the realized action coordinate, a_r is the adjusted command sent to the low-level controller, and $J, K_p, K_d > 0$ are unknown execution parameters. The policy provides a fixed desired command a_π. Following the main text, define the tracking error and sliding variable

$$e_a = a_\pi - a, \qquad s = \dot{a} - \dot{\bar{a}}, \tag{11}$$

with the auxiliary smooth reference $\bar{a}$ generated by

$$\dot{y} = -K_1 y - K_1 e_a, \qquad \dot{\bar{a}} = -K_2 y, \qquad \ddot{\bar{a}} = K_1 K_2 (y + e_a). \tag{12}$$

For the Lyapunov construction we also introduce the auxiliary positional error

$$\zeta = a - \bar{a}, \tag{13}$$

which directly links the sliding variable s (i.e., $\dot{\zeta} = s$) to a positional quantity.

APEX uses the adaptive command

$$a_r = \bar{a} + \hat{w}\big[\hat{J}\ddot{\bar{a}} + \alpha e_a\big] + \hat{w}_d\dot{\bar{a}}, \tag{14}$$

with online updates

$$\dot{\hat{w}} = -\Gamma_w s\big[\hat{J}\ddot{\bar{a}} + \alpha e_a\big], \qquad \dot{\hat{w}}_d = -\Gamma_{wd} s\dot{\bar{a}}, \qquad \dot{\hat{J}} = -\Gamma_J s\ddot{\bar{a}}. \tag{15}$$

Eqs. (14)–(15) are identical to Eqs. (7)–(8) in the main text. Define the parameter estimation errors

$$\tilde{w} = \hat{w} - w, \qquad \tilde{w}_d = \hat{w}_d - w_d, \qquad \tilde{J} = \hat{J} - J, \tag{16}$$

where $w = 1/K_p$ and $w_d = K_d/K_p$. The proof only requires finite initial conditions. In implementation, we initialize $\bar{a}(0) = a(0)$, giving $\zeta(0) = 0$ and reducing the initial transient.

Substituting Eq. (14) into Eq. (10) and expanding $\hat{w} = w + \tilde{w}$, $\hat{w}_d = w_d + \tilde{w}_d$, $\hat{J} = J + \tilde{J}$, the s-dynamics become

$$\begin{aligned} J\dot{s} = &-K_d s - K_p\zeta + \alpha e_a \\ &+ K_p\tilde{w}\big[\hat{J}\ddot{\bar{a}} + \alpha e_a\big] + K_p\tilde{w}_d\dot{\bar{a}} + \tilde{J}\ddot{\bar{a}}. \end{aligned} \tag{17}$$

The parametric uncertainty enters in three places (one per adaptive coefficient), each multiplied by a measurable regressor. Since a_π is fixed, $\dot{e}_a = -\dot{a}$, and the filter-state dynamics reduce to

$$\dot{e}_a = -s + K_2 y, \tag{18}$$

$$\dot{y} = -K_1 y - K_1 e_a. \tag{19}$$

A.2 Proof of Theorem 1

Lyapunov candidate. We choose

$$\begin{aligned} V = &\underbrace{\tfrac{1}{2}Js^2}_{\text{sliding}} + \underbrace{\tfrac{1}{2}K_p\zeta^2}_{\text{position}} + \underbrace{\tfrac{1}{2}\alpha e_a^2 + \tfrac{1}{2}\alpha\tfrac{K_2}{K_1}y^2}_{\text{filter subsystem}} \\ &+ \underbrace{\tfrac{K_p}{2\Gamma_w}\tilde{w}^2 + \tfrac{K_p}{2\Gamma_{wd}}\tilde{w}_d^2 + \tfrac{1}{2\Gamma_J}\tilde{J}^2}_{\text{parameter errors}}. \end{aligned} \tag{20}$$

The filter-subsystem terms are weighted to cancel the cross-coupling between e_a and y in Eqs. (18)–(19) in the derivative. The parameter-error weights match the inverse learning rates so that the adaptive law in Eq. (15) cancels the parametric terms in $\dot{s}$. Since $J, K_p, \alpha, K_1, K_2, \Gamma_w, \Gamma_{wd}, \Gamma_J > 0$, V is positive definite in the error variables.

Derivative along trajectories. Differentiating Eq. (20) along Eqs. (17)–(19), the state cross terms cancel

$$-K_p\zeta s + K_p\zeta s = 0, \qquad +\alpha s e_a + \alpha e_a(-s) = 0, \tag{21}$$

$$\alpha K_2 e_a y - \alpha K_2 y\, e_a = 0. \tag{22}$$

The parameter-error products are cancelled by the adaptive law (15)

$$\begin{aligned} &K_p s\tilde{w}\big(\hat{J}\ddot{\bar{a}} + \alpha e_a\big) + \tfrac{K_p}{\Gamma_w}\tilde{w}\dot{\hat{w}} = 0, \\ &K_p s\tilde{w}_d\dot{\bar{a}} + \tfrac{K_p}{\Gamma_{wd}}\tilde{w}_d\dot{\hat{w}}_d = 0, \\ &s\tilde{J}\ddot{\bar{a}} + \tfrac{1}{\Gamma_J}\tilde{J}\dot{\hat{J}} = 0. \end{aligned} \tag{23}$$

The remaining terms yield

$$\dot{V} = -K_d s^2 - \alpha K_2 y^2 \le 0. \tag{24}$$

Boundedness. Since $\dot{V} \le 0$, V is non-increasing, and all error variables

$$s,\ \zeta,\ e_a,\ y,\ \tilde{w},\ \tilde{w}_d,\ \tilde{J} \tag{25}$$

are bounded. With a_π fixed, bounded $e_a = a_\pi - a$ implies bounded a. Bounded $\zeta = a - \bar{a}$ then implies bounded $\bar{a}$. From Eq. (12), bounded y and e_a imply bounded $\dot{\bar{a}}, \ddot{\bar{a}}$. Boundedness of $\hat{w}, \hat{w}_d, \hat{J}$ follows from bounded $\tilde{w}, \tilde{w}_d, \tilde{J}$. Finally, Eq. (14) gives bounded a_r, and $\dot{a} = s + \dot{\bar{a}}$ is bounded. Thus all closed-loop signals remain bounded.

Asymptotic convergence via Barbalat's lemma. Bounded right-hand sides in Eqs. (17) and (19) make $\dot{s}, \dot{y}$ bounded, so s and y are uniformly continuous. Integrating Eq. (24) from 0 to ∞,

$$K_d \int_0^\infty s^2(\tau)\, d\tau + \alpha K_2 \int_0^\infty y^2(\tau)\, d\tau \le V(0) - V(\infty) < \infty, \tag{26}$$

gives $s, y \in L_2$. Combined with uniform continuity, Barbalat's lemma (Khalil, 2002) yields

$$\lim_{t\to\infty} s(t) = 0, \qquad \lim_{t\to\infty} y(t) = 0. \tag{27}$$

Convergence of the tracking error. The (e_a, y) subsystem from Eqs. (18)–(19) is

$$\frac{d}{dt}\begin{bmatrix} e_a \\ y \end{bmatrix} = \begin{bmatrix} 0 & K_2 \\ -K_1 & -K_1 \end{bmatrix}\begin{bmatrix} e_a \\ y \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix} s. \tag{28}$$

The system matrix has trace $-K_1 < 0$ and determinant $K_1 K_2 > 0$, hence is Hurwitz for any $K_1, K_2 > 0$. Since $s(t) \to 0$ acts as a vanishing input and the state is bounded, standard input-to-state arguments for LTI systems (Khalil, 2002) give

$$\lim_{t\to\infty} e_a(t) = 0, \tag{29}$$

which together with $s(t) \to 0$ completes the nominal proof.

Extension to bounded disturbances. When the lumped disturbance $d(t)$ in Eq. (3) is nonzero but bounded, $|d(t)| \le \bar{d}$ with $\bar{d} > 0$, the s-dynamics in Eq. (17) acquire an additional term $d(t)$ on the right-hand side, and the *Lyapunov* derivative becomes

$$\dot{V} = -K_d s^2 - \alpha K_2 y^2 + s\, d(t). \tag{30}$$

Applying *Young*'s inequality, $s\, d(t) \le \frac{K_d}{2} s^2 + \frac{1}{2K_d} d^2(t) \le \frac{K_d}{2} s^2 + \frac{\bar{d}^2}{2K_d}$, yields

$$\dot{V} \le -\tfrac{K_d}{2} s^2 - \alpha K_2 y^2 + \tfrac{\bar{d}^2}{2K_d}. \tag{31}$$

Standard input-to-state stability arguments (Khalil, 2002) then ensure that all closed-loop signals remain bounded and the sliding variable and tracking error converge to a residual set whose size scales linearly with the disturbance bound:

$$\limsup_{t\to\infty} |s(t)| \le c_1\, \bar{d}, \qquad \limsup_{t\to\infty} |e_a(t)| \le c_2\, \bar{d}, \tag{32}$$

for constants $c_1, c_2 > 0$ depending on the design gains. As $\bar{d} \to 0$, the residual set shrinks to the origin, recovering the nominal result $s(t), e_a(t) \to 0$. □

Theorem 1 assumes a fixed a_π, while a deployed policy issues a sequence of commands. At each policy update, the physical states $(a, \dot{a})$, filter state y, reference $\bar{a}$, command a_r, and adaptive parameters $\hat{w}, \hat{w}_d, \hat{J}$ remain finite. Only $e_a = a_\pi - a$ jumps when a_π changes. Provided the policy outputs are bounded, each held-command interval begins from a finite initial condition, and Theorem 1 applies interval-wise. This yields stabilization and error attenuation within each interval rather than global convergence to an arbitrary time-varying sequence. The empirical evaluation in Sec. 5 measures the resulting finite-time error reduction over full rollouts.

B Experimental Setup

This appendix provides the implementation details behind the experimental results in the main paper. For each experiment family, we specify the simulator or hardware platform, task suite, policy class, low-level execution interface, and evaluation protocol used for comparing direct execution with APEX.

B.1 Simulation Environments

Fig. S1 summarizes the simulated environments, which include four ManiSkill tasks and three LIBERO Spatial tasks. In the LIBERO experiments, we evaluate the $\pi_{0.5}$ VLA, whose end-effector delta actions are first converted into joint-space commands through inverse kinematics before being passed to the APEX correction layer.

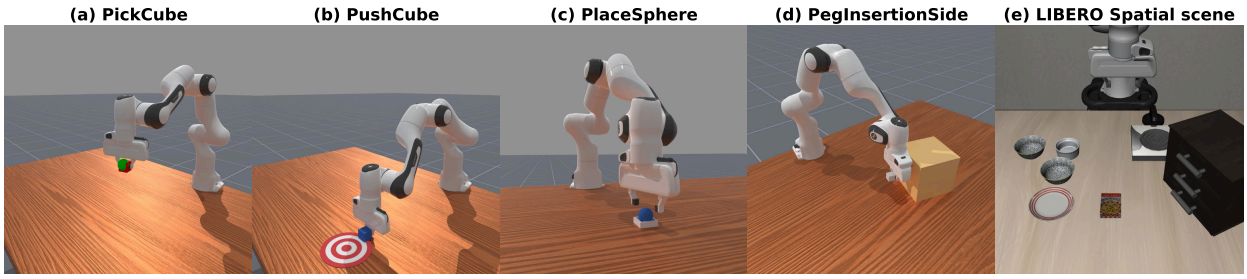

Figure S1 Simulation tasks and scenes used in our experiments.

B.2 Real-robot Environments

Hardware platform. All real-robot experiments are conducted on a 6-DoF UR5 manipulator equipped with a Robotiq Hand-E adaptive parallel-jaw gripper. Visual observations are captured by two Intel RealSense D435i (eye-in-hand) & D455 (eye-to-hand) cameras. Policy inference and execution-side correction run on a workstation with an NVIDIA RTX 3090.

Control stack. The DP and FM policies output end-effector delta actions, which are mapped to joint-space commands via inverse kinematics before execution-side correction. As a commercial platform, the UR5 exposes only a position-command interface and does not expose or allow modification of its internal controller gains. The execution-side terms of APEX are therefore estimated online from the measured tracking error rather than from known controller parameters.

Tasks & dataset. We evaluate three manipulation tasks of increasing precision demand: PushBox (planar pushing of a box to a goal region), PickCube (grasping a cube and placing it in a box), and PegInsertion (inserting a peg into a hole whose radius exceeds the peg radius by 3 mm). For each task, we collect 60 demonstrations via teleoperation at 30 Hz, recording RGB images and proprioceptive joint states. The tasks differ markedly in how strongly their outcome depends on execution accuracy: PushBox succeeds under coarse motion, whereas PegInsertion is dominated by millimeter-level alignment and is therefore the most sensitive to residual tracking error.

Evaluation protocol. We train one DP (Chi et al., 2025) and one FM (Lipman et al., 2023) policy per task using the best validation epoch. Each task-policy pair is evaluated over 20 trials. Each episode is capped at 200 inference steps, beyond which the trial is counted as a failure. The Baseline corresponds to the original policy executed by the joint-position servo without APEX. All other factors (controller, initial-pose distribution, success criterion) are held identical, so that APEX is the only variable.

B.3 Evaluation Metrics

We evaluate each method using three metrics: success rate, action-tracking error, and paired outcome changes. The tracking error measures the discrepancy between the action reference and the realized action. The reference is the expert trajectory in replay experiments and the policy output in learned-policy rollouts. For paired outcome changes, we count both cases where APEX turns a failed baseline rollout into a success and cases where it turns a successful baseline rollout into a failure. ManiSkill results report joint-space tracking error in mrad, whereas LIBERO results report end-effector tracking error in millimeters; we therefore summarize them separately. Each ManiSkill learned-policy setting is evaluated with 100 paired episodes, and each LIBERO $\pi_{0.5}$ setting with 20 paired episodes. When computing ManiSkill tracking-error summaries, we exclude rare anomalous steps where the policy command, executed command, or observed joint state exceeds 4 rad. This removes obvious policy-output or simulator failures from the controller-tracking metric, while leaving terminal success evaluation unchanged.

C Implementation Details of APEX

The dynamic filter in Sec. 4 is introduced for theoretical completeness: it provides smooth reference derivatives and yields a clean stability analysis. In the experiments, we use a simpler discrete-time realization that preserves the same adapter role but estimates derivatives by finite differences and uses a re-parameterized set of bounded adaptive coefficients. We found this implementation robust across the replay, learned-policy, and controller-sweep settings.

At each control step, APEX receives the policy-level action reference a_π and the realized action state $(a, \dot{a})$. Using the tracking error $e_a = a_\pi - a$, the adapter sends the corrected command

$$a_r = a_\pi + \hat{\alpha} e_a + \hat{\theta}_v \dot{a} + \hat{\theta}_a \ddot{a}_\pi^{\mathrm{fd}}, \tag{33}$$

where $\ddot{a}_\pi^{\mathrm{fd}}$ is a finite-difference estimate from recent policy references. The adaptive coefficients $\hat{\alpha}$, $\hat{\theta}_v$, and $\hat{\theta}_a$ correspond to position-, velocity-, and acceleration-level correction terms. The coefficients are updated online from tracking feedback, following the same adaptation principle as Sec. 4.

C.1 Hyperparameters

Tab. S1 summarizes the main APEX hyperparameters used in replay and ACT controller-grid experiments. We use a single adaptive preset across these settings. All correction coefficients are initialized from zero, so the

Table S1 Main APEX hyperparameters.

Parameter	Value
$\hat{\alpha}_0$	0
$\hat{\theta}_{v,0}$	0
$\hat{\theta}_{a,0}$	0
γ_α	30
γ_v	0.1
γ_a	0.1
λ_s	30.0
$c_{\max}$	0.10

gains come from online adaptation rather than a tuned warm start. The update gains control the adaptation rates of the position-, velocity-, and acceleration-level terms, λ_s denotes the sliding-variable bandwidth, and $c_{\max} = 0.10$ denotes the maximum absolute command compensation used to clip the final correction for improved robustness.

D Fixed Correction Cannot Replace Test-time Adaptation

This section examines whether the gains of APEX can be attributed to online adaptation rather than to a fixed correction preset. If the correction with constant coefficients were sufficient, a single set of correction coefficients should remain effective when the same reference trajectory is executed under different low-level controller gains.

We conduct this ablation on PegInsertionSide-v1 replay, where the action reference is an expert motion-planning trajectory, and policy prediction error is removed. We vary the test-time proportional gain $K_p \in \{500, 1000, 1500\}$ while keeping $K_d = 100$ fixed. For the fixed-correction variant, all adaptive updates are disabled and the coefficients are swept over

$$\hat{\alpha} \in \{0, 0.2, 0.4, 0.6\}, \quad \hat{\theta}_v \in \{0, 0.05, 0.1, 0.2\}, \quad \hat{\theta}_a \in \{0, 0.01, 0.02, 0.05\}.$$

This yields 64 fixed configurations, each evaluated with 30 paired replay episodes. Adaptive APEX is evaluated from the neutral initialization $\hat{\alpha}_0 = \hat{\theta}_{v,0} = \hat{\theta}_{a,0} = 0$, using $\gamma_\alpha = 30$, $\gamma_a = 0.1$, and $\gamma_v = 0.05$. Thus, any improvement of adaptive APEX must arise from online parameter updates during execution rather than from a tuned warm start.

The sweep shows that fixed correction is only effective within a limited region of the controller-gain space. The best fixed configuration achieves a mean improvement of +16.7 pp across the evaluated K_p values, whereas adaptive APEX reaches +23.3 pp despite starting from zero correction. These results support the claim that APEX benefits from adapting to the current execution dynamics online, rather than applying a transferable fixed correction.

E Physics Perturbation Robustness

We conduct this study on PegInsertionSide-v1 replay with $K_p = 700$, a mildly under-stiffened controller setting where direct replay is possible but unreliable. The expert reference, controller gains, and rollout seeds are kept fixed, and only the joint-level physics parameters are varied. We consider two perturbation families. First, we vary the dimensionless joint Coulomb friction coefficient over $\{0, 5, 10, 15, 20, 25, 30\}$ while keeping armature fixed; this coefficient is the simulator's dry-friction parameter, which bounds the friction torque opposing joint motion in proportion to the joint load, so a larger value imposes proportionally stronger resistance. Second, we vary rotor armature, which changes motor-side inertia, over $\{0, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30\}$ kg·m^2 while keeping friction fixed. The armature sweep covers and extends beyond the typical Franka range of 0.05–0.15 kg·m^2. For each setting, we compare direct replay and APEX using paired rollouts, terminal success, and action-tracking error.

Table S2 Complete fixed-correction sweep on PegInsertionSide replay. Each entry reports the mean success-rate change over $K_p \in \{500, 1000, 1500\}$ in percentage points. All fixed configurations disable online adaptation and vary only the fixed values of the position-, velocity-, and acceleration-level correction coefficients.

$\hat{\alpha}$	$\hat{\theta}_v$	$\hat{\theta}_a = 0$	$\hat{\theta}_a = 0.01$	$\hat{\theta}_a = 0.02$	$\hat{\theta}_a = 0.05$
0.0	0.00	0.0	−10.0	−11.1	−22.2
0.0	0.05	−13.3	−13.3	−13.3	−11.1
0.0	0.10	−37.8	−35.6	−42.2	−52.2
0.0	0.20	−58.9	−56.7	−57.8	−60.0
0.2	0.00	0.0	0.0	0.0	−6.7
0.2	0.05	0.0	−2.2	2.2	−3.3
0.2	0.10	−32.2	−33.3	−33.3	−33.3
0.2	0.20	−62.2	−60.0	−62.2	−60.0
0.4	0.00	10.0	5.6	5.6	1.1
0.4	0.05	5.6	11.1	7.8	8.9
0.4	0.10	−33.4	−32.2	−31.1	−31.1
0.4	0.20	−57.8	−57.8	−60.0	−56.7
0.6	0.00	15.6	**16.7**	13.4	6.7
0.6	0.05	13.3	13.3	12.2	16.7
0.6	0.10	−35.6	−32.2	−32.2	−30.0
0.6	0.20	−57.8	−60.0	−55.6	−57.8

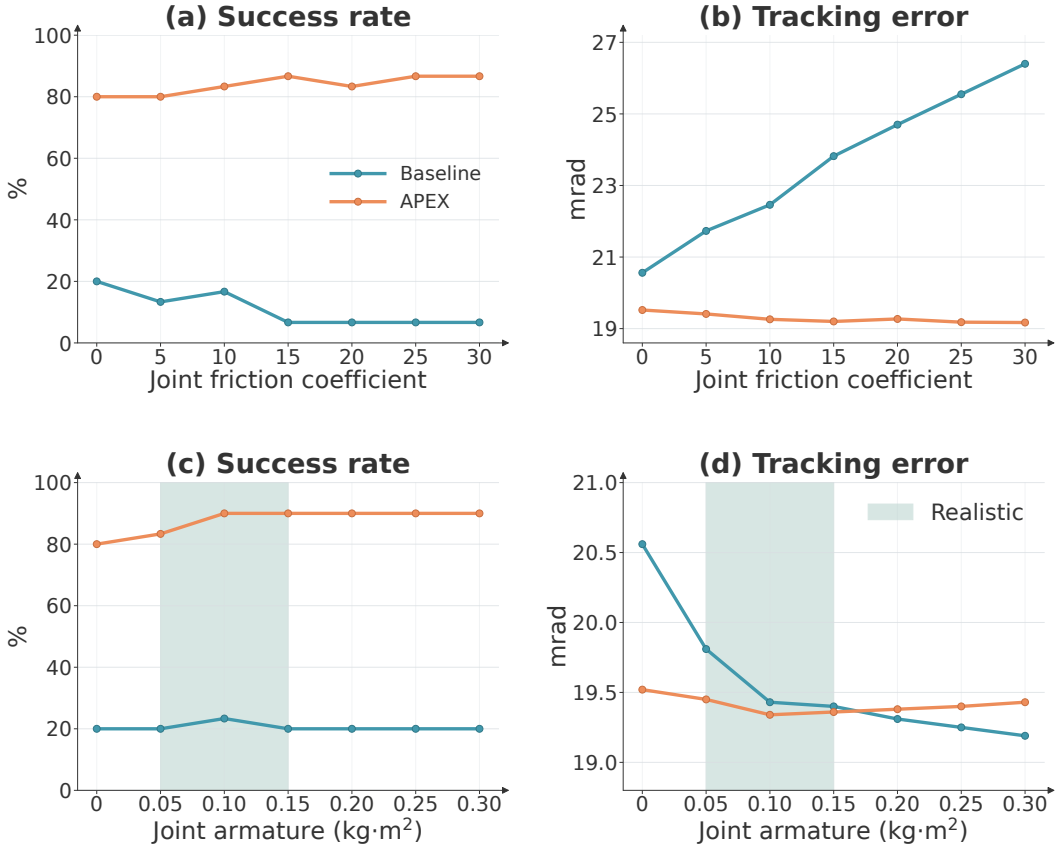

Figure S2 Physics perturbation robustness on PegInsertionSide-v1 replay. Panels (a,b) vary the joint Coulomb friction coefficient; panels (c,d) vary the rotor armature. Baseline is replay without APEX, and the shaded region marks the realistic Franka armature range.

Under increasing friction, direct replay degrades substantially. As the friction coefficient increases from 0 to 30, its success rate drops from 20% to 7%, and its tracking error increases from 20.6 to 26.4 mrad. In

contrast, APEX maintains 80–87% success rate, with tracking error remaining between 19.2 and 19.5 mrad. Consequently, the success gain of APEX increases from +60 to +80 pp as friction becomes more severe.

The armature perturbation exhibits a different failure mode. Mean tracking error changes only mildly for both direct replay and APEX, yet the terminal outcomes remain clearly separated. Direct replay achieves only 20–23% success across the sweep, whereas APEX maintains 80–90% success, including all settings within the typical Franka armature range.

These results indicate that APEX is not merely tuned to the default simulator dynamics. When friction increases, APEX prevents the tracking-error degradation observed under direct replay. When motor-side inertia changes, APEX preserves high task success even though the mean tracking error is less sensitive to this parameter. Within the PegInsertion replay setting, APEX therefore remains robust to the tested perturbations in joint friction and rotor armature.

F ACT ManiSkill Results

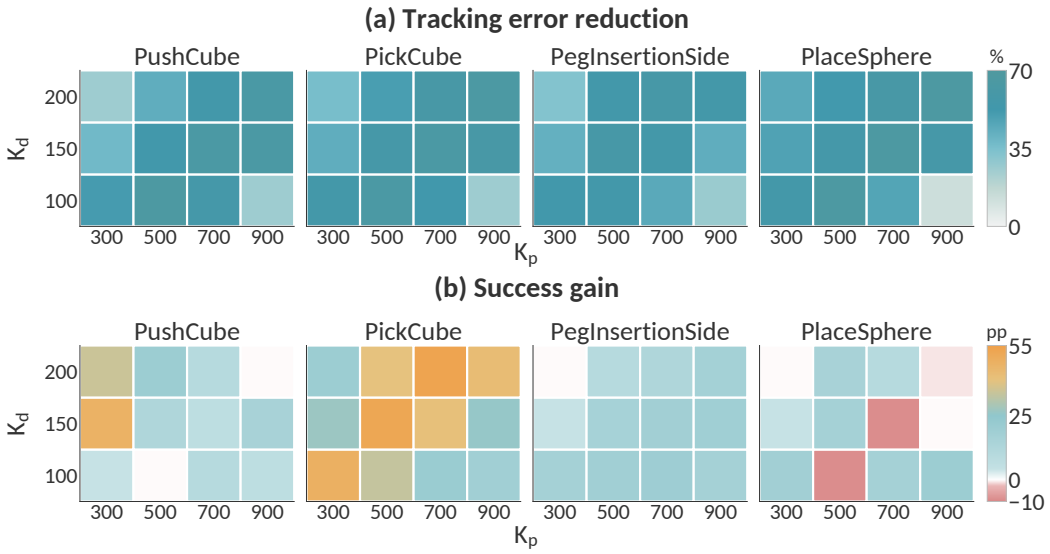

Figure S3 Full ACT performance grid. **(a)** Tracking-error reduction in percentage. **(b)** Success-rate gain in percentage points. Each heatmap sweeps K_p and K_d for one task.

Fig. S3 provides the complete ACT controller-gain sweep corresponding to Sec. 5.2. For each task, the policy is fixed and only the low-level PD gains are varied. Each setting is evaluated with 100 paired rollouts using matched initial seeds for direct execution and APEX. The heatmaps sweep K_p along the horizontal axis and K_d along the vertical axis, reporting tracking-error reduction in the top row and success-rate gain in the bottom row. Across the 48 task-controller settings, APEX improves the mean success rate from 27.4% to 40.6%. It increases success in 40 settings, leaves 5 unchanged, and decreases success in only 3, showing that the correction is broadly effective across controller-gain mismatch rather than tuned to a single configuration.

G $\pi_{0.5}$ LIBERO Results

Fig. S4 provides the complete $\pi_{0.5}$ controller-gain sweep corresponding to Sec. 5.3. For each LIBERO Spatial task, the policy is fixed and only the proportional-gain scale of the low-level controller is varied. Each setting is evaluated with 20 paired rollouts using matched initial seeds for direct execution and APEX. The curves sweep the K_p scale, reporting task success in the top row and end-effector tracking error in the bottom row. Here, tracking error is measured as the positional deviation between the target and realized end-effector positions during policy rollout.

The evaluated LIBERO Spatial tasks are listed in Tab. S3. Across the 12 task-controller settings, APEX improves the mean success rate from 72.1% to 97.9%. It increases success in 11 settings and leaves 1 unchanged, indicating that the execution-side correction remains effective for a VLA policy with end-effector delta actions and inverse-kinematics-based execution.

Table S3 $\pi_{0.5}$ LIBERO Spatial tasks used in the controller-gain sweep.

Task	Language instruction
1	Pick up the black bowl between the plate and the ramekin and place it on the plate.
2	Pick up the black bowl next to the ramekin and place it on the plate.
3	Pick up the black bowl from the table center and place it on the plate.

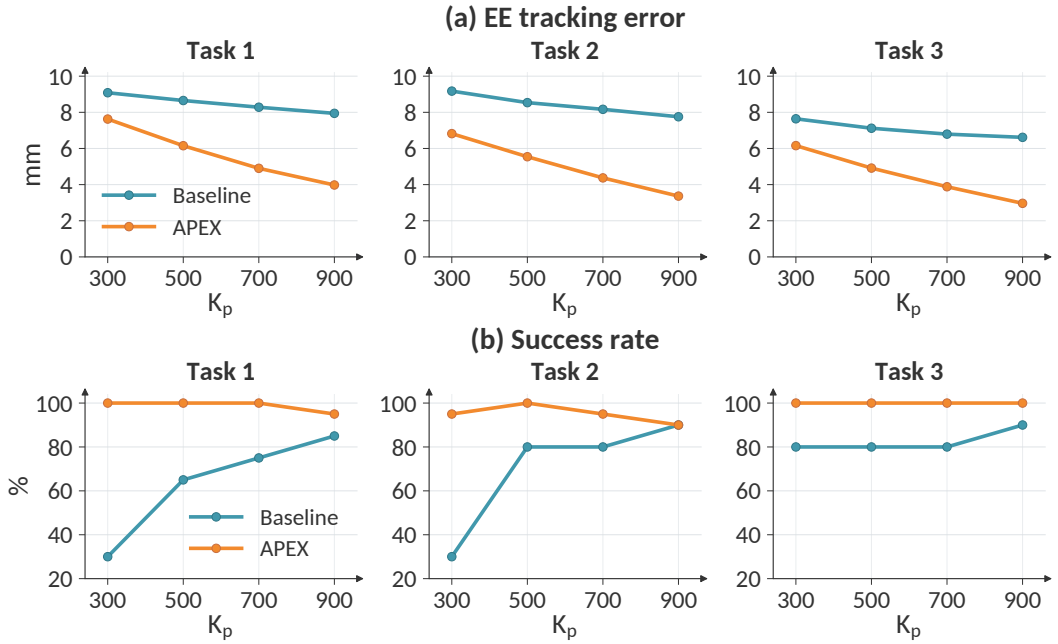

Figure S4 Full $\pi_{0.5}$ performance grid on LIBERO Spatial tasks. **(a)** End-effector tracking error. **(b)** Task success rate. Each column corresponds to one task, and each curve sweeps the K_p scale under the same fixed policy.