RobotUse: Allocating Computation, Context, and Decisions

Anonymous authors
Paper under double-blind review

Abstract

Robot agents must connect their intended actions to observed outcomes while retaining the context needed to revise their choices over repeated attempts. Existing interfaces often leave these choices inside predefined tools or require agents to manage detailed execution code and its growing history. We introduce RobotUse, a robot agent harness that organizes computation, context, and decisions around specifying and revising physical actions. Agents visually select targets and poses, while the backend handles geometry, motion planning, and control. Subagents retain detailed interactions within each subgoal and return the information needed for subsequent decisions. Continual harnessing lets agents learn from execution by updating a persistent playbook. On RoboLab, RobotUse achieves 45% task success, outperforming CaP-X by 6.7 percentage points while maintaining compact decision contexts and reducing reliance on predefined action abstractions. Furthermore, we show that RobotUse learns from real-world execution despite imperfect feedback and transfers what it learns to subsequent tasks. Project page is available at https://anonrobotuse.github.io/RobotUse/.

1 Introduction

Everyday actions give concrete form to simple intentions: a door opened just enough to pass through, a dial turned until the sound is right. These intentions guide what to change, by how much, and when to adjust, without requiring every movement to be consciously specified.

General-purpose agent design requires deciding which responsibilities belong to model reasoning and which should be delegated. A model's ability to perform a computation does not mean that the computation should be performed through model inference. Programs can process information outside the active context, while subagents can conduct detailed interactions and return the results needed for subsequent decisions (Zhang et al., 2026; Bengre & Curme, 2026). Recent agent harnesses organize computation and context to support long-horizon, multi-step work and retain experience across interactions (Karten et al., 2026a;b). The design question is how to allocate computation and context so that the model retains the decisions that matter without repeatedly re-deriving the backend computations needed to realize them.

However, physical action requires coordinating high-dimensional, continuous motion under geometric and contact constraints. Even a simple goal entails choices about position, orientation, and approach that must change with the scene and observed outcomes. Code-based policies express these choices through programs over perception and control tools (Liang et al., 2023), leaving the agent to connect physical intentions and failures to execution logic. On the other hand, high-level skills delegate execution but can conceal choices needed for adjustment (Ahn et al., 2023). Experience can improve programs and skills (Lu et al., 2026; Chen et al., 2026), yet accumulating abstractions can make the connection between internal decisions and physical outcomes harder to inspect and repair. A robot harness must therefore delegate execution complexity while retaining the context needed to interpret outcomes and guide subsequent actions.

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.

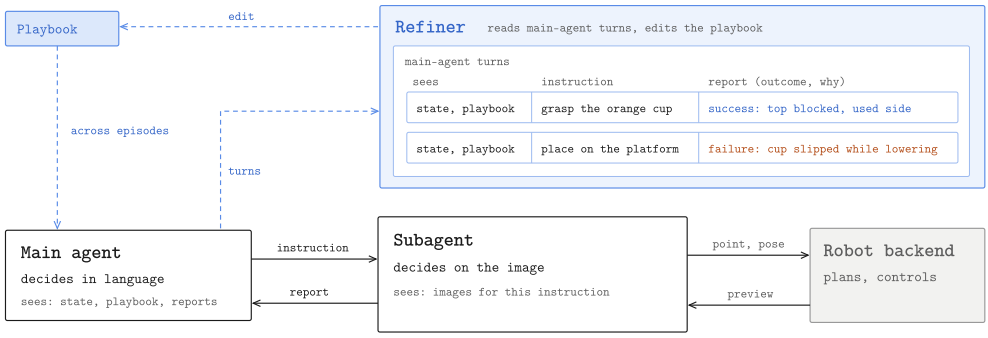


Figure 1: **RobotUse.** The main agent decides in language: from the current state, the playbook, and returned reports, it issues an instruction. The subagent decides on the image, selecting points and poses that the robot backend plans and executes, and returns a report of the outcome and its cause. Between episodes (dashed), a refiner reads only the main-agent turns and edits the playbook.

We propose RobotUse, an intent-based agent harness that organizes computation, context, and decisions around specifying and revising physical intent (Fig. 1). The main agent maintains task intent in language and delegates subgoals and constraints to a general-purpose subagent. The subagent grounds this intent in visual observations by selecting and refining task-relevant targets, positions, and orientations. The backend translates these visually specified choices into executable motion through perception, geometric computation, motion planning, and control. Detailed observation, selection, and recovery histories remain in the subagent's local context, while outcomes, current state, and unresolved constraints return to the main agent to guide subsequent intent and delegation. A playbook guides decomposition and delegation, and accumulates experiential guidance about what worked, under which conditions, and what to try differently. Continual harnessing updates this guidance while keeping the execution backend fixed, allowing experience to change subsequent decisions without rewriting robot control code.

Our evaluation compares task completion, inference cost, token usage, and execution time on RoboLab. We examine dependence on the provided grasp tool and track task completion across successive harness revisions. Across successive playbook refinements, accumulated experience improves task performance. Together, these studies examine how the reasoning demands of robot manipulation depend on the responsibilities assigned to the model and how execution experience can improve the guidance governing those responsibilities.

2 Related Work

Agent harnesses. Agent harnesses organize the computation and information through which a language model acts. ReAct interleaves reasoning with actions and observations to revise plans during interaction (Yao et al., 2023). SWE-agent studies how the design of an agent's tools and environmental feedback shapes its behavior and performance (Yang et al., 2024). Recursive Language Models treat long inputs as an external environment that the model can inspect programmatically and decompose through recursive calls (Zhang et al., 2026). Prime Agent extends this approach with persistent execution, recursive subagent sessions, and retained histories, allowing the model to construct strategies while the harness manages execution and state (Karten et al., 2026a). Deep Agents similarly uses delegation to isolate detailed work from the supervisor's context, with context inheritance available when a subagent needs to continue prior work (Bengre & Curme, 2026). Reflexion retains verbal feedback in episodic memory to improve decisions in subsequent trials (Shinn et al., 2023). Continual Harness updates prompts, subagents, skills, and memory from execution trajectories (Karten et al., 2026b). Our work brings this allocation of computation and context to physical action. We study the boundary at which geometric execution can be

delegated while task-relevant choices remain accessible to the agent, and local interaction histories can be separated while preserving the information needed for subsequent decisions. Experience is retained as playbook guidance over this interface with the execution backend held fixed.

LLMs for robotic manipulation. Vision-language-action models map images and instructions to robot actions, as in RT-2 (Brohan et al., 2023), OpenVLA (Kim et al., 2025), and π_0 (Black et al., 2025b); we include $\pi_{0.5}$ (Black et al., 2025a) as a representative baseline. Early language-model robot agents separate decisions about what to do from the motor skills that carry them out. SayCan grounds skill selection in language relevance and affordance estimates (Ahn et al., 2023), while Inner Monologue uses environmental feedback to revise task plans (Huang et al., 2023). PIVOT exposes continuous action choices to vision-language models by rendering candidate actions on images and refining them through iterative visual selection (Nasiriany et al., 2024). Research has also expanded the agent's responsibility to constructing and improving executable behavior. Code as Policies generates programs that compose perception and control APIs (Liang et al., 2023). ASPIRE diagnoses failures from multimodal execution traces, repairs control programs, and accumulates validated improvements in a reusable skill library (Lu et al., 2026). GaP constructs graphs of perception, planning, and control modules and refines their structure and parameters through simulation rehearsal (Chen et al., 2026). These approaches support flexible behavior by making executable programs and skill compositions objects of generation and revision. Inspired by the earlier separation of decision-making and execution, we develop a harness that retains concrete physical choices at the agent interface while delegating their geometric realization to the backend. Subgoal interactions remain in local contexts, and experience accumulates as guidance about which choices worked and how to adjust them, without modifying the underlying execution code.

3 Problem Formulation

Robot harness. A language-model agent selects commands from a robot interface using a task goal g and the observations and interaction history h_t provided to it. For an interface I with command space $\mathcal{U}_I$, this decision takes the form

$$u_t \sim \pi_\theta(\cdot \mid g, h_t), \qquad u_t \in \mathcal{U}_I. \tag{1}$$

The execution backend interprets the command and performs the computation and control needed to realize it. We represent backend execution together with the physical environment's response as

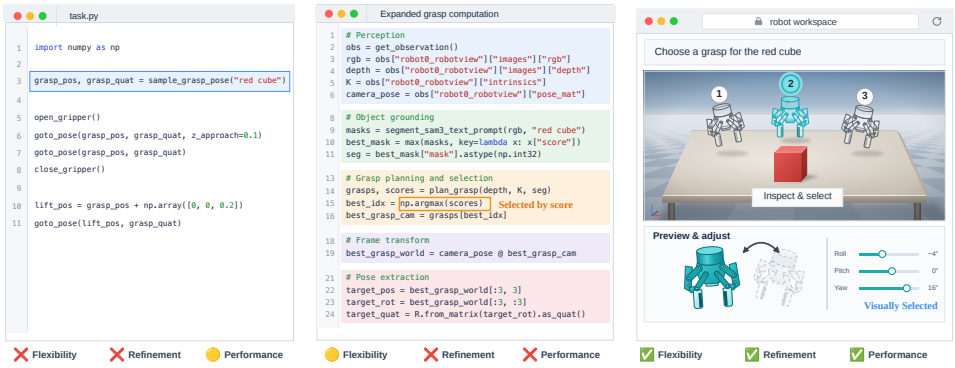
$$s_{t+1} \sim P_I(\cdot \mid s_t, u_t), \tag{2}$$

where s_t is the physical state. The index t denotes an agent interaction step, so a command can span multiple robot control steps. Observations and execution feedback contribute to the history provided for subsequent decisions.

A robot harness organizes this interaction: it determines the available commands, their realization through the backend, and the observations and history supplied to the agent. Even with the same robot capabilities, commands may expose skill invocations, executable code, individual movements, or target poses. The harness therefore allocates decisions between the agent and the execution backend.

Challenges. *Delegating execution while retaining decisions.* Code-, skill-, and tool-based interfaces delegate perception, geometric computation, and control (Liang et al., 2023; Fu et al., 2026; Ahn et al., 2023), but can also embed task-relevant choices, such as grasp location and approach direction, in high-level units or internal scoring rules (Fig. 2a,b). The harness must determine which computations to delegate and which choices to retain for the agent.

Connecting physical outcomes to revisions. A failed grasp may originate in target selection, approach direction, or contact location. The agent needs feedback that connects outcomes to these choices and an interface that lets it revise them.



(a) Code-as-policy (b) Naive expansion (c) RobotUse

Figure 2: **From code-based execution to agent-specified actions.** (a) Conventional code-as-policy composes high-level calls that determine grasp poses internally. (b) A naive expansion exposes perception, grasp generation, and coordinate transformations, but the illustrated routine still selects the highest-scoring grasp; the agent's intended grasp has no explicit selection interface. (c) RobotUse exposes grasp candidates for visual selection and pose previews for adjustment. The agent specifies the intended action, and the backend computes and executes the corresponding motion.

Maintaining context for revision. Revision requires current observations, previous choices, execution results, and attempted adjustments. As this history grows, the harness must retain local execution details while passing the information needed for subsequent task decisions.

We study how to organize commands and context so that the agent can specify and revise task-relevant choices while delegating the computation required to realize them.

4 RobotUse

4.1 Design Principle

Figure 2 illustrates the interface problem in the code-based manipulation (Fu et al., 2026). Expanding a high-level grasp call can expose its implementation while leaving the grasp choice to a score-based rule. RobotUse makes the candidate and its pose explicit objects of agent selection and revision.

RobotUse retains task-relevant physical choices at the agent interface and delegates their geometric realization to the execution backend. The agent chooses targets, locations, orientations, and adjustments; the backend performs geometric conversion, motion planning, and control. For example, grasping a cup establishes an intended outcome, while the selected grasp location and orientation specify how the gripper should engage it. These choices remain available for inspection and revision throughout execution (Fig. 2c).

For spatial execution commands, we instantiate u_t in Eq. 1 as a specification z_t. The division between selecting a specification and realizing it is

$$z_t \sim \pi_\theta(\cdot \mid g_i, h_t^{(i)}, P), \qquad \xi_t \sim B(\cdot \mid s_t, z_t). \tag{3}$$

Here g_i is the current subgoal, $h_t^{(i)}$ its local interaction history, and P the persistent playbook. The agent selects a specification z_t, and the backend B generates the motion ξ_t. Unlike B, P_I in Eq. 2 also includes the environment's response. Pointing, candidate selection, and pose editing let the agent revise the specification using visual and diagnostic feedback.

Accurate specification can require repeated observation and adjustment. RobotUse therefore couples this decision boundary with a context boundary: detailed interactions remain within the subgoal they serve, while returned reports carry the outcomes and constraints needed for subsequent planning. The playbook guides both delegation and local execution, retaining conditional guidance across episodes.

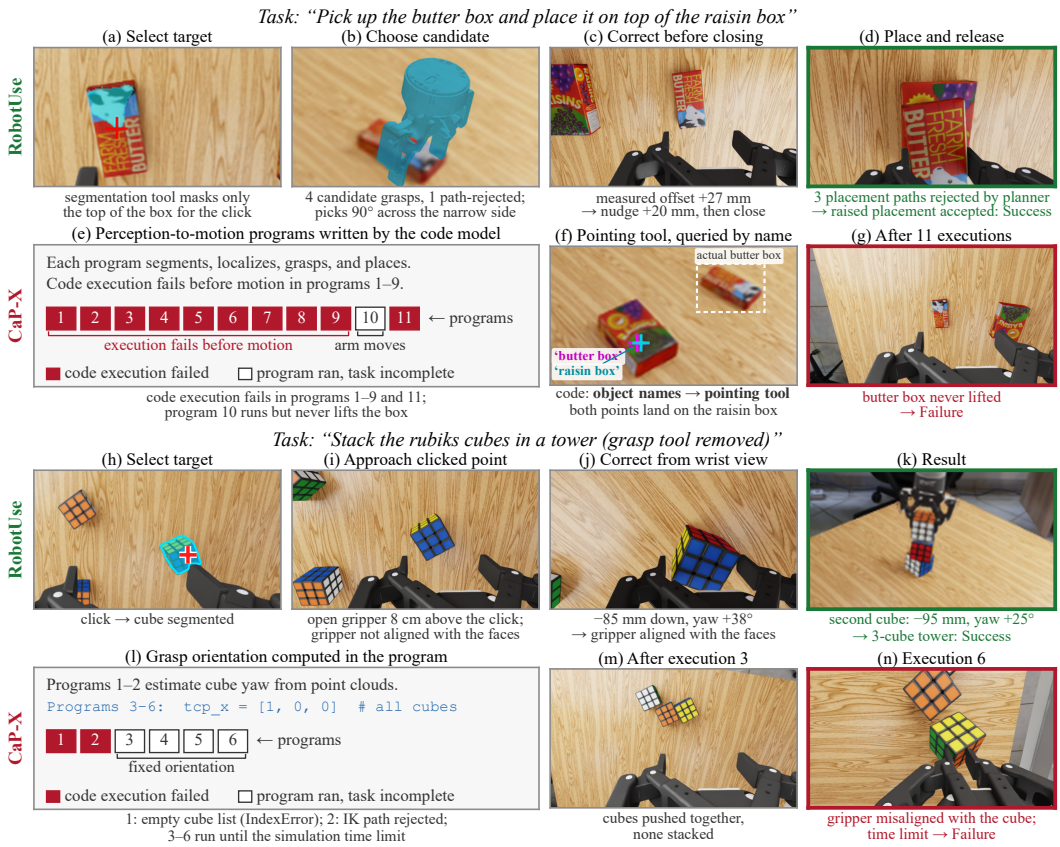


Figure 3: **Qualitative comparison with CaP-X.** (a–g) RobotUse checks imperfect tool results in the images and succeeds (3/3); CaP-X's code execution fails in 10 of 11 attempts, and its name-queried pointing tool marks the wrong box (0/3). (h–n) Without the grasp tool, RobotUse corrects its grasp visually; CaP-X uses one orientation for every cube in programs 3–6.

4.2 Handoff Unit

A handoff consists of a language-defined subgoal g_i, its constraints, and the relevant task context. The main agent selects the subgoal using the overall goal, the playbook, and its history H_i. A general-purpose subagent maintains a separate context $h_t^{(i)}$ for execution. Its returned report updates the main context:

$$H_{i+1} = H_i \oplus \big(g_i, \rho(h^{(i)})\big). \tag{4}$$

Here $\rho(h^{(i)})$ is the report from the local interaction history, and $\oplus$ appends the handoff and report. Both agents use the same model π_θ with separate contexts.

The subagent returns when the subgoal is completed or progress requires reconsidering the subgoal or its constraints. Its report describes the outcome and reason, provides the current image, and records relevant selections, revisions, and unresolved issues. Detailed interaction histories remain local; the main agent receives the returned report.



(a) RoboLab simulation (b) Franka Panda real robot

Figure 4: **Simulation and real-robot evaluation environments.** (a) RoboLab scenes: food pickup, shelf placement, and bottle collection (left to right). (b) Franka Panda setups: Pick and Place, Stack Cube, and Press Button (left to right).

4.3 Closed-Loop Interaction

Within a handoff, the subagent repeatedly observes the scene, specifies a spatial choice, inspects feedback, and revises or executes the choice. Pointing identifies targets or locations, candidate selection chooses among backend-generated alternatives, and pose editing adjusts position or orientation. The backend handles geometric conversion and motion generation for the selected specification. Language supplies the subgoal and its constraints, while visual representations expose the spatial choices used to realize it.

Inspection and execution provide different feedback. Previews and feasibility diagnostics help the agent examine a requested specification before motion; an infeasible request can return diagnostics without executing it. After execution, the agent uses the resulting observation and diagnostic feedback to assess progress and decide whether to revise the specification. For example, an obstructed approach can motivate a change in direction, while an incorrectly selected destination can motivate a position adjustment. The specification provides a common reference for these corrections; feedback must still be interpreted in the current scene.

The local context retains observations, selected candidates, adjustments, and execution feedback across this loop. Subsequent decisions can therefore refer to what was attempted and why it was changed. If a correction requires changing the delegated goal or constraints, the subagent returns the relevant evidence for a new task-level decision.

Implementation. We implement spatial specification as structured tool calls over calibrated RGB-D observations. Target selections are registered as references that subsequent candidate-generation calls consume. Each candidate has a reference used for inspection, adjustment, and execution. Inspecting a candidate produces a pose card with side, top, and closing-plane views of measured points and the robot gripper; an adjustment produces an updated card. The agent can translate the pose in robot-base axes and rotate it about the gripper's contact center before requesting motion. The backend resolves these requests into robot poses and returns planning or execution feedback. This makes the selected candidate the link between visual inspection and the motion request. Appendix B describes tool contracts, context allocation, and playbook updates.

4.4 Continual Harness

RobotUse retains experience across episodes through a persistent playbook (Karten et al., 2026b). Each round, an outer-loop refiner updates the playbook from main-agent trajectories containing delegations, returned reports, and subsequent task-level decisions. Local execution details are available to the refiner through the returned reports.

Refinement records the conditions under which particular spatial choices and recovery strategies were effective, the context required for a delegation, and the findings needed for subsequent planning. New evidence can qualify or revise earlier guidance. The resulting playbook guides both agents in delegating, selecting, and revising actions. The refiner edits the playbook while leaving model parameters and backend code unchanged; the development protocol is detailed in Appendix A.

Table 1: **Task success on RoboLab.** Language-agent methods use 120 episodes each; direct-action policies use 400. Success rates are reported overall and by task difficulty.

Method	Task success (%) ↑			
	Overall (40 tasks)	Simple (21 tasks)	Moderate (13 tasks)	Complex (6 tasks)
Direct-action policies				
$\pi_{0.5}$	30.75	30.48	33.08	26.67
Cosmos 3	42.25	44.76	**45.38**	26.67
Code- and skill-based agents				
mini-SWE	20.83	25.40	17.95	11.11
CaP-X	38.33	42.86	35.90	27.78
Open Robot Skill	20.00	31.75	10.26	0.00
Intent-based agent harness				
RobotUse	**45.00**	**53.97**	35.90	**33.33**

5 Evaluation

We evaluate whether RobotUse lets an agent carry out varied manipulation tasks while retaining control over how it acts. We then examine how context harnessing supports these decisions and how execution experience improves the guidance used in later attempts.

5.1 Experimental Setup

Environments. To evaluate generality across manipulation tasks, we use RoboLab (Yang et al., 2026), whose tabletop tasks range from picking individual objects to arranging multiple objects under spatial and ordering constraints. We select 40 tasks spanning 21 Simple, 13 Moderate, and 6 Complex tasks, all with their default English instructions. To study improvement through physical execution, we use a Franka Emika Panda for Pick and Place, Stack Cube, and Press Button (Figure 4). Objects are randomly positioned within a 50×50 cm workspace observed by a wrist camera and two fixed cameras.

Baselines. To compare how agents use robot capabilities, we include CaP-X (Fu et al., 2026) and mini-SWE (Yang et al., 2024), which generate code, and Open Robot Skill (graph-robots, 2026). Language agents use Gemini-3.8-Flash; mini-SWE shares the low-level backend used by CaP-X. We also include $\pi_{0.5}$ (Black et al., 2025a) and dense Cosmos 3 (NVIDIA, 2026a) as direct-action policies on the same task catalog. The $\pi_{0.5}$ evaluation uses the public OpenPI checkpoint (Physical Intelligence, 2025; NVIDIA, 2026b). Method-specific execution and stopping rules are given in Appendix A.

Metrics. We measure full-task success using the final task verifier. Language-agent results use three trials per task (120 episodes per method); direct-action policies use ten trials per task (400 episodes). The real-robot comparison uses 25 trials per method and task, with RobotUse's playbook frozen after a separate learning phase. For learning from experience, we examine performance across revisions together with the resulting changes in the playbook and subsequent execution.

5.2 Task Performance

Overall Performance. We first compare task completion across the RoboLab catalog to assess how well the complete harness supports manipulation (Table 1). RobotUse succeeds in 54/120 episodes (45.00%), exceeding CaP-X by 6.67 percentage points and Open Robot Skill by 25.00 points. It also exceeds mini-SWE (20.83%), $\pi_{0.5}$ (30.75%), and Cosmos 3 (42.25%). RobotUse has the highest overall, Simple, and Complex success rates, while Cosmos 3 leads on Moderate tasks. Appendix Table 8 reports the detailed task-attribute results.

Flexibility. RobotUse lets the agent adjust its actions from visual feedback, supporting continued execution when tool outputs are imperfect or a tool is unavailable. When stacking one object on another, the agent adjusts its grasp position and placement height from the images to complete the task despite an incomplete segmentation mask and rejected motion paths (Figure 3(a–g)). When we remove the grasp tool, the agent specifies and adjusts the grasp position and orientation directly (Figure 3(h–n)). Across 40 tasks, RobotUse retains 75% of its original success rate, falling from 50.00% to 37.50%; CaP-X falls from 45.00% to 17.50% (Figure 5).

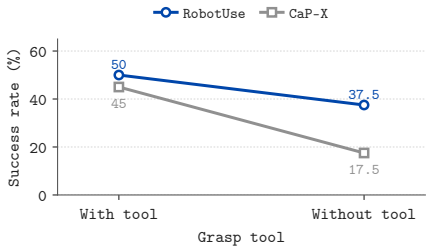


Figure 5: **Robustness of RobotUse.** Success with and without the grasp tool; 40 tasks per condition.

5.3 Context Harnessing

Fig. 6 shows the effect of context handoff across 120 episodes. We compare RobotUse with a single agent that performs all work otherwise handled by subagents, using the same tools and role instructions within one continuous conversation. Without subagents, mean input per call increases by 4.2× and the average episode peak by 3.5×, raising total token usage by 4.9× and API cost by 2.4×. Task success also drops from 45.0% to 40.8%. These results suggest that language reports from local visual interactions preserve sufficient context for subsequent decisions, while accumulating the full interaction history may burden execution. Appendix A.4 gives the execution protocol and Table 3 reports the numerical results.

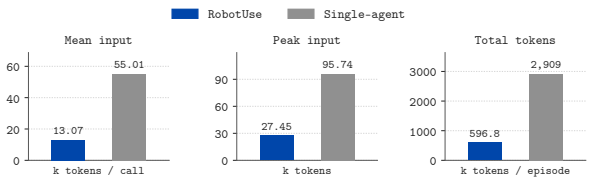


Figure 6: **Context handoff.** Token usage for RobotUse and single-agent execution across 40 tasks and three seeds (120 episodes per condition). Mean and peak input are averaged across episodes; total tokens sum input and output within each episode.

5.4 Learning from Experience

Learning from Failures. Connecting the agent's choices to their observed outcomes makes execution failures informative: the agent can examine what went wrong and retain the resulting lessons in its playbook to guide subsequent choices. We trace this process across four harness revisions on the same 40-task RoboLab catalog. Losing track of targets during execution motivates a goal checklist. Misinterpreting spatial directions motivates guidance that defines them in the robot frame, while leaving cubes at the original stack motivates re-observation before declaring completion. These lessons change how the agent interprets goals and judges the results of its actions. Figure 8 illustrates the resulting choices when placing an object "in front of the bowl" and completing an unstacking task. Success increases from 19/40 to 22/40 across revisions (Figure 7); round 3 evaluates one frozen policy in 40 new episodes. Appendix A.6 details the revision protocol.

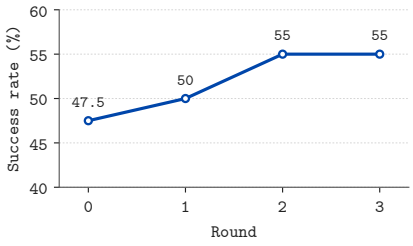


Figure 7: **Learning from experience.** Success over 40 tasks per revision; protocol in Appendix A.

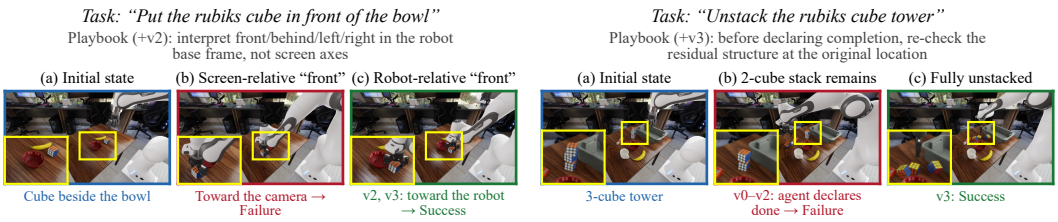


Figure 8: **Qualitative examples of playbook changes.** Left: robot-frame guidance grounds "in front of the bowl" relative to the robot. Right: checking for remaining cubes enables the agent to complete unstacking.

Rapid Adaptation. On the Franka Panda, RobotUse adapts through playbook refinement while keeping model parameters and the execution backend fixed. We compare it with fine-tuned $\pi_{0.5}$ and Cosmos 3 baselines (Figure 9). RobotUse first learns to pick up a cube and place it in a tray by reviewing each attempt and revising its playbook until the first success. The accumulated guidance also supports rapid adaptation to new tasks: carrying the playbook over to stacking a cube on another cube yields success on the first attempt. On the subsequent button-pressing task, only the first attempt fails, and the agent succeeds on its second attempt after further playbook refinement. With the playbook frozen at each task's first success, RobotUse succeeds in all 25 separate evaluation trials per task. These results show how experience retained in the playbook supports reliable execution and rapid adaptation to subsequent tasks without model fine-tuning.

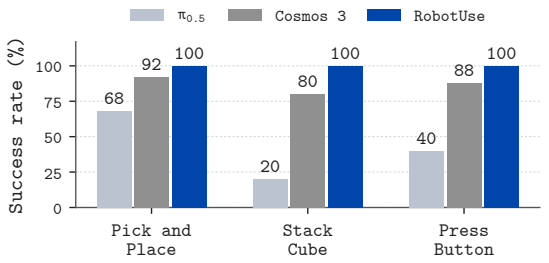


Figure 9: **Real-robot task success on Franka Panda.** RobotUse freezes its playbook at the first task success and succeeds in all 25 separate evaluation trials per task.

6 Conclusion

We introduced RobotUse, a robot agent harness that allocates computation, context, and decisions around specifying and revising physical actions. Visual action representations let agents express and revise how the robot should act, subgoal-local contexts retain the details needed for execution, and a persistent playbook carries experience into subsequent attempts. On RoboLab, RobotUse achieves 45% task success and supports manipulation when the grasp tool is removed. Our context study shows lower token usage with subgoal delegation, while simulation and real-robot experiments illustrate how execution experience informs later decisions through playbook refinement. These findings support harness design as a means of improving robot agents: delegating geometric computation and control, preserving the physical choices agents need to revise, and retaining experience in guidance that shapes future action.

AI Use Statement

We used LLM-based assistants (Claude by Anthropic and ChatGPT by OpenAI) during the preparation of this work. Specifically, these tools were used to (i) polish the grammar, phrasing, and clarity of the manuscript text, and (ii) assist with writing and debugging code used in our experiments. All research ideas, experimental designs, and analyses were conceived and directed by the authors; LLM assistance was limited to implementation support carried out under our explicit instruction and supervision. The authors have reviewed and verified all AI-assisted content and take full responsibility for the contents of this paper.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances. In Karen Liu, Dana Kulic, and Jeff Ichnowski (eds.), *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.

Thushanth Bengre and Chester Curme. Organizing context in a multi-agent harness. LangChain Blog, September 2026. URL https://www.langchain.com/blog/organizing-context-in-a-multi-agent-harness.

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 Joseph Lim, Shuran Song, and Hae-Won Park (eds.), *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*, Los Angeles, CA, USA, June 2025b. doi: 10.15607/RSS.2025.XXI.010. URL https://www.roboticsproceedings.org/rss21/p010.html.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-2: Vision-language-action

models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *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.

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*, 2026. URL https://arxiv.org/abs/2607.05369.

Christopher Clark, Jieyu Zhang, Zixian Ma, Jae Sung Park, Mohammadreza Salehi, Rohun Tripathi, Sangho Lee, Zhongzheng Ren, Chris Dongjoo Kim, Yinuo Yang, Vincent Shao, Yue Yang, Weikai Huang, Ziqi Gao, Taira Anderson, Jianrui Zhang, Jitesh Jain, George Stoica, Winson Han, Ali Farhadi, and Ranjay Krishna. Molmo2: Open weights and data for vision-language models with video understanding and grounding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2026. URL https://arxiv.org/abs/2601.10611.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi "Jim" Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026. URL https://openreview.net/forum?id=4JRO9plGAI.

graph-robots. open-robot-skills: Skill and tool bundles for GaP. GitHub repository, 2026. URL https://github.com/graph-robots/open-robot-skills. Accessed September 26, 2026.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In Karen Liu, Dana Kulic, and Jeff Ichnowski (eds.), *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.

Seth Karten, Alex L. Zhang, Kevin Thomas, Sebastian Müller, Elie Bakouch, Daniel Auras, Mika Senghaas, Fares Obeid, Konstantin Dunas, Johannes Hagemann, and Sami Jaghouar. Prime agent: A self-improving RLM harness. *arXiv preprint arXiv:2608.23552*, 2026a. URL https://arxiv.org/abs/2608.23552.

Seth Karten, Joel Zhang, Tersoo Upaa, Jr., Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents. *arXiv preprint arXiv:2605.09998*, 2026b. URL https://arxiv.org/abs/2605.09998.

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 Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *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.

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 *2023 IEEE International Conference on Robotics and Automation*

(*ICRA*), pp. 9493–9500. IEEE, 2023. doi: 10.1109/ICRA48891.2023.10160591. URL https://doi.org/10.1109/ICRA48891.2023.10160591.

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.

Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, Quan Vuong, Tingnan Zhang, Tsang-Wei Edward Lee, Kuang-Huei Lee, Peng Xu, Sean Kirmani, Yuke Zhu, Andy Zeng, Karol Hausman, Nicolas Heess, Chelsea Finn, Sergey Levine, and Brian Ichter. PIVOT: Iterative visual prompting elicits actionable knowledge for VLMs. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 37321–37341. PMLR, 2024. URL https://proceedings.mlr.press/v235/nasiriany24a.html.

NVIDIA. Cosmos 3: Omnimodal world models for physical AI. *arXiv preprint arXiv:2606.02800*, 2026a. URL https://research.nvidia.com/labs/cosmos-lab/cosmos3/technical-report.pdf.

NVIDIA. RoboLab: Pi0 family (OpenPI) policy documentation. GitHub documentation, 2026b. URL https://github.com/NVlabs/RoboLab/blob/main/policies/pi0_family/README.md. Documents the pi05_droid_jointpos checkpoint. Accessed September 26, 2026.

Physical Intelligence. OpenPI: Open-source models and packages for robotics. GitHub repository, 2025. URL https://github.com/Physical-Intelligence/openpi. Accessed September 26, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

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 *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2026. URL https://arxiv.org/abs/2604.09860.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2026. URL https://arxiv.org/abs/2512.24601.

A Additional Evaluation Details

A.1 Evaluation Protocols

Tasks and protocol. We evaluate 40 RoboLab tasks with their default English instructions: 21 Simple, 13 Moderate, and 6 Complex tasks. The main comparison aggregates three runs indexed by seeds 0–2, with one episode per task in each run (120 episodes per method). Full-task success requires a successful final verifier result; all other episodes count as failures. Seed 0 uses fixed initial placements, while seeds 1 and 2 randomize dynamic-object positions independently within ±0.10 m in X and Y.

Baselines and execution conditions. We compare RobotUse, CaP-X, mini-SWE, and Open Robot Skill. mini-SWE v1.17.5[1] uses the CaP-X M4 low-level interface with a persistent Python session. The completed language-agent reports use Gemini-3.8-Flash through OpenRouter for language-model inference. These are system-level comparisons with method-specific stopping rules. CaP-X uses the default set by its authors: an initial code execution and ten regenerations. RobotUse uses a 60-minute wall-clock limit per episode. Open Robot Skill uses a $1.50 API-cost limit per episode. We set this limit because, without it, some Open Robot Skill episodes kept running in repeated loops and exceeded $50 without succeeding, whereas successful episodes completed within $1. The final API response may exceed this limit before further requests are stopped.

CaP-X motion on RoboLab. CaP-X's RoboLab integration reuses the simulator interface, joint controller, and inverse-kinematics solver of RobotUse's backend. For a fair comparison, its motion primitives apply the same path check as that backend: each commanded motion follows a straight-line Cartesian path interpolated at 3 mm and 0.025 rad, and is rejected when inverse kinematics fails or a joint changes by more than 0.2 rad between waypoints. The upstream implementation instead moves in joint space to an inverse-kinematics solution. A rejected motion raises an exception in the generated program, which CaP-X can address when it regenerates code.

Direct-action reference. We additionally evaluate $\pi_{0.5}$ (Black et al., 2025a) using the public `pi05_droid_jointpos` checkpoint through the RoboLab-native OpenPI client (Physical Intelligence, 2025; NVIDIA, 2026b). Each of the same 40 tasks runs in ten parallel environments, yielding 400 completed episodes with the default English instruction and native simulation limits. The runner uses native initialization and exposes no seed argument, so this reference uses a separate initialization and repetition protocol from the three language-agent runs. We aggregate the native binary success field across all episodes; the separately logged partial-task score is not used for success rates. $\pi_{0.5}$ achieves 123/400 successes (30.75%): 64/210 Simple (30.48%), 43/130 Moderate (33.08%), and 16/60 Complex (26.67%). Agent turns and external API cost are inapplicable to this local policy. Its batched timing records are kept separate from the per-episode agent process-time comparison.

Cosmos 3. The dense Cosmos 3 (NVIDIA, 2026a) baseline evaluates each of the selected 40 tasks with ten rollouts, yielding 400 episodes. Success counts are 94/210 on Simple tasks, 59/130 on Moderate tasks, and 16/60 on Complex tasks, totaling 169/400.

Resource accounting. We average reported API cost, turns, tokens, and process time over all episodes. Effective tokens equal total tokens minus cached input tokens; k denotes 1,000 tokens. Local perception-model and GPU costs are excluded. The main-agent comparison counts only RobotUse main requests and CaP-X code-generation requests, as defined below. Cost per success divides a run's total episode cost by its confirmed successes; Table 7 averages these ratios across the three runs. Process time measures per-episode elapsed time rather than parallel batch duration.

[1] https://github.com/SWE-agent/mini-swe-agent/tree/v1.17.5

A.2 Harness Revision Protocol

This sequence measures iterative system development on the evaluated task catalog. All rounds use a 60-minute wall-clock limit per episode. Round 3 uses a single frozen policy. Evaluation costs exclude development probes and playbook-update costs.

A.3 Grasp-Tool Comparison

RobotUse uses a click location and relative height for the approach, with position and rotation adjustments, in its no-grasp run. The compared runs share seed 1 and randomized initial placements. These completed runs show a smaller observed success reduction for RobotUse.

A.4 Context Handoff Ablation

Single-agent execution. The single-agent condition in Section 5.3 uses the model, tools, role instructions, playbook v0, turn budgets, and execution backend of RobotUse. One agent performs all work otherwise handled by subagents within its own conversation. Each subgoal and its subsequent local interactions enter that conversation, and each role's instructions enter once, when the role is first used. Per-turn budgets and current images are supplied as in RobotUse. We evaluate all 40 tasks across three seeds (0–2), with one episode per task and seed (120 episodes per condition), using the existing RobotUse main-evaluation records for comparison. Seed 0 uses fixed placements, while seeds 1 and 2 randomize dynamic-object positions in X and Y as described above. The single-agent condition has a wall-clock limit of 60 minutes per task, with a 120-minute limit for CleanUpToysTask. Full-task success requires a successful final verifier result; failed and unverified endings remain in the denominator.

Accounting. For the input statistics in Table 3, we compute each episode's mean and maximum recorded input length, including cached input, then average each statistic equally across episodes. Total tokens include input and output over all model calls per episode, and API cost includes all agent roles. RobotUse uses 596.8k tokens and $0.511 per episode, compared with 2,909k tokens and $1.221 for single-agent execution. The average numbers of model calls per episode are 41.6 and 41.1, respectively. Success counts are 54/120 for RobotUse and 49/120 for single-agent execution.

A.5 Main-Agent Turn and Context Accounting

We recount the main agent from unique `agent_input` events identified by session and step with role `prime`. A started decision with an unanswered request remains a turn; HTTP transport retries do not add turns.

Context length uses the provider's `prompt_tokens` for each main-agent request, including cached inputs. We compute each episode's average and observed maximum over recorded usage, then average each statistic equally across its 40-episode campaign. Completion tokens and subagent calls are excluded from these context statistics but remain in the system-wide resource accounting.

For CaP-X seed 0, we read the archived `provider_inputs.jsonl` and `provider_audit.jsonl` files and count unique request identifiers with role `code`. This yields 259 main requests with complete input-token usage. VDM (Fu et al., 2026) and Molmo (Clark et al., 2026) calls are excluded from main-agent statistics. The cross-method main-agent comparison covers seed 0.

A.6 Context and Learning Results

Table 3 gives the numerical context comparison in Section 5.3. Input lengths are computed per episode and then averaged across all 120 episodes. Tables 4–6 retain the detailed results and revision records for Section 5.4.

Table 2: **Additional RobotUse main-agent accounting.** Each row contains 40 episodes. Definitions and coverage follow Table 9. Baseline seed 0 also serves as refinement round 0.

Run	Main turns/ep.	Mean input (k)	Peak input (k)	Coverage
Baseline seed 0 / round 0	10.62	9.12	12.58	424/425
Baseline seed 2	12.40	9.75	13.79	496/496
Refinement round 1	11.08	10.01	13.64	443/443
Refinement round 2	10.70	13.38	18.73	427/428
Refinement round 3	11.25	14.37	20.13	450/450

Table 3: **Context handoff ablation on RoboLab.** Forty tasks across seeds 0–2, one episode per task and seed (120 episodes per condition). Input/call is the mean input tokens per model call and Peak is the largest input to a single call, both computed per episode and averaged across episodes. Total tokens sum input and output over all calls per episode.

Configuration	Success (%) ↑	Mean input (k/call)	Peak input (k)	Total tokens (k/ep.)
RobotUse	**45.0**	**13.07**	**27.45**	**596.8**
Single-agent	40.8	55.01	95.74	2909

Table 4: **Continual harnessing on RoboLab.** Each round contains 40 seed-0 episodes. Cost includes all agent roles; turns count main-agent decisions. Protocol details appear in Appendix A.

	Success (%) ↑				Usage per episode	
Round	Overall	Simple	Moderate	Complex	Cost ($)	Main turns
0	47.50	61.90	30.77	33.33	0.4787	10.62
1	50.00	66.67	38.46	16.67	0.5409	11.08
2	**55.00**	71.43	46.15	16.67	0.5277	10.70
3	**55.00**	76.19	30.77	33.33	0.5401	11.25

A.7 Additional Resource and Difficulty Breakdowns

Effective tokens equal total tokens minus cached input tokens; k denotes 1,000 tokens. Cost per success is the mean of run-level total-cost-to-success ratios. RobotUse uses more API cost and time than CaP-X and less API cost than Open Robot Skill.

Success by task attribute. Table 8 breaks the success rates of Table 1 down by RoboLab's task attributes, grouped into its visual, relational, and procedural categories. Among language/code agents, RobotUse has the highest or tied-highest success on nine of ten attributes; the exception is reorientation, which contains two tasks. Cosmos 3 leads on the three relational attributes and on semantics and affordance.

Main-agent interaction and context. Table 9 compares the main agent in RobotUse with the code-generating agent in CaP-X over the same 40 seed-0 tasks. RobotUse uses 10.62 main requests per episode and CaP-X uses 6.48. Their episode-averaged input contexts are 9.12k and 7.37k tokens per main request, respectively; average observed peaks are 12.58k and 11.90k. These statistics describe the input received directly by the main agent. CaP-X processes images through VDM and passes its descriptions to the code agent; the comparison therefore separates main-agent context burden from system-wide visual processing. Both runs use fixed initial placements and their recorded method-specific stopping rules. Appendix A.5 gives the aggregation rules and additional RobotUse runs.

Table 5: **Evaluation usage across harness revisions.** Per-episode token counts are in thousands. Cost/success divides evaluation costs by confirmed successes.

	Tokens per episode (k)		Time (s/ep.)	Cost ($/success)
Round	Total	Effective		
0	555.69	502.59	1774.55	1.0077
1	628.00	562.84	2082.74	1.0817
2	632.88	552.59	1924.91	0.9594
3	660.72	566.46	1785.98	0.9820

Table 6: **Playbook changes across harness revisions.** Representative additions (+), removals (−), and modifications (M).

Version	Cause (observed issue)	Change
v0	—	Baseline, brief skeletal guide.
v1	Targets were lost or miscounted mid-task.	(+) Build a checklist of targets, counts, order, and relations.
	Rotation restrictions blocked a needed pose.	(−) Remove the yaw-only constraint and allow tilt.
v2	Screen and robot directions were confused.	(+) Define front/back/left/right in the robot frame.
	Rejected grasps were retried unchanged.	(+) Diagnose the rejection before retrying.
v3	Residual stacked objects were overlooked.	(+) Re-check the original location before declaring unstacking complete.
	Gripper closure was mistaken for reorientation.	(M) Confirm orientation changes from observations.

Table 7: **Additional inference usage.** Means over three runs; mini-SWE costs $0.1706 per episode; token counts are in thousands per episode and time is seconds per episode. Cost/success is the mean of the three run-level ratios.

Method	$/ep.	Total tokens (k)	Effective (k)	Time (s)	$/success
CaP-X	0.1672	119.94	119.20	642.80	0.4417
Open Robot Skill	1.3800	5779.41	1230.81	1185.77	7.1346
RobotUse	0.5110	596.80	555.33	1880.51	1.1984

B RobotUse Implementation

Tool interface and spatial references. The implementation separates model-facing tool schemas from backend geometry and motion routines. A selected target is registered with the current observation and passed to subsequent tools by reference. The grasp and placement tools return candidate references, camera projections, and measured metadata. The agent inspects a candidate before selecting or editing it; the backend produces the detailed pose card on inspection and caches it for that candidate. An edit creates a revised candidate and its updated preview. Reference checks associate selections with the current interaction state, preventing an old candidate from being treated as a fresh executable selection after the scene changes.

Pose inspection and execution. Detailed cards combine side, top, and closing-plane views of measured points and the robot gripper mesh. Translation edits use robot-base XYZ axes in millimetres, while rotation edits use local gripper axes about the gripper contact

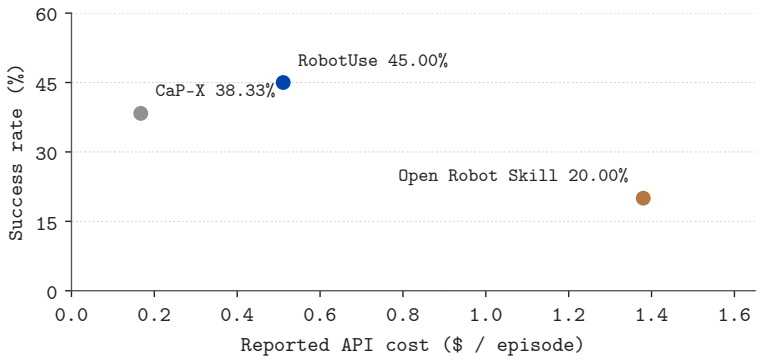


Figure 10: **Task completion against reported API cost.** Each point summarizes the three reported runs of RobotUse, CaP-X, or Open Robot Skill. API cost averages all 120 episodes per method and excludes local perception and GPU costs. RobotUse attains higher success with higher reported cost than CaP-X and lower cost than Open Robot Skill.

Table 8: **Task success by RoboLab task attribute.** Success rates (%) over the episodes of Table 1. Attributes and categories follow RoboLab's task metadata; a task can carry several attributes, and parentheses give the number of tasks.

	Visual			Relational			Procedural			
Method	Color (8)	Sem. (16)	Size (4)	Conj. (4)	Count. (3)	Spatial (8)	Afford. (4)	Reor. (2)	Sort. (4)	Stack. (3)
Direct-action policies										
$\pi_{0.5}$	21.25	26.88	55.00	50.00	83.33	13.75	17.50	25.00	**22.50**	16.67
Cosmos 3	28.75	**31.88**	42.50	**72.50**	**90.00**	**48.75**	**30.00**	20.00	**22.50**	6.67
Language agents										
mini-SWE	33.33	10.42	8.33	33.33	22.22	33.33	8.33	0.00	0.00	22.22
CaP-X	37.50	29.17	25.00	58.33	66.67	25.00	25.00	**33.33**	0.00	**66.67**
Open Robot Skill	16.67	6.25	33.33	58.33	22.22	16.67	0.00	**33.33**	0.00	0.00
RobotUse	**41.67**	31.25	**66.67**	58.33	88.89	33.33	25.00	16.67	16.67	**66.67**

Table 9: **Main-agent requests and input context on the 40 seed-0 tasks.** RobotUse counts role `prime`; CaP-X counts role `code`, excluding VDM and Molmo calls. Mean input averages per-episode mean input lengths; peak input averages observed per-episode maxima. Input lengths include cached tokens and use thousands of tokens. Coverage counts main requests with recorded input usage.

Method	Main turns/ep.	Mean input (k)	Peak input (k)	Coverage
CaP-X	6.48	7.37	11.90	259/259
RobotUse	10.62	9.12	12.58	424/425

center. Grasp and placement heights can be specified relative to a supported measured reference. The backend resolves the reference into a pose, checks the request, and computes the motion. Placement motion keeps the gripper closed, with release requested separately after observation. Tool feedback reports planning or execution status; task completion is determined by the native task verifier.

Context allocation. The runtime implements delegation with a fresh message list initialized from a role-scoped prompt and the delegated task. Pointing, grasp selection, placement, and local pose review use prompts and tool subsets for their respective operations. These prompt roles use the general-purpose language model specified in the evaluation setup. A

Table 10: **Performance with and without the grasp tool.** Each condition contains 40 seed-1 tasks. CaP-X success decreases by 27.50 percentage points and RobotUse by 12.50 points. RobotUse specifies no-grasp motions through a clicked location, height, and pose adjustments.

Method	Grasp tool	Overall (%)	Simple	Moderate	Complex	$/ep.
CaP-X	With	45.00%	52.38%	38.46%	33.33%	0.1704
CaP-X	Without	17.50%	23.81%	7.69%	16.67%	0.2024
RobotUse	With	50.00%	47.62%	53.85%	50.00%	0.5704
RobotUse	Without	37.50%	47.62%	23.08%	33.33%	0.4100

Table 11: **Resource use in the grasp-tool comparison.** Tokens include all agent roles and are thousands per episode.

Method	Grasp tool	Turns	Total (k)	Effective (k)	Time (s)	$/success
CaP-X	With	13.70	126.43	125.52	622.63	0.3788
CaP-X	Without	15.75	145.66	143.09	963.83	1.1568
RobotUse	With	—	641.37	584.98	1760.33	1.1407
RobotUse	Without	—	420.13	370.87	1359.01	1.0933

child accumulates its own tool interactions; the parent receives the returned selection, status, reason, and relevant feedback rather than the full child transcript. Images and references are passed through the handoff and tool results. This implements the separation between task-level history and the detailed context used to make a spatial decision.

Playbook loading and revision records. The playbook is a versioned Markdown document with common guidance and role-specific sections. Each agent prompt receives the common section and the section for its current role. The loader records a SHA-256 hash of the document, and episode records preserve the selected version and configuration. Later revisions therefore change the guidance supplied to subsequent decisions while preserving earlier documents for comparison. The reported refinement campaign is detailed in Appendix A.