PACE: Phase-Aware Chunk Execution for Robot Policies with Action Chunking

Junnan Nie[1,2], **Jiayi Li**[2], **Chenghao Liu**[1], **Junyi Lao**[1],
Jiachen Zhang[1], **Tianle Zhang**[2], **Songfang Huang**[1✉]
[1]Peking University [2]JD Explore Academy

Abstract

Action-chunking robot policies predict sequences of future actions, but deployment requires choosing how many actions to execute before replanning. The execution horizon trades off feedback frequency against motion continuity. However, success varies non-monotonically with the execution horizon, and the best-performing horizon differs across tasks. These effects make a single fixed horizon unreliable as a default execution strategy across tasks. We propose PACE (Phase-Aware Chunk Execution), a training-free method that selects execution horizons online from predicted action chunks. PACE uses prominent low-speed valleys in the predicted speed profile as candidate replanning boundaries, exploiting the phase-dependent kinematics of manipulation trajectories. By default, PACE calibrates the valley threshold from each task's existing training demonstrations, without task-specific fixed-horizon tuning. On the 50-task RoboTwin2.0 benchmark, PACE improves average success from 57.8% to 64.2% over the strongest fixed-horizon baseline and outperforms the adaptive execution baselines AAC and AutoHorizon. With GR00T N1.7 on LIBERO, PACE achieves the highest overall success rate among the evaluated strategies, supporting its applicability across base policies, benchmarks, and action representations. On real robots, PACE improves the average task score from 60.7 to 77.7 and success rate from 50.7% to 70.4% over full-chunk execution. Code is available on our project page.

1 Introduction

Recent progress in vision-language-action models and diffusion-based robot policies has largely come from training-side gains. Larger backbones, broader demonstration data, and stronger multi-modal pretraining have all improved predictive performance [1–7]. At deployment, however, performance depends not only on how accurately a policy predicts, but also on how those predictions are executed. To improve local motion continuity, these policies often adopt action chunking [4, 8], where each query predicts a short sequence of future actions rather than a single-step command.

Yet an action chunk only specifies what the policy predicts, not how much of that prediction the robot should execute. Its length fixes the prediction horizon, but a separate choice must still be made about how many actions to execute before acquiring a new observation and querying the policy again. We refer to this executed prefix length as the *execution horizon*. Unlike the prediction horizon, the execution horizon is neither learned nor prescribed by the policy [4, 8]. It is an inference-time choice that trades off feedback frequency against within-chunk motion continuity. As such, the execution horizon is not a mere implementation detail, but a central variable for reliable deployment of action-chunking robot policies.

✉Corresponding author: Songfang Huang (hsf@pku.edu.cn).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

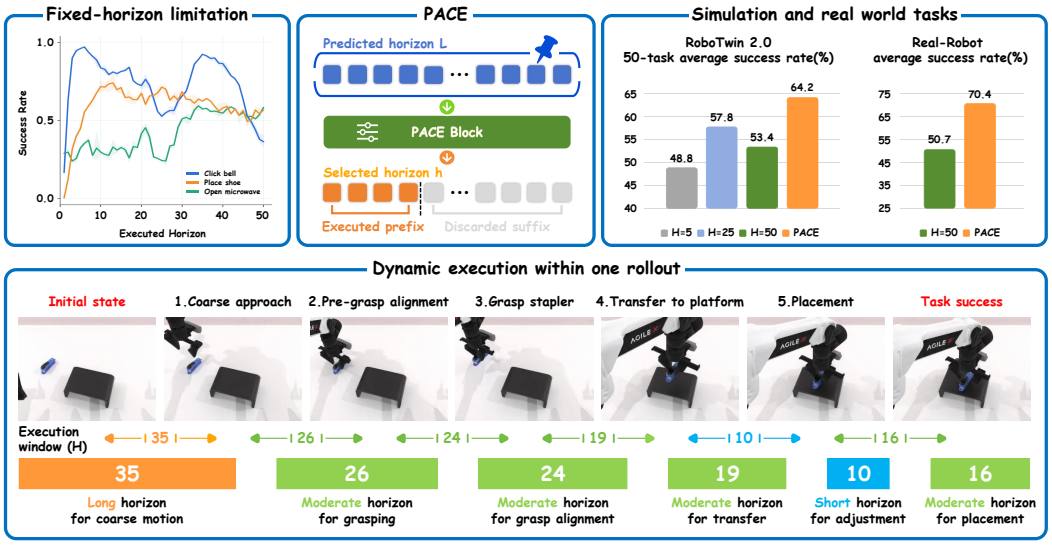


Figure 1: **Overview of PACE.** Fixed horizons can be unreliable because success varies with H across tasks. PACE selects the executed prefix online from each predicted chunk, improving performance in simulation and real-robot experiments while adapting the horizon in a rollout.

In practice, the execution horizon is typically fixed to a constant H throughout an episode [4, 8]. At each replanning step, the robot executes the first H actions of the predicted chunk before querying the policy again, tying all chunks to the same replanning interval regardless of the motion they predict. A smaller H incorporates new observations more often but can interrupt a coherent local motion, while a larger H follows the predicted chunk longer but delays reactive correction. Without a principled selection criterion, H is usually chosen empirically.

Sweeping H from 1 to 50 on three RoboTwin2.0 tasks reveals strongly non-monotonic success rates and task-dependent preferred horizons (Fig. 1, left). On Click Bell, success peaks near $H \approx 6$, drops by nearly 40 points around $H \approx 25$, and rebounds to a second peak near $H \approx 35$. A fixed horizon can therefore cause substantial performance degradation on some tasks. The best horizon found offline on one task also does not transfer to new tasks or real-robot deployment. These findings indicate that the execution horizon must be selected online rather than fixed in advance.

We propose PACE (Phase-Aware Chunk Execution), a test-time method that selects the execution horizon online from the speed profile of each predicted action chunk. Manipulation trajectories consist of locally coherent motion segments, with transitions often accompanied by deceleration. Policies trained via imitation learning preserve this structure in their predicted action chunks, where phase transitions can appear as low-speed valleys in the speed profile. PACE treats such valleys as candidate replanning points and executes the chunk up to the earliest accepted valley, aiming to preserve coherent motion within a segment and re-query the policy before the next begins. PACE uses no model-internal signals or learned components and requires no policy changes or retraining.

On the 50-task RoboTwin2.0 benchmark [9], PACE improves the average success rate of $\pi_{0.5}$ from 57.8% to 64.2% over the best fixed-horizon baseline and outperforms the adaptive baselines AAC and AutoHorizon. On real robots, it improves average task score from 60.7 to 77.7 and success rate from 50.7% to 70.4% over full-chunk execution. With GR00T N1.7 on LIBERO, PACE achieves the highest overall success rate among the evaluated strategies, supporting its applicability across policies, benchmarks, and action representations. Ablations, sensitivity analyses, and fixed-horizon sweeps on representative tasks assess its design choices, while analyses of successful and failed rollouts reveal its working mechanism and failure modes. Our contributions are summarized as follows:

- We identify the execution horizon as a critical yet overlooked test-time variable, and show empirically that its effect on success is both task-dependent and non-monotonic, indicating that no single fixed horizon provides a reliable default across tasks.

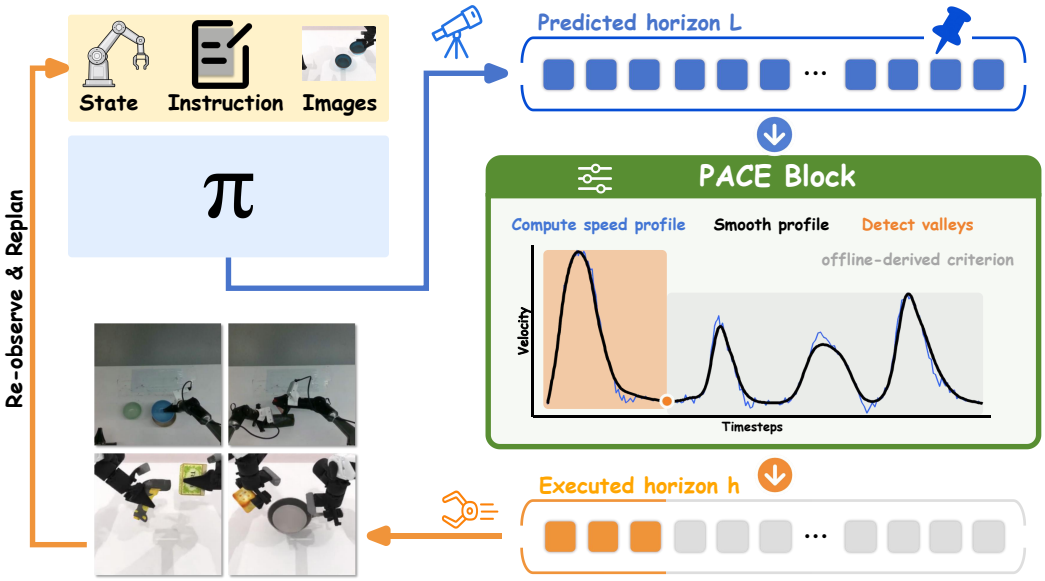


Figure 2: **PACE framework.** PACE selects an execution horizon from low-speed valleys in the predicted chunk's smoothed speed profile. The robot executes the selected prefix, discards the suffix, and then queries the policy again.

- We propose PACE, a plug-and-play test-time method that selects the executed prefix online from the phase-dependent kinematic structure of predicted action chunks. It requires no model-internal signals, no learned module, and no modification or retraining of the policy.
- We validate PACE with $\pi_{0.5}$ on all 50 RoboTwin2.0 tasks, with GR00T N1.7 on LIBERO, and on real robots. Comparisons with fixed-horizon and adaptive baselines demonstrate its effectiveness, while ablations and rollout analyses explain its performance gains.

2 Related work

Robot policies with action chunking. Action chunking has become a common interface in robot policy learning. It was popularized by ACT for fine-grained bimanual imitation learning [8] and Diffusion Policy for receding-horizon action-sequence generation [4]. Subsequent work applies this interface to mobile manipulation [10], 3D visuomotor control [11], and large-scale bimanual policy learning [12]. Generalist policies such as Octo, π_0, and $\pi_{0.5}$ extend chunk prediction to diverse tasks and environments [3, 13, 14]. VLA research also explores efficient action tokenization [15] and faster inference for action prediction [16, 17]. While these works focus on generating, representing, or scaling action chunks, PACE addresses the complementary deployment-time question of how much of each predicted chunk to execute before incorporating new observations and replanning.

Test-time execution of action chunks. Many policies execute action chunks using fixed test-time rules: ACT smooths overlapping predictions through temporal ensembling, while Diffusion Policy executes a fixed action horizon in a receding-horizon loop [4, 8]. Recent inference-time methods reduce latency-induced discontinuities or stabilize chunk execution [18, 19]. Another line modifies training or model design to reason over multiple horizons [20]. More closely related are methods that adapt the execution horizon at inference time. Adaptive Action Chunking (AAC) [21] estimates prefix-wise action entropy from multiple sampled chunks and selects the horizon subject to a minimum-movement constraint. AutoHorizon [22] estimates the execution horizon from action-token self-attention in flow-based VLAs, treating attention as a proxy for the model's predictive limit. PACE instead selects replanning boundaries from prominent low-speed valleys in the kinematic profile of a single decoded action chunk, without requiring model-internal signals or additional chunk samples.

3 Method

3.1 Problem formulation

We consider a manipulation policy with action chunking. At the i-th policy query, the policy receives the observation o_{τ_i} and the language instruction ℓ, and outputs

$$\mathbf{A}_i = \pi_\theta(o_{\tau_i}, \ell) \in \mathbb{R}^{L \times d_a}, \qquad \mathbf{A}_i = (a_{i,1}, \ldots, a_{i,L}), \tag{1}$$

where L is the prediction horizon and d_a is the action dimension. Let $h_i \in \{1, \ldots, L\}$ denote the execution horizon, i.e., the number of predicted actions executed before the next policy query. After executing the prefix $(a_{i,1}, \ldots, a_{i,h_i})$, the next query occurs at $\tau_{i+1} = \tau_i + h_i$.

A fixed-horizon rule sets $h_i^{\text{fixed}} = H$ for all i, whether the predicted chunk describes a smooth transit or a contact event. PACE instead conditions the execution horizon on the predicted chunk,

$$h_i = g(\mathbf{A}_i), \qquad g : \mathbb{R}^{L \times d_a} \to \{1, \ldots, L\}. \tag{2}$$

The deterministic, training-free function g uses fixed hyperparameters (Secs. 3.2–3.3) without modifying the trained policy π_θ. The prediction horizon L remains the policy's training horizon (Sec. 4.3).

3.2 Kinematic profile extraction

Expert manipulation trajectories are rarely uniform in speed. They organize into alternating fast and slow segments, where low-speed regions typically arise under contact, stabilization, or sub-stage transitions. An imitation-trained policy tends to preserve this structure in its predicted action chunks. This makes the speed profile a kinematic cue for where one locally coherent motion segment ends and the next begins, and thus a natural basis for the selection rule g.

Concretely, for each controllable arm group $b = 1, \ldots, B$, let $q_{i,j}^b \in \mathbb{R}^{d_q}$ denote the joint command at predicted step j. We define the speed profile $\mathbf{v}_i^b = (v_{i,1}^b, \ldots, v_{i,L-1}^b)$ by

$$v_{i,j}^b = f_s \, \|q_{i,j+1}^b - q_{i,j}^b\|_2, \qquad j = 1, \ldots, L-1, \tag{3}$$

where f_s is the control frequency. Raw speed profiles contain high-frequency fluctuations that create spurious local valleys and, if untreated, trigger overly frequent replanning. We therefore apply a Savitzky–Golay filter $\mathcal{S}$ to obtain the smoothed profile $\tilde{\mathbf{v}}_i^b = \mathcal{S}(\mathbf{v}_i^b)$, whose low-speed valleys serve as candidate replanning boundaries for the selection rule in Sec. 3.3. We use joint-space speed by default; a Cartesian variant yields comparable success after smoothing (Sec. 4.3), and filter parameters are given in Appendix A.

3.3 Phase-aware execution horizon selection

Given the kinematic profile from Sec. 3.2, we now ask where inside the chunk the policy should stop and replan. Our guiding observation, illustrated in Fig. 2, is that phase boundaries such as contacts, grasps, and releases can manifest as pronounced low-speed valleys in $\tilde{\mathbf{v}}_i^b$: the end-effector decelerates as it approaches the event and accelerates again once it is resolved. The function g therefore searches for the earliest such valley and treats it as a candidate point to hand control back to the policy.

Not every local minimum qualifies: a valley is informative only when preceded by a clearly faster segment, otherwise it merely reflects noise on an already slow trajectory. Let $\mathcal{V}_i^b \subseteq \{1, \ldots, L-1\}$ denote the local minima of $\tilde{\mathbf{v}}_i^b$ satisfying a minimum spacing constraint. For each candidate $r \in \mathcal{V}_i^b$ we define the preceding maximum

$$m_i^b(r) = \max_{1 \le j \le r} \tilde{v}_{i,j}^b, \tag{4}$$

and the contrast score

$$\Delta_i^b(r) = \min\{\, m_i^b(r) - \tilde{v}_{i,1}^b,\ m_i^b(r) - \tilde{v}_{i,r}^b \,\}, \tag{5}$$

which requires the speed to both rise substantially before the valley and drop substantially into it, so that r sits at the bottom of a genuine deceleration rather than on a flat region. A valley is accepted when $\Delta_i^b(r) \ge \delta_{\mathcal{T}}$, where $\delta_{\mathcal{T}}$ is calibrated once per task from offline training trajectories using a fixed percentile rule shared across tasks.

Table 1: **RoboTwin2.0 results with $\pi_{0.5}$.** Success rates (%) on six representative tasks, grouped by task duration, and all 50 tasks. Overall is the mean success rate over all 50 tasks. Bold marks the highest mean in each column. Red values after the slash report PACE improvements over the best of $H = 5, 25, 50$ in each column (percentage points, computed before rounding).

Method	Short		Medium		Long		Overall (50 tasks)
	Pick Bottles	Grab Roller	Place A2B	Dump Bin	Put Bottles	Stack Bowls	
$H = 5$	13.3	89.9	20.0	58.7	0.3	88.7	48.8
$H = 25$	45.3	96.2	47.8	85.1	79.3	88.4	57.8
$H = 50$	38.7	96.2	36.8	84.3	70.8	85.1	53.4
AutoHorizon	41.6	96.9	43.1	88.1	74.3	91.0	59.0
AAC	**48.9**	97.7	46.7	89.1	76.6	88.2	59.1
PACE (Ours)	48.2 / +2.9	**98.6** / +2.3	**54.6** / +6.8	**93.0** / +7.9	**88.9** / +9.6	**95.6** / +6.9	**64.2** / +6.4

Accepted valleys from all arm groups are pooled into

$$\mathcal{R}_i = \bigcup_{b=1}^{B} \{\, r \in \mathcal{V}_i^b \;:\; \Delta_i^b(r) \geq \delta_{\mathcal{T}} \,\}. \tag{6}$$

We take the union because a candidate phase transition on any single arm is already a reason to replan, and we select the earliest candidate, aiming to query the policy before, rather than after, a boundary is crossed with stale observations. When no valley is accepted, PACE falls back to the maximum execution horizon, giving

$$h_i = g(\mathbf{A}_i) = \begin{cases} \min \mathcal{R}_i, & \mathcal{R}_i \neq \emptyset, \\ H_{\max}, & \mathcal{R}_i = \emptyset, \end{cases} \tag{7}$$

where $H_{\max} \leq L$ is the execution horizon used when no valley is accepted. Calibration of $\delta_{\mathcal{T}}$, the valley-spacing constant, and the value of $H_{\max}$ are deferred to Appendix A.

4 Simulation experiments

Experimental setup. We evaluate PACE on 50 RoboTwin2.0 bimanual manipulation tasks, training one $\pi_{0.5}$ checkpoint per task under the standard protocol. All methods share the same checkpoint and differ only in test-time execution. Unless otherwise noted, our *full evaluation protocol* uses 900 episodes per task (three seeds, each with three batches of 100 episodes), with success rates averaged over all 50 tasks. Task-level tables show six representative tasks balanced across short-, medium-, and long-duration groups. Full results and training details are in Appendix B.

Overview of experiments. We compare PACE as a default execution strategy with fixed-horizon and adaptive baselines (Sec. 4.1) and analyze its horizon adaptation within rollouts (Sec. 4.2). We ablate the phase profile, training prediction horizon, and threshold sharing (Sec. 4.3), and use task-specific fixed-horizon sweeps as a diagnostic upper bound (Sec. 4.4).

4.1 Simulation results

We compare PACE against fixed-horizon execution with $H \in \{5, 25, 50\}$, corresponding to short, intermediate, and full-chunk prefixes (recall H_{train}=50). We also evaluate Adaptive Action Chunking (AAC) [21] and AutoHorizon [22] using the same task-specific $\pi_{0.5}$ checkpoints. Our bimanual joint-space adaptation of AAC estimates action entropy from 20 sampled chunks per policy query. PACE calibrates one threshold per task from the checkpoint's existing training demonstrations, without using evaluation rollouts, success labels, or a fixed-horizon search (Appendix A.3).

Table 1 shows that PACE achieves the highest 50-task average, exceeding the best global fixed horizon ($H = 25$) by 6.4 points and AAC and AutoHorizon by approximately 5.1 and 5.2 points, respectively. PACE also exceeds the strongest fixed horizon on every displayed task and achieves the highest mean on five of the six. AAC leads on Pick Bottles (48.9% versus 48.2%).

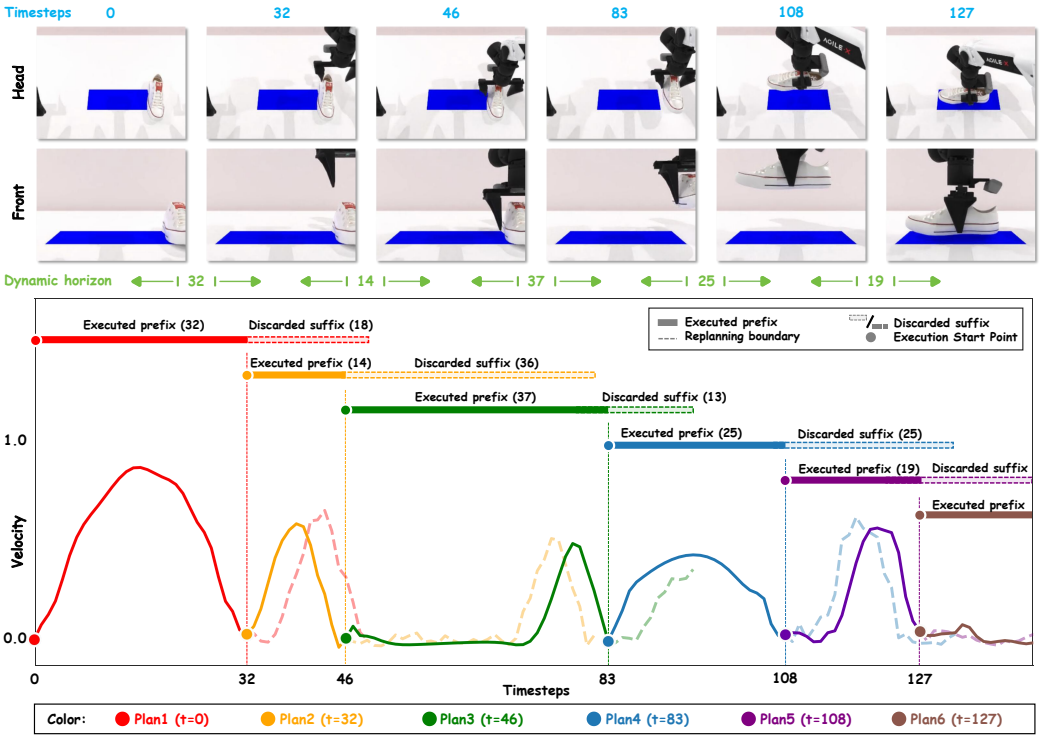


Figure 3: **Rollout-level behavior of PACE.** The rollout is from `place_shoe`. The top rows show head and front camera observations at the six replanning timesteps ($t = 0, 32, 46, 83, 108, 127$), with the selected execution horizon annotated between consecutive queries. The bottom panel shows the six predicted action chunks along the rollout timeline, one color per plan. For each plan, the horizontal bar marks the executed prefix (solid) and discarded suffix (dashed), and the curve below shows the corresponding predicted velocity profile under the same solid/dashed convention. Circles mark the execution start point of each plan, and vertical dashed lines mark the replanning boundaries.

Cross-policy evaluation. We evaluate PACE with GR00T N1.7 [23] on all four LIBERO suites [24], covering 40 tasks. GR00T predicts 16-step Cartesian-delta chunks. Without retraining or modifying GR00T, PACE constructs its motion profile from the translational-delta norm and applies the same valley-based selection rule. PACE achieves the highest overall success rate, 95.33%, compared with 94.47% for the strongest global fixed horizon ($H = 16$) and 94.92% for AutoHorizon. These results support the applicability of PACE across different base policies, benchmarks, action representations, and chunk lengths. The horizon selector takes 0.0566 ms per query, approximately 0.044% of the time required for a single policy forward pass (Appendix B.2).

4.2 Visualizing adaptive execution horizons

We visualize a representative `place_shoe` rollout in Fig. 3 to examine how PACE adapts its execution horizon. Selected horizons range from 14 to 37 steps, with longer horizons covering coherent motion (the approach in Plan 3, 37 steps) and shorter horizons near low-speed transitions (Plan 2, 14 steps). A fixed horizon cannot adapt to this within-episode structure.

The discarded suffixes show how PACE limits open-loop execution as the scene changes. PACE replans at boundaries selected from the predicted motion structure, and the new plan often differs from the discarded suffix over the overlapping interval, as seen in the mismatched solid and dashed curves of adjacent plans around t=83. This limits reliance on stale observations while preserving long horizons for coherent motion, helping explain the gains over fixed-horizon baselines in Sec. 4.1. Appendix D.3 quantifies the alignment of replanning boundaries with motion-phase transitions.

Table 2: **Ablation studies. Left:** phase-profile ablation on all 50 RoboTwin2.0 tasks; Avg. Horizon is the mean selected execution horizon. **Right:** training-horizon ablation on six representative tasks; each row fixes H_{eval} and varies H_{train}. Gain@50 is the gain of H_{train}=50 over the shortest feasible setting. Success rates are in %. † marks the default.

Configuration	Succ.	Avg. Horizon
Cartesian, raw	62.2	11.3
Cartesian, smoothed	64.5	19.5
Joint, raw	60.4	16.7
Joint, smoothed†	64.2	24.3

$H_{\text{eval}} \backslash H_{\text{train}}$	10	20	30	40	50	Gain@50
10	57.2	68.7	69.9	74.6	77.9	**+20.7**
20	–	68.7	72.6	73.8	75.3	**+6.6**
30	–	–	70.2	71.4	73.8	**+3.6**
40	–	–	–	70.1	73.4	**+3.3**

4.3 Ablation analysis

We examine how motion-profile construction, the training prediction horizon, and threshold sharing affect PACE's performance. We evaluate motion-profile variants and compare shared versus per-task thresholds on all 50 RoboTwin2.0 tasks, with unchanged policy checkpoints. To study the training prediction horizon H_{train}, we train policies to predict action chunks of different lengths. We compare their success rates at the same fixed execution horizon to assess whether supervision over longer action sequences improves the quality of the executed prefix. Since each setting requires retraining, we evaluate them on six representative tasks. Further details and results appear in Appendix C.

Phase-profile construction. We compare joint-space and Cartesian speed profiles, with and without smoothing, to examine whether PACE depends on a specific motion representation and how smoothing affects boundary selection. The Cartesian variant computes end-effector linear speed through forward kinematics. We keep the valley-selection rule and calibration percentile fixed across all four variants. Table 2 (left) shows that smoothing improves success in both spaces and increases the mean selected execution horizon. This pattern suggests that smoothing suppresses noisy local valleys that would otherwise trigger overly frequent replanning. After smoothing, Cartesian and joint-space profiles achieve comparable success, while the joint-space variant selects longer execution horizons on average. We adopt the smoothed joint-space profile by default. It is within 0.3 points of the best variant, matches the policy action space, and avoids forward kinematics at every replanning query.

Training prediction horizon. Each row of Table 2 (right) fixes H_{eval} and varies H_{train}; pairs with $H_{\text{train}} < H_{\text{eval}}$ are infeasible because the policy predicts too few actions. Success increases monotonically with H_{train}, and the gain of H_{train}=50 over the shortest feasible target shrinks from $+20.7$ at H_{eval}=10 to $+3.3$ at H_{eval}=40. This trend shows that longer training targets provide stronger sequence-level supervision: they constrain not just the action after the observation, but how motion evolves across a longer expert segment. Since the executed prefix is predicted jointly with the rest of the chunk, it inherits this structure. We adopt H_{train}=50 as the default prediction horizon.

Table 3: **Shared versus per-task threshold calibration.** Success rates (%), reported as mean ± sample SD across three evaluation seeds.

Execution	Threshold calibration	Success (%)
Fixed $H = 25$	Not applicable	57.80 ± 1.49
PACE, shared	Pooled training demonstrations	62.44 ± 0.92
PACE, per-task	Each task's training demonstrations	$\mathbf{64.20} \pm 0.88$

Shared versus per-task thresholds. By default, PACE calibrates a separate valley-acceptance threshold from each task's training demonstrations. To assess how much PACE depends on per-task calibration, we pool the training demonstrations from all 50 tasks to calibrate a single threshold and apply it unchanged across tasks. All other PACE settings and evaluation conditions remain unchanged. The shared threshold governs valley acceptance, while execution horizons still adapt to the speed profile of each predicted action chunk. Table 3 shows that the shared threshold retains most of PACE's gain over fixed $H = 25$, improving success by 4.64 percentage points. Per-task calibration provides a further gain of approximately 1.8 points and remains our default (Appendix C.4).

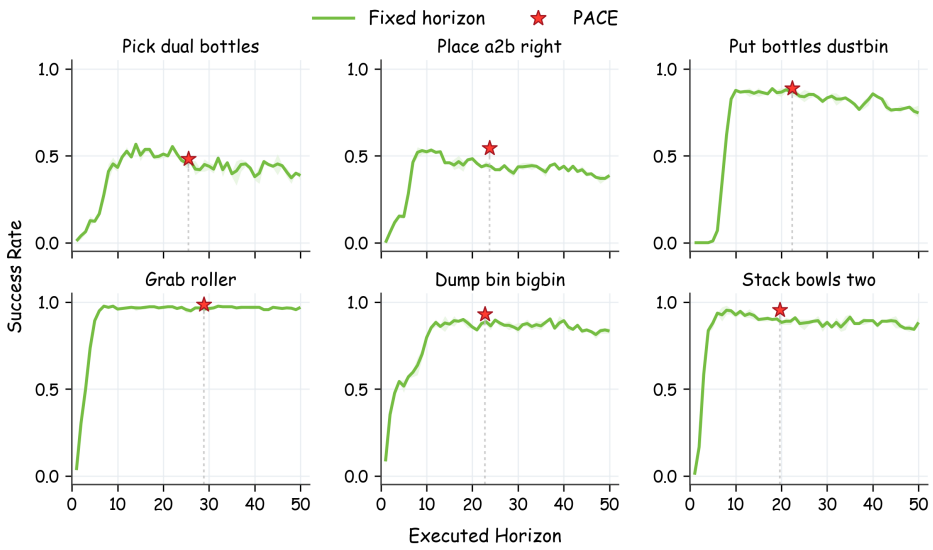


Figure 4: **PACE compared with fixed-horizon sweeps.** Each panel shows one of the six representative tasks. The green curve reports success as the fixed execution horizon H varies from 1 to 50. The red star marks PACE's success rate at its mean selected execution horizon.

4.4 PACE versus fixed-horizon sweeps

We compare PACE with task-specific fixed-horizon sweeps on six representative tasks. For each task, we evaluate $H = 1, \ldots, 50$ to obtain the success-rate curve and the best observed fixed-horizon performance. These results serve as diagnostic references for evaluating PACE, which selects execution horizons online without task-specific fixed-horizon search.

Figure 4 shows that the success curves and preferred horizons vary substantially across tasks. For example, the best fixed horizon is $H = 8$ on Stack Bowls Two but $H = 37$ on Dump Bin Bigbin. PACE reaches the high-success region of each task without task-specific fixed-horizon tuning, with mean execution horizons ranging from 19.7 to 28.9 steps. These results are achieved without uniformly short execution prefixes.

Table 4: **Matched-horizon comparison on RoboTwin2.0.** For each task, the fixed horizon is obtained by rounding PACE's mean execution horizon to the nearest integer. Red values after the slash report PACE's success-rate gains over the matched baseline.

Setting	Short		Medium		Long	
	Pick Bottles	Grab Roller	Place A2B	Dump Bin	Put Bottles	Stack Bowls
Fixed H	26	29	24	23	22	20
PACE mean H	25.5	28.9	23.8	22.7	22.3	19.7
Fixed (%)	46.7	96.0	44.0	88.7	87.0	88.3
PACE (%)	**48.2** / +1.6	**98.6** / +2.6	**54.6** / +10.6	**93.0** / +4.3	**88.9** / +1.9	**95.6** / +7.2

Matched execution horizons. To test whether PACE's performance can be reproduced by a constant execution length, we round its mean horizon on each task to the nearest integer and evaluate that fixed horizon (Appendix D). Despite similar mean horizons, the baseline replans at fixed intervals, whereas PACE selects a horizon from each predicted chunk. Table 4 shows higher observed success for PACE on all six tasks. On Place A2B Right, PACE achieves 54.6% success with a mean horizon of 23.8 steps, versus 44.0% for fixed $H = 24$. These results suggest that matching the mean horizon alone is insufficient to reproduce PACE's performance.

Table 5: **Real-robot results.** Task scores and full-task success rates (%) on RoboChallenge and the Franka task. Red values after the slash indicate gains over fixed H = 50.

Method	Stack Bowls		Put Pen in Case		Place Object on Plate	
	Score	Succ.	Score	Succ.	Score	Succ.
Baseline	73.0	70.0	37.2	10.0	72.0	72.0
PACE	**93.5** / +20.5	**90.0** / +20.0	**51.7** / +14.5	**33.3** / +23.3	**88.0** / +16.0	**88.0** / +16.0

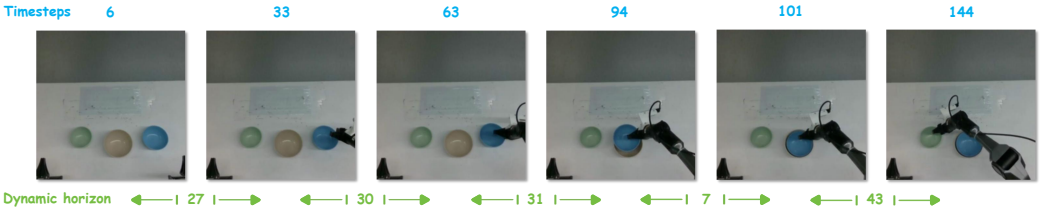


Figure 5: **Successful rollout.** The rollout is from `stack_bowls`. Blue: policy-query timestep. Green: horizon length. The horizon is long during approach and transport (27, 30, 31), short at contact-sensitive alignment (7), and long again afterward (43).

5 Real-robot experiments

We evaluate PACE on real robots in two settings: RoboChallenge [25] and our in-lab `place_object_on_plate` task family. RoboChallenge is a public real-robot benchmark; we evaluate `stack_bowls` and `put_pen_into_pencil_case` with 30 trials per method per task. In `place_object_on_plate`, a Franka robot places one of five objects onto a plate in our laboratory setup. Within each task, all methods use the same fine-tuned $\pi_{0.5}$ checkpoint and differ only in the test-time execution scheme. We report task Score and full-completion rate Succ. for both settings; task definitions, scoring rules, training details, and per-object in-lab results appear in Appendix E.

5.1 Real-robot results

Table 5 compares PACE with the full-chunk execution baseline ($H = 50$) on the three real-robot tasks. PACE improves both task score and success rate on every task, raising the average success rate from 50.7% to 70.4% and the average task score from 60.7 to 77.7, with each task weighted equally.

To assess whether more frequent replanning with a fixed horizon yields similar performance, we further evaluate $H = 5$ and $H = 25$ on the Franka task. PACE achieves an 88% success rate, compared with 87% for $H = 5$ and 82% for $H = 25$. It thus attains a success rate comparable to $H = 5$ while requiring 84.6% fewer policy queries per successful episode (Appendix E.3).

5.2 PACE horizon selection in a successful rollout

Figure 5 visualizes a successful `stack_bowls` rollout and shows that PACE adjusts its execution horizon according to the local manipulation phase rather than a fixed replanning cadence. During approach and transport, the method selects long horizons (27, 30, 31 actions), allowing the robot to maintain continuous motion while open-loop execution remains reliable. Near contact-sensitive alignment, where small pose errors determine whether the bowl stacks, the horizon contracts to 7 actions so the next query can incorporate an updated observation. Once alignment is resolved, the horizon expands again to 43, showing that the method does not remain in a high-frequency replanning mode when a coherent motion segment becomes available.

This behavior mirrors the simulation analysis in Sec. 4.2 and helps explain the real-robot gains in Table 5: PACE introduces feedback where stale observations are most costly, while avoiding unnecessary queries during smooth motion. Additional rollout visualizations for representative RoboTwin2.0 and RoboChallenge tasks are provided in Appendix F.

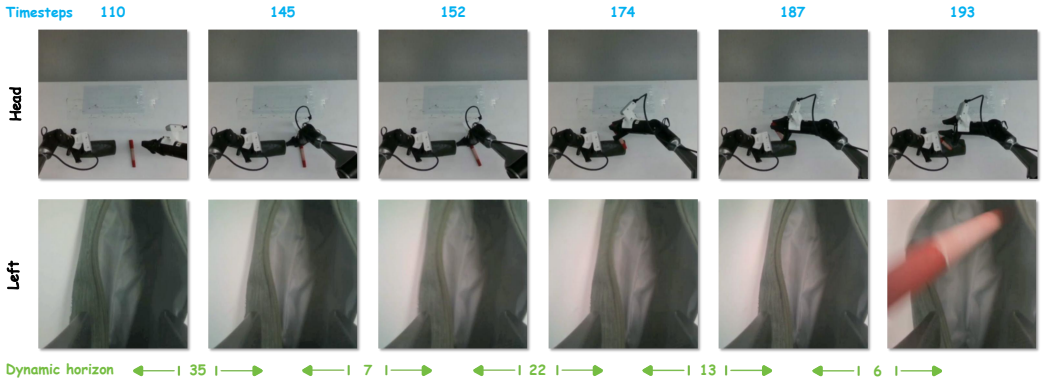


Figure 6: **Failure case.** The rollout is from `put_pen_into_pencil_case` (top: head view; bottom: left-wrist view). Blue: policy-query timestep. Green: horizon length. The pencil case remains only partially opened while the right arm moves the pen toward it.

5.3 Failure case and scope of test-time execution control

Figure 6 shows a failed `put_pen_into_pencil_case` rollout. The task is strictly sequential: the left gripper must fully open the pencil case before the right arm inserts the pen. In this rollout, PACE selects short horizons (7 and 6 actions) around the insertion phase, so the policy refreshes its action chunk multiple times from updated observations. Yet the pencil case remains only partially opened throughout the segment, and the task fails.

The failure is therefore not due to feedback timing, but to the learned action distribution of the base policy. PACE already shortens the execution horizon near insertion and refreshes the chunk from updated observations, but the refreshed chunks are still produced by the same policy and do not yield task-completing behavior in this rollout. This case highlights the boundary of test-time execution control: it can decide when to query the policy, but it cannot compensate when the policy itself fails to generate actions sufficient for task completion.

6 Limitations

PACE cannot resolve failures when the base policy continues to predict incomplete or incorrect actions (Sec. 5.3). Its replanning rule can miss phase transitions without clear low-speed valleys. If no valley is accepted, PACE falls back to the maximum execution horizon. Effectiveness with policy families and action representations beyond those evaluated here remains to be assessed. Threshold calibration requires training demonstrations, either per task or pooled across tasks for a shared threshold. Sharing avoids separate per-task calibration but retains this data requirement. Earlier replanning can increase policy queries relative to full-chunk execution, raising inference cost even though the selector is inexpensive.

7 Conclusion

We showed that the execution horizon is a critical deployment variable with strongly task-dependent and non-monotonic effects on success. PACE selects execution horizons online from the phase-dependent kinematics of predicted action chunks, using prominent low-speed valleys as candidate replanning boundaries without additional training. On RoboTwin2.0, PACE improves the average success rate of $\pi_{0.5}$ over both fixed-horizon and adaptive execution baselines. Applying PACE to GR00T N1.7 on LIBERO also yields the highest overall success rate among the evaluated strategies, supporting its applicability across different base policies, benchmarks, and action representations. Real-robot evaluations further demonstrate improvements in task score and success rate over full-chunk execution. Ablations support motion-profile smoothing and longer training prediction horizons, while a shared threshold retains most gains. Gains over matched fixed-horizon baselines further support adaptation within a rollout. PACE offers a practical way to improve task performance through adaptive execution without modifying or retraining the policy.

Acknowledgments and Disclosure of Funding

This work was carried out at Peking University and JD Explore Academy. This work was supported by the Beijing Natural Science Foundation International Scientists Project (Grant No. IS26042). The authors declare no other competing interests.

References

[1] Anthony Brohan et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control, 2023. URL https://arxiv.org/abs/2307.15818.

[2] Moo Jin Kim et al. OpenVLA: An Open-Source Vision-Language-Action Model, 2024. URL https://arxiv.org/abs/2406.09246.

[3] Kevin Black et al. π_0: A Vision-Language-Action Flow Model for General Robot Control, 2024. URL https://arxiv.org/abs/2410.24164.

[4] Cheng Chi et al. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion, 2023. URL https://arxiv.org/abs/2303.04137.

[5] Tianle Zhang et al. JoyAI-RA 0.1: A Foundation Model for Robotic Autonomy, 2026. URL https://arxiv.org/abs/2604.20100.

[6] JoyAI-RA Team. JoyAI-RA 0.5: Scaling Robot Manipulation Learning via Dual Action Alignment, 2026. URL https://arxiv.org/abs/2608.05674.

[7] Yixue Zhang et al. RoboGene: Diversity-Driven Agentic Task Generation Framework for VLA Pre-training. In *Proceedings of the 10th Conference on Robot Learning (CoRL)*, 2026. URL https://arxiv.org/abs/2602.16444.

[8] Tony Z. Zhao et al. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware, 2023. URL https://arxiv.org/abs/2304.13705.

[9] Tianxing Chen et al. RoboTwin 2.0: A Scalable Data Generator and Benchmark with Strong Domain Randomization for Robust Bimanual Robotic Manipulation, 2025. URL https://arxiv.org/abs/2506.18088.

[10] Zipeng Fu et al. Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation, 2024. URL https://arxiv.org/abs/2401.02117.

[11] Yanjie Ze et al. 3D Diffusion Policy: Generalizable Visuomotor Policy Learning via Simple 3D Representations, 2024. URL https://arxiv.org/abs/2403.03954.

[12] Songming Liu et al. RDT-1B: A Diffusion Foundation Model for Bimanual Manipulation, 2024. URL https://arxiv.org/abs/2410.07864.

[13] Octo Model Team et al. Octo: An Open-Source Generalist Robot Policy, 2024. URL https://arxiv.org/abs/2405.12213.

[14] Physical Intelligence et al. $\pi_{0.5}$: A Vision-Language-Action Model with Open-World Generalization, 2025. URL https://arxiv.org/abs/2504.16054.

[15] Karl Pertsch et al. FAST: Efficient Action Tokenization for Vision-Language-Action Models, 2025. URL https://arxiv.org/abs/2501.09747.

[16] Moo Jin Kim et al. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success, 2025. URL https://arxiv.org/abs/2502.19645.

[17] Mustafa Shukor et al. SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics, 2025. URL https://arxiv.org/abs/2506.01844.

[18] Kevin Black et al. Real-Time Execution of Action Chunking Flow Policies, 2025. URL https://arxiv.org/abs/2506.07339.

[19] Haoxuan Wang et al. Real-Time Robot Execution with Masked Action Chunking, 2026. URL https://arxiv.org/abs/2601.20130.

[20] Dong Jing et al. Mixture of Horizons in Action Chunking, 2025. URL https://arxiv.org/abs/2511.19433.

[21] Yuanchang Liang et al. Adaptive Action Chunking at Inference-time for Vision-Language-Action Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20802–20811, June 2026. URL https://arxiv.org/abs/2604.04161.

[22] Haoxuan Wang et al. VLA Knows Its Limits: Adaptive Execution Horizons for Robot Policies, 2026. URL https://arxiv.org/abs/2602.21445. ECCV 2026.

[23] NVIDIA. NVIDIA Isaac GR00T N1.7. https://github.com/NVIDIA/Isaac-GR00T, 2026.

[24] Bo Liu et al. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning, 2023. URL https://arxiv.org/abs/2306.03310.

[25] Adina Yakefu et al. RoboChallenge: Large-scale Real-robot Evaluation of Embodied Policies, 2025. URL https://arxiv.org/abs/2510.17950.

[26] Alexander Khazatsky et al. DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset, 2024. URL https://arxiv.org/abs/2403.12945.

[27] Qingwen Bu et al. AgiBot World Colosseo: A Large-Scale Manipulation Platform for Scalable and Intelligent Embodied Systems. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2025. URL https://doi.org/10.1109/IROS60139.2025.11247088.

[28] Kun Wu et al. RoboMIND: Benchmark on Multi-Embodiment Intelligence Normative Data for Robot Manipulation. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025. URL https://www.roboticsproceedings.org/rss21/p152.html.

[29] Shihan Wu et al. RoboCOIN: An Open-Sourced Bimanual Robotic Data Collection for Integrated Manipulation, 2025. URL https://arxiv.org/abs/2511.17441.

A PACE implementation details

This appendix specifies the default configuration, online selection algorithm, and demonstration-based threshold calibration for PACE.

A.1 Default configuration

Table 6 summarizes the default PACE configuration for the $\pi_{0.5}$ experiments on RoboTwin2.0. The GR00T N1.7 configuration on LIBERO is described in Appendix B.2.

Table 6: **Default PACE configuration for $\pi_{0.5}$ on RoboTwin2.0.**

Component	Default value
Prediction horizon L	50
Maximum execution horizon $H_{\max}$	50
Control frequency f_s	50 Hz
Speed signal	Joint-space speed
Smoothing	Savitzky–Golay filter
Filter window / polynomial order	11 / 3
Minimum valley spacing $d_{\min}$	10 steps
Valley score	Absolute peak-height contrast
Threshold percentile	5th percentile from training demonstrations
Multi-arm fusion	Earliest accepted boundary across arms

The default speed signal is computed from arm joint commands and excludes gripper commands. This matches the policy action space and avoids forward kinematics during deployment.

A.2 Online horizon selection

Algorithm 1 gives the test-time procedure at one policy query. Let $\mathcal{B}$ denote the set of controllable arm groups. In our bimanual RoboTwin2.0 experiments, $\mathcal{B} = \{\text{left}, \text{right}\}$. For each arm group, PACE scans low-speed valleys in temporal order and accepts the first valley whose peak-height contrast exceeds the task threshold. The final execution horizon is the earliest accepted boundary across arms. If no valid valley is found, the method falls back to full-chunk execution.

Algorithm 1 PACE at a policy query

Require: Predicted action chunk $\mathbf{A}_i$, arm groups $\mathcal{B}$, maximum horizon $H_{\max}$, threshold $\delta_{\mathcal{T}}$
Ensure: Execution horizon h_i

for each arm group $b \in \mathcal{B}$ **do**
  Compute speed profile $\mathbf{v}_i^b$ from $\mathbf{A}_i$
  Smooth the profile: $\tilde{\mathbf{v}}_i^b \leftarrow \mathcal{S}(\mathbf{v}_i^b)$
  Detect candidate valleys $\mathcal{V}_i^b$ with spacing $d_{\min}$
  Set $h_i^b \leftarrow H_{\max}$
  for valley $r \in \mathcal{V}_i^b$ in temporal order **do**
    if $r > H_{\max}$ **then**
      break
    end if
    Compute peak-height contrast $\Delta_i^b(r)$
    if $\Delta_i^b(r) \geq \delta_{\mathcal{T}}$ **then**
      $h_i^b \leftarrow r$
      break
    end if
  end for
end for
return $h_i = \min\left(\{h_i^b : b \in \mathcal{B}\} \cup \{H_{\max}, L\}\right)$

For a single-arm policy or a single-arm task, we simply set $\mathcal{B} = \{b\}$, where b is the executed arm. No dummy second arm is introduced. The final horizon then reduces to

$$h_i = \min\{h_i^b, H_{\max}, L\}. \tag{8}$$

Thus, the single-arm implementation is the same algorithm without the cross-arm minimum. If the single arm has no accepted valley, the fallback remains $h_i = H_{\max}$.

A.3 Offline threshold calibration from demonstrations

The only task-specific quantity in PACE is the valley-acceptance threshold $\delta_{\mathcal{T}}$. We calibrate this threshold once per task from training demonstrations and then keep it fixed for evaluation. This calibration does not use evaluation rollouts, success labels, or fixed-horizon sweeps.

For each task, we compute speed profiles from 50 training demonstrations using the same signal type, smoothing filter, and minimum-spacing parameter as the online controller. Candidate valleys divide each demonstration into motion segments. For a segment $s = [u, v]$, we compute its peak-height contrast as

$$\Delta(s) = \min\left\{ \max_{u \le j \le v} \tilde{v}_j - \tilde{v}_u,\ \max_{u \le j \le v} \tilde{v}_j - \tilde{v}_v \right\}. \tag{9}$$

We collect positive contrasts across training segments and set the task threshold to

$$\delta_{\mathcal{T}} = \mathrm{Percentile}_5\left(\{\Delta(s) : s \in \text{training segments of task } \mathcal{T}\}\right). \tag{10}$$

The same calibration procedure is used for single-arm and multi-arm settings; the only difference is the arm set $\mathcal{B}$ used to construct speed profiles.

We also pool training demonstrations from all 50 RoboTwin2.0 tasks and apply the same calibration procedure to obtain a shared threshold (Appendix C.4).

B Simulation protocols and results

B.1 RoboTwin2.0

B.1.1 Evaluation protocol

We evaluate all methods on the 50-task RoboTwin2.0 benchmark under the `demo_clean` setting. Each task uses one trained $\pi_{0.5}$ checkpoint with prediction horizon $L = 50$. Within each task, all compared execution schemes share the same checkpoint, and only the test-time execution scheme is changed.

For each task, we fine-tune the pretrained $\pi_{0.5}$ model on 50 `demo_clean` demonstrations. All model parameters are updated using AdamW with a global batch size of 32 and a fixed training seed of 42. The learning rate is warmed up to 2.5×10^{-5} over 1,000 steps and then decayed to 2.5×10^{-6} following a cosine schedule. We evaluate the checkpoint saved at step 30,000, using exponential moving average weights with a decay of 0.99.

We compare PACE against fixed-horizon execution with $H \in \{5, 25, 50\}$, AAC, and AutoHorizon. The $H = 50$ baseline corresponds to full-chunk execution. For fixed-horizon execution, the robot executes the first H actions of each predicted chunk before re-querying the policy. For PACE, the executed prefix is selected by the rule in Appendix A. The AAC and AutoHorizon implementations are described in Appendix B.1.3. In all methods, the executed prefix is clamped by the returned chunk length.

For each task and method, we evaluate seeds $0, 1, 2$. For each seed, we run three batches of 100 episodes. Thus, each task-method pair is evaluated over $3 \times 3 \times 100 = 900$ episodes. Success is measured using the RoboTwin2.0 task success predicate. The task-level success rate is averaged over the nine 100-episode runs, and the 50-task average is the equal-weight average over all task-level success rates.

B.1.2 Full results and aggregate statistics

Table 7 reports success rates for PACE, fixed-horizon execution, AAC, and AutoHorizon on all 50 RoboTwin2.0 tasks. The best single global fixed-horizon baseline is $H = 25$, with a 50-task average success rate of 57.8%. PACE improves the average success rate to 64.2%.

Table 7: **Results on all 50 RoboTwin2.0 tasks.** Success rates (%). Bold marks the highest mean in each row. The final rows report the mean and sample SD of the three seed-level overall success rates.

Task	$H = 5$	$H = 25$	$H = 50$	AutoHorizon	AAC	PACE
adjust_bottle	99.0	98.8	96.1	98.7	**99.7**	99.6
beat_block_hammer	**81.1**	67.7	66.6	71.3	64.8	80.7
blocks_ranking_rgb	7.3	50.0	36.1	**66.1**	52.4	61.4
blocks_ranking_size	23.9	18.6	9.2	25.9	20.2	**32.9**
click_alarmclock	**92.4**	65.3	65.9	71.4	86.7	87.6
click_bell	**97.8**	49.2	41.4	70.8	64.3	91.4
dump_bin_bigbin	58.7	85.1	84.3	88.1	89.1	**93.0**
grab_roller	89.9	96.2	96.2	96.9	97.7	**98.6**
handover_block	3.6	33.2	52.6	43.1	43.2	**53.1**
handover_mic	77.2	97.7	96.8	97.7	96.8	**99.4**
hanging_mug	16.2	15.8	16.7	20.1	**23.6**	21.8
lift_pot	50.9	61.2	62.0	62.2	63.1	**65.4**
move_can_pot	55.4	**74.3**	58.2	56.8	60.2	72.3
move_pillbottle_pad	34.8	**49.9**	29.1	44.3	46.6	45.0
move_playingcard_away	79.2	**79.7**	69.6	72.1	70.4	79.3
move_stapler_pad	12.6	11.4	7.1	13.6	10.6	**18.6**
open_laptop	84.9	84.8	88.8	**89.6**	89.3	88.0
open_microwave	33.3	26.0	**60.3**	55.4	52.4	45.0
pick_diverse_bottles	13.4	23.9	24.1	26.4	27.9	**35.7**
pick_dual_bottles	13.3	45.3	38.7	41.6	**48.9**	48.2
place_a2b_left	54.6	52.7	43.2	46.4	50.4	**55.9**
place_a2b_right	20.0	47.8	36.8	43.1	46.7	**54.6**
place_bread_basket	**55.3**	53.9	30.3	42.6	46.2	49.1
place_bread_skillet	38.4	39.8	29.0	33.2	**42.0**	37.0
place_burger_fries	88.9	89.3	85.8	87.1	**91.9**	91.3
place_can_basket	18.1	56.8	38.2	52.1	**66.3**	53.6
place_cans_plasticbox	85.1	83.0	81.8	85.2	85.8	**90.6**
place_container_plate	88.3	81.7	91.6	91.2	89.1	**96.1**
place_dual_shoes	19.9	**40.6**	33.7	31.0	40.3	37.3
place_empty_cup	**59.4**	48.1	45.8	51.8	45.9	58.8
place_fan	35.3	34.0	25.8	31.4	35.7	**39.1**
place_mouse_pad	4.3	14.1	12.8	16.3	15.1	**17.8**
place_object_basket	35.9	59.4	58.0	67.8	64.8	**70.2**
place_object_scale	23.8	29.4	18.9	23.8	24.8	**40.7**
place_object_stand	**64.2**	57.3	53.4	56.6	54.9	59.2
place_phone_stand	52.9	**59.9**	54.7	54.7	55.4	58.7
place_shoe	43.9	**64.4**	56.1	58.6	63.0	62.8
press_stapler	**68.2**	60.6	56.0	59.8	61.6	61.8
put_bottles_dustbin	0.3	79.3	70.8	74.3	76.6	**88.9**
put_object_cabinet	7.6	51.9	50.6	50.6	48.7	**57.4**
rotate_qrcode	48.9	69.9	66.1	70.2	68.4	**71.7**
scan_object	20.9	**43.1**	28.9	39.7	39.9	40.3
shake_bottle	95.2	95.7	96.7	96.7	96.2	**98.4**
shake_bottle_horizontally	97.3	96.8	96.9	97.1	98.6	**98.8**
stack_blocks_three	39.7	43.2	28.7	54.6	39.1	**55.2**
stack_blocks_two	12.4	80.1	63.9	**82.9**	77.3	82.3
stack_bowls_three	45.0	61.0	67.6	70.8	64.0	**79.8**
stack_bowls_two	88.7	88.4	85.1	91.0	88.2	**95.6**
stamp_seal	**67.4**	34.2	24.3	37.0	32.1	47.8
turn_switch	34.4	39.4	37.1	41.8	36.7	**42.6**
Mean	48.79	57.80	53.36	59.03	59.07	**64.20**
Seed SD	0.89	1.49	0.44	0.53	0.56	0.88

For each evaluation seed, we average the three batches within each task and then average equally over all 50 tasks. The final two rows of Table 7 summarize the resulting three overall success rates. Each task uses the same checkpoint across seeds, so the SD reflects evaluation variability.

PACE achieves a higher mean success rate than AutoHorizon on 46 of the 50 tasks and than AAC on 39 of the 50 tasks.

Using unrounded success rates, PACE improves overall success by 6.40 percentage points over fixed $H = 25$ and by 5.13 points over AAC. Paired bootstraps over the 50 task-level mean differences yield 95% confidence intervals of $[+4.21, +8.86]$ and $[+3.22, +7.10]$, respectively. Against AutoHorizon, PACE gains 5.18 percentage points, with a 95% confidence interval of $[+3.62, +6.78]$ from a paired hierarchical bootstrap. These confidence intervals describe uncertainty in the gains over each baseline, while the SD in Table 7 describes variability across evaluation seeds.

B.1.3 Adaptive baseline implementations

Shared experimental setting. AutoHorizon and AAC use the same task-specific $\pi_{0.5}$ checkpoints as the fixed-horizon baselines and PACE, without additional policy training. Both use ten flow-sampling steps to predict 50-step chunks of decoded 14-D actions, with six absolute joint-position commands and one gripper command per arm. They follow the evaluation protocol in Appendix B.1.1. For each baseline, the three batches within a task and evaluation seed reuse the same set of 100 evaluation scenes. Their selector settings are fixed across tasks and are not tuned using evaluation success labels.

AutoHorizon. We implement AutoHorizon [22] in JAX following the official implementation. The selector uses model-internal action-token self-attention from denoising step 3 and applies the bidirectional soft-pointer rule with $\tau = 0.3$, entropy quantile 0.9, and run length 1. The robot executes the selected prefix of the decoded action chunk, discards the remaining actions, and queries the policy again using a new observation.

AAC adapted to bimanual joint-space actions. We adapt AAC [21] to RoboTwin's bimanual joint-space actions and sample 20 candidate chunks per policy query. At each prediction step, we estimate Gaussian entropy from the six normalized joint commands of each arm across the candidates, with covariance regularization $10^{-6}I$. For each gripper, we binarize its decoded commands at 0.5 and compute Bernoulli entropy.

Movement is measured from the decoded absolute joint targets relative to the current state, summing the arm-wise displacements and weighting gripper toggles by 0.2. We use the movement threshold of 3.0 reported in the AAC supplementary material. The final execution horizon is the larger of the horizon selected by AAC's prefix-entropy turning-point rule and the shortest prefix satisfying the movement constraint.

To generate AAC's 20 candidate chunks, our $\pi_{0.5}$ implementation encodes the image–language–state prefix once per query and reuses its KV cache across candidates. The action head processes candidate 0 separately and the remaining 19 candidates as a batch. This grouping preserves candidate 0's agreement with standard $\pi_{0.5}$ sampling. The robot executes the selected prefix of candidate 0.

B.2 GR00T N1.7 on LIBERO

Policy and evaluation protocol. We evaluate PACE using the official NVIDIA/LeRobot GR00T N1.7 checkpoints [23] on LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-10 [24]. The four suites contain 10 tasks each, for 40 tasks in total. Within each suite, all execution strategies use the same checkpoint without retraining or modifying the base policy. We evaluate each task on 50 initial states for each of three evaluation seeds, yielding 150 episodes per task and 6,000 episodes per strategy.

For each seed, Overall is the equal-weight mean success rate across all 40 tasks. We report the mean and sample standard deviation across the three seeds for each suite and for Overall.

Adapting PACE. GR00T N1.7 predicts 16-step Cartesian-delta chunks, whereas the $\pi_{0.5}$ policies evaluated on RoboTwin2.0 predict 50-step joint-space chunks. PACE constructs the motion profile from the norm of the predicted translational action deltas and applies the same valley-based horizon-selection rule. PACE calibrates its acceptance thresholds from LIBERO training demonstrations using valley contrasts normalized by peak speed.

We compare PACE with fixed horizons $H = 2, 4, 8, 16$ and AutoHorizon under the same evaluation protocol. The $H = 16$ baseline executes the full predicted chunk. AutoHorizon follows its published horizon-selection rule adapted to GR00T N1.7.

Table 8: **GR00T N1.7 results on LIBERO.** Success rates (%), reported as mean ± sample SD across three evaluation seeds. Bold marks the highest mean in each column.

Execution strategy	Spatial	Object	Goal	LIBERO-10	Overall
H = 2	97.20 ± 0.92	91.47 ± 0.50	85.87 ± 1.17	77.73 ± 1.33	88.07 ± 0.28
H = 4	$\mathbf{97.67} \pm 0.31$	94.33 ± 0.90	93.13 ± 1.62	83.33 ± 1.92	92.12 ± 0.30
H = 8	96.87 ± 0.90	96.20 ± 0.72	96.00 ± 0.53	88.00 ± 0.87	94.27 ± 0.37
H = 16	96.27 ± 0.99	97.00 ± 0.92	96.53 ± 1.01	88.07 ± 0.31	94.47 ± 0.23
AutoHorizon	96.13 ± 0.42	96.67 ± 0.31	95.80 ± 0.80	$\mathbf{91.07} \pm 1.45$	94.92 ± 0.44
PACE	97.00 ± 0.72	$\mathbf{97.73} \pm 0.23$	$\mathbf{96.93} \pm 0.90$	89.67 ± 1.17	$\mathbf{95.33} \pm 0.18$

Results and interpretation. PACE achieves the highest overall success rate among the evaluated strategies (Table 8). Using unrounded rates, it exceeds $H = 16$, the strongest global fixed-horizon baseline, by 0.87 percentage points. A paired hierarchical bootstrap stratified by suite gives a 95% confidence interval of $[+0.10, +1.70]$ percentage points for this gain. Its overall success rate is also 0.42 percentage points higher than AutoHorizon. Across individual suites, PACE leads on Object and Goal, fixed $H = 4$ on Spatial, and AutoHorizon on LIBERO-10.

Policy-query and inference overhead. We measure inference latency on a dedicated NVIDIA H20 using the LIBERO setup described above. Across the four suite-specific GR00T N1.7 checkpoints, model-forward latency ranges from 127.55 to 130.06 ms per query. PACE's horizon selector takes 0.0566 ± 0.0046 ms per query, approximately 0.044% of the time required for a policy forward pass. The total selector time per successful episode is 0.595 ± 0.001 ms.

Table 9 reports policy queries and total inference time per successful episode for each execution strategy. These metrics are computed separately for each evaluation seed using that strategy's successful episodes. We report the mean and sample standard deviation across the three seeds. The reported times cover inference computation only.

Table 9: **Policy queries and inference time with GR00T N1.7 on LIBERO.** Total inference time includes policy forward passes and, for PACE, horizon selection.

Execution strategy	Queries per successful episode	Total inference time per successful episode (s)
H = 2	78.21 ± 0.43	10.075 ± 0.056
H = 4	38.49 ± 0.10	4.959 ± 0.013
H = 8	19.05 ± 0.08	2.455 ± 0.010
H = 16	9.69 ± 0.01	1.248 ± 0.002
PACE	10.53 ± 0.02	1.357 ± 0.003

Compared with full-chunk execution ($H = 16$), PACE improves success by 0.87 percentage points while increasing total inference time per successful episode by 8.66%. This increase comes almost entirely from the 0.83 additional policy queries per successful episode, with horizon selection contributing little. All differences are computed from unrounded measurements.

C Ablation and robustness studies

The phase-profile and valley-selection studies follow the full RoboTwin2.0 protocol in Appendix B.1.1: 50 tasks and 900 episodes per task across three seeds. Average horizon denotes the mean planned execution length. The training-horizon study requires retraining and uses six representative tasks, seed 0, and three 100-episode batches per setting.

C.1 Phase-profile construction

We compare Cartesian and joint-space speed, with and without Savitzky–Golay smoothing, while keeping the valley-selection rule and calibration percentile fixed. The Cartesian variant computes end-effector linear speed using forward kinematics; the raw variants detect valleys without smoothing. Table 2 (left) reports the results. Smoothing improves both representations, consistent with suppressing spurious short-range valleys. The smoothed Cartesian variant has a slight success-rate advantage, while the default smoothed joint-space variant produces longer execution horizons and avoids forward kinematics at test time.

C.2 Training prediction horizon

We study how the policy prediction horizon used during training affects fixed-horizon execution at test time. Here H_{train} denotes the supervised action-chunk length during policy training, while H_{eval} denotes the fixed executed prefix length at evaluation. Entries with $H_{\text{train}} < H_{\text{eval}}$ are infeasible because the policy does not predict enough actions to execute the requested prefix. We sweep both axes on a finer grid, from 5 to 50 in steps of 5, using six representative tasks, seed 0, and three 100-episode batches per setting.

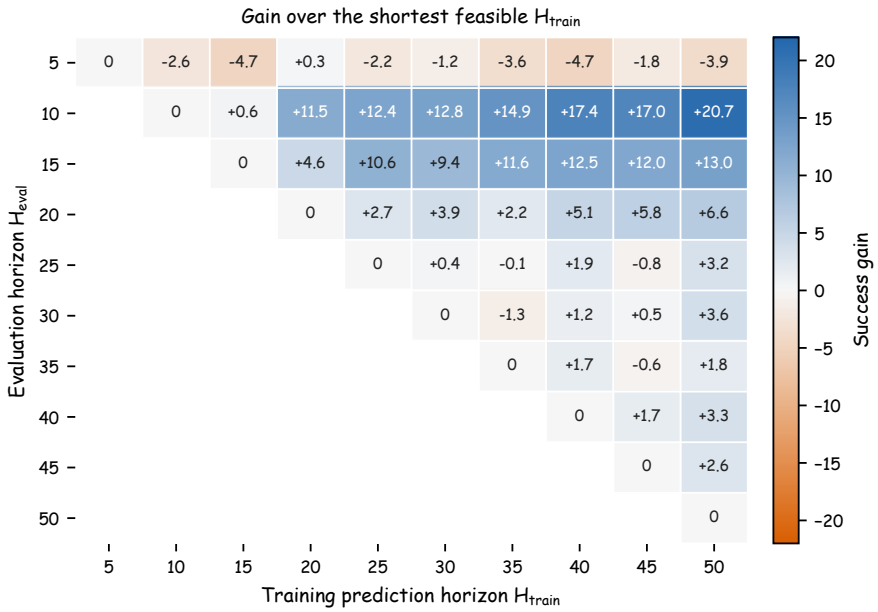


Figure 7: **Training prediction horizon ablation.** Each cell shows the success gain of training horizon H_{train} (columns) relative to the shortest feasible training horizon for a given evaluation horizon H_{eval} (rows), i.e., H_{train}=H_{eval}. Blue indicates a gain, orange a loss; the diagonal is zero by construction. Cells below the diagonal are infeasible and left blank.

Figure 7 shows that, for $H_{\text{eval}} \geq 10$, increasing H_{train} generally improves success, with gains reaching $+20.7$, $+13.0$, and $+6.6$ points at $H_{\text{eval}} = 10, 15, 20$, respectively. Very short execution is an exception: at $H_{\text{eval}} = 5$, most gains lie between -4.7 and 0. For $H_{\text{eval}} > 20$, gains remain below $+7$ points and become more variable, including near-zero and slightly negative values. Longer training horizons thus benefit short but non-trivial execution prefixes most, motivating the default $H_{\text{train}} = 50$.

C.3 Valley-selection sensitivity

We vary the minimum spacing between candidate valleys and the calibration percentile ρ used to set the contrast threshold δ, keeping all other settings fixed. Larger ρ produces a stricter threshold, which accepts fewer valleys and therefore tends to increase the selected execution horizon.

Table 10: **Sensitivity to valley-selection parameters.** All results use the smoothed joint-space phase profile and are averaged over all 50 RoboTwin2.0 tasks.

Min. spacing	Percentile ρ	Succ.	Avg. Horizon
5	0	**64.3**	18.1
5	5	63.5	23.4
5	10	63.3	24.6
5	25	62.5	31.9
10	0	64.0	19.8
10	$5^{\dagger}$	64.2	24.3
10	10	63.0	25.5
10	25	62.0	32.9

PACE is not sensitive to a single narrowly tuned setting. Several nearby configurations achieve similar success rates. Very strict thresholds, such as $\rho = 25$, select substantially longer execution windows and reduce success, suggesting that rejecting too many phase valleys delays feedback in contact-sensitive stages. We use minimum spacing 10 and $\rho = 5$ as the default because it provides high success while keeping the average execution horizon moderate.

C.4 Shared versus per-task threshold calibration

To evaluate whether PACE can use one valley-acceptance threshold across tasks, we compare per-task and shared-threshold calibration. The default setting calibrates a separate threshold from each task's training demonstrations. For the shared setting, we pool training demonstrations from all 50 RoboTwin2.0 tasks and apply the same calibration procedure (Appendix A.3), retaining the speed-profile construction and 5th-percentile rule. The resulting threshold is used unchanged across all 50 tasks. All compared execution strategies use the same task-specific $\pi_{0.5}$ checkpoints, and all other PACE settings and evaluation conditions remain unchanged.

Following the evaluation protocol in Appendix B.1.1, we compute the equal-weight mean success rate over all 50 tasks for each of three evaluation seeds. Table 3 reports the mean and sample standard deviation of these three averages.

Shared-threshold PACE improves success over fixed $H = 25$ by 4.64 percentage points, with a 95% confidence interval of $[+2.61, +6.99]$. Per-task calibration provides an additional gain of approximately 1.8 points over the shared threshold, with a 95% confidence interval of $[+1.18, +2.35]$. We compute each interval with a paired bootstrap over the 50 task-level differences using unrounded mean success rates. These results show that a single shared threshold preserves most of PACE's improvement over fixed $H = 25$.

C.5 Calibration stability and trajectory sensitivity

Data and sampling. To assess the stability of PACE's horizon decisions, we conduct an offline analysis of trajectories from DROID [26], AgiBot World Beta [27], RoboMIND [28], and RoboCOIN [29]. We use stratified sampling by task and robot configuration, covering 1,384 eligible strata across 31 dataset-defined robot configurations. For each stratum, we use 50 episodes for calibration and up to 50 held-out episodes for evaluation. The resulting sample contains 135,046 distinct episodes and 263,319 fixed 50-step evaluation windows.

Calibration resampling. For each stratum, we first compute a reference valley-acceptance threshold from its 50 calibration demonstrations using the fifth-percentile rule. We then resample these demonstrations with replacement 500 times and recompute the threshold for each sample. On the same held-out evaluation windows, we compare the horizons selected by each recalibrated threshold with those selected by the reference threshold.

Trajectory perturbation. To assess sensitivity to trajectory fluctuations, we double the high-frequency residual in the joint-position action sequences. This residual is the difference between each sequence and its trend obtained with a five-step, second-order Savitzky–Golay filter. We perturb both

the calibration trajectories and the held-out evaluation windows, and recalibrate the threshold on the perturbed demonstrations. On the perturbed evaluation windows, we measure horizon agreement within five control steps of the original decisions.

Table 11: **Offline stability of PACE's horizon decisions.** For each metric, we average equally over task/configuration strata within each dataset family and then equally across the four families.

Test	Metric	Result
Calibration resampling	Threshold coefficient of variation (dataset-family-balanced)	18.7%
	Identical horizon	99.08%
	Absolute horizon shift > 5 steps	0.78%
	Mean absolute horizon shift	0.14 steps
High-frequency residual doubled	Absolute horizon shift ≤ 5 steps	80.6%

Table 11 reports horizon consistency under calibration resampling and trajectory perturbation. On a fixed evaluation window, threshold changes leave the horizon unchanged whenever the same earliest valley remains accepted or all candidates remain rejected.

D Execution-horizon analysis

D.1 Fixed-horizon sweeps

For each of the six representative tasks in Sec. 4.4, we evaluate every integer fixed horizon $H \in \{1, \ldots, 50\}$. Each fixed-horizon setting uses seed 0 and three 100-episode batches, giving 300 episodes per horizon. PACE uses three seeds with three 100-episode batches per seed, giving 900 episodes per task. Its mean execution horizon is averaged over all selected execution windows. The best swept horizon $H^\star$ is selected using test-task outcomes and serves as a diagnostic reference within fixed-horizon schedules, not a deployable selection rule.

Table 12: **Diagnostic fixed-horizon sweep summary.** $H^\star$ denotes the best fixed horizon in the sweep, with ties listed together. All success rates are percentages.

Task	$H^\star$	Fixed at $H^\star$	PACE
Pick Dual Bottles	14	56.7	48.2
Grab Roller	7, 9, 32	97.7	98.6
Place A2B Right	11	53.3	54.6
Dump Bin Bigbin	37	90.3	93.0
Put Bottles Dustbin	18	88.7	88.9
Stack Bowls Two	8	95.3	95.6

Table 12 shows substantial task dependence in $H^\star$, ranging from short horizons such as 7 or 8 to 37. Grab Roller has three tied best horizons, illustrating the non-monotonic effect of execution length.

D.2 Matched execution horizons

We obtain the matched-horizon results in Table 4 from the sweep by selecting the fixed horizon nearest to PACE's mean execution horizon on each task. PACE achieves a higher observed success rate on all six tasks, with gains of 1.6–10.6 percentage points computed from unrounded values. These results suggest that matching the mean execution horizon alone is insufficient to reproduce PACE's performance.

Pick Dual Bottles illustrates the difference between the two fixed-horizon comparisons. PACE achieves 48.2% success, exceeding the matched fixed horizon $H = 26$ (46.7%) but remaining below the best swept horizon $H^\star = 14$ (56.7%).

In the seed-0 evaluation, PACE achieved 43.7% success, compared with 46.0% for fixed $H = 25$, with 300 episodes per method. Across all three seeds (900 episodes per method), the corresponding success rates are 48.2% and 45.3% (Appendix B), reversing the seed-0 ordering.

D.3 Phase alignment

To examine whether PACE replans near motion-phase transitions, we compare its executed valley-triggered boundaries with independently annotated transitions. The analysis covers 89 successful rollouts from three RoboTwin2.0 tasks: Place A2B Right, Place Phone Stand, and Pick Dual Bottles.

Blind annotation and matching. Annotators identified motion-phase transitions from paired head-camera and external-observer videos without access to PACE's replanning boundaries or boundary overlays. After the annotations were finalized, we extracted the executed valley-triggered boundaries from the rollout logs. We matched boundaries one-to-one with annotated transitions within ± 5 control steps (± 0.1 s at 50 Hz). We chose the matching with the largest number of pairs, breaking ties by the smallest total temporal distance. The one-to-one constraint prevents multiple boundaries from being matched to the same transition.

Of the 391 executed replanning boundaries, six were $H_{\max}$ fallbacks and were counted separately. The primary alignment metric uses the remaining 385 valley-triggered boundaries.

Metrics and temporal controls. Alignment precision is the fraction of executed valley-triggered boundaries matched to annotated transitions. Transition coverage is the fraction of annotated transitions matched to a boundary. For each task, we obtain 95% confidence intervals for PACE precision by bootstrapping the audited successful rollouts at the trajectory level.

To assess whether the observed alignment exceeds chance levels, we compare PACE with two offline controls that retain the same number of boundaries. The first samples boundary times uniformly at random with a minimum spacing of 10 control steps. The second circularly shifts PACE's boundaries while preserving their relative spacing. Each control is evaluated over 10,000 randomizations.

Table 13: **Phase alignment on successful rollouts.** The last three columns report alignment precision (%). Overall is computed from pooled counts across tasks.

Task	Annotated transitions	Aligned / valleys	PACE precision (95% CI)	Uniform random	Circular shift
Place A2B Right	182	135 / 172	78.5 [71.1, 85.1]	45.9	46.1
Place Phone Stand	145	96 / 123	78.0 [72.1, 84.0]	43.7	43.8
Pick Dual Bottles	120	76 / 90	84.4 [78.5, 90.2]	41.9	42.3
Overall	447	307 / 385	79.7	–	–

Results. Table 13 shows that 307 of 385 valley-triggered boundaries align with annotated motion-phase transitions, yielding 79.7% precision. These matches cover 68.7% of the 447 annotated transitions. Matched boundaries lie an average of 2.3 control steps from their corresponding transitions. PACE achieves higher alignment precision than both the uniform-random and circular-shift controls on all three tasks, with $p \leq 0.0001$ in all six randomization tests.

Interpretation. The observed alignment on successful rollouts from these three tasks supports PACE's use of prominent low-speed valleys as cues for candidate replanning boundaries. Assessing the effect of individual boundary choices on task success would require controlled interventions on boundary timing.

E Real-robot experiment details

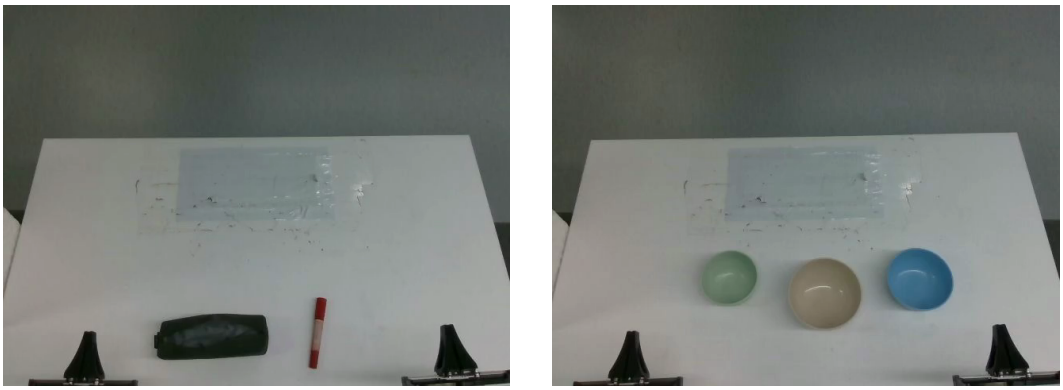
E.1 Tasks, scenes, and scoring rules

Within each task, all execution strategies use the same fine-tuned $\pi_{0.5}$ checkpoint and differ only in the test-time execution rule. Table 14 summarizes the robots, demonstrations, and trial counts.

Table 14: **Real-robot task and data summary.** Demos denotes the number of demonstration episodes used for fine-tuning.

Task	Robot	Demos	Trials / method
`stack_bowls`	ALOHA	1,047	30
`put_pen_into_pencil_case`	ALOHA	1,220	30
`place_object_on_plate`	Franka	301	5×20

RoboChallenge. We evaluate `stack_bowls` and `put_pen_into_pencil_case` on RoboChallenge Table30 V1, following its task prompts, reset procedures, and grading rules. Figure 8 shows the initial scenes. We report the official partial-credit Score and full-completion rate Succ., using the scoring steps below.



`put_pen_into_pencil_case` `stack_bowls`

Figure 8: **Initial frames of the RoboChallenge tasks.** Left: `put_pen_into_pencil_case`. Right: `stack_bowls`. Both tasks use an ALOHA robot.

`stack_bowls`**.**

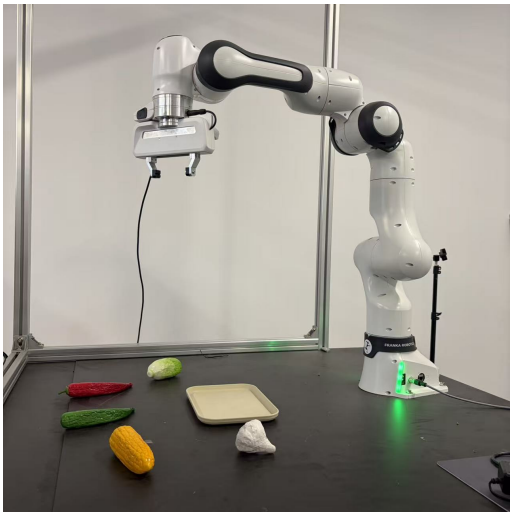
- The task is to nest the bowls from largest to smallest.
- 2.0×2 points: pick up a bowl with the gripper.
- 2.5×2 points: successfully stack the bowls.
- 1.0 point: retract the gripper to its original position.

`put_pen_into_pencil_case`**.** *Note:* The official RoboChallenge description labels the pencil-case opening step with the opposite arm; in the evaluation setup and rollout videos, the left gripper opens the pencil case while the right arm manipulates the pen. We therefore report the scoring steps using the observed arm assignment.

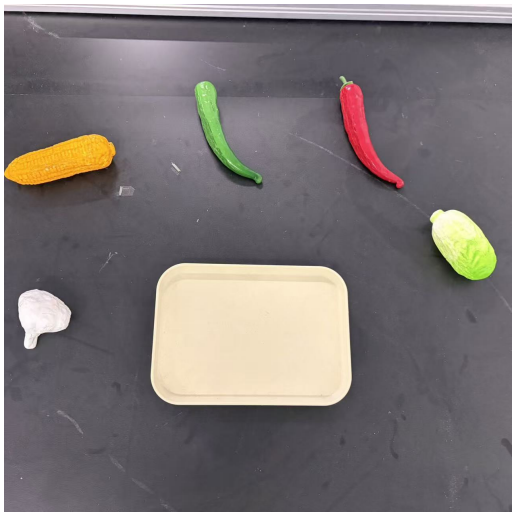
- The task is to place the pen on the table into the pencil case.
- 2.0 points: left gripper reaches and opens the pencil case.
- 2.0 points: the pencil case is sufficiently open for insertion.
- 2.0 points: right arm picks up the pen.
- 3.0 points: right arm places the pen into the pencil case.
- 1.0 point: reset both arms.

Franka `place_object_on_plate`**.** The robot picks up one of five objects—corn, cabbage, green pepper, red pepper, and garlic—and places it onto a fixed plate (Fig. 9). A single fine-tuned checkpoint is shared across all five object variants. A trial succeeds only if the object has been released and remains fully inside the plate at the end of the trial; otherwise it fails. Successful trials receive 1 point

and failures 0, so Score equals the success rate in percentage points. Table 15 reports the per-object outcomes.



Object and plate layout



Franka tabletop setup

Figure 9: **In-lab `place_object_on_plate` setup.** The robot places one of five objects onto a fixed target plate.

Table 15: **Per-object results for `place_object_on_plate`.** Each object variant is evaluated over 20 trials per method.

Object	Baseline	PACE	Δ
Corn	16 / 20 (80.0%)	**18 / 20 (90.0%)**	+10.0
Cabbage	15 / 20 (75.0%)	**17 / 20 (85.0%)**	+10.0
Green pepper	13 / 20 (65.0%)	**18 / 20 (90.0%)**	+25.0
Red pepper	14 / 20 (70.0%)	**17 / 20 (85.0%)**	+15.0
Garlic	14 / 20 (70.0%)	**18 / 20 (90.0%)**	+20.0
Total	**72 / 100 (72.0%)**	**88 / 100 (88.0%)**	+16.0

E.2 Fine-tuning configuration

For each RoboChallenge task, we fine-tune a task-specific $\pi_{0.5}$ policy initialized from the PI0.5 base checkpoint. Training is performed on four H200 GPUs with FSDP and bfloat16 precision. We use AdamW with a global batch size of 256, global gradient clipping at norm 1.0, EMA decay 0.99, and eight data-loader workers. The learning rate follows a cosine decay schedule with 1000 warmup steps, peak learning rate 2.5×10^{-5}, decay horizon 30,000 steps, and final learning rate 2.5×10^{-6}. For all reported RoboChallenge trials, we deploy the 30,000-step checkpoint.

For `place_object_on_plate`, we fine-tune a single $\pi_{0.5}$ checkpoint over all five object variants using 301 demonstrations. The training recipe follows the RoboChallenge setting except that we use a batch size of 32 and train on one RTX A6000 Pro GPU. We also deploy the 30,000-step checkpoint for all in-lab trials.

E.3 Evaluation and fixed-horizon comparisons

On RoboChallenge, we compare PACE with full-chunk execution ($H = 50$). On Franka, we additionally evaluate fixed $H = 5$ and $H = 25$. PACE selects the executed prefix using the default configuration in Appendix A, without task-specific horizon tuning during evaluation.

Table 16 reports success rates and policy queries per successful episode on Franka.

Table 16: **Comparison with fixed-horizon execution on Franka.** Query counts are averaged over each strategy's successful episodes.

Execution	Successes / trials	Fixed H / PACE mean H	Queries / successful episode
Fixed $H = 5$	87 / 100	5	36.22
Fixed $H = 25$	82 / 100	25	6.36
Fixed $H = 50$	72 / 100	50	3.14
PACE	**88 / 100**	28.64	5.57

PACE achieves a success rate comparable to fixed $H = 5$ (88% versus 87%), while using 84.6% fewer policy queries per successful episode. Relative to fixed $H = 25$, PACE improves success by 6 percentage points and reduces policy queries per successful episode by 12.4%. Fixed $H = 50$ requires the fewest queries per successful episode, but its success rate is lower at 72%.

F Additional PACE rollout visualizations

These additional RoboTwin2.0 and RoboChallenge rollouts show the policy-query timesteps and execution horizons selected by PACE. Each caption summarizes how the horizon changes across the task's motion phases.

F.1 RoboTwin2.0 rollouts

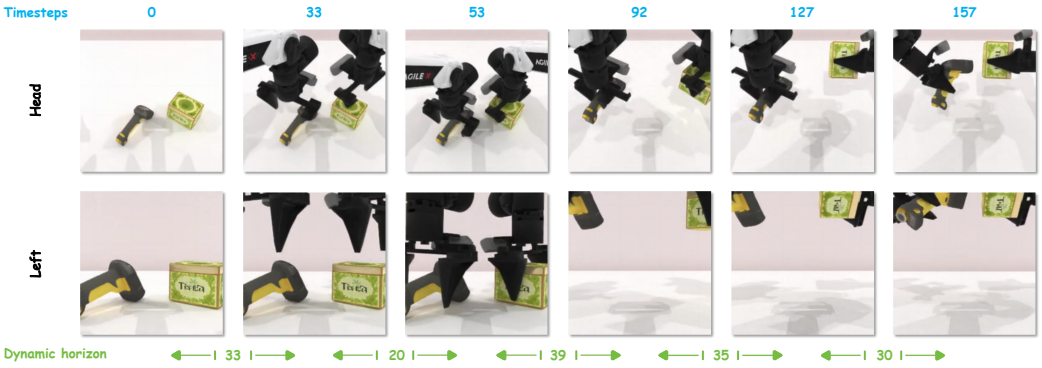


Figure 10: **PACE rollout visualization on `scan_object`.** PACE selects execution horizons from the predicted kinematic profile, allowing the rollout to keep longer open-loop segments when motion is smooth and to replan earlier around phase transitions.

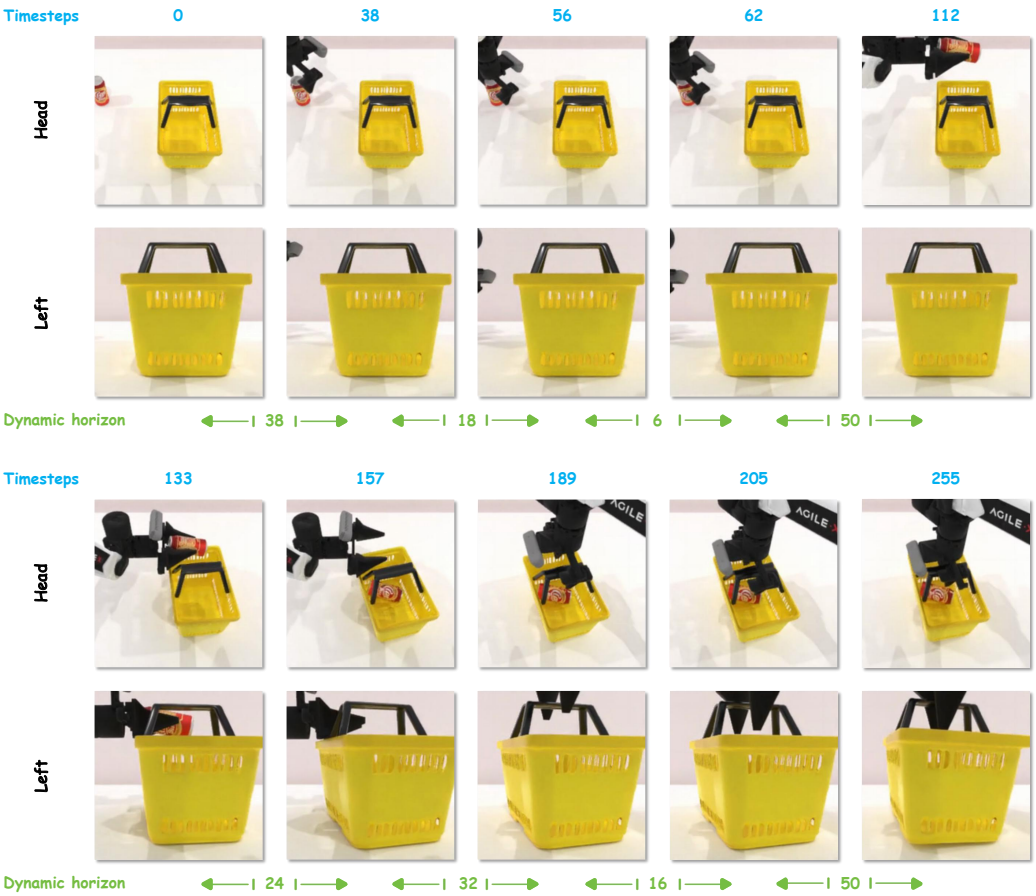


Figure 11: **PACE rollout visualization on** `place_can_basket`**.** The selected execution horizon adapts to the local phase structure of the rollout, with shorter prefixes near the placement transition and longer prefixes during smooth motion segments.



Figure 12: **PACE rollout visualization on** `hanging_mug`**.** The execution horizon is shortened around the task transition that requires precise alignment and interaction, and lengthened during more stable motion segments.

F.2 RoboChallenge rollout

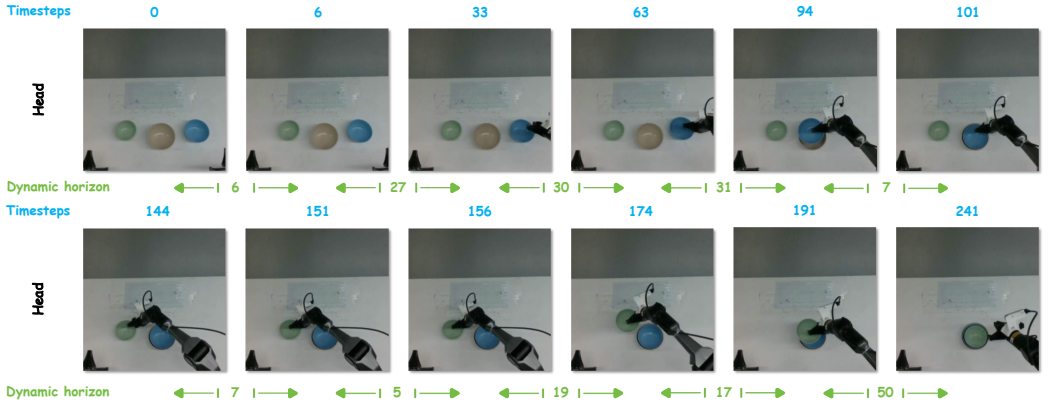


Figure 13: **PACE rollout visualization on RoboChallenge** `stack_bowls`**.** The selected execution horizon is longer during smooth approach and transport, and shorter near the stacking phase, reflecting the same phase-aware replanning rule used throughout the paper.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The abstract and Section 1 state the main claims: execution horizon is a critical test-time variable, fixed horizons are task-dependent and non-monotonic, and PACE improves simulation and real-robot execution without retraining. These claims are supported by the simulation results in Section 4.1, real-robot results in Section 5.1, and diagnostic analyses in Sections 4.2 and 4.4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: Sec. 6 discusses PACE's dependence on the base policy, reliance on low-speed valley cues, and need for demonstration-based calibration. It also addresses the scope of the empirical evaluation and the inference cost of additional policy queries.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A].

   Justification: The paper does not present theoretical results or formal proofs. The equations in Sections 3 and A define the action-chunking setup, speed profiles, and deterministic PACE selection rule.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes].

   Justification: Sections 3 and A specify the PACE algorithm, speed signal, smoothing, threshold calibration, and multi-arm fusion rule. Sections 4, B, C, D, and E describe the benchmarks, seeds, trial counts, baselines, training settings, and evaluation protocols.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No].

   Justification: Code and usage instructions are available through the PACE project page. The release covers a subset of the reported experiments.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: The main text and appendix specify the prediction horizon, execution baselines, PACE configuration, threshold calibration from demonstrations, seeds, episode counts, ablation settings, and real-robot trial counts. Appendix E.2 reports optimizer, batch size, learning-rate schedule, precision, checkpoint step, and hardware for real-robot fine-tuning.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No].

   Justification: The simulation results and phase-alignment analysis include uncertainty estimates (Appendices B.1.2, B.2, C.4, and D.3). The main real-robot results are reported without uncertainty estimates.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No].

   Justification: Appendix E.2 reports the GPU setup used for real-robot fine-tuning, and the simulation and test-time execution experiments were run on internal GPU servers with two Intel Xeon Platinum 8468V sockets, approximately 1.9 TiB system memory, and eight NVIDIA H20 GPUs with 97,871 MiB memory each. However, the current manuscript does not provide precise wall-clock runtimes, GPU-hours, or total compute budget for each experiment.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The work uses simulation benchmarks and controlled robot manipulation experiments, and does not involve private data, human-subject studies, or deceptive user-facing deployment. We have reviewed the NeurIPS Code of Ethics and are not aware of any deviation.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No].

    Justification: The current manuscript focuses on test-time execution for robot policies and does not include a dedicated broader-impact discussion. Potential positive impacts include more reliable robot execution without retraining, while negative impacts would mainly arise from deploying more capable robot policies without appropriate real-world safety controls.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: We release PACE code and task-specific RoboTwin policy checkpoints for manipulation research. The release does not include high-risk models or datasets.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No].

    Justification: The paper cites the original sources. OpenPI, LeRobot, GR00T, and AutoHorizon code use Apache License 2.0. AAC's LIBERO implementation, LIBERO code, and RoboTwin2.0 code/data use MIT. The PaliGemma backbone of $\pi_{0.5}$ is subject to the Gemma Terms of Use. GR00T N1.7 weights use the NVIDIA Open Model License Agreement, distinct from the code license. LIBERO demonstrations and DROID data use CC BY 4.0. We access DROID through the DreamZero conversion. RoboMIND 1.x, RoboMIND 2.0, and RoboCOIN data use Apache License 2.0. AgiBot World Beta data and RoboChallenge inference code use CC BY-NC-SA 4.0, with attribution and noncommercial-use conditions, and share-alike conditions when adaptations are shared. The consulted Table30 V1 demonstration releases do not specify a data license.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes].

    Justification: The released code includes usage and licensing documentation. The RoboTwin policy checkpoints are accompanied by a model card describing training and usage, together with the applicable model terms.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

Justification: The paper does not involve crowdsourcing, user studies, or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A].

Justification: The paper does not involve crowdsourcing, user studies, or research with human subjects, so IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes].

Justification: The paper studies vision-language-action robot policies and describes the use of language-conditioned robot policies such as $\pi_{0.5}$ throughout Sections 1, 2, 3, and 5. LLMs used only for writing, editing, or formatting are not material to the research claims.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.