CO-EVOLVING ROBOT ORCHESTRATORS AND POLICIES THROUGH DEPLOYMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) policies trained on large datasets are capable within their training domains, yet they still fail to generalize to the variety of situations a robot meets in real-world deployment. Agentic robot systems complement the policy with a vision-language model (VLM) orchestrator that learns when to call the policy, how to instruct it, and when to use scripted skills instead. However, because the harness is built around a frozen policy that has limited language steerability, the orchestrator can avoid the policy's failures but never overcome them. The policy becomes the bottleneck of the whole system. Fine-tuning the policy can remove this bottleneck, but updating it alone decouples it from an orchestrator tuned to its old behavior. We propose ROBO-COP, in which the orchestrator and policy co-evolve during deployment. ROBO-COP curates skill demonstrations from its own executions, fine-tunes the policy when this data can address recurring failures, and adopts each new policy only after it improves the skills it was trained for. Across ten simulated RoboLab tasks, ROBO-COP raises mean held-out success from 64.8% to 73.8% over the same harness with a frozen policy, while fine-tuning on a fixed schedule without verification reaches only 65.8%. On three real-world tasks, ROBO-COP raises held-out success from 38.3% to 50.0%. ROBO-COP turns deployment into a self-improving flywheel in which robots learn by doing, with each improvement in execution producing better data for the next round of learning. Videos and code are available at `https://robo-cop.pages.dev/`.

1 INTRODUCTION

Large pre-collected datasets have made robot policies far more general (Kim et al., 2024; Black et al., 2025; Physical Intelligence et al., 2025; Khazatsky et al., 2024), but no dataset can enumerate the diverse situations a robot will face once deployed, and even strong policies can degrade under distribution shift (Fei et al., 2025; Chen et al., 2026c). Agentic robot systems handle such distribution shifts at the orchestration level, composing existing capabilities in new ways and adapting during deployment without retraining. In these systems, a language model writes code that combines learned policies with scripted motion primitives, covering cases where the policy alone may fail (Ahn et al., 2022; Liang et al., 2023; Xu et al., 2023; Huang et al., 2022).

Such systems improve by reflecting on failures during deployment (Liu et al., 2023; Zha et al., 2024; Lu et al., 2026) and learning when to delegate to a vision-language-action (VLA) policy (Galanti et al., 2026; Huang et al., 2026; Zhang et al., 2026). Over many episodes, the harness adapts to the current policy, learning which parts of a task to give the policy and which to handle with scripted skills. The orchestrator can rephrase instructions or add supporting motions, but VLAs have limited language steerability (Fei et al., 2025; Kwok et al., 2026), so the harness can route around the policy's failures rather than fix them, and the policy caps what the whole system can achieve.

Updating the policy from deployment experience could raise this ceiling (Sukhija et al., 2026), but inside an agentic system it poses three problems. First, the training data must come from deployment itself, and those executions interleave policy and scripted actions. An episode can also fail after several sub-steps succeeded, such as dropping a block that was grasped correctly, and training only on successful episodes discards those correct sub-steps. Second, a fine-tuned checkpoint may be worse than the one it replaces. Third, updating the policy alone decouples it from the orchestrator,

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.

whose memory, such as advice to grasp a block with scripted motion because earlier VLA grasps failed, was formed around the old checkpoint. Prior harnesses assume a fixed policy (Galanti et al., 2026; Huang et al., 2026) and therefore do not face these problems. Once the policy changes, this coupling must become dynamic, keeping the orchestrator and policy matched as both improve. We therefore ask: *how can the orchestrator and its policy co-evolve during deployment?*

We introduce ROBO-COP (Co-evolving Orchestrator and Policy), which updates both orchestrator and policy during deployment (Figure 1). It turns its own deployment executions into new training data for the policy by splitting each episode into skills and keeping every skill that succeeded, even when the episode as a whole failed, and it decides when this data warrants training. After fine-tuning, it tests each candidate policy during deployment, adopting it if it improves the skills it was trained for and reverting to the current policy otherwise. Either way, it flags memory advice against using the policy as outdated, so the orchestrator can retry the policy on skills it had previously avoided. Unlike skill discovery methods that spend a separate budget on exploration (Lu et al., 2026), ROBO-COP learns entirely during deployment, so every trial is an attempt at the task. We evaluate on ten simulated tasks chosen to span three regimes: tasks the pretrained VLA solves alone, tasks that scripted primitives solve, and tasks that need both. We also test the full loop on three real-world tasks.

Our contributions are fourfold:

- **Co-evolving orchestrator and policy.** We frame deployment-time learning as improving an agentic harness and its policy together.
- **Agentic data curation.** We turn deployment executions into skill demonstrations without manual labels, keeping successful skills even from failed episodes, and train only when the data warrants it.
- **Skill verification and memory revision.** We verify each policy candidate before adoption and revise policy-dependent memory after every update.
- **Performance gains in simulation and the real world.** ROBO-COP is competitive on all three regimes, raising mean held-out success from 64.8% to 73.8% in RoboLab simulation and from 38.3% to 50.0% on the real-world DROID setup over a frozen-policy harness.

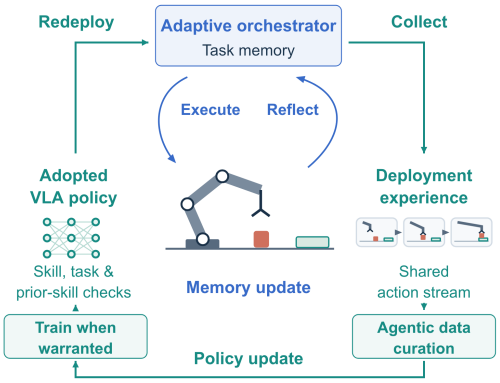

Figure 1: **Two coupled loops for deployment-time improvement.** The inner loop adapts the harness through episode reflection. The outer loop uses collected experience to update the policy, then revises memory so the harness can reassess the policy's capabilities.

2 RELATED WORK

Agentic robot orchestration. Language-model planners compose motion primitives and learned policies into task-solving programs (Huang et al., 2022; Ahn et al., 2022; Liang et al., 2023; Huang et al., 2023b; Xu et al., 2023). More recently developed frameworks also improve during deployment: they revise failed programs and retain reusable skills (Lu et al., 2026; Li et al., 2026; Chen et al., 2026a), or learn when to delegate to a policy rather than an analytic tool (Galanti et al., 2026; Chen et al., 2026b; Huang et al., 2026; Hu et al., 2026; Zhao et al., 2026). All of these adapt how a fixed set of capabilities is used, so the system can do no better on a skill than its policy allows. ROBO-COP keeps this adaptive harness but also improves the policy it calls, and revises the harness's memory whenever that policy changes.

Generating robot-policy data. Improving a policy requires data that covers the situations it fails on. Simulation provides scalable supervision (Pfaff et al., 2026), but transferring learned behavior to physical robots remains difficult (Huang et al., 2023a; Zhang et al., 2025; Choi et al., 2026). Real-world datasets and collection interfaces avoid this gap at the cost of human effort (Khazatsky et al., 2024; Chi et al., 2024; Xu et al., 2025), and demonstration-generation methods stretch a small set of human examples further (Mandlekar et al., 2023; Li et al., 2025). Closer to our setting, agents and

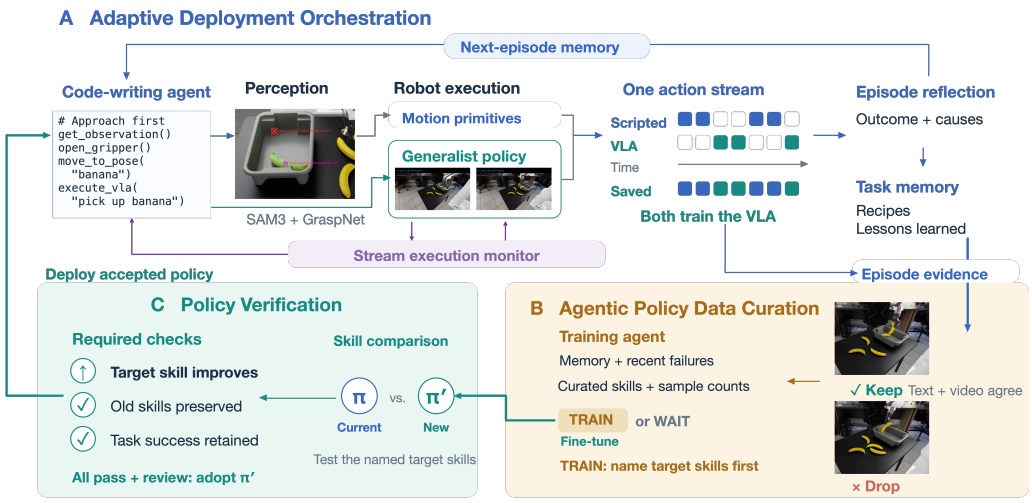

Figure 2: **The ROBO-COP deployment-to-learning loop. A:** The orchestrator mixes scripted motion and VLA calls, logs one action stream, and updates memory. **B:** Judges segment the stream into labeled skills; a controller decides when to train. **C:** Candidates are verified before adoption, and policy-dependent memory is revised.

policies can collect and filter their own experience (Chen et al., 2025; Zhou et al., 2024; Zhang & Shah, 2026; Xiao et al., 2026). ROBO-COP draws its data from deployment itself, where executions mix policy and scripted actions, and curates skill-level demonstrations from them, including correct skills performed within failed trials.

Policy improvement during deployment. Planner distillation and autonomous collection turn robot executions into policy supervision (Ha et al., 2023; Dalal et al., 2023; Bousmalis et al., 2024; Zhou et al., 2024), real-world reinforcement learning updates policies from interaction and reward (Mark et al., 2024; Wagenmaker et al., 2025; Lei et al., 2025), and VLM-guided exploration and skill acquisition generate training data for adapting VLA policies (Sukhija et al., 2026; Wang et al., 2026). These methods typically deploy each updated policy, while test-time verification selects instructions or actions for a policy that stays fixed (Kwok et al., 2026; Zhang & Shah, 2026). ROBO-COP instead treats each update as a candidate: it decides when training is warranted, verifies the whole candidate policy before adoption, and revises the memory that the harness built around the previous policy.

3 METHOD

ROBO-COP updates both the VLA policy and the adaptive harness that orchestrates its use (Figure 2). The harness logs policy and scripted actions as one stream (Section 3.1), curates it into skill demonstrations (Section 3.2), and verifies each new policy before adoption, revising the memory tied to the old one (Section 3.3).

3.1 ADAPTIVE POLICY ORCHESTRATION

Acting and reflecting. ROBO-COP is an embodied coding agent built on the CaP-X execution tools (Fu et al., 2026). Its orchestrator writes a program that calls scripted motion primitives, perception functions, and the VLA policy, conditioned on the task goal, current observation, episode history, and task memory M, as in prior harnesses (Galanti et al., 2026; Chen et al., 2026b; Huang et al., 2026). A vision-language model (VLM) monitors execution and can interrupt a call or end the episode. Whenever execution stops, a visual judge checks the outcome and either ends the episode or asks for revised code. It credits the VLA and scripted actions separately, so a successful skill keeps its credit even if a later step fails. At the end of the episode, the orchestrator records what worked and what failed in M, a form of episodic memory that lets agentic robot systems improve without changing any weights (Liu et al., 2023; Zha et al., 2024; Lu et al., 2026; Huang et al., 2026).

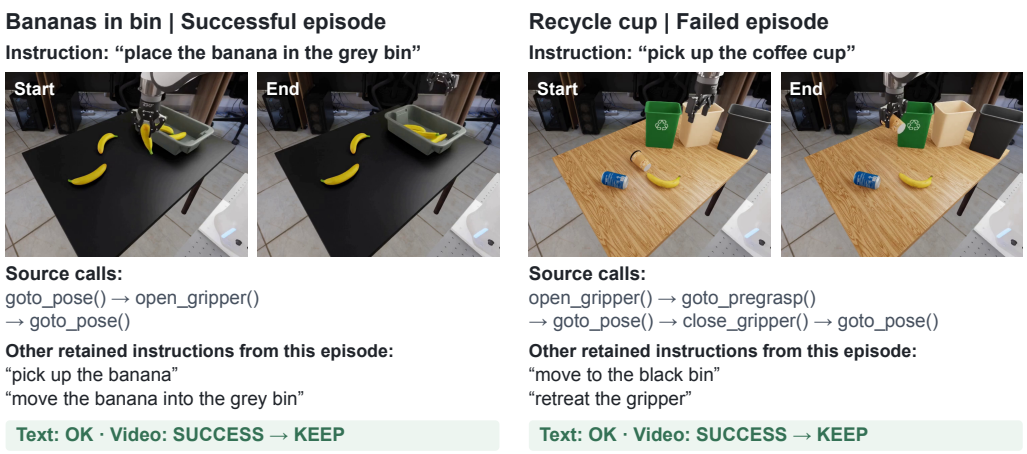

Figure 3: **Curating skills from deployment episodes.** Each episode is split into skills, and each skill is labeled and judged on its own. The failed episode still contributes the skills that succeeded before the failure. Source calls list the executed motion APIs with arguments omitted.

Action unification. Training data for the VLA has to come from these executions, yet many useful motions are produced by scripted primitives rather than by the policy. Within one execution, the orchestrator may alternate between VLA calls and scripted primitives. For example, the robot may approach a red block with scripted primitives and then call the VLA to lower and close the gripper. We record the commanded joint targets, gripper commands, observations, and robot states at each step in execution order, which yields a continuous action stream in the VLA's training format. The recorded steps can then become a demonstration labeled *grasp the red block*, even though part of it was produced by scripted primitives.

3.2 Experience curation and policy improvement

From execution to labeled skills. The unified action stream contains interleaved VLA and scripted actions, which ROBO-COP curates into skill demonstrations (X, ℓ), where $X = (o_t, s_t, a_t)_{t=1}^{T}$ is a contiguous trajectory and ℓ is its language instruction, and collects the accepted ones in a training set D.

For each execution, a text judge uses the executed code, console output, and episode feedback to identify each segment's intended skill and assess its outcome. For VLA calls, it keeps the instruction passed to the policy. For scripted primitives, it groups consecutive calls into a motion-level skill, such as the scripted approach above, and a segment labeled *pick up the red block* must also include the lift. A video judge then checks sampled frames against the proposed instruction, and a segment enters D only when the text judge accepts it and the video judge confirms success or clear partial progress. Because curation evaluates segments independently of the episode outcome, a successful pickup can remain a training example even if the block is later dropped, and a successful episode does not make every segment valid. Figure 3 shows segments retained from a successful and a failed episode with their source API calls and labels, which range from object-level skills such as picking up a banana to supporting motions such as lifting or retreating. We therefore expect fine-tuning to improve most on skills that recur across episodes, such as grasping, which the failure analysis in Section 4.2 examines. Appendix C gives the segmentation rules and judge outputs.

Deciding when to train. Training on a fixed schedule causes two potential problems: it trains on tasks the harness already solves, where an update can only add risk, and it trains before enough demonstrations have accumulated to support a useful update. ROBO-COP therefore lets a language-model controller decide when to train. It reviews the curated demonstrations alongside recent performance, task memory, failures, and earlier attempts, and returns TRAIN or WAIT with the desired improvement or the evidence still needed. We expect the controller to skip training on tasks the harness already solves, and to wait on the others until the curated demonstrations cover the skills that keep failing (Section 4.2).

Algorithm 1 ROBO-COP Self-Improving Pipeline

$\pi \leftarrow \pi_{\text{base}}$; $M, D, S \leftarrow \emptyset$
while deployment trials remain **do**
  $\tau \leftarrow \textsc{Execute}(\pi, M)$; $M \leftarrow \textsc{Reflect}(M, \tau)$; $D \leftarrow D \cup \textsc{Curate}(\tau)$
  if $\textsc{Controller}(D, M)$ returns TRAIN with targets E **then**
    $\pi' \leftarrow \textsc{FineTune}(\pi_{\text{base}}, D)$
    $v \leftarrow \textsc{Verify}(\pi', \pi, E, S)$ ▷ runs over the next W trials
    if $v =$ ADOPT **then**
      $\pi \leftarrow \pi'$; $S \leftarrow S \cup E$
    end if
    if $v \neq$ STARVED **then**
      $M \leftarrow \textsc{ReviseMemory}(M)$
    end if
  end if
end while

A TRAIN decision names up to four target skill instructions E, which guide verification. Training itself uses all curated data collected so far. We fine-tune a candidate π' from the pretrained checkpoint with LoRA (Hu et al., 2022) and the policy's original flow-matching objective. Deployment on the task pauses while π' trains. Afterward, π stays in service as the baseline against which π' is evaluated. Appendices D and F give the controller and training details.

3.3 POLICY VERIFICATION AND INTEGRATION

Candidate policy verification. We evaluate the candidate π' during a window of W trials while retaining the deployed policy π as a fallback. For target skills, the orchestrator first tries π' and falls back to π or scripted motion if needed. Because fallbacks can complete a task after π' fails, task success alone does not reflect the candidate's own performance, so verification is performed at the skill level rather than only at the episode level. Since each candidate is fine-tuned from base policy on all curated data, it could also lose a skill that an earlier update added, so a verifier checks for regressions on skills adopted with earlier candidates, which we call *admitted* skills, S.

Let b be the incumbent's recent task success rate and $\text{succ}(W)$ the task success rate during the candidate window. For each admitted skill s, let $r_{\text{inc}}(s)$ and $r_c(s)$ be the observed per-call finish rates of the incumbent and candidate, where a call finishes when its completion check fires or the arm becomes still. We apply the skill check to S_{obs}, the admitted skills with sufficient candidate calls and an incumbent reference, and require

$$\text{succ}(W) \geq b - \delta_{\text{task}}, \qquad r_c(s) \geq r_{\text{inc}}(s) - \delta_{\text{skill}} \quad \forall s \in S_{\text{obs}}. \tag{1}$$

If the candidate receives too few calls to assess, verification returns STARVED, and the candidate is discarded and memory is left unchanged. If either inequality fails, it returns REVERT and π stays deployed without updating policy weights.

For the target skills E, a language-model reviewer checks what each candidate call actually achieved, given the generated code, output logs, and visual judge outputs. Verification does not rely on episode success alone but checks whether the skills actually meet the expectations. A candidate is adopted only if it satisfies all requirements, and otherwise the system reverts to the current policy (Figure 4). Adoption is decided only at the end of the evaluation window, whereas reversion can happen earlier if the candidate clearly fails during evaluation. Appendix E specifies the window size, thresholds, and stopping rules.

Evidence Curated demonstrations cover the target skill.

Train Target: place the green block in the red bowl.

Verify Numerical safeguards pass; semantic review credits the candidate. Adopt and revise memory.

Figure 4: Illustrative training and adoption reasoning. Prompt details appear in Appendix G.

Policy adoption and memory revision. A candidate that passes verification replaces π, and its target skills are added to S. If no candidate is ever adopted, the system keeps the base policy and

improves through orchestration alone. Changing the deployed policy does not change the advice stored in task memory. For example, memory may recommend scripted motion for grasping a red block because earlier VLA attempts failed. Following that recommendation after an update would keep the harness from discovering that the new policy can successfully grasp. The same problem arises when a candidate is rejected.

During its evaluation window, episode reflection records the candidate's failures as advice against using the VLA. After the incumbent is restored, that advice would wrongly steer every later episode away from it. After adoption or reversion, we therefore rewrite references to the candidate to name the policy in service and mark advice that steers the orchestrator away from the VLA as unverified for the current checkpoint. The advice is marked rather than deleted, since the update may not have fixed the skill, so later episodes can retest it while its history stays in memory. Algorithm 1 summarizes the full loop.

4 EXPERIMENTS

Through this section, we aim to address three main questions. **Q1:** Does co-evolution improve held-out success over a frozen-policy harness and fixed-schedule training, and in which task regimes? **Q2:** Does the system decide correctly when to train and whether to adopt? **Q3:** Does learning during deployment cost task success along the way? We answer them on ten simulated tasks and then test the full loop on three real-world tasks.

4.1 EXPERIMENTAL SETUP

Tasks and system. We evaluate on ten RoboLab (Yang et al., 2026) tasks with randomized object configurations, spanning retrieval, grouping, placement, and recovery. We selected them before running ROBO-COP to represent the three regimes introduced in Section 1. Within every regime, the controller ends up waiting on one task, adopting a candidate on another, and reverting at least one candidate (Table 4), so the results cover each decision it can make. Figure 5 shows representative scenes, and Appendix H.1 shows successful episodes and randomized resets for all ten tasks. The initial VLA is $\pi_{0.5}$ trained on DROID (Physical Intelligence et al., 2025; Khazatsky et al., 2024) and served at its training control rate, and Gemini 3.8 Flash runs the orchestrator. Appendices B and F give the execution tools and fine-tuning settings.

Baselines. $\pi_{0.5}$ *only* runs the base policy without an orchestrator, memory, or updates. *Frozen-policy control* uses the same harness and episode reflection as ROBO-COP but never changes the VLA weights, isolating what memory adaptation alone achieves. *Fixed schedule* uses the same recorder, curator, fine-tuning procedure, episode reflection, and memory revision as ROBO-COP, but trains every 30 trials and deploys each checkpoint without verification, isolating the contribution of the controller and verifier. *ASPIRE* diagnoses failures, revises robot programs, and retains validated solutions as reusable skills (Lu et al., 2026). Since ASPIRE did not evaluate on RoboLab, we adapt its algorithm to our benchmark and API setting.

Protocol and metrics. Frozen-policy control, fixed schedule, and ROBO-COP each run 100 deployment trials (episodes) per task on matched initializations, sharing their first 30 trials before any training decision. *Deployment success* is the success rate over these trials, which include data collection and candidate evaluation. We then freeze each system's policy and memory, or ASPIRE's skill library after its 100-execution search, and measure *held-out success* on the same 50 new initializations per task, following the single-learning-run protocol of prior agentic robot-learning work (Lu et al., 2026; Xiao et al., 2026). The standalone $\pi_{0.5}$ is tested only on the held-out trials.

4.2 SIMULATION RESULTS

Q1: Does co-evolution improve held-out success, and where? Co-evolution improves held-out success over both controls, and the gains are largest when ROBO-COP updates the policy. ROBO-COP reaches 73.8% mean held-out success, versus 64.8% for the frozen-policy control and 65.8% for the fixed schedule, which recovers only 1.0 of the 9.0 points (Figure 6). The standalone VLA solves every held-out trial of Bananas in bin but succeeds on at most 8% of trials on six tasks. The



Figure 5: Representative RoboLab tasks. Object configurations vary across deployment and held-out evaluation.

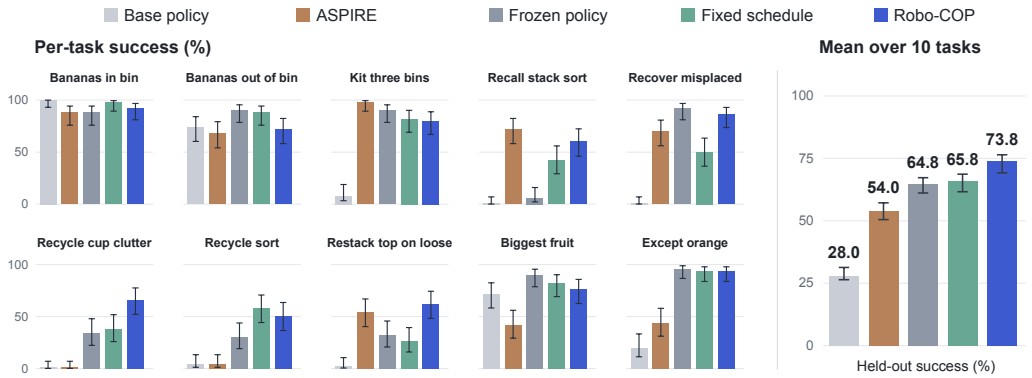

Figure 6: Held-out success on 50 new initializations per task. Left: per-task success for all five methods. Right: equally weighted ten-task means; whiskers are 95% beta-posterior credible intervals over the fixed task set (Appendix H).

gain concentrates where the policy changed: on the six tasks where ROBO-COP adopted an update, it averages 11.3 points over the frozen-policy control, versus 5.5 points on the four tasks where it kept the base policy, where the systems differ only in their memory and deployment histories. Omitting any single task leaves a mean gain of 4.0–12.0 points over the frozen-policy control and 4.9–10.7 over the fixed schedule (Figure 10).

The regimes separate the methods as anticipated in Section 1 (Table 1). Each method leads where its adaptation matches the task, and ROBO-COP is the only one that stays competitive in all three regimes. Where the VLA alone suffices, every harness that can call the policy stays close to $\pi_{0.5}$ (82.0%): the frozen and fixed harnesses reach 89.3% and ROBO-COP 80.0%, while ASPIRE, which cannot call the VLA, falls to 66.0%. Where scripted primitives suffice, ASPIRE leads (73.5%) because it searches over programs built from exactly those primitives, and ROBO-COP comes close (72.0%), while the frozen and fixed harnesses reach 55.0% and 50.0%. Where neither suffices, ASPIRE falls to 16.0% because its programs cannot call the VLA, while ROBO-COP fine-tunes the missing skills on the two tasks that need them and reaches 70.0%, 16.7 points above the frozen-policy control.

Across regimes, the skill that fine-tuning improves most is grasping. ROBO-COP fails 131 of 500 held-out trials versus 176 for the frozen-policy control, and grasp or drop, the dominant failure in both (40 and 68), accounts for 28 of the 45 fewer failures: every first training request targeted a pickup skill (Table 4), and fine-tuning on curated grasps, including those from failed episodes, makes the policy's grasps more reliable.

Q2: Does the system decide correctly when to train and whether to adopt? The controller trains where training helps, and the verifier rejects candidates that would have hurt. The controller trained on the seven tasks where the harness still failed often, where the frozen-policy control succeeds on 45–79% of deployment trials, and waited on the three where it rarely failed (80–96%), as its prompt directs (Appendix G). On those three tasks, the fixed schedule's nine training runs bring no net gain over the frozen-policy control. In every first training request (Table 4), the targets include

Table 1: **Held-out success by task regime (%).** Regimes group tasks by what solves them: the VLA alone (Bananas in bin, Bananas out of bin, Biggest fruit), scripted primitives alone (Kit three bins, Recall stack sort, Recover misplaced, Restack top on loose), or neither (Recycle cup clutter, Recycle sort, Except orange). The last three columns count tasks by ROBO-COP's final policy decision.

Regime	Tasks	$\pi_{0.5}$	ASPIRE	Frozen	Fixed	Ours	No training	Adopted	Reverted (final)
VLA solves alone	3	82.0	66.0	**89.3**	**89.3**	80.0	1	2	0
Primitives solve	4	2.5	**73.5**	55.0	50.0	72.0	1	2	1
Neither alone	3	8.0	16.0	53.3	63.3	**70.0**	1	2	0



Put the cup and tape on the towel (40 s) | *Move the lid to the towel and the contents to the plate* (60 s) | *Put the mustard in the bowl and the carrots in the pot* (90 s)

Figure 8: **Real-world tasks on a DROID setup.** Each task requires placing multiple objects, and a trial succeeds only if every condition is met within the time limit shown.

a pickup skill, which matches the grasp and drop failures that dominate held-out errors in the failure analysis. Verification then rejected 5 of 12 candidates. On the four tasks whose first candidate was reverted, the fixed schedule, which deploys every checkpoint, falls below the frozen-policy control on three (Table 6). Rejection also does not end learning: the controller waits for new demonstrations and trains again, and on three of those four tasks the next candidate was adopted (Figure 7).

Q3: Does learning cost deployment success? ROBO-COP learns without sacrificing success during deployment. It succeeds on 73.2% of 100 deployment trials, versus 68.7% for the frozen-policy control and 70.3% for the fixed schedule. The gap appears once the runs diverge after trial 30 (72.9% versus 66.4% and 68.7% over trials 31–100) and holds over the final 25 trials (73.6% versus 67.6% and 67.2%) and every other late window we examined (Appendix H). ASPIRE spends its 100-execution budget exploring rather than attempting the task, so it has no comparable deployment success.

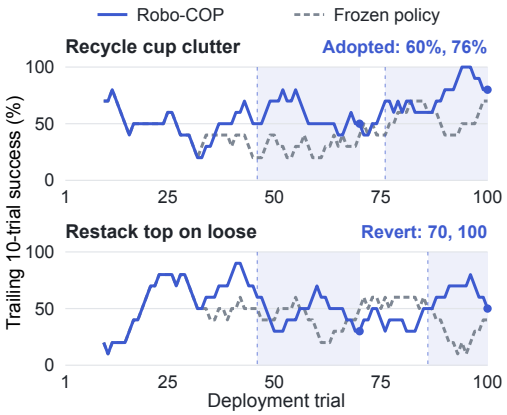

Figure 7: Adoption and reversion during deployment. Shading marks candidate evaluation.

4.3 REAL-WORLD EXPERIMENTS

We run the same protocol on three physical tasks with 40-, 60-, and 90-second limits (Figure 8), with 50 learning trials and 20 held-out tests per task, comparing ROBO-COP with a frozen-policy control that uses the same policy, tools, and memory adaptation. Because each task has only 50 learning trials, we shorten the adoption window to $W = 10$ trials. A trial succeeds only if every requested condition is met within its episode time budget. Appendix I shows successful executions.

Results. The real-world results reproduce each controller outcome seen in simulation, and only the verified update changes held-out success(Table 2). On the 60-second task, the controller trains at trial 30, verification adopts the candidate, and held-out success doubles from 35% to 70%. On the 40-second task, the controller also trains at trial 30, but the candidate fails on several evaluation trials. Verification reverts it and deployment continues with the incumbent, so learning (44% versus

Table 2: **Real-world success rates (%) and ROBO-COP's training decisions.** Frozen uses the same harness but never updates the policy.

	ROBO-COP decisions		Learning (50 trials)		Test (20 trials)	
Task	Train request	Outcome	Frozen	ROBO-COP	Frozen	ROBO-COP
Cup and tape (40 s)	Train@30	Revert	46	44	40	40
Lidded pot (60 s)	Train@30	Adopt	32	38	35	70
Mustard and carrots (90 s)	Wait	n/a	32	32	40	40
Mean			36.7	**38.0**	38.3	**50.0**

46%) and held-out success (40% versus 40%) stay level with the frozen-policy control, and the rejected update costs little. On the 90-second task, the controller never trains, so both systems share one learning trace and reach 40% held-out. A separate diagnostic that forced training on this task and deployed the checkpoint without verification reached only 20% held-out, half of either system, confirming that waiting was the right decision. Across the three tasks, ROBO-COP raises mean held-out success from 38.3% to 50.0%.

5 CONCLUSION

ROBO-COP co-evolves a robot's orchestrator and policy during deployment, turning the deployment experience into steady improvement of the system. Across ten simulated manipulation tasks, ROBO-COP improves held-out success by 9.0 points over the same harness with a frozen policy, while fine-tuning on a fixed schedule without verification recovers only 1.0 of those points. Verification rejected 5 of 12 candidates, and the controller skipped training on exactly the tasks where the harness already succeeded on most trials. In the real world, ROBO-COP improves performance on the task where it adopts an update, loses little on the task where it reverts, and forcing an update the controller rejected hurts performance.

Limitations. ROBO-COP can learn only skills that its policy and scripted tools already demonstrate at least partially, since every training example comes from its own executions. Physical deployment requires human resets, and each policy update adds curation, training, and evaluation cost. Where scripted primitives already cover a task, ASPIRE's program search outperforms ROBO-COP (Kit three bins and Recall stack sort), because ROBO-COP's orchestrator adapts through memory rather than structurally searching over programs. Finally, even though all design components have been verified through experiments, the scalability of ROBO-COP still needs to be validated on larger experiments to confirm that the results hold at scale.

Future work. ROBO-COP is a step toward robot systems that improve themselves in the real world through deployment. A fully self-improving system requires every component to improve. ROBO-COP only updates part of the whole system, the policy and the orchestrator's memory, while its scripted skills, its judges, and the orchestrator's own reasoning stay fixed. Structured skill exploration, as in ASPIRE, diagnoses failures, proposes new programs, and retains the validated ones as reusable skills. Running such exploration inside the deployment loop, directed at the task at hand, would give the orchestrator better tools whose successful executions become training data for the policy, and the orchestrator itself could be better aligned with reinforcement learning from deployment outcomes. Because ROBO-COP curates demonstrations at the skill level, training a single policy on skills pooled from many tasks is a natural path to cross-task generalization, so that a system deployed long enough improves at every task it encounters.

ETHICS STATEMENT

This work does not involve human subjects, personal data, or crowdsourced annotation. Simulation experiments use the public RoboLab benchmark, and the initial policy is the publicly released $\pi_{0.5}$ model trained on DROID. All physical experiments used a single Franka arm manipulating household objects under the supervision of a human operator, who performed resets and could stop the

robot at any time. Because ROBO-COP updates a deployed policy, a poor update could degrade robot behavior; candidate verification, which rejects updates that reduce task success, is designed partly to limit this risk, but deployment in unstructured environments would still require additional safety measures.

REPRODUCIBILITY STATEMENT

Section 3 and Algorithm 1 describe the full loop. Appendix A lists every constant used in the simulation experiments, Appendix B gives the orchestrator's tool interface, memory format, and recorded fields, Appendix C gives the curation and segmentation rules, Appendices D and E specify the training controller and adoption checks, Appendix F gives the fine-tuning configuration, and Appendix G quotes the prompts for every language-model component. Appendix H reports per-task results and deployment curves, and Appendices H.1 and I show the tasks and example executions. We will release our code, prompts, and task configurations.

DISCLOSURE OF AI USE

Language models are components of the proposed system: Gemini 3.8 Flash serves as the orchestrator, curation judges, training controller, and adoption reviewer, as described in Section 3 and Appendix G. Separately, in preparing this paper, we used generative AI tools to edit and polish the manuscript text and to format tables and figures in LaTeX. We did not use generative AI tools to generate research ideas, design experiments, or produce or analyze experimental results. All authors reviewed the AI-assisted text, verified every reported number against our experimental logs, and take full responsibility for the content of this paper.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022. URL https://arxiv.org/abs/2204.01691.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems (RSS)*, 2025. URL https://arxiv.org/abs/2410.24164.

Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X. Lee, et al. RoboCat: A self-improving generalist agent for robotic manipulation. *Transactions on Machine Learning Research (TMLR)*, 2024. URL https://arxiv.org/abs/2306.11706.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, et al. SAM 3: Segment anything with concepts. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2511.16719.

Annie S. Chen, Alec M. Lessing, Yuejiang Liu, and Chelsea Finn. Curating demonstrations using online experience. *arXiv preprint arXiv:2503.03707*, 2025. URL https://arxiv.org/abs/2503.03707.

Kaiyuan Chen, Shuangyu Xie, Letian Fu, Justin Yu, William Pacini, Sandeep Bajamahal, Hudson Kim, Jaimyn Drake, Daehwa Kim, Haoru Xue, Jonathan Francis, Christian Juette, Peter Schaldenbrand, Muhammet Yunus Seker, Ruwan Wickramarachchi, Uksang Yoo, Guanzhi Wang, Adithyavairavan Murali, Balakumar Sundaralingam, S. Shankar Sastry, Spencer Huang, Yuke Zhu, Linxi Fan, and Ken Goldberg. GaP: A graph-as-policy multi-agent self-learning harness for variational automation tasks. *arXiv preprint arXiv:2607.05369*, 2026a. URL https://arxiv.org/abs/2607.05369.

Siyi Chen, Hugo Hadfield, Alex Zook, Mikaela Angelina Uy, Chan Hee Song, Erwin Coumans, Xuning Yang, Faisal Ladhak, Qing Qu, Stan Birchfield, Jonathan Tremblay, and Valts Blukis.

VoLo: A physical orchestrator for open-vocabulary long-horizon manipulation. *arXiv preprint arXiv:2606.07723*, 2026b. URL https://arxiv.org/abs/2606.07723.

Yuhao Chen, Zhihao Zhan, Xiaoxin Lin, Zijian Song, Hao Liu, Qinhan Lyu, Yubo Zu, Xiao Chen, Zhiyuan Liu, Tao Pu, Tianshui Chen, Keze Wang, Liang Lin, and Guangrun Wang. RADAR: Benchmarking vision-language-action generalization via real-world dynamics, spatial-physical intelligence, and autonomous evaluation. *arXiv preprint arXiv:2602.10980*, 2026c. URL https://arxiv.org/abs/2602.10980.

Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. In *Robotics: Science and Systems (RSS)*, 2024. URL https://arxiv.org/abs/2402.10329.

Andrew Choi, Xinjie Wang, Zhizhong Su, and Wei Xu. Scaling sim-to-real VLA reinforcement learning with generative 3d worlds. *arXiv preprint arXiv:2603.18532*, 2026. URL https://arxiv.org/abs/2603.18532.

Murtaza Dalal, Ajay Mandlekar, Caelan Reed Garrett, Ankur Handa, Ruslan Salakhutdinov, and Dieter Fox. Imitating task and motion planning with visuomotor transformers. In *Conference on Robot Learning (CoRL)*, 2023. URL https://arxiv.org/abs/2305.16309.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, et al. Molmo and PixMo: Open weights and open data for state-of-the-art vision-language models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. URL https://arxiv.org/abs/2409.17146.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. LIBERO-Plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025. URL https://arxiv.org/abs/2510.13626.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026. URL https://arxiv.org/abs/2603.22435.

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

Huy Ha, Pete Florence, and Shuran Song. Scaling up and distilling down: Language-guided robot skill acquisition. In *Conference on Robot Learning (CoRL)*, 2023. URL https://arxiv.org/abs/2307.14535.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? In *Conference on Language Modeling (COLM)*, 2024. URL https://arxiv.org/abs/2404.06654.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Jiaheng Hu, Mohit Shridhar, Caden Lu, Dhruv Shah, Hao-Tien Lewis Chiang, Jie Tan, and Annie Xie. What matters in orchestrating robot policies: A systematic study of hierarchical VLA agents. *arXiv preprint arXiv:2606.10267*, 2026. URL https://arxiv.org/abs/2606.10267.

Jinbang Huang, Yuanzhao Hu, Zhiyuan Li, Ran Qi, Yixin Xiao, Zhanguang Zhang, Mark Coates, Tongtong Cao, and Yingxue Zhang. RoboHarness: Memory-driven orchestration of heterogeneous robot policies for long-horizon planning. *arXiv preprint arXiv:2607.18060*, 2026. URL https://arxiv.org/abs/2607.18060.

Peide Huang, Xilun Zhang, Ziang Cao, Shiqi Liu, Mengdi Xu, Wenhao Ding, Jonathan Francis, Bingqing Chen, and Ding Zhao. What went wrong? closing the sim-to-real gap via differentiable causal discovery. In *Conference on Robot Learning (CoRL)*, volume 229 of *Proceedings of Machine Learning Research*, pp. 734–760, 2023a. URL https://proceedings.mlr.press/v229/huang23c.html.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning (ICML)*, 2022. URL https://arxiv.org/abs/2201.07207.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3d value maps for robotic manipulation with language models. In *Conference on Robot Learning (CoRL)*, 2023b. URL https://arxiv.org/abs/2307.05973.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, et al. DROID: A large-scale in-the-wild robot manipulation dataset. In *Robotics: Science and Systems (RSS)*, 2024. URL https://arxiv.org/abs/2403.12945.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Pannag 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 *Conference on Robot Learning (CoRL)*, 2024. URL https://arxiv.org/abs/2406.09246.

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 *European Conference on Computer Vision (ECCV)*, 2026. doi: 10.1007/978-3-032-37016-7_3. URL https://arxiv.org/abs/2602.12281.

Kun Lei, Huanyu Li, Dongjie Yu, Zhenyu Wei, Lingxiao Guo, Zhennan Jiang, Ziyu Wang, Shiyu Liang, and Huazhe Xu. RL-100: Performant robotic manipulation with real-world reinforcement learning. *arXiv preprint arXiv:2510.14830*, 2025. URL https://arxiv.org/abs/2510.14830.

Bowen Li, Mayank Mishra, Y. Isabel Liu, Stone Tao, Nishanth Kumar, Alexander G. Gray, Ruwan Wickramarachchi, Jonathan Francis, Sebastian Scherer, and Tom Silver. Recover, discover, plan: Learning skills and concepts from robot failures. *arXiv preprint arXiv:2606.18328*, 2026. URL https://arxiv.org/abs/2606.18328.

Chengshu Li, Mengdi Xu, Arpit Bahety, Hang Yin, Yunfan Jiang, Huang Huang, Josiah Wong, Sujay Garlanka, Cem Gokmen, Ruohan Zhang, Weiyu Liu, Jiajun Wu, Roberto Martín-Martín, and Li Fei-Fei. MoMaGen: Generating demonstrations under soft and hard constraints for multi-step bimanual mobile manipulation. *arXiv preprint arXiv:2510.18316*, 2025. URL https://arxiv.org/abs/2510.18316.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023. URL https://arxiv.org/abs/2209.07753.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. URL https://arxiv.org/abs/2307.03172.

Zeyi Liu, Arpit Bahety, and Shuran Song. Reflect: Summarizing robot experiences for failure explanation and correction. In *Conference on Robot Learning (CoRL)*, pp. 3468–3484, 2023. URL https://proceedings.mlr.press/v229/liu23g.html.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Yuke Zhu, Linxi Fan, and Guanzhi Wang. ASPIRE: Agentic skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026. URL https://arxiv.org/abs/2607.00272.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. MimicGen: A data generation system for scalable robot learning using human demonstrations. In *Conference on Robot Learning (CoRL)*, 2023. URL https://arxiv.org/abs/2310.17596.

Max Sobol Mark, Tian Gao, Georgia Gabriela Sampaio, Mohan Kumar Srirama, Archit Sharma, Chelsea Finn, and Aviral Kumar. Policy agnostic RL: Offline RL and online RL fine-tuning of any class and backbone. *arXiv preprint arXiv:2412.06685*, 2024. URL https://arxiv.org/abs/2412.06685.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. SceneSmith: Agentic generation of simulation-ready indoor scenes. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2602.09153.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. URL https://arxiv.org/abs/2504.16054.

Bhavya Sukhija, Oliver Groth, Mohit Shridhar, Tim Hertweck, Michael Bloesch, Markus Wulfmeier, Abbas Abdolmaleki, and Martin Riedmiller. EXIMO: VLM guided exploration of VLA policies. *arXiv preprint arXiv:2608.19891*, 2026. URL https://arxiv.org/abs/2608.19891.

Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. CuRobo: Parallelized collision-free robot motion generation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8112–8119, 2023. doi: 10.1109/ICRA48891.2023.10160765. URL https://curobo.org.

Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-GraspNet: Efficient 6-DoF grasp generation in cluttered scenes. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13438–13444, 2021. doi: 10.1109/ICRA48506.2021.9561877. URL https://arxiv.org/abs/2103.14127.

Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2025. URL https://arxiv.org/abs/2506.15799.

Maggie Wang, Lars Osterberg, Stephen Tian, Ola Shorinwa, Joseph Bowkett, Daniel Pastor, Jiajun Wu, and Mac Schwager. InSight: Self-guided skill acquisition via steerable VLAs. *arXiv preprint arXiv:2606.24884*, 2026. URL https://arxiv.org/abs/2606.24884.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi Fan, Yuke Zhu, and Guanya Shi. ENPIRE: Agentic robot policy self-improvement in the real world. *arXiv preprint arXiv:2606.19980*, 2026. URL https://arxiv.org/abs/2606.19980.

Mengda Xu, Han Zhang, Yifan Hou, Zhenjia Xu, Linxi Fan, Manuela Veloso, and Shuran Song. DexUMI: Using human hand as the universal manipulation interface for dexterous manipulation. In *Conference on Robot Learning (CoRL)*, 2025. URL https://arxiv.org/abs/2505.21864.

Mengdi Xu, Peide Huang, Wenhao Yu, Shiqi Liu, Xilun Zhang, Yaru Niu, Tingnan Zhang, Fei Xia, Jie Tan, and Ding Zhao. Creative robot tool use with large language models. *arXiv preprint arXiv:2310.13065*, 2023. URL https://arxiv.org/abs/2310.13065.

Xuning Yang, Rishit Dagli, Alex Zook, Hugo Hadfield, Ankit Goyal, Stan Birchfield, Fabio Ramos, and Jonathan Tremblay. RoboLab: A high-fidelity simulation benchmark for analysis of task generalist policies. In *Robotics: Science and Systems (RSS)*, 2026. doi: 10.15607/RSS.2026.XXII.096. URL https://www.roboticsproceedings.org/rss22/p096.pdf.

Lihan Zha, Yuchen Cui, Li-Heng Lin, Minae Kwon, Montserrat Gonzalez Arenas, Andy Zeng, Fei Xia, and Dorsa Sadigh. Distilling and retrieving generalizable knowledge for robot manipulation via language corrections. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024. doi: 10.1109/ICRA57147.2024.10610455. URL https://arxiv.org/abs/2311.10678.

Mingtong Zhang and Dhruv Shah. Visual verification enables inference-time steering and autonomous policy improvement. *arXiv preprint arXiv:2606.18247*, 2026. URL https://arxiv.org/abs/2606.18247.

Xilun Zhang, Shiqi Liu, Peide Huang, William Jongwon Han, Yiqi Lyu, Mengdi Xu, and Ding Zhao. Dynamics as prompts: In-context learning for sim-to-real system identifications. *IEEE Robotics and Automation Letters (RA-L)*, 10(4):3190–3197, 2025. URL https://arxiv.org/abs/2410.20357.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Yi Nie, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Jiaji Rao, Zhengru Fang, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering frozen VLAs into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026. URL https://arxiv.org/abs/2607.08448.

Linfeng Zhao, Haojie Huang, Jiayuan Mao, Weiyu Liu, Mykel Kochenderfer, and Lawson L. S. Wong. Retriever: Composing the perception-reasoning-action loop for long-horizon manipulation. *arXiv preprint arXiv:2607.17213*, 2026. URL https://arxiv.org/abs/2607.17213.

Zhiyuan Zhou, Pranav Atreya, Abraham Lee, Homer Walke, Oier Mees, and Sergey Levine. Autonomous improvement of instruction following skills via foundation models. In *Conference on Robot Learning (CoRL)*, 2024. URL https://arxiv.org/abs/2407.20635.

A CONSTANTS

All reported simulation results use the settings in Table 3.

Table 3: Constants defining the loop.

Constant	Value	Raising it
Turns per episode	20	more deliberation per episode, fewer episodes per budget
Memory budget	8000 char	more recall, less of the prompt in the attended region
Stillness exit	16 steps	fewer premature exits, more budget spent on stalled bursts
Held frames per segment	15	stronger stopping behaviour, more static frames in D
Action horizon H	15	longer commitment per inference, coarser reaction
Curated floor / new since last	30 / 10	later first training run, more evidence behind it
Named instructions per run	≤ 4	broader expectation, weaker per-skill evidence
Adoption window W	up to 25 episodes	more evidence, fewer episodes for further updates
Candidate-exposed trial floor	5	stronger exposure requirement, more starved rounds
Prior-skill call floor	5	more evidence per check, fewer prior skills checked
Margins $\delta_{\text{task}}, \delta_{\text{skill}}$	0.1	more tolerated decline, weaker safeguards
Task bar window	30 episodes	steadier bar, slower to reflect a real change
Attempts per task	3	more chances to improve, more episodes spent on windows

B ORCHESTRATOR INTERFACE

The orchestrator's programs run in a sandbox that exposes only the following functions: `get_observation`, `get_object_pose`, `sample_grasp_pose`, `compute_place_pose`, `goto_pose`, `goto_pregrasp`, `open_gripper`, `close_gripper`, and `execute_vla`. Documentation for each function is generated from its signature and docstring. Object poses come only from the perception functions; the sandbox has no access to ground-truth poses and blocks imports of simulator packages. Each episode allows at most twenty turns.

Execution tools and monitoring. Scripted execution selects targets with segmentation (Carion et al., 2026), visual pointing (Deitke et al., 2025), grasp proposals (Sundermeyer et al., 2021), and placement geometry, and executes them with a collision-aware planner (Sundaralingam et al., 2023) and gripper commands. The policy is called through `execute_vla(instruction, max_steps, policy)`. It outputs arm and gripper commands. We refer to a single invocation as a policy call. A call ends when its completion check fires, when it reaches its step limit, or when the VLM monitor stops it. An arm that stops moving while still holding an object is not counted as a completed placement. Program variables persist across turns.

Episode memory and policy reconciliation. After each episode, a script logs the outcome, the instruction and exit condition of each policy call, and the scripted grasp poses. A reflection model then updates two memory sections, *what works* and *known failure modes*, and may propose deleting specific lines from a *turn-verified recipe*. Deletions are applied by code, so recipe steps are never dropped silently. Memory is capped at 8000 characters with a fixed allocation per section to keep prompts short (Liu et al., 2024; Hsieh et al., 2024); similar memory designs appear in Lu et al. (2026); Li et al. (2026). After each adoption or reversion, entries that discourage using the policy are marked UNVERIFIED for the current checkpoint, and references to the candidate are rewritten to name the policy in service. These entries keep their counts and last-observed episode as history, but the counts no longer set the orchestrator's confidence. All other entries are left unchanged.

Recording fields. The recorder logs every actuated step, whether it comes from the inverse-kinematics servo, the motion planner, a gripper command, or the policy. Each row stores both

camera views, the end-effector pose, the measured joint state, the commanded joint target with an eighth channel for the gripper, and the turn index.

C CURATION

Curation runs in two stages: a text judge proposes and labels skills, and a visual judge checks them against video.

Stage A (text). For each turn, the text judge reads the code, console output, and outcome assessment, and returns an instruction and a verdict. Each instruction names one skill using a canonical verb and its required arguments. When the turn called the policy with an instruction string, that string is kept verbatim.

Stage A (skill grouping). For scripted turns, we mark each primitive call in the code and ask the judge to group consecutive primitives into physically meaningful skills, named with the same rule. Function names may not appear in instructions.

Stage B (video). The visual judge samples six frames at native resolution from the segment's video, receives the Stage A instruction, and returns one of five verdicts: success, useful partial progress, no progress, failure, or unclear.

Filtering. A segment is kept when Stage A returns OK and Stage B returns success or useful partial progress. Turns whose only instruction opens or closes the gripper are dropped before Stage B. Both verdicts are cached with their reasons, so the filter can be retuned offline without new model calls.

Segmentation. Accepted rows are split wherever the instruction changes, a rejected row intervenes, or the joint state jumps by more than 0.3 rad. Segments shorter than two steps are discarded. Each remaining segment is padded with fifteen held frames that repeat the final observation and use the final joint state and gripper command as the action, so that the policy learns to stop at the end of a skill.

D CONTROLLER

Before the language model is consulted, code applies several guards. No decision is made while a candidate is under evaluation. Training waits until at least thirty episodes contain a kept segment and at least ten such episodes have arrived since the last attempt. The controller makes at most three training attempts per task, and it does not train when the incumbent's task-success reference has fewer than five trials or the deployment budget is spent. Two further caps, on total training runs and total episodes per task, bound the overall experiment.

Once the guards pass, the model sees the task memory, trial statistics, previous attempts, the curated instructions with their segment counts, and the code and judge output from up to three recent failures. It returns TRAIN or WAIT with at most four target instructions. If the verdict cannot be parsed, the controller waits; an instruction list that cannot be parsed is treated as empty. Appendix G gives the prompt.

Table 4: **First training request and later decisions by task.** Trial, targets, and data (curated episodes / kept segments) refer to the first training request; arrows show later decisions. Six tasks end with an updated policy. A dash means the controller never requested training.

Task	Trial	Selected target instructions	Data (ep./seg.)	Decision
Bananas in bin	–	–	–	Base retained
Bananas out of bin	45	Pick up the banana; place the banana on the table	43/185	Adopt
Kit three bins	–	–	–	Base retained
Recall stack sort	60	Place the green block in the red bowl; place the blue block on the table; pick up the red block	59/280	Adopt
Recover misplaced	60	Pick up the blue can; place the blue can in the grey bin	58/220	Revert→Adopt
Recycle cup clutter	45	Pick up the coffee cup; place the coffee cup in the black bin	43/194	Adopt→Adopt
Recycle sort	45	Pick up the blue can; place the blue can in the green bin	39/176	Revert→Adopt
Restack top on loose	45	Pick up the corn can; place the corn can on the tuna can	42/121	Revert→Revert
Biggest fruit	45	Pick up the orange; place the orange into the white bowl	42/135	Revert→Adopt
Except orange	–	–	–	Base retained

E ADOPTION ARITHMETIC

The evaluation window is 25 episodes, or fewer if the deployment budget runs out. Execution, recording, and reflection continue as usual during evaluation. The orchestrator is asked to try the candidate at least once on a relevant skill before falling back, but it is not required to use every target instruction. A round in which fewer than five episodes contain a candidate call that commits an action is STARVED. Both margins in Eq. 1 are 0.1.

The task reference b is computed before training from up to thirty recent episodes served by the incumbent. For an adopted checkpoint these include its own evaluation window; for the base checkpoint they include only episodes it served outside candidate evaluation windows. The reference therefore measures the whole system running that checkpoint.

For the prior-skill check, candidate calls are grouped by the instruction strings registered at each earlier adoption. A group is checked only if it has at least five candidate calls and an incumbent finish rate from before the window; smaller groups are reported but cannot block adoption. The numerical check matches instructions by exact string, and the semantic reviewer handles paraphrases. A call counts as finished when its completion check fires or the arm stops moving. The stillness exit is enabled only for checkpoints trained with held frames. The semantic reviewer is called only when the exposure floor and both safeguards pass, and a response it cannot parse counts as a veto.

Early stopping. After five episodes with candidate calls, a mid-window reviewer may stop evaluation if the candidate has repeatedly failed at its primary skill, the failures are clearly its own, and further trials seem unlikely to change the outcome. Low task success, an unrelated later failure, or missing evidence is not enough to stop, and unclear or unparseable responses let evaluation continue. A stopped round never ends in adoption, even if the partial tallies would pass. If the per-task episode cap is reached mid-window, the round is settled on the episodes collected so far, provided the exposure floor is met.

F FINE-TUNING

The candidate is optimized on the curated dataset with the policy's flow-matching objective:

$$\mathcal{L}(\theta) = \mathbb{E}_{(X,\ell)\sim D}\, \mathbb{E}_t \left[\mathcal{L}_{\mathrm{FM}}(\pi_\theta(o_t, s_t, \ell), a_{t:t+H-1}) \right]. \tag{2}$$

Here $H = 15$ is the prediction horizon. The dataset stores absolute joint targets and gripper commands. Before training, joint targets are converted to offsets from the observed joint state, while

gripper commands stay absolute; held frames therefore have zero joint offset. We train LoRA adapters in the language backbone (rank 16) and the action expert (rank 32) and freeze their remaining parameters, while the vision encoder, action input/output projections, and time-conditioning MLP are fully fine-tuned. Training uses batch size 32, AdamW with gradient clipping at 1.0, and a cosine schedule from 5×10^{-5} to 5×10^{-6}. The number of steps scales with the size of the curated set and is clamped to $[500, 10000]$. Normalization statistics are recomputed for each curated dataset, and training does not start if they are out of date. Every run initializes from the pretrained checkpoint.

G AGENT PROMPTS AND DECISION CONTRACTS

Quoted text below is verbatim from our prompts; everything else summarizes them. Curation, training control, and adoption review are separate model calls with different inputs and outputs. The training controller and adoption reviewer use Gemini 3.8 Flash at temperature 0.2, with reasoning enabled and a 1024-token output limit.

Skill grouping and visual filtering. The text judge receives the task goal, the generated code annotated with executed primitive indices, console output, and the turn's outcome assessment. Its prompt asks:

> "Your job is to GROUP the marked primitives above into however many distinct LOW-LEVEL SKILLS actually happened in this turn"

Each returned skill contains `segment_indices`, `instruction`, `verdict`, and `reason`, with verdict `OK`, `FAILED`, or `UNCLEAR`. Every marked primitive must belong to exactly one group, and each label must describe the motion actually executed; for example, closing the gripper on an object is labeled differently from lifting it. The visual judge receives the instruction and six sampled frames and is told:

> "If the decisive moment for this instruction falls between frames, that is UNCLEAR, not SUCCESS."

It returns `verdict` and `reason`, with verdict `SUCCESS`, `USEFUL_PARTIAL`, `NO_PROGRESS`, `FAILED`, or `UNCLEAR`. For primitive groups, the judge also sees frames from the rest of the turn to check whether an apparent outcome persists; these frames are not added to the training segment. Missing frames and unparseable responses yield `UNCLEAR`.

Training request. The controller receives curated episode counts (total and new), the most frequent curated instructions with their segment counts, trial statistics, previous attempts, recent failure code and assessments, and task memory. The prompt opens:

> "You decide whether to spend GPU hours fine-tuning a robot policy for ONE task. Fine-tuning helps when the task's skills are demonstrable from the curated data and the current approach is failing or slow; it wastes money when orchestration already succeeds cheaply or the data is thin/stale."

The response follows this format:

```
VERDICT: TRAIN or WAIT
EXPECT: intended skill improvement and failure to address;
  on WAIT, evidence that would change the decision.
EXPECT_INSTRUCTIONS: representative primary-capability
strings;
  on WAIT, an empty list.
One short paragraph of rationale.
```

The prompt asks for one representative string per primary skill, copied from the supplied vocabulary, and the parser keeps at most four. The semantic reviewer judges these targets by meaning, and supporting motions are listed only if they are themselves the reason to train. If the instruction list is missing, verification falls back to task-level counts. Each request is logged with its rationale, targets, and inputs.

Adoption review. The reviewer receives the original expectation, the target instructions, per-call records, executed code, and matched outcome assessments. The prompt states:

> "Mechanical completion (including park-exit) is NOT verified skill success."
>
> "if attribution to the challenger is ambiguous, classify UNCERTAIN."
>
> "Do not count UNCERTAIN as success."

The reviewer groups paraphrases and constituent motions into primary skills, separates a successful early skill from a later episode failure, and does not credit the candidate for work completed by scripted recovery. Explicit gripper operations count as orchestration, so for a placement the reviewer checks whether the candidate reached a usable release pose. Its response begins with `ADJUDICATION: CONFIRM` or `ADJUDICATION: VETO`, followed by its reasoning.

H ADDITIONAL SIMULATION RESULTS

Table 5 summarizes the simulation comparison arms, and Table 6 lists per-task success over all 100 deployment trials and the 50 held-out trials.

Table 5: Simulation comparison arms. Frozen, Fixed, and ROBO-COP have matched 100-trial deployment sequences; all methods are tested on held-out initializations.

Arm	Closest setting	Actuation	Adaptation	Deployment
$\pi_{0.5}$ only	base VLA	policy	none	eval only
Frozen-policy control	memory-adaptive harness	both	memory	100 trials
Fixed schedule	scheduled policy update	both	memory+policy	100 trials
ASPIRE	agentic skill discovery	programs	skill library	100 search budget
ROBO-COP	full method	both	memory+policy	100 trials

Table 6: **Simulation success rates (%).** Deployment uses 100 trials per task and held-out evaluation uses 50. ASPIRE is evaluated on held-out trials only. Bold marks the best result in each group.

	Deployment			Held-out				
Task	Frozen	Fixed	Ours	$\pi_{0.5}$	Frozen	Fixed	ASPIRE	Ours
Bananas in bin	80	88	**91**	**100**	88	98	88	92
Bananas out of bin	69	**74**	66	74	**90**	88	68	72
Kit three bins	**96**	84	94	8	90	82	**98**	80
Recall stack sort	61	**62**	58	0	6	42	**72**	60
Recover misplaced	69	71	**82**	0	**92**	50	70	86
Recycle cup clutter	46	50	**60**	0	34	38	0	**66**
Recycle sort	51	**56**	54	4	30	**58**	4	50
Restack top on loose	45	**51**	**51**	2	32	26	54	**62**
Biggest fruit	**79**	73	**79**	72	**90**	82	42	76
Except orange	91	94	**97**	20	**96**	94	44	94
Ten-task mean	68.7	70.3	**73.2**	28.0	64.8	65.8	54.0	**73.8**

Late-deployment windows. Table 7 reports deployment success over the final k trials for several window lengths, each an equally weighted mean over the ten tasks. ROBO-COP has the highest mean in every window.

Deployment curves and intervals. Figure 9 shows every deployment trajectory in the main-text mean, one run per method and task. On Recall stack sort, all three methods decline late in deployment. Held-out evaluation starts from the memory saved after trial 100, when the frozen-policy control was already performing poorly; it falls from 61% during deployment to 6% held-out.

Table 7: Deployment success (%) over the final k trials.

Final k trials	10	20	25	30	50
Frozen-policy control	69.0	63.5	67.6	68.7	66.2
Fixed schedule	68.0	65.5	67.2	68.7	68.4
ROBO-COP	**75.0**	**75.5**	**73.6**	**72.7**	**73.4**

Task-level variation. Figure 10 shows the held-out difference from each control on every task, and the range of the mean difference when one task is omitted. Starred tasks are those where ROBO-COP never requested training.

H.1 SUCCESSFUL EPISODES AND RANDOMIZED RESETS

Figures 11 and 12 show one successful deployment episode per task, five frames each. Figure 13 plots the initial object positions over 50 seeded resets per task.

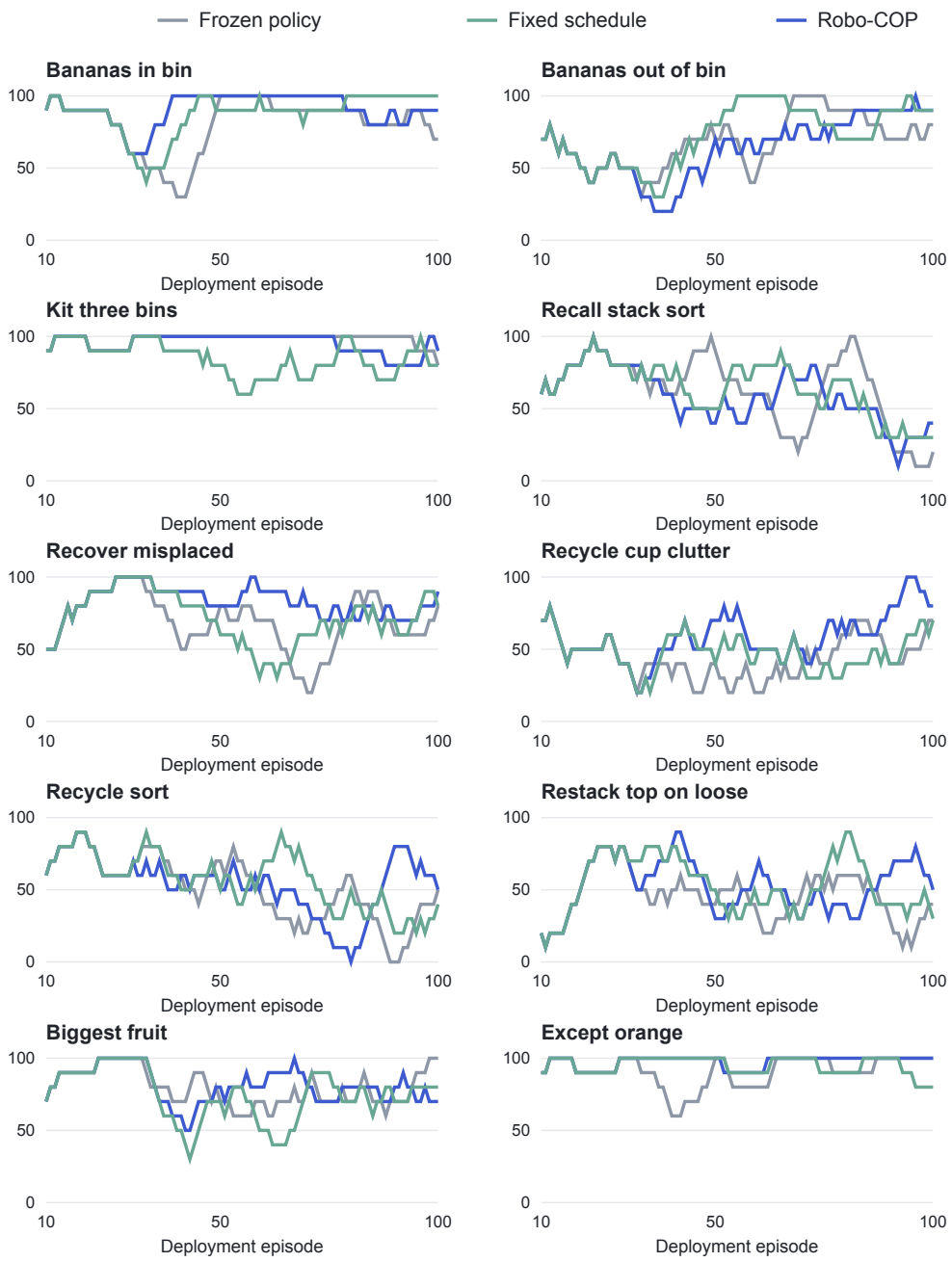

Figure 9: Trailing 10-episode deployment success on each task.

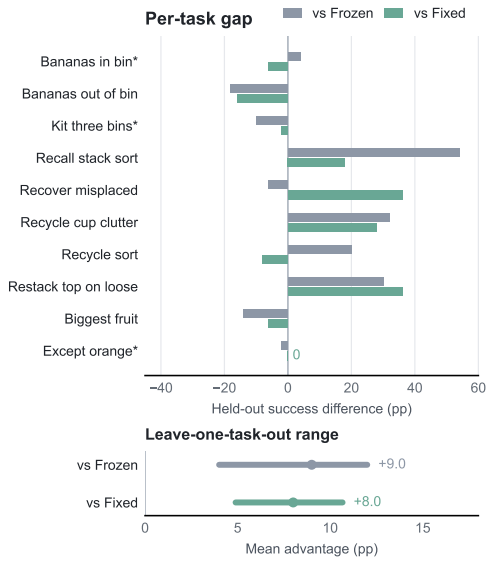

Figure 10: **Held-out task differences.** Bars show ROBO-COP minus each control, in percentage points. The zero for Except orange versus fixed schedule is a tie. The lower panel shows the range of the mean gap when one task is omitted; dots mark the full ten-task means.

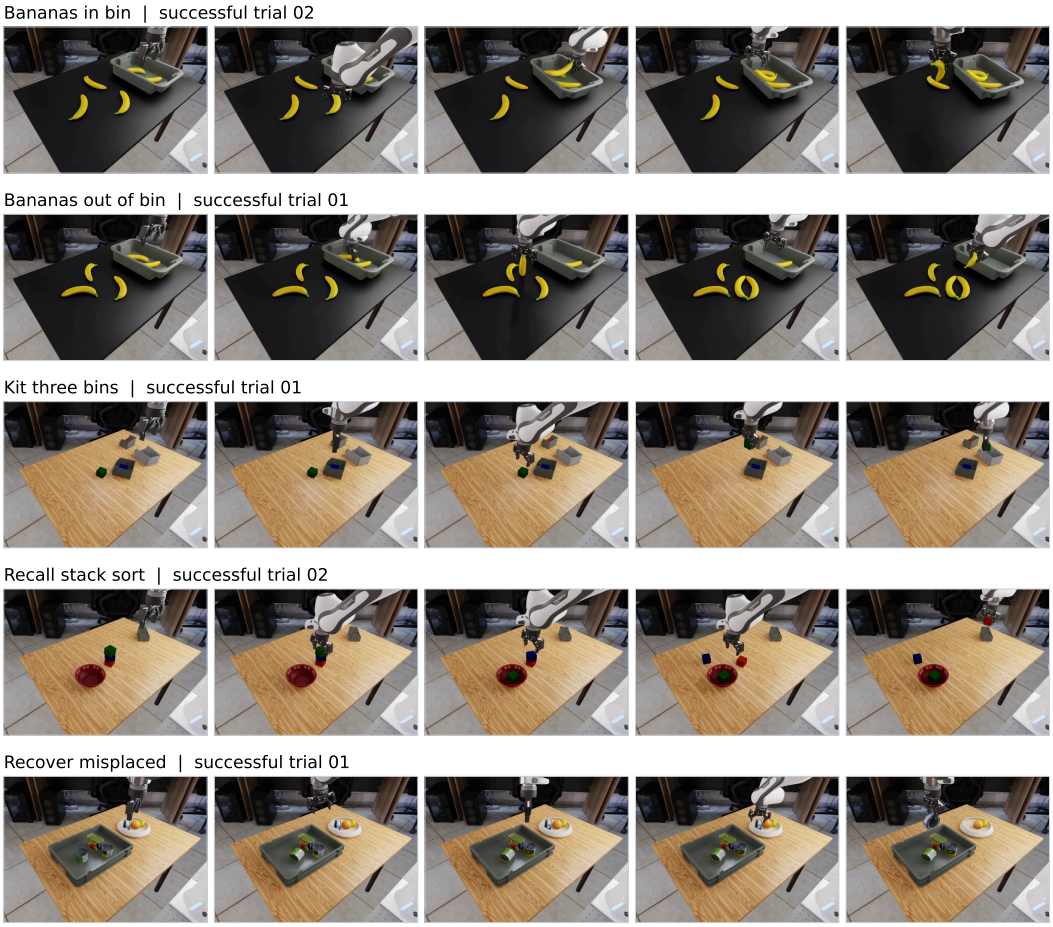

Figure 11: Successful episodes for tasks 1–5. Each row shows five frames from one trial, at roughly 2, 25, 50, 75, and 98% of its duration.

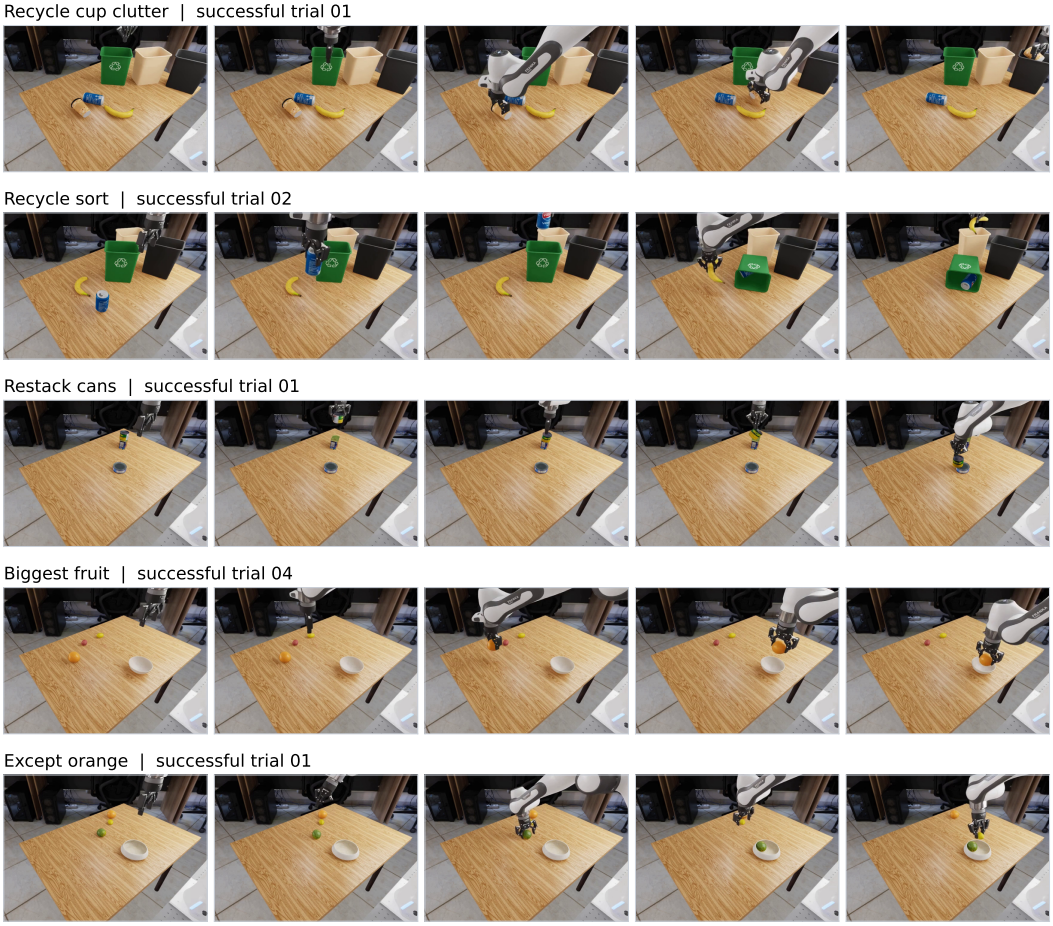

Figure 12: Successful episodes for tasks 6–10, sampled as in Figure 11.

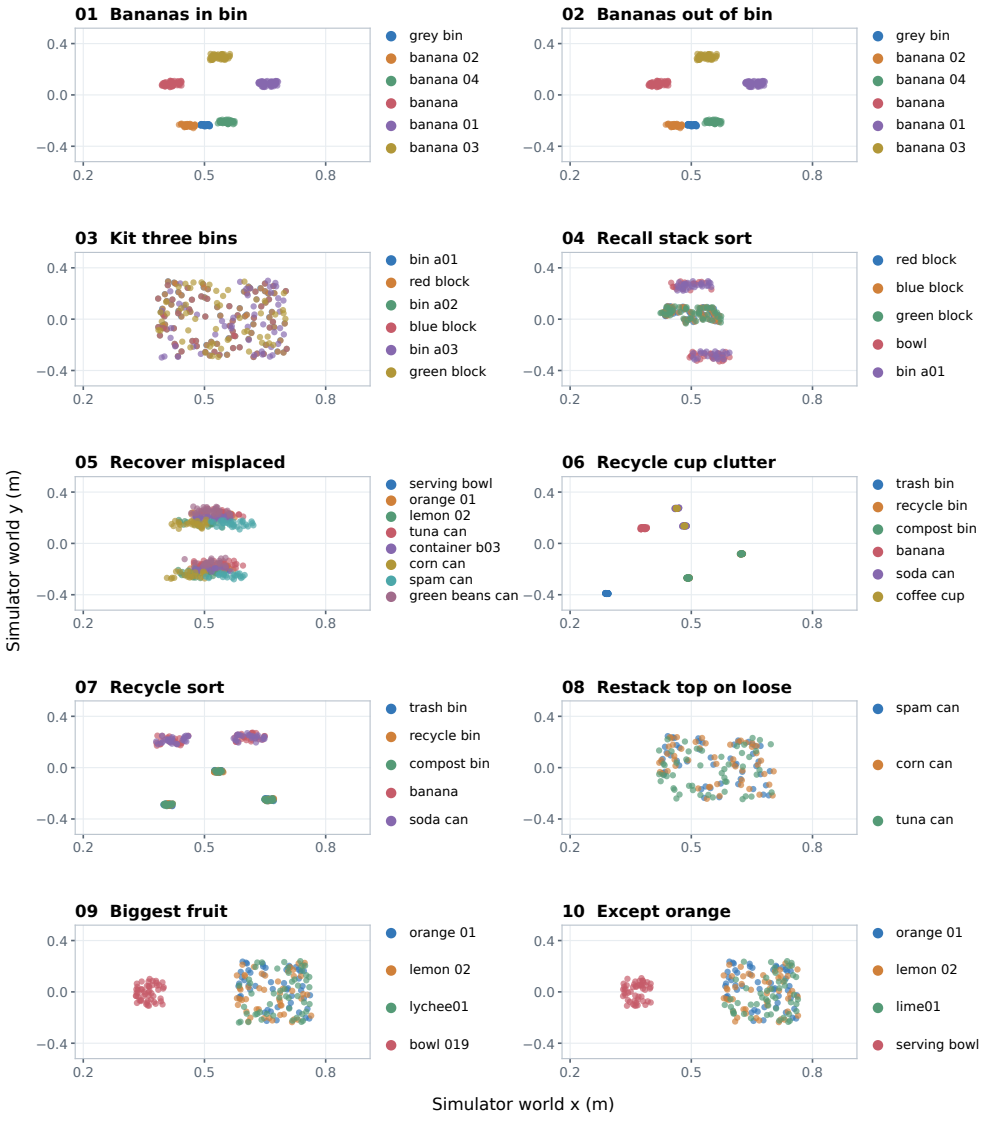

Figure 13: Initial object positions after 50 seeded resets per task, in simulator world coordinates (m). Colors distinguish objects within each task.

I REAL-WORLD EXECUTIONS

Figure 14 shows successful executions of the three real-world tasks in Section 4.3.

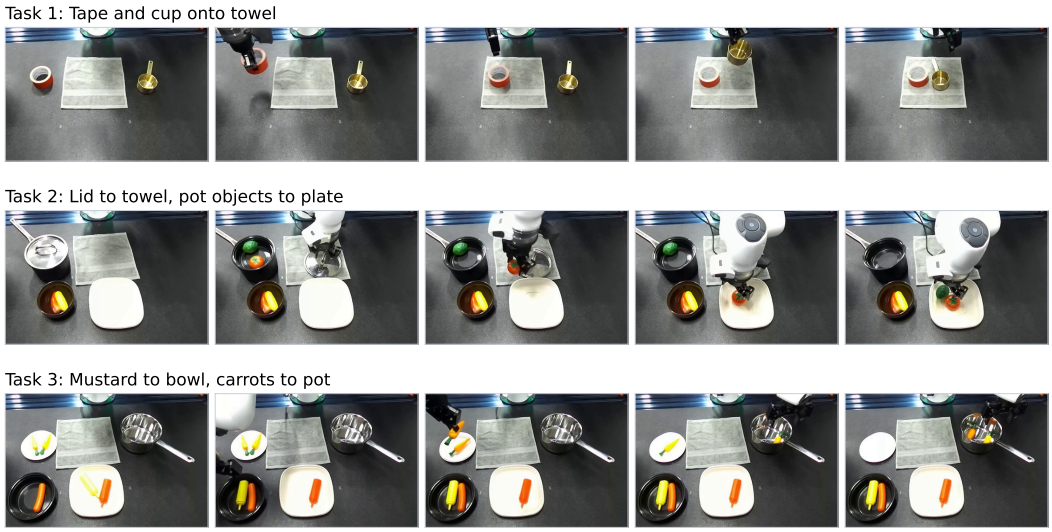

Figure 14: Successful real-world executions of the three tasks, with frames shown in temporal order.