ACRO: ACTOR–CRITIC ROLLOUT ORCHESTRATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Executing a vision-language-action (VLA) policy involves deciding when to continue an attempt and when to intervene to improve its chances of success. We introduce ACRO, an Actor–Critic rollout orchestration layer that makes these decisions for a fixed VLA. ACRO seeks interventions that offer a more promising continuation, are physically feasible, and leave sufficient time to complete the task. A Critic learned from policy rollouts estimates success prospects to decide when to intervene and select a re-entry target along the executed trajectory. Trajectory Retraction shortcuts the visited path to return the robot to that target, where the same VLA resumes execution from a fresh observation. We evaluate ACRO with GR00T N1.7 and $\pi_{0.5}$ across SimplerEnv, RoboCasa, and real-robot manipulation. Under the same physical execution budget, including all return motion, ACRO improves success by 10.4, 7.3, and 21.7 percentage points, respectively. It also completes additional tasks as the execution budget grows. These results show that controlling when and where a policy resumes can turn its existing capabilities into more successful task executions.

1 INTRODUCTION

Teaching a robot to act leaves open the question of when to trust its policy to continue and when to intervene. Vision-language-action (VLA) models map visual observations and language instructions to robot actions (Zitkovich et al., 2023; Kim et al., 2025; Black et al., 2025b). These models can succeed on a task in one execution and fail on the same task in another, suggesting that the outcome of a single execution may fall short of the policy's capabilities. Orchestration that translates this potential into successful task completion could improve the performance of an existing VLA without retraining the policy.

Effective orchestration requires two decisions: when to intervene, and how far that intervention should go. RoboMonkey and CoVer evaluate and select action proposals from the current observation (Kwok et al., 2025; 2026), while AutoHorizon adjusts how much of a predicted action chunk to execute (Wang et al., 2026). These decisions improve the policy's next execution, but evaluating an unfavorable proposal does not itself specify a physical intervention that would enable a better continuation. External orchestrators such as Pigey organize subgoals and skill calls, verify their outcomes, and apply retry and fallback rules (Galanti et al., 2026). In Pigey's VLA interface, however, the orchestrator reassesses execution after a bounded rollout returns. Such verification and routing rules do not directly determine when to interrupt an ongoing VLA execution or how much to change its state before returning control. Intervening only as much as is useful requires evaluating the policy's prospects both if the current action proceeds and if execution resumes after an intervention.

We introduce ACRO, an Actor–Critic rollout orchestration layer that makes these decisions according to the deployed VLA's probability of task completion. ACRO learns action- and state-value estimates from execution outcomes. The Critic evaluates the VLA's proposed action to determine when to interrupt execution. To turn this assessment into a tractable intervention, we introduce *Trajectory Retraction*, which reconstructs the observed execution path into a constrained region for intervention search. Within this region, ACRO uses state values to select a promising target reachable under the intervention budget. The Actor moves the robot to the selected configuration and returns control to the VLA, which observes the resulting state and generates a new continuation. Repeating this process within the remaining budget allows a minimal intervention layer to support alternative successful executions while leaving task execution to the existing policy.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

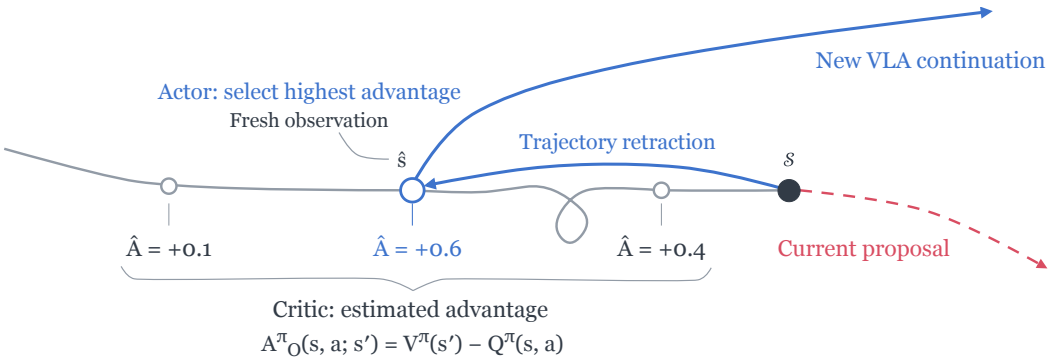


Figure 1: **Actor–Critic rollout orchestration.** The orchestration advantage $A^{\pi}_{\mathcal{O}}(s, a; s') = V^{\pi}(s') - Q^{\pi}(s, a)$ compares task-success probabilities after re-entry and after executing the current proposal, within the available time. Once intervention is activated, the Actor selects the recorded candidate with the highest estimated advantage; the current Q is fixed during this comparison. Trajectory Retraction shortens the return path, and the VLA resumes from a fresh observation.

We evaluate ACRO in simulation and on real-robot manipulation tasks. ACRO improves task success under the same execution budget, even when the physical steps used for trajectory retraction are included. These gains extend to the real robot, where success increases from 43.3% to 65.0%. As the execution budget grows, ACRO achieves further gains in task completion, demonstrating test-time scaling through rollout orchestration.

Our contributions are as follows:

- We formulate rollout orchestration through the difference in task-success probability between continuing a proposed action and resuming the policy after a physical intervention.
- We introduce ACRO, which uses a shared Q/V Critic to decide when to intervene and Trajectory Retraction to select and reach a promising continuation point.
- We show that ACRO improves success with fixed VLAs on SimplerEnv, RoboCasa, and real robots under matched physical execution budgets that include retraction.

2 RELATED WORK

Robot architectures and orchestration. Robot architectures such as CLARAty, CRAM, and hierarchical planning in the now connect planning to execution through runtime feedback and adjustment (Volpe et al., 2001; Beetz et al., 2010; Kaelbling & Lozano-Pérez, 2011). More recent systems organize learned robot capabilities: SayCan grounds skill selection in affordance values (Ichter et al., 2023), Inner Monologue incorporates execution feedback into language planning (Huang et al., 2023), and Pigey coordinates subgoals, skill calls, and retries (Galanti et al., 2026). These mechanisms support task-level coordination, but do not directly specify when to interrupt an ongoing policy action or how much physical intervention would improve its continuation.

Test-time control of VLA execution. Test-time methods improve deployed VLAs by controlling action selection and execution. V-GPS and RoboMonkey use learned evaluators to select action proposals (Nakamoto et al., 2025; Kwok et al., 2025), while CoVer jointly selects instructions and action chunks through alignment scoring (Kwok et al., 2026). AutoHorizon adjusts how much of a predicted chunk to execute before replanning (Wang et al., 2026). These methods refine execution from the current observation, leaving open how to select a physical intervention when continuing from that state offers poor prospects for success.

Failure detection and recovery. Runtime failure detection provides signals for intervention. SAFE predicts failure from VLA features (Gu et al., 2025), while FIPER combines observation novelty with action uncertainty (Römer et al., 2025). Such signals alone do not determine where the policy should resume. CycleVLA combines a progress-aware VLA with an external VLM that predicts failures and selects subtasks for backtracking, followed by minimum Bayes risk decoding

for retries (Ma et al., 2026). Recovery decisions therefore depend on an external VLM's assessment of subtask preconditions, whose connection to the deployed policy's actual ability to recover remains unclear. Moreover, organizing intervention around subtask boundaries makes its timing depend on the chosen task decomposition; these boundaries need not coincide with the moments when intervention would most benefit execution.

3 METHOD

3.1 PROBLEM FORMULATION

Goal Our goal is to improve a VLA's task success through orchestration. To be useful in real robotic deployments, orchestration must account for both the time required to intervene and the physical feasibility of the intervention. Meaning, interventions must be physically executable and timely: assessing the ongoing execution, selecting a re-entry target, and moving the robot must leave sufficient time for task completion. We therefore seek interventions that enable more promising policy continuations within the available time.

Objective Our goal can be viewed as finding an intervention whose resulting continuation offers an advantage over executing the currently proposed action. Let s denote the available execution context, including the observation history, task instruction, and elapsed time, and let $a \sim \pi(\cdot \mid s)$ denote the action proposed by the VLA. We define $Q^\pi(s, a)$ as the probability of task completion within the time limit after executing a and then following π, and $V^\pi(s)$ as the corresponding probability when π selects its continuation from s, following the standard state- and action-value definitions (Sutton & Barto, 2018). For an intervention that reaches s', including the elapsed time at policy resumption, we define its advantage over executing a as

$$A^\pi_{\mathcal{O}}(s, a; s') = V^\pi(s') - Q^\pi(s, a). \tag{1}$$

Here, $\mathcal{O}$ denotes an orchestration decision that changes where the VLA resumes execution.

Challenges However, realizing this advantage at test time is not straightforward. We must estimate whether continuing with the proposed action is likely to succeed. If the prospects are low, we need to find a state s' that offers a better continuation. Searching more broadly may uncover better targets, but takes time and can lead to states where value estimates are less reliable. We must then move the robot toward the selected target and leave enough time to complete the task. The scene may also change during the move, affecting the continuation we expected.

3.2 ACRO

Actor–Critic Orchestration. We implement the objective in Eq. 1 through an Actor–Critic structure. The Critic evaluates the success prospects of policy continuations, and the Actor uses these evaluations to select and execute a repositioning motion. Directly evaluating the orchestration advantage requires searching for alternative targets and assessing their continuation values. Doing so at every policy step would spend time even when the current execution is likely to succeed. We therefore adopt a sparse Actor–Critic structure: a Q-based assessment identifies when to activate the Actor, which then searches for a promising target using V. Once activated, the current action value is fixed, so maximizing V over candidate targets also maximizes their estimated orchestration advantage.

Evaluating Continuations. Repositioning and retries consume time that could otherwise be used to complete the task. A useful continuation must therefore offer a high probability of success within the available time. We learn Q and V from rollouts of the deployed VLA by modeling the time to task completion. Q conditions on the observation history and proposed action, while V uses the observation history alone. Both predict completion hazard rates, yielding success probabilities over a forecast horizon h:

$$\widehat{V}(s) = 1 - e^{-\int_0^h \lambda_V(u|s)\,du}, \qquad \widehat{Q}(s, a) = 1 - e^{-\int_0^h \lambda_Q(u|s,a)\,du}. \tag{2}$$

Here, λ_V and λ_Q are the predicted completion rates conditional on the task not yet being completed. We parameterize them as piecewise-constant rates over time intervals (Kvamme & Borgan, 2021).

The forecast horizon h is the interval over which the Critic predicts success; the episode budget counts all physical steps taken by the policy and by retraction.

These predictions must account for both observed successes and executions that end before success is observed. A successful rollout provides a completion time; an incomplete rollout indicates only that completion did not occur during the observed interval. We incorporate both through the censored time-to-success loss

$$\ell_{\text{surv}}(\lambda; d, \delta) = \int_0^d \lambda(u)\, du - \delta \log \lambda(d), \tag{3}$$

where d is the duration from a policy decision to completion or observation cutoff, and δ indicates whether completion was observed. We apply this loss to both Q and V.

The value of a proposed action should also agree with the continuation it produces. For an action interval of duration Δt without termination, the finite-horizon Bellman relation gives (Sutton & Barto, 2018)

$$Q_h^\pi(s, a) = \mathbb{E}\left[V_{h-\Delta t}^\pi(s') \mid s, a\right]. \tag{4}$$

We encourage this consistency by additionally training Q toward a stop-gradient V prediction at the next observed state, reducing the forecast horizon by the elapsed action duration. Observed success in the final action interval supplies a target of one. Censored endpoints without a subsequent observation are excluded from this additional bootstrap term.

Before target search, the alternative continuation value is unavailable. We therefore use Q to predict whether the current execution will fail within the task horizon. With the Critic fixed, we train a logistic classifier on recent Q estimates using rollout-level failure labels. Its prediction determines when to activate the Actor. This provides a proxy for identifying opportunities for positive advantage while reserving target search and physical repositioning for selected moments.

Value-Guided Repositioning. The Actor must find a promising target and reach it with enough time left for task completion. Searching more broadly may reveal better targets, but increases search time and exposes selection to value errors at unfamiliar states. We therefore use configurations recorded along the active execution trajectory as candidates. These records provide observations and value estimates for target selection, together with a path for guiding physical movement.

Among the eligible recorded contexts $\mathcal{R}$, the Actor selects the target with the highest estimated V. The recorded value serves as a proxy for the prospects of resuming from its associated robot configuration. To reduce movement time, Trajectory Retraction shortcuts the recorded path while keeping each segment near the visited positions. The target-selection rule and shortcut condition are

$$\hat{s} \in \arg\max_{s' \in \mathcal{R}} \widehat{V}(s'), \qquad \max_{p \in [p_i, p_j]} \operatorname{dist}(p, \mathcal{P}) \le \rho. \tag{5}$$

Here, $\mathcal{P}$ is the set of recorded end-effector positions along the selected trajectory segment, and ρ is the allowed deviation. After target selection, we greedily choose the farthest waypoint toward the target whose connecting segment satisfies the condition, approximated by sampling points along the segment. The Actor follows the resulting path, and the VLA receives a fresh observation of the reached state to generate a new continuation. Subsequent execution updates the active trajectory for later orchestration decisions. Algorithm 1 summarizes the online orchestration loop.

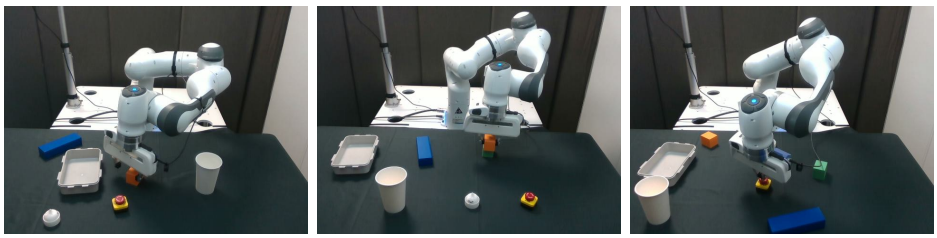
4 Experiments

We evaluate ACRO across pretrained policies, simulated environments, and a real robot. Section 4.2 examines its use across these execution settings and the data needed to prepare the Critic. Section 4.3 studies Critic-guided recovery and the efficiency of execution after returning. Section 4.4 examines test-time scaling and physical return cost. Appendix B examines Critic architectures and intervention triggers.

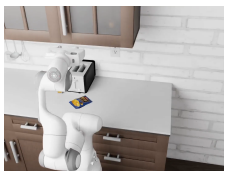
Algorithm 1 Actor–Critic Rollout Orchestration (ACRO)

Initialize trajectory history $\mathcal{B}$, Q history $\mathcal{Q}$, and elapsed steps $n \leftarrow 0$
while $n < T$ and the episode has not terminated **do**
  Observe s from the reached scene; sample $a \sim \pi(\cdot \mid s)$
  $q \leftarrow \widehat{Q}(s, a)$, $v \leftarrow \widehat{V}(s)$ ▷ Evaluate continuations
  Store (s, v) and the robot pose in $\mathcal{B}$; append q to $\mathcal{Q}$
  if $g(\mathcal{Q}) > \tau$ and re-entry is available **then**
    $\hat{s} \leftarrow \arg\max_{s' \in \mathcal{R}} \widehat{V}(s')$ ▷ Select an eligible recorded target
    $\mathcal{P} \leftarrow$ recorded path from the current pose toward $\hat{s}$
    Shortcut $\mathcal{P}$ within distance ρ of visited positions ▷ Trajectory Retraction
    Follow the shortened path; let Δn be the physical steps used
  else
    Execute a until the next query; let Δn be the steps used
  end if
  Update the active trajectory $\mathcal{B}$; $n \leftarrow n + \Delta n$
end while

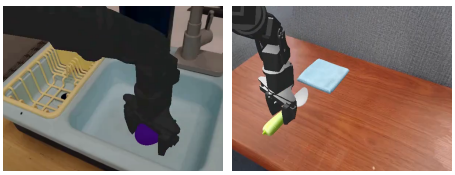
The budget T counts all physical steps; motion stops at termination or budget exhaustion. Re-entry availability includes a nonempty eligible set $\mathcal{R}$ and the intervention conditions in Appendix A.3–A.4.



(a) Real Robot



(b) RoboCasa



(c) SimplerEnv Bridge

Figure 2: Evaluation environments. (a) Real robot: pick and place, cube stacking, and button pressing. (b) RoboCasa (Nasiriany et al., 2024): cabinet interaction and pick and place. (c) SimplerEnv Bridge (Li et al., 2025): eggplant in basket and spoon on towel. Scenes are ordered left to right within each group; simulation images are from official project demonstrations.

4.1 Experimental Setup

Models and Datasets. We use two pretrained VLAs with their policies fixed throughout execution: GR00T N1.7 (NVIDIA, 2026) on SimplerEnv Bridge (Li et al., 2025), and $\pi_{0.5}$ (Black et al., 2025a) on RoboCasa (Nasiriany et al., 2024). SimplerEnv Bridge tests fine-grained tabletop manipulation, while RoboCasa covers a broader range of household object interactions. We also evaluate $\pi_{0.5}$ on a real robot to test recovery under physical sensing and actuation conditions. Figure 2 introduces the three evaluation environments.

Baselines. We compare three ways of using the execution budget: uninterrupted policy execution, value-ranked action selection, and recovery through physical repositioning. Base VLA executes the underlying policy without intervention. RoboMonkey (Kwok et al., 2025) selects among candidate actions at the current state. Our adaptation uses ACRO's Q estimate as its verifier to rank eight action chunks at each policy query. This compares using the same Critic for action selection with using it to guide ACRO's interventions, while requiring eight policy samples per query. CycleVLA (Ma et al., 2026) uses a VLM to decompose the instruction into subtasks, check completion at subtask boundaries, and return the robot to the start of a failed subtask before retrying. This provides a comparison with recovery guided by an external VLM. Our adaptation keeps the base policy fixed and omits the

Table 1: Task success rate (%) in simulation. All methods run the same base policy under the benchmark's standard physical step limit, including recovery motion: 300 steps on SimplerEnv Bridge and each task's registered limit on RoboCasa. Overall pools all trials of a benchmark. Best result per row in bold. †Adapted baselines; see Section 4.1.

			Baselines			
Policy	Task	N	Base	RoboMonkey†	CycleVLA†	ACRO
GR00T N1.7 SimplerEnv Bridge	Carrot on Plate	60	50.0	31.7	48.3	**68.3**
	Eggplant in Basket	60	36.7	50.0	**66.7**	51.7
	Spoon on Towel	60	**93.3**	90.0	**93.3**	**93.3**
	Stack Cube	60	18.3	18.3	13.3	**26.7**
	Overall	240	49.6	47.5	55.4	**60.0**
$\pi_{0.5}$ RoboCasa	Open/Close/Slide	250	64.4	66.8	65.6	**76.0**
	Pick-and-Place	250	64.0	**68.0**	56.0	65.2
	Control Operation	100	43.0	52.0	40.0	**55.0**
	Overall	600	60.7	64.8	57.3	**68.0**

original policy fine-tuning. On RoboCasa, Periodic rewind provides a fixed-schedule recovery baseline: it returns along the recorded path at fixed intervals to a fixed look-back distance. For real-robot experiments, we compare ACRO with Base VLA. Appendix A gives the implementation details of all methods.

Implementation Details. For a fair comparison, all methods use the same evaluation identities and receive the same physical step budget, which includes intervention motion. Additional policy and VLM queries do not count toward this physical step budget. We use each benchmark's standard episode limit: 300 steps on SimplerEnv, the official GR00T evaluation limit, and each task's registered limit on RoboCasa. To learn when and where to intervene, we train the Critic on uninterrupted rollouts of the base policy (1230 on SimplerEnv and 1200 on RoboCasa) using AdamW with a learning rate of 10^{-3}, and set the intervention threshold by calibrating on held-out rollouts so that about 20% of executions that would succeed raise an alarm. We measure task success over 240 trials on SimplerEnv, 600 on RoboCasa, and 60 on the real robot, pooling trials within each environment for overall performance. The scheduled-rewind and outcome analyses reuse all 600 RoboCasa evaluation trials at the same standard physical budget H. For the separate Critic and path ablations, we use $\pi_{0.5}$ on three RoboCasa tasks, one from each category: Turn On Electric Kettle (Control), Open Stand Mixer Head (Open), and Pick-and-Place Counter to Cabinet (Pick and Place).

4.2 Across Policies and Execution Environments

On Simulation Benchmarks. We evaluate ACRO across two pretrained VLAs and two simulation environments to test its use across different policies and manipulation settings. Table 1 covers tabletop manipulation with GR00T N1.7 on SimplerEnv Bridge and household interactions with $\pi_{0.5}$ on RoboCasa. ACRO raises overall success from 49.6% to 60.0% on SimplerEnv and from 60.7% to 68.0% on RoboCasa. It also exceeds RoboMonkey and CycleVLA overall in both settings. RoboMonkey ranks eight policy samples per query with the same Critic, while CycleVLA uses external VLM inference for recovery. With each base policy fixed and all return motion charged to the physical budget, these gains show that ACRO applies across distinct VLA backbones, environments, and task types.

Real-Robot Evaluation. Physical retraction must bring the robot back to a useful configuration despite sensing noise and actuation error. We test this requirement on a real robot using $\pi_{0.5}$ for pick and place, cube stacking, and button pressing (Table 2). ACRO improves success on all three tasks, with the largest gain on cube stacking (20.0% to 55.0%). Across 60 trials, it completes 39 tasks compared with 26 for Base VLA, raising success from 43.3% to 65.0%. The improvement under real-world execution conditions shows that physical return and renewed policy execution can be used together for successful recovery.

Table 2: Real-robot success rate (%).

Task	N	Base	ACRO
Pick and Place	20	70.0	80.0
Stack Cube	20	20.0	55.0
Press Button	20	40.0	60
Overall	60	43.3	65

Critic Training Cost. The Critic is trained from rollouts of the deployed policy. To measure the data needed to build it, we vary the training set from 8 to 60 seeds per task across 12 tasks, training one Critic with joint Q and V heads per size and fixing the validation and calibration sets at 180 and 300 rollouts. We evaluate on three RoboCasa tasks with 150 trials per task at $2H_{\text{ref}}$ (Appendix A.1). Figure 3 shows that ACRO exceeds Base VLA at every tested size. With 8 seeds per task, it reaches 76.7% success using 96 training rollouts; AUROC ranges from 0.897 to 0.921 across sizes. These results show that a small training set is sufficient to build a Critic that improves task completion.

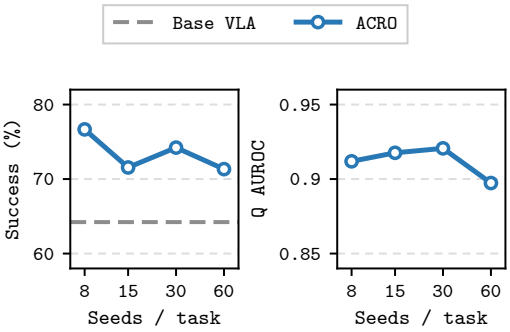


Figure 3: **A small rollout set suffices to train the Critic.** Success and AUROC of minimum rollout Q over three tasks at $2H_{\text{ref}}$.

4.3 Critic-Guided Recovery

The preceding results show that ACRO improves completion across the evaluated policies and environments. We now examine the role of its recovery strategy, the successes gained through intervention, and the execution that follows a return.

Scheduled Rewinds. A fixed rewind schedule also gives the policy repeated attempts. We compare this strategy with ACRO to examine the benefit of using Critic estimates to guide recovery. Periodic rewind uses fixed intervals, fixed look-back distances, and reverse replay, while ACRO uses Q to activate recovery and V to select a target reached through Trajectory Retraction. On the same 600 RoboCasa trials at the standard physical budget H, Periodic completes 59.5% of trials, compared with 60.7% for Base VLA and 68.0% for ACRO (Table 3). Periodic and ACRO initiate similar numbers of returns per trial (1.04 and 1.06), while ACRO uses about half as many return steps (46.3 versus 92.7). The Critic-guided strategy thus turns a similar number of recovery attempts into more completed tasks with less return motion.

Table 3: **ACRO improves on scheduled rewinds.** All 600 RoboCasa trials at H; returns and return steps are trial means.

Method	Success (%)	Returns	Steps
Base VLA	60.7	0	0
Periodic	59.5	1.04	92.7
ACRO	68.0	1.06	46.3

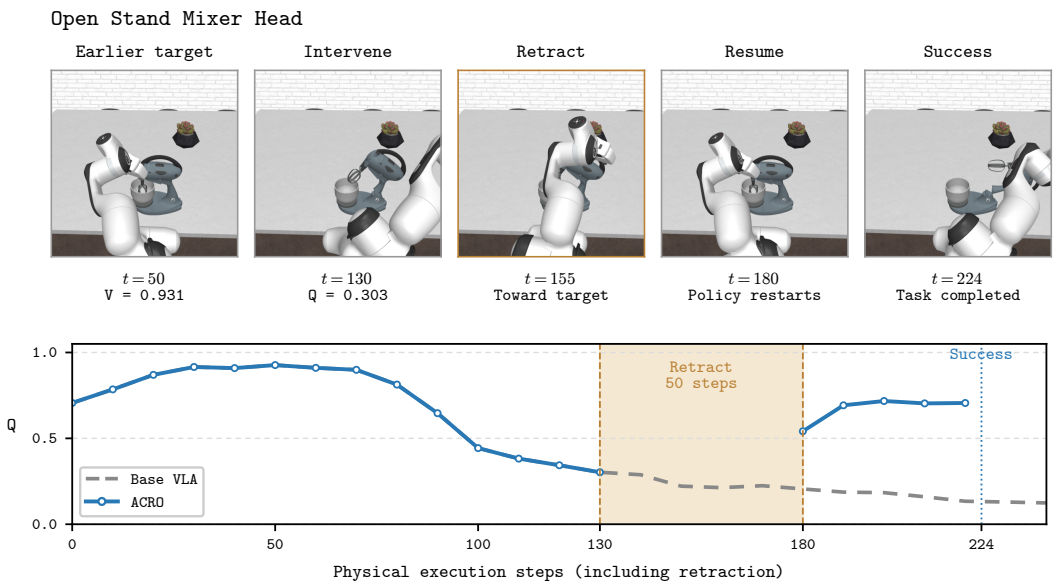


Figure 4: **Retraction enables a successful re-approach.** ACRO intervenes at step 130, spends 50 steps returning toward the target at step 50, and succeeds at step 224; paired Base VLA times out at 1,000. Q curves show the first 240 steps, with no queries during retraction. Frames replay recorded actions, verified against per-step success flags.

Gained and Lost Successes. An intervention can recover a failed execution, but it can also interrupt one that would succeed. To examine this balance, Table 4 decomposes the outcome differences over all 600 RoboCasa task–seed identities at H. ACRO succeeds on 70 seeds where Base fails and fails on 26 where Base succeeds, giving 44 additional completions. Periodic rewind gains 50 successes and loses 57. ACRO's net gain is positive in every category: +29 on open/close, +12 on control, and +3 on pick and place. Thus, the overall improvement reflects more gained than lost successes across the three categories. The comparison protocol is detailed in Appendix D.

Table 4: **Gained successes exceed losses.** All 600 RoboCasa seeds at H, relative to Base. Policy-step means use the 46 shared successes after ACRO's first alarm.

	Outcome changes		Mean rollout steps	
Category	Gained	Lost	Base	ACRO
Open/close	33	4	362.6	296.1
Pick/place	17	14	387.0	345.5
Control	20	8	139.3	157.7
Overall	**70**	**26**	**327.9**	**287.3**

Recovery through a New Approach. Figure 4 shows how a return can lead to a successful continuation on Open Stand Mixer Head. ACRO intervenes at step 130 as estimated continuation success declines, then spends 50 steps returning toward an earlier target. The policy resumes and opens the mixer head after another 44 steps. Base VLA continues with low Q estimates and fails within 1,000 steps. In this example, returning to a previously visited state allows the frozen policy to make a new approach and complete the task.

Execution Efficiency after Returning. Returning moves the robot to an earlier point along its path. If the policy then needs fewer execution steps to finish, that return has provided a more productive starting point for the next attempt. We examine the 46 trials in which both Base and ACRO succeed after ACRO's first-alarm step. From that step onward, Base uses 327.9 policy steps on average and ACRO uses 287.3, counting every retry (Table 4). The reduction comes from open/close and pick-and-place tasks; control tasks use more policy steps under ACRO. Including return motion gives ACRO 339.9 physical steps. The lower policy-step count shows how execution after returning can reach success with fewer policy execution steps, while the physical total accounts for the movement needed to reach the re-entry state.

4.4 Test-Time Scaling and Retraction Cost

ACRO uses additional execution time for renewed attempts. We examine how task completion scales with this budget and how Trajectory Retraction reduces the steps spent returning between attempts.

Test-Time Scaling. Additional execution time gives ACRO more opportunities to return and try again. We examine test-time scaling by increasing the physical execution budget and measuring how many additional tasks are completed. Figure 5 scores the same 240 SimplerEnv executions at successive step limits up to 300, including all retraction steps. Within the first 100 steps, ACRO and Base VLA complete 101 and 94 trials. Over the remaining 200 steps, ACRO adds 43 completions and Base adds 25. On Eggplant in Basket, Base adds no success after step 100 while ACRO adds five; task-level curves appear in Fig. 7. Thus, ACRO converts additional execution budget into further successes through renewed attempts.

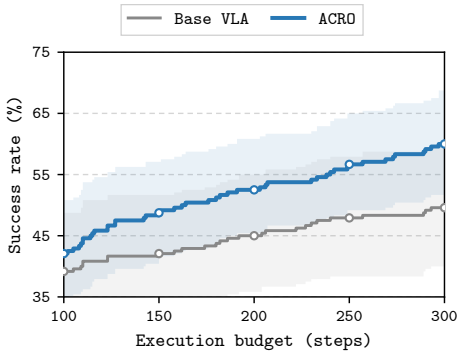


Figure 5: **Success grows with budget.** Bands: 95% bootstrap intervals.

Retraction Path. Replaying the recorded path spends steps on detours made during the original attempt. We compare reverse replay with an RDP-simplified path, a shortcut within a tube around visited positions, and a straight line to the target. All variants use three RoboCasa tasks with 150 trials per task at $2H_{\text{ref}}$ (Appendix A.1). Table 5 shows that the visited-tube shortcut reduces the mean return from 62.0 to 32.4 steps while retaining similar target reach (86.2% versus 85.4%). A straight line is shorter (20.6 steps), but reaches the target in only 64.1% of returns. Task success stays between 70.7% and 71.3% across paths. The visited-tube shortcut therefore saves return steps while preserving reliable access to the target.

Table 5: **Shortcuts reduce return cost.** Paths at $2H_{\text{ref}}$: success (%), mean return steps, and target reach (%).

Path	Success	Steps	Reach
Reverse replay	70.7	62.0	86.2
RDP-simplified	70.9	43.4	80.5
Straight line	70.7	20.6	64.1
Ours	71.3	32.4	85.4

5 Conclusion

We introduced ACRO, an Actor–Critic rollout orchestration layer that improves task completion by controlling when to intervene and where a fixed VLA resumes execution. The Critic guides recovery toward a promising continuation, and Trajectory Retraction makes the physical return efficient by using the visited path. Across two VLA backbones, simulation benchmarks, and real-robot manipulation, ACRO improves success under the same physical budget, including return motion. Additional execution time enables further successes through renewed attempts. These findings show that evaluating and redirecting an ongoing execution can make better use of a learned policy's capabilities, connecting execution-time orchestration to more reliable task completion.

AI USE STATEMENT

We used ChatGPT to assist with manuscript drafting and revision, discussion of mathematical formulations, literature discovery, and code for figures and tables. We did not use ChatGPT for method or experimental design, implementation or debugging of research code, or interpretation of experimental results. The authors reran the experiments to check the reported results and take responsibility for the final content of this work, including all text, claims, and artifacts produced with AI assistance.

ETHICS STATEMENT

This study did not involve human-subject research or the collection of personal data. Real-robot experiments were conducted in a separated workspace under researcher supervision, with an emergency stop immediately accessible to the supervising researcher.

REPRODUCIBILITY STATEMENT

Section 4.1 describes the policies, environments, baselines, and execution budgets used in our experiments. Appendix A provides the evaluation protocol, Critic data splits and training settings, intervention rules, retraction procedure, and baseline implementations. Appendix C describes the task-level budget analysis and resampling procedure. Code is not included with this submission. Upon acceptance, we plan to release the code, trained Critic checkpoints, and experimental configurations to support reproduction of the results.

REFERENCES

Michael Beetz, Lorenz Mösenlechner, and Moritz Tenorth. CRAM—a cognitive robot abstract machine for everyday manipulation in human environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010. doi: 10.1109/IROS.2010.5650146. URL https://www.open-ease.org/papers/beetz10cram.pdf.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025a. URL https://proceedings.mlr.press/v305/black25a.html.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, 2025b. doi: 10.15607/RSS.2025.XXI.010.

Liane Galanti, Dhruv Shah, and Tri Dao. Addressing the orchestration gap in generalist robots via physical agency, 2026. URL https://arxiv.org/abs/2607.21725.

Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. SAFE: Multitask failure detection for vision-language-action models. In *Advances in Neural Information Processing Systems*, volume 38, pp. 40041–40076. Curran Associates, Inc., 2025. doi: 10.52202/085713-1337. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/392d0d05e2f514063e6ce6f8b370834c-Paper-Conference.pdf.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda

Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 287–318. PMLR, 2023. URL https://proceedings.mlr.press/v205/ichter23a.html.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *IEEE International Conference on Robotics and Automation*, 2011. URL https://people.csail.mit.edu/tlp/pdf/2011/hpnICRA11Final.pdf.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Håvard Kvamme and Ørnulf Borgan. Continuous and discrete-time survival prediction with neural networks. *Lifetime Data Analysis*, 27(4):710–736, 2021. doi: 10.1007/s10985-021-09532-6. URL https://link.springer.com/article/10.1007/s10985-021-09532-6.

Jacky Kwok, Christopher Agia, Rohan Sinha, Matt Foutter, Shulu Li, Ion Stoica, Azalia Mirhoseini, and Marco Pavone. RoboMonkey: Scaling test-time sampling and verification for vision-language-action models, 2025. URL https://arxiv.org/abs/2506.17811.

Jacky Kwok, Xilun Zhang, Mengdi Xu, Yuejiang Liu, Azalia Mirhoseini, Chelsea Finn, and Marco Pavone. Scaling verification can be more effective than scaling policy learning for vision-language-action alignment. In *Computer Vision – ECCV 2026*, volume 17071 of *Lecture Notes in Computer Science*, pp. 37–55. Springer, 2026. doi: 10.1007/978-3-032-37016-7_3. URL https://link.springer.com/chapter/10.1007/978-3-032-37016-7_3.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3705–3728. PMLR, 2025. URL https://proceedings.mlr.press/v270/li25c.html.

Chenyang Ma, Kai Lu, Guangyu Yang, Jiuming Liu, Shitong Xu, Bill Byrne, Ioannis Havoutis, Niki Trigoni, and Andrew Markham. CycleVLA: Proactive self-correcting vision-language-action models via subtask backtracking and minimum bayes risk decoding, 2026. URL https://arxiv.org/abs/2601.02295.

Mitsuhiko Nakamoto, Oier Mees, Aviral Kumar, and Sergey Levine. Steering your generalists: Improving robotic foundation models via value guidance. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 4996–5013. PMLR, 2025. URL https://proceedings.mlr.press/v270/nakamoto25a.html.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-Scale Simulation of Household Tasks for Generalist Robots. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024. doi: 10.15607/RSS.2024.XX.050.

NVIDIA. Isaac GR00T N1.7. Software and model release, 2026. URL https://github.com/NVIDIA/Isaac-GR00T. Accessed September 25, 2026.

Ralf Römer, Adrian Kobras, Luca Worbis, and Angela Schoellig. Failure prediction at runtime for generative robot policies. In *Advances in Neural Information Processing Systems*, volume 38, pp. 7631–7670. Curran Associates, Inc., 2025. doi: 10.52202/085713-0260. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/0b7cb3b8cc44e652761245537027db44-Paper-Conference.pdf.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018. URL https://mitpress.mit.edu/9780262039246/reinforcement-learning/.

Richard Volpe, Issa Nesnas, Tara Estlin, Darren Mutz, Richard Petras, and Hari Das. The CLARAty architecture for robotic autonomy. In *IEEE Aerospace Conference Proceedings*, 2001. URL https://www-robotics.jpl.nasa.gov/media/documents/01_volpe_claraty_aerospace.pdf.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Ramana Rao Kompella, and Gaowen Liu. VLA knows its limits: Adaptive execution horizons for robot policies. In *Computer Vision – ECCV 2026*, volume 17064 of *Lecture Notes in Computer Science*, pp. 607–623. Springer, 2026. doi: 10.1007/978-3-032-37196-6_33. URL https://link.springer.com/chapter/10.1007/978-3-032-37196-6_33.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

Table 6: RoboCasa tasks and their registered episode limits.

Category	Task	Limit (steps)
Open/Close/Slide	Close Fridge	1200
	Open Cabinet	1400
	Open Drawer	1000
	Open Stand Mixer Head	600
	Slide Dishwasher Rack	600
Pick-and-Place	Counter to Cabinet	1000
	Counter to Stove	800
	Drawer to Counter	1000
	Sink to Counter	1200
	Toaster to Counter	800
Control Operation	Turn On Electric Kettle	600
	Turn On Sink Faucet	800

A Implementation Details

This section describes the evaluation protocol, the components of ACRO, and our instantiation of the baselines. Table 7 summarizes the hyperparameters.

A.1 Evaluation Protocol

Pairing. For each trial, methods use the same task, environment seed, and policy noise seed. The RoboCasa analysis retains all 600 task–seed identities; initial-observation and action-prefix checks are reported in Appendix D. Action selection can change a baseline trajectory from its first policy query; matching initial conditions does not imply identical action prefixes. Budgets count physical steps, including the steps spent moving the robot during an intervention. Additional policy or VLM queries are not charged.

SimplerEnv Bridge. We use the official GR00T N1.7 SimplerEnv Bridge checkpoint and its four standard tasks. The initial configurations of each task, defined by object placement and orientation, are split into disjoint sets for Critic training, Critic validation, threshold calibration, and evaluation, in proportion 5:2:2:3; all orientations of a placement fall in the same set. We evaluate on six held-out configurations per task, with ten trials per configuration (60 trials per task, 240 in total). For Eggplant in Basket, which has nine held-out configurations, we use the first six. The policy is queried every 4 steps and executes the first 4 steps of each 16-step action chunk. The budget is 300 steps, the limit of the official GR00T evaluation.

RoboCasa. We use the $\pi_{0.5}$ RoboCasa checkpoint released with LeRobot and 12 atomic tasks, evaluated on 50 environment seeds each (600 trials in total). Critic training uses a disjoint range of environment seeds. The evaluation budget H is each task's registered episode limit, listed in Table 6. RoboCasa and the Bridge tasks other than Eggplant in Basket were run with a longer budget and scored at the standard limit. We retain the original fixed intervention schedules and score the saved per-step success traces up to H, including all executed recovery motion.

Analysis and ablations. The Critic and path ablations use three RoboCasa tasks, one per category: Turn On Electric Kettle, Open Stand Mixer Head, and Counter to Cabinet, each on 150 environment seeds disjoint from the Critic's (450 paired trials per variant). The reference horizon H_{ref} of each task is the 95th percentile of the lengths of successful Base VLA executions on the Critic's seeds, rounded up to a multiple of 50 steps (550, 500, and 850 steps), and every variant has a budget of $2H_{\mathrm{ref}}$ (1,100, 1,000, and 1,700 steps). Since a variant produces the same actions as Base VLA until its first intervention, we replay its trigger on the recorded Base VLA executions and simulate only the trials in which it fires; the other trials keep the Base VLA outcome.

A.2 CRITIC

Data. The Critic is trained on uninterrupted rollouts of the base policy: 1,230 on SimplerEnv (780 for training, 150 for validation, 300 for calibration) and 1,200 on RoboCasa (720, 180, and 300). No rollout from the evaluation placements or seeds is used.

Architecture. At each query, the input is the mean-pooled output of the VLA backbone concatenated with the robot's proprioceptive state. A linear projection with layer normalization embeds each of the last eight inputs, and a single-layer GRU summarizes them. The V head predicts the value of the current state from this summary. The Q head adds a residual, computed from the summary and a projection of the proposed action chunk, to the V prediction. Both heads output hazard rates of a piecewise-exponential survival model over the next steps, with bin edges at $\{0, 10, 25, 50, 100, 200, 400, 600\}$, so that the success probability can be read at any horizon.

Training and selection. We train with AdamW (learning rate 10^{-3}, weight decay 10^{-3}, batch size 32, gradient clipping at 5) on the censored negative log-likelihood of both heads, for up to 120 epochs with early stopping after 12 epochs without improvement. We train eight variants (MLP or GRU summary, width 64 or 128, input noise 0 or 0.03, two seeds) and select the one with the lowest validation likelihood loss. The selected Critic is a GRU of width 64 on SimplerEnv and of width 128 on RoboCasa. On validation rollouts, the minimum Q over a rollout separates eventual successes from failures with an AUROC of 0.94 and 0.97, respectively.

A.3 INTERVENTION TRIGGER

Rather than thresholding a single Q value, ACRO scores the last eight Q values with a logistic regression that predicts whether the execution will fail. The regression is fit on the validation rollouts with an L2 penalty of 10^{-3}. Its threshold τ is set on the calibration rollouts as the $(1 - \alpha)$ quantile of the maximum score reached by each successful rollout, with $\alpha = 0.2$, so that about one in five executions that would succeed raises an alarm. On the calibration rollouts, the trigger detects 76% of failures on SimplerEnv and 85% on RoboCasa. After an intervention, the trigger re-arms once two consecutive queries rise at least 0.02 above the threshold, or after 16 consecutive queries below it. The Critic is queried at every policy query on SimplerEnv and every 10 steps on RoboCasa.

A.4 RETRACTION AND RE-ENTRY

Target selection. The k-th intervention in an execution searches a look-back window of the executed path and moves the robot to the state with the highest predicted value in that window. The windows grow with k: 24 steps, 60 steps, and then the start of the execution on SimplerEnv, and 120 steps, 300 steps, and then the start on RoboCasa.

Path. Trajectory Retraction replaces the recorded end-effector path with straight segments that stay within 2 cm of positions the robot has already visited, which removes loops and hesitations while staying in explored space. The robot follows this path with end-effector pose commands, clipped per step to 3 cm and 0.3 rad on SimplerEnv and to 5 cm and 0.5 rad on RoboCasa. The gripper keeps its last commanded state, the mobile base and torso stay fixed, and objects are not reset.

Limits. On RoboCasa, ACRO intervenes at most three times per execution, and a single retraction is capped at 200 steps. On SimplerEnv, the number of interventions is bounded only by the budget. The action queue is cleared after retraction so that the next proposal uses the reached observation.

A.5 BASELINES

Periodic rewind. This baseline uses a reference horizon H_{ref} fixed from development rollouts. At each 10-step query, it triggers once $\lfloor H_{\mathrm{ref}}/2 \rfloor$ physical steps have elapsed since the preceding return ended, or since episode start for the first return. It selects the recorded position $\lfloor H_{\mathrm{ref}}/4 \rfloor$ logical forward steps back, bounded by episode start, and follows the recorded path in reverse. The original intervals and offsets are held fixed when scoring at the standard evaluation budget H. In the

Table 7: Hyperparameters of ACRO and the baselines.

	SimplerEnv	RoboCasa
Base policy	GR00T N1.7	$\pi_{0.5}$
Step budget	300	registered limit (Table 6)
Critic		
Rollouts (train / validation / calibration)	780 / 150 / 300	720 / 180 / 300
Summary network	GRU, width 64	GRU, width 128
History length	8 queries	8 queries
Query interval	4 steps	10 steps
Optimizer	AdamW, lr 10^{-3}, weight decay 10^{-3}, batch 32	
Trigger		
History length	8 Q values	8 Q values
α	0.2	0.2
τ	2.11	1.48
Retraction		
Look-back windows	24, 60, start	120, 300, start
Path tube radius	2 cm	2 cm
Max. interventions	–	3
Baselines		
RoboMonkey candidates	8	8
CycleVLA VLM / retries	Qwen3.8-Flash / 2	Qwen3.8-Flash / 2

task order of Table 6, the intervals are 500, 500, 250, 250, 150, 425, 325, 475, 375, 325, 275, and 200 steps. Its trigger, target, and path are fixed rules, so the comparison measures the full recovery strategy. The policy weights and physical-step accounting are shared with ACRO.

RoboMonkey. At each policy query, we draw eight action chunks: the chunk that Base VLA would execute and seven more sampled with different noise seeds. The ACRO Critic scores each chunk as the Q value of executing it from the current state history, and the robot executes the chunk with the highest score. On SimplerEnv, a sampled chunk replaces the default only if its score is strictly higher. The original RoboMonkey trains a VLM-based verifier on preference data. Using our Critic instead lets RoboMonkey and ACRO rely on the same value estimates. RoboMonkey queries the policy eight times as often as Base VLA.

CycleVLA. A VLM (Qwen3.8-Flash) decomposes each instruction once into a short sequence of subtasks, using a fixed vocabulary of verbs (move, rotate, open, close, grasp, lift, place, release). We freeze these decompositions before evaluation, so every trial of a task uses the same subtasks. A change in the sign of the gripper command that persists for two steps marks the end of a subtask. At the next policy query, the VLM receives the instruction, the subtask list, and the current camera image, and returns whether the subtask succeeded and, if not, which subtask to resume from. On a failure, the robot moves back along its recorded path to the start of that subtask and the policy continues from there. Each subtask is retried at most twice and is then treated as complete. If the VLM does not return a valid answer, execution continues without intervention. Unlike the original work, we do not fine-tune the policy with a progress head, since every method keeps the base policy fixed. The SimplerEnv decompositions are: Carrot on Plate, (grasp carrot, move carrot to plate, place carrot on plate); Spoon on Towel, (grasp the spoon, place the spoon on the towel, release the spoon); and four subtasks each for Eggplant in Basket and Stack Cube.

B Ablation Studies

Overview. We vary the Critic architecture and intervention trigger, changing one component at a time while keeping the rest at the default configuration. All variants use the same three RoboCasa analysis tasks and $2H_{\text{ref}}$ budgets as the Critic training-data experiment in Section 4.2 and the retraction experiment in Section 4.4.

Table 8: Critic architecture. The MLP and KAN read only the latest query; the GRU and Transformer read the last eight. Success rate (%), averaged over three tasks.

Critic	Params	Q AUROC	Success
GRU (ours)	0.44M	0.897	71.3
MLP	0.35M	0.886	71.1
KAN	1.17M	0.939	77.6
Transformer	0.62M	0.915	73.6

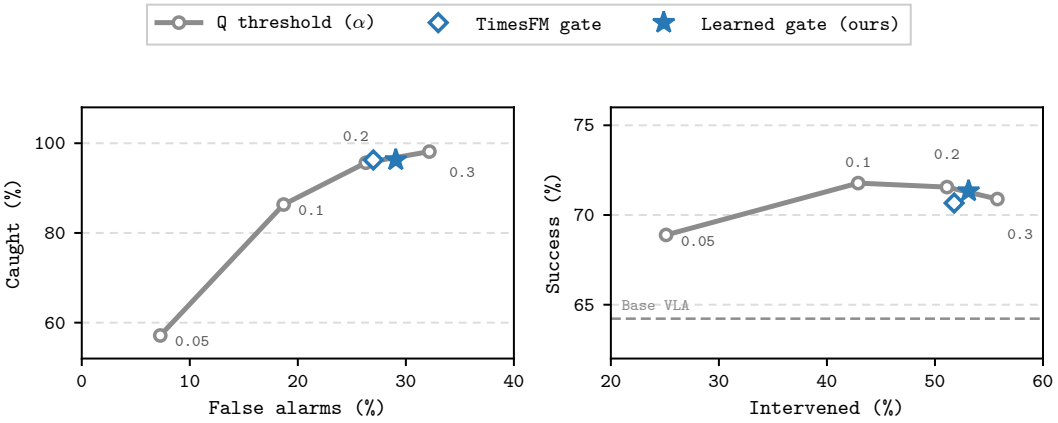


Figure 6: **Intervention trigger.** Gray points threshold Q at conformal level α; both gates are calibrated at $\alpha = 0.2$. Failures caught and false alarms are the fractions of executions that fail or succeed under Base VLA in which the trigger intervenes. Averaged over three tasks (450 paired trials).

Critic Architecture. We also test whether ACRO depends on a particular Critic design. We replace the GRU with an MLP and a KAN that read only the latest query, and with a Transformer over the same eight-query window, all reading the same policy features. Table 8 shows that every Critic improves over Base VLA and that success follows the Critic's AUROC. The MLP matches the GRU (71.1% versus 71.3%), the Transformer reaches 73.6%, and the KAN attains both the highest AUROC (0.939) and the highest success (77.6%), gaining 63 trials and losing 35 relative to the GRU. ACRO improves success with every tested architecture; across these fits, higher Critic AUROC is associated with higher task success.

Intervention Trigger. Finally, the trigger trades missed failures against unnecessary interruptions of executions that would have succeeded. We vary the conformal level α of a Q threshold and compare it with our learned gate on the recent Q history and with a gate that forecasts Q using TimesFM. Figure 6 plots the fraction of failures each trigger catches against its false-alarm rate, and success against how often it intervenes. The learned gate (96.3% with 29.1%) and the TimesFM gate (96.3% with 27.0%) fall on the same trade-off as the threshold at $\alpha = 0.2$ (95.7% with 26.3%). Success saturates once most failures are caught: every trigger with $\alpha \geq 0.1$ reaches 70.7% to 71.8%, whereas $\alpha = 0.05$ reaches 68.9%. Thus, on these tasks, how many failures the trigger catches matters more than whether it reads the level or the trend of Q.

C Success over the Execution Budget

The curves in Fig. 7 disaggregate Fig. 5 by task. Every method is evaluated on the same 60 trials per task and all physical movement counts toward the 300-step budget. The shaded intervals resample object placements, retaining all ten trials of each sampled placement.

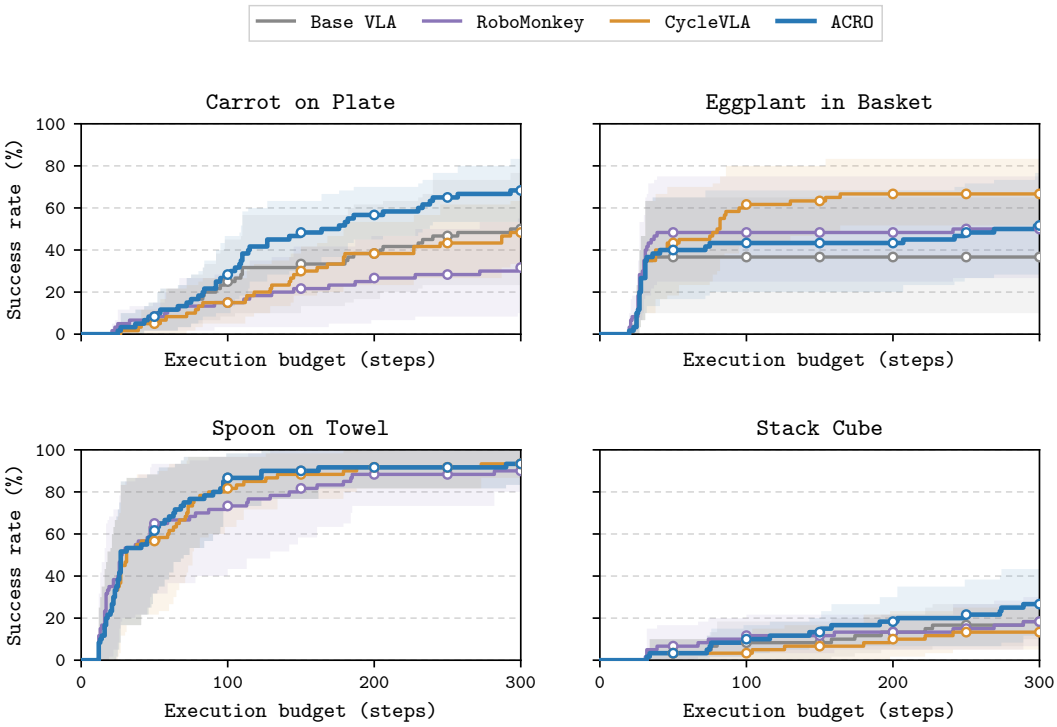


Figure 7: **Success over the execution budget by task.** Sixty paired trials per task on SimplerEnv Bridge, comparing Base VLA, RoboMonkey, CycleVLA, and ACRO. All panels use the same success-rate scale. Bands show 95% bootstrap intervals over object placements.

D Recovery Outcomes and Execution Cost at the Standard Budget

Cohort and budget. This analysis uses all 600 RoboCasa task–seed identities from the main evaluation: 12 tasks and 50 environment seeds per task. Each execution is scored at the task's registered physical-step limit H in Table 6. Success is read from the saved per-step environment success array. All 3,000 records across the five methods cover their scoring horizon or succeed before it, and their saved terminal success flags agree with the full success arrays. The source manifest records commit `f2493a4` and the frozen `lerobot/pi05_robocasa` policy. The original fixed schedules are preserved during prefix scoring. The separate three-task ablations use their original longer budgets.

Outcome accounting. ACRO completes 408 trials, Base 364, and Periodic rewind 357. Relative to Base, ACRO gains 70 successes and loses 26; Periodic gains 50 and loses 57. Thus, the net changes are $+44$ and -7. The ACRO comparison contains 338 shared successes and 166 shared failures. These four outcome groups sum to 600 without excluding any identity.

Pairing checks. Task and policy seeds are shared, but exact observations and action prefixes are not identical for every execution. Compared with Base, 45 ACRO records and 60 Periodic records have different initial-observation hashes. Action arrays before the first changed action differ for 57 ACRO and 75 Periodic records. The full 600-identity outcome comparison retains these records and describes observed same-seed differences.

Timing after the first alarm. ACRO initiates a return before success or cutoff on 284 of the 600 identities. Of the 338 shared successes, 46 succeed in both executions after the step index of ACRO's first alarm. We use that index as the time origin and subtract all ACRO return actions before success to obtain policy execution steps. Every retry remains in the count. This gives 327.9 Base policy steps and 287.3 ACRO policy steps on average; including return motion gives 339.9 ACRO physical steps. The median paired policy-step difference is -5 steps. Four of these 46 records have

Table 9: Shared successes after ACRO's first alarm at H. Means count every policy attempt; ACRO physical steps also include return motion.

		Policy steps		Physical steps
Category	N	Base	ACRO	ACRO
Open/Close/Slide	20	362.6	296.1	365.2
Pick-and-Place	17	387.0	345.5	391.2
Control	9	139.3	157.7	187.0
Overall	46	327.9	287.3	339.9

different initial-observation hashes and five have different action prefixes, so the timing comparison is descriptive. It conditions on shared success and uses no exclusion based on the pairing checks.

Physical execution cost. Return actions count against H. For each return event, we intersect the interval from its first action through its last executed action with the interval before the first success or H. This also includes unfinished returns at the cutoff. Averaged over all 600 trials, Periodic initiates 1.04 returns and spends 92.7 steps returning; ACRO initiates 1.06 and spends 46.3 steps returning. Total executed physical steps average 553.6 for Base, 595.9 for Periodic, and 528.4 for ACRO. These totals include successes and failures; they describe physical execution, without measuring inference latency or wall-clock duration.