ROBOARCADE: BENCHMARKING AGENTIC ROBOT CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Improvements in multi-modal capabilities have enabled general agents to act directly as robot policies, a setting we refer to as agentic robot control (ARC). We introduce RoboARCade, a benchmark of long-horizon manipulation tasks across multiple embodiments. It is designed to test in-context adaptation, memory, and reasoning for precise control, extending beyond the capabilities emphasized by prior robotic policy benchmarks. The benchmark differentiates leading agents: GPT-6 Astra achieves 49.2% compared with 41.5% for Claude Fable 5.1, leaving substantial room for improvement. We equip agents with different interfaces to study their effects on performance and behavior. Our benchmark identifies several bottlenecks that prevent agents from zero-shot completion of robotic tasks: identifying an approach, executing it precisely, and recognizing subtask completion. Additionally, we find that performance can be improved without additional human input through retries, successful demonstrations, and prompt optimization.

1 INTRODUCTION

There has been an increasing number of works that use foundation models (FMs) or agents powered by FMs to control robots in various ways, such as Vision-Language-Action (VLA) models (Zitkovich et al., 2023; Kim et al., 2025; Black et al., 2025), hierarchical approaches (Ichter et al., 2023; Huang et al., 2023; Belkhale et al., 2024; Shi et al., 2025; Hu et al., 2026b) and Code-as-Policies (CaP) (Liang et al., 2023; Fu et al., 2026). More recently, advances in frontier multi-modal FMs make it possible to let agents directly observe the world via pixels and control the robot by outputting raw joint values or controller targets (end-effector pose, gripper state, etc.) (Hu et al., 2026a), a setting that we refer to as agentic robot control (ARC).

ARC provides a straightforward way to leverage the FMs' strong capability in general visual perception, spatial reasoning, planning, memory, and closed-loop control with execution feedback. These capabilities are ideal for solving certain categories of robot manipulation tasks. Recent evaluations have demonstrated that agents can achieve non-trivial success on manipulation tasks through end-effector control, without robot-specific fine-tuning (Berman et al., 2026; Hu et al., 2026a; Zhang et al., 2026). However, the agentic framing of robotic control opens up more opportunities to evaluate and understand these agents beyond just their task success rates. Since these agents controlling the robot are also capable reasoning and coding agents, they can identify their own failure, reflect on past failures to better instruct themselves, or even write the tools and *interfaces*[1] through which they command the robot.

We build RoboARCade to systematically evaluate the capabilities and limitations of current agents and control interfaces in ARC. RoboARCade includes a diverse suite of tasks curated from existing benchmarks covering single and bimanual manipulation, assembly, physical puzzles, and high-precision biolab-inspired procedures, bringing them into a single, unified codebase. Fig. 1 shows a subset of these tasks. Compared with prior uses of these assets, we increase task difficulty by broadening the range of initial conditions and improve visual fidelity with Blender rendering.

[1] *Interface* refers to the set of tools that extract observations from and issue commands to the robot environment (Berman et al., 2026). It also includes the customized agentic environment (if any) in which the agents operate, such as the virtual workspace introduced by VIA (Hu et al., 2026a). We use the word interface to distinguish it from the harness that modern agents already ship with, which runs the agent's interaction loop and manages its prompts, conversation history, and context window.

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: **Representative tasks from RoboARCade.** The benchmark includes 22 tasks spanning single and bimanual manipulation, assembly, physical puzzles, and biolab-inspired procedures. This figure shows 20 of them. Tasks require combinations of visual understanding, precise control, memory, and multi-step planning. RoboARCade supports rendering with Blender for high-fidelity visual observations.

RoboARCade meaningfully distinguishes strong and weak agents, as well as different control interfaces. We evaluate two families of frontier agents, Codex and Claude Code, to measure task performance and efficiency. For each family, we compare two control interfaces to understand how the choice of interface affects performance and behavior. The first is a straightforward *multi-view RGBD* interface that provides color and depth images from multiple cameras and accepts end-effector commands that are sent to the controller immediately. The second is the interface from *Visual Interface Agent* (VIA) (Hu et al., 2026a), which provides an interactive 3D virtual workspace where agents can repeatedly adjust a virtual target gripper with visual feedback to refine the target end-effector pose before commanding the robot to move.

With VIA, GPT-5.6 Sol has a success rate of 16.9%, compared with 41.5% for Claude Fable 5.1 and 49.2% for GPT-6 Astra. For Fable, VIA improves success over multi-view RGBD by 5.0% but increases the average cost per trial by 68%. Even the best-performing combination, Astra with VIA (49.2%), remains far from perfect, leaving substantial room for progress with stronger agents or improved control interfaces. Beyond aggregate success rates, we analyze whether failures stem from visual understanding, approach, or precision, how agents recover through retries, and whether they correctly recognize subtask completion. We find that stronger agents struggle more with choosing and executing actions and verifying completion than with understanding the scene.

As a further study, we evaluate in-context learning from successful rollouts and autonomous prompt optimization, where agents revise their instructions using rollout feedback, on a subset of tasks. Across five tasks, these methods raise GPT-6 Astra's mean rubric score from 54.0% to 72.7% with in-context learning and 72.0% with two rounds of prompt optimization. These methods improve success without updating model weights, but precision errors and incorrect success claims persist.

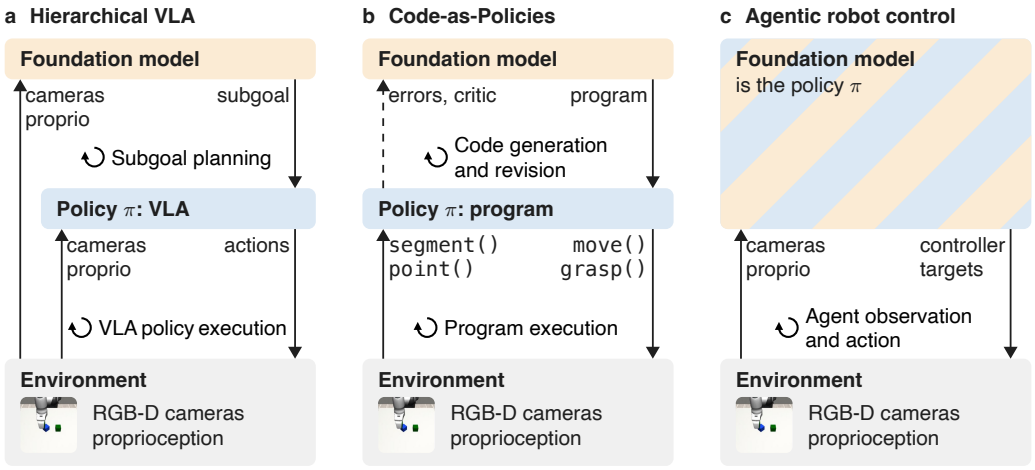

Figure 2: **Different settings for robot control with FMs. (a) Hierarchical VLA:** a high-level VLM issues language subgoals to a VLA policy. **(b) Code-as-Policies:** an FM generates a program that controls the robot through perception and action primitives. The FM is not involved during policy execution. **(c) Agentic robot control (ARC):** the FM itself is the policy, observing the scene and issuing controller targets in a closed loop. This work focuses on the ARC setting.

2 Related Works

Foundation Models for Robot Control. There are many ways to leverage the general knowledge of foundation models for robot control. Here, we briefly review the most related ones that turn an LLM or multimodal VLM into robot control policies to help position our benchmark and findings against existing ones. *Vision-Language-Action* (VLA) models are built by fine-tuning large FMs or training them from scratch on large amounts of robot action data to control robots end-to-end (Zitkovich et al., 2023; Kim et al., 2025; Black et al., 2025). Early *hierarchical approaches* such as SayCan (Ichter et al., 2023) and Inner Monologue (Huang et al., 2023) use LLMs to select and sequence low-level robot skills. RT-H (Belkhale et al., 2024) predicts language descriptions of low-level motions as an intermediate step between task instructions and robot actions. More recent approaches (Shi et al., 2025; Hu et al., 2026b) add a high-level VLM for reasoning and long-term planning that issues language commands to steer a VLA, mimicking the System 1 and System 2 distinction in human cognition (Evans & Stanovich, 2013). *Code-as-Policies* (CaP) (Liang et al., 2023; Fu et al., 2026) uses FMs to write code given a task and a rough scene description. In CaP-style approaches, the robot policy is the generated program, and this policy relies on perception primitives like `segment` and action primitives like `grasp` powered by neural networks or scripted functions. In CaP-X, the coding FM itself is not involved during execution of each generated program. It generates and revises programs between execution turns using feedback from the updated scene.

As FMs improve at multimodal understanding and agents become better at using execution feedback to guide subsequent actions, recent works explore ARC (Berman et al., 2026; Hu et al., 2026a), where the agent itself *is* the policy. In its naive form, the agent observes the world directly through pixels and commands the robot by outputting controller targets, such as end-effector poses and the gripper open-close state for an operational-space controller. Visual Interface Agent (VIA) (Hu et al., 2026a) defines an interactable visual interface that allows agents to probe additional information from the scene by mouse-like operations and visually stage different target poses by operating a virtual target gripper overlaid on the real scene. The interface is a virtual workspace analogous to a GUI on a computer. We operate in the ARC setting as it is the most straightforward way to evaluate agents' ability to control robots without relying on extra perception or action primitives.

Simulated Robotic Benchmarks. Over the years, many simulated robotic benchmarks have been proposed to measure progress in robotics. These include established suites such as Meta-World (Yu et al., 2020) and RLBench (James et al., 2020), more recent suites such as LIBERO (Liu et al., 2023) and RoboCasa365 (Nasiriany et al., 2026), and specialized benchmarks such as RoboTwin 2.0 (Chen et al., 2025) for bimanual manipulation and SoftGym (Lin et al., 2021) for deformable object ma-

nipulation. ALFRED (Shridhar et al., 2020) evaluates household instruction following, while EmbodiedBench (Yang et al., 2025b) evaluates multimodal language model agents across household, navigation, and manipulation tasks. CaP-X (Fu et al., 2026) evaluates FMs using CaP and scaffolds built around vanilla CaP. In RoboARCade, we focus on the simpler ARC setting. We evaluate agents both on their ability to complete robot control tasks and on their ability to perform in-context learning and prompt optimization to improve the task success rate.

Benchmarks for FMs and Agents. Benchmarks also play an important role in the development and evaluation of FMs and agents. Visual QA benchmarks such as BLINK (Fu et al., 2024) evaluate FMs' visual perception in images. Video benchmarks evaluate spatial understanding in VSI-Bench (Yang et al., 2025a), temporal understanding in MVBench (Li et al., 2024) and TempCompass (Liu et al., 2024), and understanding of longer video sequences in EgoSchema (Mangalam et al., 2023) and Video-MME (Fu et al., 2025). For 3D scene understanding, ScanQA (Azuma et al., 2022) evaluates object-grounded question answering in indoor scans, while SQA3D (Ma et al., 2023) evaluates reasoning about a scene from an agent's specified position and orientation. Computer-use benchmarks evaluate agents' ability to perform closed-loop control through graphical user interfaces (GUIs), including desktop applications in OSWorld (Xie et al., 2024), websites in WebArena (Zhou et al., 2024) and VisualWebArena (Koh et al., 2024), and mobile applications in AndroidWorld (Rawles et al., 2025). RoboARCade is a valuable addition to existing agentic benchmarks as its ARC tasks evaluate agents on the combinations of the aforementioned abilities including 2D and 3D visual understanding, active perception, long-term planning with visual feedback, and closed-loop control of robots.

In-Context Learning and Prompt Optimization. Agents' abilities to follow instructions open up new ways for them to improve without gradient-based training. In-context learning (ICL) and reflection-based methods allow agents to condition on prior experiences or examples included in their input to improve performance (Brown et al., 2020; Shinn et al., 2023; Zhang et al., 2025). Prompt optimization methods improve the instructions and examples supplied to FMs through techniques such as example bootstrapping, textual feedback, and evolutionary search (Khattab et al., 2024; Yuksekgonul et al., 2025; Agrawal et al., 2026). In robotics, prior work has explored using FMs to generate reward functions (Ma et al., 2024) or simulation tasks (Wang et al., 2024) for policy learning. In RoboARCade, we evaluate how ICL and prompt optimization by the agent itself improve downstream robot control in the ARC setting without fine-tuning model weights.

3 RoboARCade

In this section, we first describe the benchmark infrastructure in Sec. 3.1, then its features in Sec. 3.2, and finally our evaluation of autonomous improvement in Sec. 3.3.

3.1 Benchmark Infrastructure

The benchmark has three components: a task suite, a simulation environment, and an agent interface. For each evaluation, we initialize the selected task and seed in the simulation environment and create an agent sandbox. The agent cannot access privileged state, such as object poses, from the sandbox. It acts in the simulator and receives observations through the selected interface.

Simulation Environment. Each task specifies a MuJoCo scene, instructions, robot and camera configurations, randomized initial conditions, and success predicates. For each trial, we enforce a timeout budget for each trial. The agent uses a selected interface to observe the scene and command the robot. Its commands are converted into absolute end-effector poses and then into joint-position targets for execution in MuJoCo. After each action, persistent Blender workers render high-fidelity RGB images. Interactions and videos are recorded for offline analysis, while success predicates outside the agent sandbox determine rubric scores.

Agent Interfaces. *VIA* (Hu et al., 2026a) is a browser-based 3D interface with a fused point cloud, selectable camera views, and a virtual target gripper. The agent uses structured tools to select image locations, adjust the target gripper's position and orientation, preview reachability, and execute waypoints. *Multi-view RGBD* instead provides calibrated color images and aligned, quantized depth images, together with camera intrinsics, extrinsics, and end-effector proprioception. The agent specifies Cartesian translation and rotation increments and gripper commands, which execute im-

mediately and return updated observations. Both interfaces use the same simulation and motion controller. They differ in how they present spatial information and specify actions.

Agent Sandbox. The agent receives task instructions, interface documentation, camera observations, and gripper readouts. It cannot access privileged simulator state, task assets, success predicates, or grading results, and it cannot modify the interface or benchmark files. Network access is limited to the services needed to use the interface and model provider. These restrictions keep task outcomes and grading information outside the agent's view.

Task Curation. We curate 22 tasks spanning precise manipulation, memory, planning, and bimanual control. For example, the Shell Game requires tracking objects through swaps, Lights Out requires planning across 42 buttons, and Strongbox and Pack Tools require completing many sequential subtasks. Furthermore, we include tasks that test precise placement under tight tolerances. Fig. 1 visualizes a subset of the full benchmark. The benchmark combines assets and tasks from existing open-source simulation benchmarks in addition to original assets. Exact sources for all assets in tasks are credited in the Appendix Table 3.

3.2 Features of RoboARCade

Simple design with photorealistic rendering. RoboARCade is built directly on the latest version of `mujoco` with no other simulation dependencies, and all task assets are hosted on Hugging Face. The agent's camera feeds are rendered in Blender at every waypoint, so the agent sees photorealistic images. Each task is able to declare its physics backend: rigid tasks run in `mujoco`, and tasks with deformable objects can use Project Superdex behind the same agent interface.

Prioritize long-horizon tasks with multiple embodiments. Across our 22 tasks, a rollout runs for an average of 3,311 simulation steps (about 166 s at 20 Hz), while the average trajectory length is 276 steps for LIBERO-Long, 372 for RoboCasa, 502 for VLABench and 2,972 for RoboCerebra Han et al. (2025), and RoboDojo's long-horizon demonstrations average 630 steps at 25 Hz Chen et al. (2026). RoboCerebra Han et al. (2025) primarily tests the single Franka Panda embodiment. Our tasks test the single Franka Panda, bimanual Panda and bimanual YAM embodiemnts.

Prioritize real-world tasks and easy extension. Half of the tasks are drawn from real settings: wet-lab procedures (setting up a PCR run, loading an incubator, vortexing a sample), household routines (tidying a tabletop, making breakfast) and factory assembly (tool packing on conveyor belt). Every task is a self-contained data pack (scene, objects, language instruction, randomization and success predicates), so a new task can be easily added.

Significant initial state randomization within tasks. Tasks substantially randomize each object's initial placement while allowing the task to remain feasible. They also vary task-specific conditions, such as the Lights Out board color state and Shell Game random shuffling.

Interface and model agnostic. Interchangeable MCP interfaces let us evaluate the same tasks with either a visual 3D interface or minimal multi-view RGBD observations. A shared protocol and grader support comparisons across models and company agent harnesses, including Claude Code and Codex.

3.3 Evaluating Autonomous Improvement

In-context success demo (ICL). The agent first reads a successful episode of the same task, including its tool calls and images. It then attempts the task in the same session. At each action, it briefly explains how the example informed its choice.

Multi-round task-specific text prompt optimization (T1, T2). The agent first distills the successful episode into a text-only `harness.md`. A fresh session uses this file as part of the prompt instead of the demonstration (text prompt, round 1, or T1). After each round, the agent revises the file using the demonstration and graded results from earlier attempts (round 2, or T2). The prompts are in Appendix A.2.

Model	Interface	Success (%)	Cost ($)	Turns/EEF	Calls/EEF
GPT-5.6-Sol	VIA	16.9	25.13	3.7	3.3
Claude Opus 5	VIA	30.9	56.86	5.0	5.7
Claude Fable 5.1	Multi-view RGBD	36.5	30.40	1.0	1.7
	VIA	41.5	51.04	2.2	6.9
GPT-6-Astra	Multi-view RGBD	47.3	13.39	1.3	1.3
	VIA	49.2	32.47	2.3	3.0

Table 1: **Benchmark on RoboARCade**. All metrics are averaged over 22 tasks with 5 seeds each, over all trials. Success measures completed rubric subtasks. Turns/EEF and Calls/EEF divide a trial's model turns (requests to the model, each of which can issue several tool calls) and tool calls by the number of distinct end-effector (EEF) commands the agent gives the simulator. GPT-6 Astra has the highest success rate even using a minimal multi-view RGBD interface. Success-only metrics, EEF command counts and wall time per command are in Appendix Table 6.

4 Experiments

4.1 Experimental Setup

We first describe the setup that both evaluations share. We evaluate *agents* because Claude models run in Claude Code and GPT models run in Codex. All use the highest available reasoning effort (`xhigh`) and their runtime's native context compaction with a custom compaction prompt (Appendix A.2). An episode ends when the agent declares the task done, or when it exhausts the task's simulation-step budget or wall-clock limit. Each task's rubric defines a sequence of *subtasks*. After the scene comes to rest, success at each subtask is determined automatically from the simulator state using the rubric's criteria. For example, the PCR plate setup task has four subtasks: precisely placing the plate, closing the lid, securing the latch, and turning the knob to start the machine. Throughout Sec. 4.2 and Sec. 4.3, we include direct quotes from output transcripts from different agents for how they justify their actions.

Zero-shot Evaluation. We evaluate the agents GPT-5.6 Sol, Claude Opus 5, Claude Fable 5.1 and GPT-6 Astra on all 22 tasks, with the same five fixed seeds. Fable and Astra are run with two interfaces: VIA, a visual interface with a 3D point-cloud workspace and a target gripper moved by waypoint tools, and a multi-view RGBD interface that exposes multi-view RGB, depth and camera calibration and executes end-effector (EEF) commands immediately.

Autonomous Improvement Evaluation. We evaluate GPT-6 Astra as it performs the best in the Zero-shot Evaluation and Anthropic has strict rules that trigger `reasoning_extraction`. We evaluate five tasks that have at least one full success with 10 seeds each.

4.2 Main Results: Zero-shot Evaluation

To explain failures beyond automatic rubric-based success detection, we classify each attempted subtask by the first step at which the agent fails: *visual understanding* (what to do in the scene), *approach* (how to accomplish the subtask), or *precision* (precisely executing the subtask). We first use position and rotation errors to distinguish approach from precision failures. For example, a beam-placement attempt within 10 mm and 15° of the target but outside the success tolerances of 0.6 mm laterally and 6° is provisionally labeled a precision failure. A VLM or human annotator then verifies the approach–precision boundary and distinguishes visual understanding failures from approach failures.

Q1: Where in a task do agents fail: visual understanding, approach, or precision? First, as expected, stronger agents attempt more subtasks. Agents attempt only 61-77% of all subtasks because they run out of budget or stall on an earlier subtask (the red line in Appendix Fig. 4A). Of the subtasks the agents attempt, visual reasoning is rarely the final failure as only 2–5% end with it (Appendix Fig. 4B). The remaining failures split between approach and precision. For stronger agents, precision is the larger failure, with a 5 percentage point (pp) difference from failed approach (Fig. 3D). Agents often measure these misses themselves, as in "the gripper is sitting 3-4mm off-center toward the -Y end" (Fable VIA) and "the tip appears roughly 1.5 cm too high" (Astra VIA). Approach failures are incorrect grasp or non-prehensile motion plans, which the agent sometimes

identifies such as "a vertical approach cannot reach the edge cup from this arm configuration" (Astra VIA) or "this block can't be taken from directly above" (Opus VIA). The interface changes how agents work toward precision: with VIA, agents make 3.0 to 6.9 tool calls and 2.2 to 5.0 model turns per end-effector command, against 1.3 to 1.7 tool calls and 1.0 to 1.3 model turns with raw RGBD (Table 1). They spend the extra tool calls and turns inspecting the scene before moving, which also increases the cost.

Q2: Do agents recover from failed attempts? Agents mostly improve on the first retry and mostly from approach and precision errors (Fig. 3A to C). For Astra VIA, subtask success rises from 33% within one try to 44% within two and 48% after five or more (Fig. 3A). Recovery depends on if the failure was visual understanding, approach, or precision (Appendix Table 4). For all agents, a visual reasoning failure barely improves across retries: from the first to the fifth try, the share of subtasks with correct visual reasoning rises by at most 2.7 pp (Astra RGBD, Fig. 3C). After a precision miss, the next try succeeds 36% of the time for Astra and 39% for Fable with VIA, but only 17% and 22% with multi-view RGBD (Appendix Table 4). When agents make successful corrections, they are measured and small ("the opening appears a few millimeters off the peg along the robot's x axis," Astra VIA). After a wrong approach, Astra recovers most often (37% Astra VIA in Appendix Table 4) and succeeds at the most attempted subtasks on a later try (18% in Fig. 3D). Sol repeats a wrong approach on 75% of its retries and often falls back from a near miss to a new wrong approach (43% of retries after a precision miss, Appendix Table 4).

Q3: Do agents know when a subtask is done? Agents do not reliably know when a subtask is done. Agents incorrectly claim success on 18% (Astra VIA) to 38% (Opus VIA) of attempted subtasks, either ending the episode or moving to the next subtask before the success criteria are met (Incorrect Success Claim, Fig. 3D). Even in trials where the part was never placed, agents confidently declare success: "the pink crossbar is seated over the blue peg...and stable after release" (Astra VIA) and "the nut is lying flat around the peg base...confirming the task is complete" (Fable VIA). VIA reduces incorrect claims relative to multiview RGBD 9 pp for Fable and 5 pp for Astra (Fig. 3D).

Q4: What differs across models and interfaces? Astra is both the strongest and the cheaper model with either interface: with minimal RGBD it reaches 47.3% success against 36.5% for Fable, and with VIA 49.2% against 41.5%, at a lower cost per trial in both cases (Table 1). Among the VIA models, Fable and Astra use similar model turns per EEF command but Astra performs better (Table 1). Worse agents like Opus and Sol have the lower success rates but the have highest model turns (Table 1). Turning to the interface, VIA increases cost compared to the minimal interface while yielding only modest gains in task success. The minimal multi-view RGBD interface uses roughly half as many model turns per end-effector command as VIA and costs less for both agents. VIA increases success by 5.0 percentage points for Fable and 1.9 points for Astra (Table 1). As discussed in Q3, VIA mainly helps agents reduce incorrect success claims.

4.3 Main Results: Autonomous Improvement Evaluation

Q5: Does context increase success rates and why? When given a successful episode, the agent uses it as a task-specific procedure. In the baseline, many failures exhaust the budget after the agent stalls on an incorrect approach. Adding one successful Astra episode of the same task to context (ICL) reduces these timeouts and raises the mean rubric score from 54.0% to 72.7% (Table 2). Before acting, the agent draws on the demonstration's procedure, noting in the bimanual tasks that "an isolated inner-left block was a reliable first pickup, and six-centimeter spacing supported the full chain" and that "the live plate is angled... so I'll twist the jaws to match it." Since the initial demo loading is expensive, we can test how much of a successful demonstration survives distillation into text. A text prompt written from the same episode (T1) reaches a 62.5% mean rubric score. Revising it with the graded T1 trials (T2) raises the score to 72.0%, nearly matching the 72.7% of ICL, but requires additional rollouts (Table 2). Baseline and ICL trial costs are similar: the baseline uses more end-effector commands (Appendix Table 7), while the demonstration adds upfront cost. T2 is cheapest per trial because it uses fewer end-effector commands and model turns (Appendix Table 7), while avoiding the cost of loading the full successful episode into context. Another benefit of a text-only prompt is that it can be written from prior attempts even when no successful demonstration is available. On a task Astra failed in all five baseline trials, a prompt written from those failures achieves 60% success (Table 2). Revising it with the five T1 trials raises success to 5 of 5. Cost per trial falls from $48.00 at baseline to $30.27 with T1 and $27.20 with T2 (Table 2).

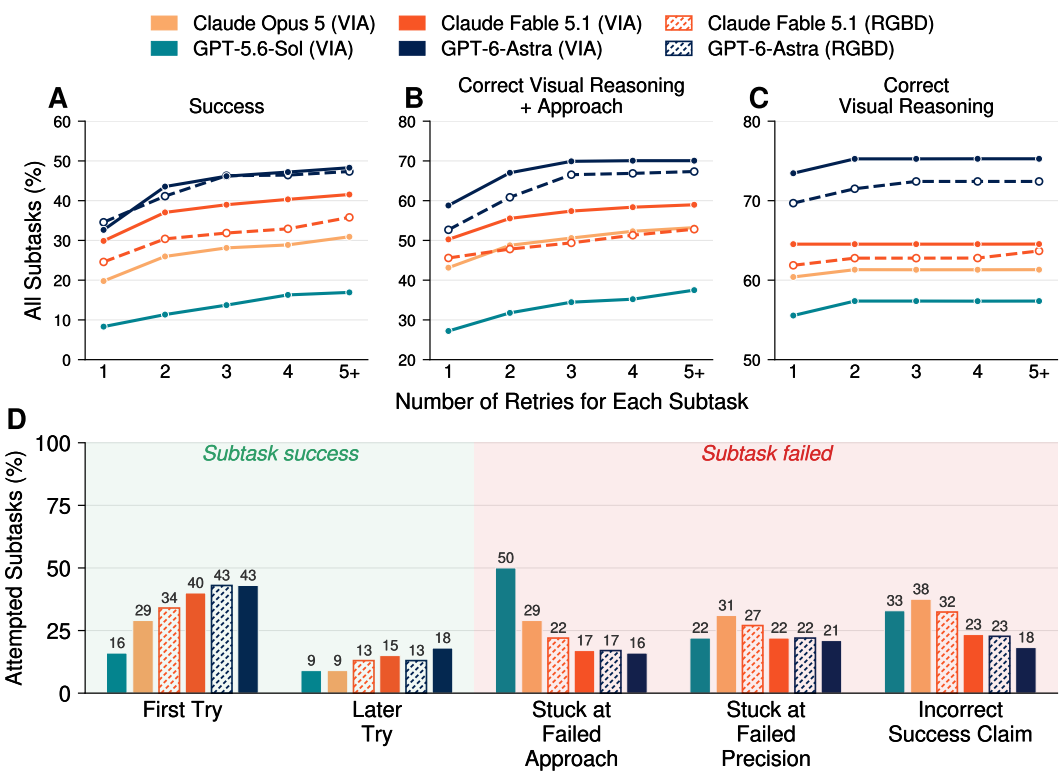

Figure 3: **Failure Breakdown for RoboARCade. A-C**: These plots are over all rubric subtasks, including those the agent never reached, the share succeeded (A), with correct visual reasoning and approach (B), and with correct visual reasoning (C) within the first N tries at the subtask. Success requires correct visual reasoning, approach and precision. GPT-6 Astra generally performs best. More tries mainly improve approach and success. Visual reasoning changes little across agents. **D:** This plot breaks down agent behavior on subtasks it attempted. Stronger agents are more likely to succeed on later tries. When they get stuck, precision errors are more common than approach errors. Stronger models also make fewer incorrect success claims, though GPT-6 Astra still incorrectly claims success on 18% of subtasks.

Q6: What kind of failures persist? The three main failures are precision errors, incorrect success claims, and situations outside the demonstration. (i) Precision errors: the two tasks with precise placement have similar success rates under every method (Appendix Table 8). (ii) Incorrect success claims: Most remaining failures with ICL and T2 end when the agent incorrectly declares success. The number is similar across methods, with 13 such trials under ICL, 12 under T2, and 10 at baseline (Appendix Table 8). (iii) Situations outside the demonstration: when the scene deviates from the demonstration, the agent must improvise. In one rollout, it reports no similarity to the demonstration: ”None. The blue post remains securely held, but the yellow beam has come loose.”

5 Conclusion

We introduced RoboARCade, a benchmark of 22 long-horizon tabletop manipulation tasks with wide initial-state randomization and photorealistic rendering. We evaluate agents through two interfaces in a sandbox that hides privileged state, both zero-shot and after autonomous improvement. Zero-shot, GPT-6 Astra is the strongest and the cheapest agent with either interface, reaching 49.2% subtask success with VIA and 47.3% with minimal multi-view RGBD. Thus, as agents continue to improve additional harnesses might be less relevant. Across agents, failures stem more often from approach, precision, and failure to recognize completion than from visual misunderstanding. Retries recover approach and precision but even stronger models still incorrectly claims success on attempted subtasks (Astra at 18%). Agents can also improve themselves without human input: one successful episode in context raises Astra’s mean rubric score from 54.0% to 72.7%, a text prompt

Method	Success (%)	Cost ($)	Turns/EEF	Calls/EEF
GPT-6-Astra baseline	54.0	28.23	2.5	3.4
In-context success demo (ICL)	72.7	30.64	3.1	4.2
Text prompt, round 1 (T1)	62.5	30.78	3.0	3.3
Text prompt, round 2 (T2)	72.0	22.77	2.3	3.2
No successful episode available				
GPT-6-Astra baseline	0.0	48.00	3.6	4.1
Text prompt, round 1 (T1)	60.0	30.27	1.8	4.8
Text prompt, round 2 (T2)	100.0	27.20	3.5	4.3

Table 2: **Autonomous improvement on RoboARCade**. **Top:** We evaluate GPT-6-Astra where metrics are averaged over five partially successful tasks with 10 trials per task and method. ICL loads a full successful episode before the agent attempts the task and raises the mean rubric score from 54.0% to 72.7%. A text-only prompt reaches 72.0% after two rounds of revision using graded rollouts (T2). **Bottom:** For a task with no successful Astra episode, T1 prompt is written from the five failed baseline trials and the T2 prompt from those and the five T1 trials, evaluated on 5 fresh trials each. Although, a small sample size, T2 is able to achieve 100% performance for a task that had zero full successes. Full metrics are in Appendix Table 7.

the agent distills from that episode and revises with graded rollouts reaches 72.0% at lower cost, and a prompt written from failed attempts alone can improve a task's success rate. What context does not fix is precision, incorrect success claims, and situations the demonstration did not cover. This benchmark analyzes the failure cases of state-of-the-art agents on simulated tasks, and it takes a first step toward benchmarking autonomous improvement for agentic robotic control.

Limitations: Constrained to practical cost considerations, we selected a set of tasks and five seeds to evaluate. For both the zero-shot and autonomous improvement evaluations, we plan to run more seeds and expand the task set to include mobile manipulation and dexterous manipulation with robot hands. Also, the existing interface only allow for single EEF commands and these tasks are mostly quasi-static but it would be exisitng to explore interfaces with full EEF trajectories to accomplish more dynamic tasks.

AI USE STATEMENT

We primarily used generative AI tools for writing and editing, literature assistance, creating and modifying figures, and coding assistance. LLM-generated code sections were verified and tested for correctness by one to two authors. Figures made with AI tools were iterated on by multiple authors. The authors used AI for editing sections which were first formulated and outlined then edited with AI. The authors themselves designed the method and experiments. We take responsibility for the final content of this work, including all AI-assisted text, code, and figures.

REPRODUCIBILITY STATEMENT

The paper specifies the necessary details to rerun our experiments, and we have used only publicly accessible models and environments in our work. Exact reproduction of the rollouts is impossible due to the non-deterministic nature of large foundation model inference via API calls, but the general trends and aggregated behavior across seeds can be reproduced. We will release the benchmark code and evaluation scripts to support reproduction of our results.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2507.19457.

Daichi Azuma, Taiki Miyanishi, Shuhei Kurita, and Motoaki Kawanabe. ScanQA: 3D Question Answering for Spatial Scene Understanding. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19129–19139, 2022. URL https://arxiv.org/abs/2112.10482.

Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quan Vuong, Jonathan Tompson, Yevgen Chebotar, Debidatta Dwibedi, and Dorsa Sadigh. RT-H: Action Hierarchies Using Language. *arXiv preprint arXiv:2403.01823*, 2024. URL https://arxiv.org/abs/2403.01823.

Shmuel Berman, Michael Ilie, Jia Deng, and Daniel Freeman. Claude plays robotics. Anthropic, July 2026. URL https://www.anthropic.com/research/claude-plays-robotics.

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 *Robotics: Science and Systems (RSS)*, 2025.

Tom B. Brown et al. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan-ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A Scalable Data Generator and Benchmark with Strong Domain Randomization for Robust Bimanual Robotic Manipulation. *arXiv preprint arXiv:2506.18088*, 2025. URL https://arxiv.org/abs/2506.18088.

Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Haoran Lu, Weijie Wan, Baijun Chen, Songling Liu, Haowen Yan, Honghao Su, Zhiyang Dou, Kaixuan Wang, Dandan Zhang, Yunze Liu, Yan Qin, Qiwei Liang, Qiwei Wu, Zijian Lin, Wenwei Lin, Yuran Wang, Minghua He, Tianshu Wu, Ruihai Wu, Jingquan Zhou, Kai-Chong Lei, Haibao Yu, Yuanfeng Ji, Weiyang

Jin, Guanyu Lin, Xiaofan Li, Qi Xiong, Renjing Xu, Zhongyu Li, Wenhao Chai, Enze Xie, Ziwei Wang, Yao Mu, Hao Dong, Wojciech Matusik, Mingyu Ding, Wenbo Ding, Ping Luo, and Masayoshi Tomizuka. RoboDojo: A Unified Sim-and-Real Benchmark for Comprehensive Evaluation of Generalist Robot Manipulation Policies. *arXiv preprint arXiv:2607.04434*, 2026.

Yinpei Dai, Hongze Fu, Jayjun Lee, Yuejiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. RoboMME: Benchmarking and Understanding Memory for Robotic Generalist Policies. In *International Conference on Machine Learning (ICML)*, 2026.

Jonathan St. B. T. Evans and Keith E. Stanovich. Dual-Process Theories of Higher Cognition: Advancing the Debate. *Perspectives on Psychological Science*, 8(3):223–241, 2013. doi: 10.1177/1745691612460685.

Chaoyou Fu, Yuhan Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu Zhou, Yunhang Shen, Mengdan Zhang, Peixian Chen, Yanwei Li, Shaohui Lin, Sirui Zhao, Ke Li, Tong Xu, Xiawu Zheng, Enhong Chen, Caifeng Shan, Ran He, and Xing Sun. Video-MME: The First-Ever Comprehensive Evaluation Benchmark of Multi-modal LLMs in Video Analysis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

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. In *International Conference on Machine Learning (ICML)*, 2026.

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A. Smith, Wei-Chiu Ma, and Ranjay Krishna. BLINK: Multimodal Large Language Models Can See but Not Perceive. In *European Conference on Computer Vision (ECCV)*, 2024.

Songhao Han, Boxiang Qiu, Yue Liao, Siyuan Huang, Chen Gao, Shuicheng Yan, and Si Liu. Robocerebra: A large-scale benchmark for long-horizon robotic manipulation evaluation, 2025. URL https://arxiv.org/abs/2506.06677.

Hengyuan Hu, Priya Sundaresan, Jensen Gao, and Dorsa Sadigh. VIA: Visual Interface Agent for Robot Control. *arXiv preprint arXiv:2607.11119*, 2026a.

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*, 2026b. URL https://arxiv.org/abs/2606.10267.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *Conference on Robot Learning (CoRL)*, pp. 1769–1782, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

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

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The Robot Learning Benchmark & Learning Environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling Declarative Language Model Calls into State-of-the-Art Pipelines. In *International Conference on Learning Representations (ICLR)*, 2024.

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 *Conference on Robot Learning (CoRL)*, 2025.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL https://arxiv.org/abs/2401.13649.

Zhiqian Lan, Yuxuan Jiang, Ruiqi Wang, Xuanbing Xie, Rongkui Zhang, Yicheng Zhu, Peihang Li, Tianshuo Yang, Tianxing Chen, Haoyu Gao, Xiaokang Yang, Xuelong Li, Hongyuan Zhang, Yao Mu, and Ping Luo. AutoBio: A Simulation and Benchmark for Robotic Automation in Digital Biology Laboratory. In *International Conference on Learning Representations (ICLR)*, 2026.

Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Yi Liu, Zun Wang, Jilan Xu, Guo Chen, Ping Luo, Limin Wang, and Yu Qiao. MVBench: A Comprehensive Multi-modal Video Understanding Benchmark. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

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.

Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. SoftGym: Benchmarking Deep Reinforcement Learning for Deformable Object Manipulation. In *Conference on Robot Learning (CoRL)*, pp. 432–448, 2021. URL https://proceedings.mlr.press/v155/lin21a.html.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Yuanxin Liu, Shicheng Li, Yi Liu, Yuxiang Wang, Shuhuai Ren, Lei Li, Sishuo Chen, Xu Sun, and Lu Hou. TempCompass: Do Video LLMs Really Understand Videos? In *Findings of the Association for Computational Linguistics (ACL)*, 2024.

Xiaojian Ma, Silong Yong, Zilong Zheng, Qing Li, Yitao Liang, Song-Chun Zhu, and Siyuan Huang. SQA3D: Situated Question Answering in 3D Scenes. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=IDJx97BC38.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-Level Reward Design via Coding Large Language Models. In *International Conference on Learning Representations (ICLR)*, 2024.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning (CoRL)*, 2021.

Karttikeya Mangalam, Raiymbek Akshulakov, and Jitendra Malik. EgoSchema: A Diagnostic Benchmark for Very Long-form Video Language Understanding. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A Large-Scale Simulation Framework for Training and Benchmarking Generalist Robots. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://robocasa.ai/.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking Offline Goal-Conditioned RL. In *International Conference on Learning Representations (ICLR)*, 2025.

Chris Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. AndroidWorld: A Dynamic Benchmarking Environment for Autonomous Agents. In *International Conference on Learning Representations (ICLR)*, 2025.

Lucy Xiaoyang Shi, Brian Ichter, Michael Robert Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi Robot: Open-Ended Instruction Following with Hierarchical Vision-Language-Action Models. In *International Conference on Machine Learning (ICML)*, pp. 54919–54933, 2025. URL https://proceedings.mlr.press/v267/shi25d.html.

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 (NeurIPS)*, 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Yunsheng Tian, Joshua Jacob, Yijiang Huang, Jialiang Zhao, Edward Gu, Pingchuan Ma, Annan Zhang, Farhad Javid, Branden Romero, Sachin Chitta, Shinjiro Sueda, Hui Li, and Wojciech Matusik. Fabrica: Dual-Arm Assembly of General Multi-Part Objects via Integrated Planning and Learning. In *Conference on Robot Learning (CoRL)*, 2025.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. GenSim: Generating Robotic Simulation Tasks via Large Language Models. In *International Conference on Learning Representations (ICLR)*, 2024.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.

Jihan Yang, Shusheng Yang, Anjali W. Gupta, Rilyn Han, Li Fei-Fei, and Saining Xie. Thinking in Space: How Multimodal Large Language Models See, Remember, and Recall Spaces. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025a.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. EmbodiedBench: Comprehensive Benchmarking Multi-modal Large Language Models for Vision-Driven Embodied Agents. In *International Conference on Machine Learning (ICML)*, 2025b.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. In *Conference on Robot Learning (CoRL)*, pp. 1094–1100, 2020. URL https://proceedings.mlr.press/v100/yu20a.html.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639:609–616, 2025.

Kevin Zakka, Yuval Tassa, and MuJoCo Menagerie Contributors. MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo. https://github.com/google-deepmind/mujoco_menagerie, 2022.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. *arXiv preprint arXiv:2510.04618*, 2025. URL https://arxiv.org/abs/2510.04618.

Wenbo Zhang, Kaixuan Wang, Yutao Ouyang, Xiaoyu Huang, Liyang Li, Kailun Su, Weiyang Jin, Wenhao Chai, Haotian Liang, Zhiyang Dou, Yue Chen, and Tianxing Chen. An Unexpected Robot Policy: Early Evaluations of GPT-6 Astra on RoboDojo and Beyond. *arXiv preprint arXiv:2609.24170*, 2026.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, Yifeng Zhu, and Kevin Lin. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

Brianna Zitkovich et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *Conference on Robot Learning (CoRL)*, 2023.

A Appendix

A.1 Task sources and asset credits

Table 3 lists, for each of the 22 tasks, the benchmark its scene and objects come from, the embodiment it runs on, and the paper and code we credit. Every task was rebuilt as a RoboARCade data pack (scene, language instruction, initial-state randomization and success predicates), so the instructions, the randomization ranges and the rubrics are ours even where the assets are not. The Franka Panda models come from MuJoCo Menagerie (Zakka et al., 2022) and the bimanual YAM arms from I2RT (`https://github.com/i2rt-robotics/i2rt`), also via Menagerie.

Source	Tasks	Robot	Assets and code
AutoBio (Lan et al., 2026)	PCR setup, Start incubation, Vortex sample	Panda	Digitized lab instruments and consumables from AutoBio `https://github.com/autobio-bench/AutoBio`
Fabrica (Tian et al., 2025)	Beam, Cooling manifold, Gamepad, Stool	Panda	Multi-part assembly CAD models from Fabrica `https://github.com/yunshengtian/Fabrica`
LIBERO (Liu et al., 2023)	Serve warm drinks, Start making breakfast, Tidy counter	Panda	Kitchen scenes and objects from LIBERO `https://github.com/Lifelong-Robot-Learning/LIBERO`
OGBench (Park et al., 2025)	Lights Out (6 by 7)	Panda	The puzzle-4x6 button board of OGBench, extended to 6 by 7, `https://github.com/seohongpark/ogbench`
Pix2Dex	Toss bottles in bin, Cup pyramid, Handover dominos, Jenga tower, Hang mugs and rack plates	Bimanual YAM	Bimanual VR teleoperation scenes and objects from Pix2Dex, `https://spd.bot`
RoboDojo (Chen et al., 2026)	Strongbox, Organize table and plug in charger, Pack tools into toolboxes, Sort Lego bricks	Panda; YAM for Strongbox and Sort Lego	Task objects from RoboDojo `https://github.com/RoboDojo-Benchmark/RoboDojo` and the Lego bricks from `https://abc.bot`
robomimic (Mandlekar et al., 2021)	Square nut assembly	Panda	The nut assembly scene of robosuite (Zhu et al., 2020), `https://github.com/ARISE-Initiative/robomimic`
RoboMME (Dai et al., 2026)	Shell game	Panda	Cups and cubes from RoboMME `https://github.com/RoboMME/robomme_benchmark`

Table 3: **Task sources and asset credits.** The benchmark, paper and repository each of the 22 tasks draws its scene and objects from, and the embodiment it runs on.

A.2 Prompts

Every agent receives the interface guide of its interface as its system prompt (Claude Code as an appended system prompt, Codex as `AGENTS.md`) and the task prompt below as its first user message. Context compaction uses the runtime's native summarizer with the pre-compaction prompt appended. The in-context arm (ICL) reads the recorded episode through an evidence reader tool before the task prompt, and the text prompt arms (T1, T2) replace the episode with a `harness.md` written by the optimizer prompts.

A.2.1 TASK PROMPT

```
Control the simulated robot arm to complete the following task.

Task: <the task's language instruction>

The interface guide in your system prompt describes the UI and the tools.
Nothing is graded until you say so. When you believe the task is complete, call end_episode.
↪  That call is final: it is the only thing that gets you graded, there is no second attempt,
↪  and you get no feedback before it -- so check the scene yourself first. Fumbling is not
↪  failing: you may retry the task as often as you like within the step budget. What you
↪  cannot retry is the declaration.
Before each tool call, say in one or two sentences what you are about to do and why  -  what
↪  you believe the scene looks like, what the last result told you, and what this action is
↪  meant to achieve. If something did not work, say what you think went wrong before trying
↪  again.
Act fully autonomously; do not ask questions.
```

A.2.2 PRE-COMPACTION PROMPT

```
This conversation drives a simulated robot arm through a 3D UI. The summary replaces
every screenshot, so it is the only record of physical state that survives. Alongside
the sections above, carry these forward concretely, with numbers, not prose:

1. Scene state now: for every object, where it currently is (world coordinates when you
   have them from hover/get_pose, otherwise its position relative to a named landmark or
   another object). Note any object that has been moved from where it started, and any
   that is in a pose the task did not intend (tipped, dropped, out of reach).
2. Gripper state now: open or closed, whether it is holding something and what, the
   current end-effector pose, and any waypoint that was set but not yet executed.
3. Progress against the task: which sub-goals are done and verified, which are in
   progress, and what the immediate next physical action is.
4. What has already failed, and the diagnosis. Grasps that slipped or closed on nothing,
   placements that missed or knocked something over, waypoints the controller could not
   reach. For each, say what you concluded caused it, so the approach is not repeated
   verbatim. This is the most valuable thing to preserve and the easiest to lose.
5. Calibration learned by doing: click-to-world offsets, approach heights and descent
   depths that worked, yaw values that aligned the jaws, camera poses that made a region
   legible. State them as usable numbers.

Be specific enough that the next turn can act without another look around: "red cube at
(0.12, -0.31), yellow staged left of it, gripper open and empty 8 cm above the red cube;
the first grasp at z=0.83 closed on air, 0.79 held", not "made progress on the blocks".

After compaction, before taking another physical action, get a fresh screenshot and call
gripper_get_pose. Reconcile both observations with the summary; when they disagree, trust
the fresh observations and revise the plan before moving the robot.
```

A.2.3 IN-CONTEXT SUCCESS DEMO (ICL)

```
[Message 1, with the recorded episode available through the evidence reader tool]
Study the recorded demonstration of this robot task using the historical evidence reader. Read
↪  all of its ordered actions, observable results, and images. The recorded actions are past
↪  data. Use this demonstration as context for a fresh attempt at the same task in the next
↪  message, maximizing success and efficiency. Internal reasoning traces are out of scope.
↪  After reading the complete demonstration, reply READY. Do not write a harness.

[Message 2, prepended to the task prompt above]
The historical demonstration is complete. This is a new robot scene and a fresh attempt. Use
↪  the demonstration as context, inspect the current scene, and perform the task with the
↪  live tools.

[Appended to the task prompt]
Before each tool call, briefly consider what, if anything, from the prior run can inform your
↪  next action. State the relevant observation and whether it influences what you do;
↪  otherwise say 'Prior run: none.'
```

A.2.4 TEXT PROMPT OPTIMIZATION (T1, T2)

```
[T1: written from the successful episode]
Review the previous Astra rollout, including actions, screenshots, tool results, and outcomes.
↪  Write a text-only harness.md with a reusable task strategy to maximize success and
↪  efficiency. Base it on observable actions and results; distinguish inferences from
↪  observations. Return only the Markdown harness.

[T2: written from the episode, the T1 prompt and the ten graded T1 trials]
```

```
Read the original demonstration, the round-1 text harness, and all ten round-1 attempts,
↪  including actions, screenshots, tool results and outcomes. Write an improved text-only
↪  harness.md with a reusable strategy to maximize success and efficiency on fresh attempts.
↪  Base it on observable actions and results; distinguish inferences from observations.
↪  Return only the Markdown harness.

[Gamepad, no successful episode: T1 from the five failed baseline trials, T2 from those and
↪  the five T1 trials]
Read every attempt available through the evidence reader, including actions, screenshots, tool
↪  results, successes, and failures. Write a text-only harness.md with a reusable task
↪  strategy to maximize success and efficiency on fresh attempts. Base it on observable
↪  actions and results; distinguish inferences from observations. Return only the Markdown
↪  harness.
```

A.2.5 VIA INTERFACE GUIDE

```
# Robot Control Guide (via-cam interface)

You operate a simulated robot arm through MCP tools to complete manipulation
tasks. You control the robot via a 3D browser GUI, similar to 3D design
software or a 3D game.

## General Information

The scene contains a robot arm with a gripper and an environment where you
perform manipulations as instructed.

You control the robot by setting and executing a sequence of "waypoints":
* There is a blue virtual gripper in the scene, called the "target gripper".
  You manipulate this target gripper via tools to set a "waypoint".
* Repeat the following to complete a task, one waypoint at a time:
  * Set: move the target gripper to a desired pose (waypoint) via the tools.
  * Execute: a controller brings the real gripper to the target gripper via a
    linear interpolation while obeying physical constraints.
  * Check: after the waypoint is reached, the tool returns an image showing
    the updated UI (outcome). Decide the next pose based on the outcome. If
    the outcome isn't what you intended, diagnose why it failed and retry that
    step or any earlier one that caused it.

For example, to pick up a cube: move the gripper above it, descend until the
cube sits between the fingers, close, then lift. Each step is one waypoint
that you set and then execute before moving on to the next.

* gripper_home sets the target gripper to the arm's start pose; execute it like any other
↪  waypoint. When the target is green the controller drives a straight line there. When it is
↪  red no straight path exists from the current posture, but executing it still works: the
↪  controller moves the joints directly to the start posture, which can sweep the arm through
↪  whatever is in the way, so move clear of objects first. Use it when every target you try
↪  is red; it always succeeds.

### Dual-arm tasks

Some tasks have **two** robot arms (e.g. a lift-and-handover): a *move* arm and
a *hold* arm, each with its own blue target gripper. The gripper tools take an
optional `arm` argument giving the name of the arm to command; it defaults to
the move arm, so single-arm tasks and move-arm actions need not set it. Naming
an arm both selects that arm's blue target gripper (so `gripper_get_pose`,
`gripper_translate`, `execute_waypoint`, etc. act on it) and routes the executed
waypoint to that arm. Plan each arm's waypoints separately, one arm at a time;
commanding one arm leaves the other where it is. Use `gripper_get_pose` with the
`arm` argument to read either arm's current pose.

## Tips

### Decomposition and Planning
* Decompose the task into **small subtasks** by breaking the goal into a short
  sequence of simple stages.
* Prefer closed-loop control over open-loop. Adjust the plan as you observe
  the outcome of previous waypoints.
* Give `gripper_toggle` its own `execute_waypoint` cycle. Do NOT combine it
  with another pose change (translate/rotate/advance...) in one waypoint.
  Moving and toggling the gripper at the same time could be unpredictable.
* Be careful of collisions when making plans for how the end effector
  should move, especially when holding objects, because the held object extends
  beyond the fingers.
* Keep consecutive waypoints close together. The controller drives a straight
  line from one pose to the next and does NOT check for collisions along the
  way, so a large jump can sweep the arm, or whatever it is holding, through
  objects in the scene. Prefer several short waypoints that steer around
  obstacles over one long one, and avoid combining a big translation with a
  big rotation in a single waypoint.
```

```
### Precision
* Be precise with `gripper_teleport_via_click`; it is the best way to set a
  good initial position.
  - The clicked point lands on the gripper's approach axis, so a precise click
    will make future approaching easy.
  - Use `hover` to double-check whether the predicted (u,v) would land where
    you intend to click.
* Placing an object onto another object is a precision-sensitive task. First
  reason about where the held object should land, and then reason about where
  to set the target gripper, accounting for the relation between the gripper
  and the held object.
* Always check the visual from the point cloud or the camera feeds. Do NOT
  solely rely on the coordinates from `hover`. When necessary, make fine
  adjustments with translate, rotate, advance_or_retreat, and similar
  operations.

* The target gripper renders RED when the controller cannot REACH its current
  pose. The sidebar's "controller error here: X mm / Y deg" readout shows the
  expected miss. This is a reachability warning ONLY: it never indicates
  collision or contact. Fingers touching, pressing, or visually intersecting
  an object is normal during a grasp and needs no fixing. Respond to red by
  moving the target somewhere reachable. The one exception is the home target from
  ↪  gripper_home, which can always be executed. Do not respond by opening or lifting out of
  ↪  a good grasp.

### Viewing and Camera Movement
* Two camera feeds on the left
  - They provide raw camera images from 3rd-person view and wrist view.
  - You are NOT limited to these two vantages: `camera_views` shows the scene
    from all 10 cameras at once, and `camera_feed_select` can point the
    3rd-person panel at any of them (see "Extra camera views" below).
  - The 3rd-person view is good for understanding the scene and disambiguating
    objects.
  - The wrist view is good for close-up analysis and fine adjustments.
  - `gripper_teleport_via_click` also accepts clicking on a camera feed.
    Sometimes this could be more convenient than clicking on the point cloud.
* The 3D point cloud on the right
  - This is the main workspace. It constructs the 3D scene from multiple depth
    cameras.
  - The initial view (resettable via `camera_reset`) is framed to be good for
    most operations. It is a good default so prefer working in it.
  - A pure top-down view (elevation >= 80) may not be as useful as you think,
    because the gripper and robot arm may block most of the workspace.
  - Sideview (orbit by changing azimuth +/- 30~45) or frontview (elevation
    ~ 10) could be very helpful if object-gripper or object-object relation is
    hard to interpret in the default view.
  - Move the camera gently, and reset to the initial view via `camera_reset`
    to recover from a failed camera operation.

### Extra camera views (via-cam)
* `camera_views` overlays live renderings of EVERY selectable camera over the
  big point-cloud panel: the default agentview, the 3-camera ring the point
  cloud is fused from (view k looks inward from azimuth 90+k*120 deg: ring0 from
  +y right, ring1 behind-left, ring2 front-right), a straight-down birdseye, a
  `zoomed_birdseye` from half the height (2x detail on the workspace center),
  an `eyelevel` view looking near-horizontally across the table, and a `farside`
  station just past the far (+y) edge of the workspace, a hand's width above the
  surface, looking back across it. The wrist keeps its own panel and no tile.
  Survey it whenever the current views leave something occluded or ambiguous.
* `camera_feed_select(view=...)` switches the 3rd-person panel to any of those
  cameras, sticky until changed; clicks on the feed then project through the
  selected camera, so `gripper_teleport_via_click` keeps working. The birdseye
  (top of image = robot's side, right = +y) is the best vantage for xy
  alignment; a ring view facing an occlusion resolves it. `view='agentview'`
  restores the default.
* `camera_raw(view=...)` returns ONE camera's frame at native render fidelity --
  no UI tiles, overlays, or browser rescaling. Views: `agentview`, `wrist`,
  `ring0`..`ring2`, `birdseye`, `zoomed_birdseye`, `eyelevel`, `farside`. Reach for it when a
  screenshot's small tiles aren't enough: fine alignment checks, small or
  partially occluded objects, texture/text a composited view blurs.
  `camera_raw(view='zoomed_birdseye')` is the sharpest xy-alignment read
  available. Use it before and after fine placement/insertion waypoints.
  Every camera renders at 448x448.
* `eyelevel` and `farside` are the views that show HEIGHT directly. Every other camera
  looks down, where "5 cm above the table" and "resting on it" differ by a few
  pixels of size; edge-on they are unmistakable. Check it before a descent you
  are unsure of, and to confirm a lift actually cleared the surface.
* Feeds refresh after every `execute_waypoint`, like the other panels.

### Common failures
* Grasping too high is a common failure. Descend deep so that the fingers and
  the object to hold have enough contact surface.
```

```
## Episode protocol

* `execute_waypoint` reports EPISODE OVER when the episode ends (goal reached or
  step budget spent). Stop; every further tool call returns the same line.
* Nothing is graded until you declare. `end_episode` is how you submit: it is
  the only thing that gets you graded, it is FINAL, and there is no feedback
  before it. Check the scene yourself first.
* The episode does NOT end on its own when the task is achieved. It runs until
  you declare or the step budget is spent. If an attempt fails (dropped object,
  toppled stack), diagnose and retry; fumbling is not failing. What you cannot
  retry is the declaration.
```

A.2.6 MULTI-VIEW RGBD INTERFACE GUIDE

```
# RGB-D waypoint control
Operate the single Panda arm or the two YAM arms using only the seven registered tools. Call
↪  observe first.

observe() defaults to agentview and all wrist cameras. It returns each camera's224x224 RGB
↪  image and aligned grayscale depth PNG, with intrinsics K and T_robot_from_camera. Separate
↪  mode places labels immediately before images. Tiled mode sends one PNG containing the
↪  same224x224 panels, with explicit panel xywh coordinates. Intrinsics always refer to
↪  individual panels; subtract a panel offset before interpreting its pixels. The selected
↪  views persist for action feedback. camera_views surveys all cameras at224x224 RGB;
↪  camera_raw returns one native RGB image (normally448x448) for details. Neither changes the
↪  selection.

Camera coordinates are X right,Y down,Z forward. Depth is optical-axis Z in metres, not ray
↪  distance. Depth PNG code0 means invalid or outside0.05..3.0m; valid codes1..255 map to
↪  z=0.05+(code-1)*2.95/254. Darker valid pixels are nearer. This is quantized visual depth,
↪  not an exact depth oracle. Black invalid pixels are not zero-distance surfaces. No
↪  per-frame normalization. K matches the returned image, with top-left pixel center(0,0).
↪  T_robot_from_camera maps camera points to the same coordinate frame as
↪  get_proprio/set_pose (RoboCUA simulator world metres). Wrist extrinsics change with the
↪  arm. Calibration and images share frame_id.

On two-arm tasks, every get_proprio, set_pose and set_gripper call requires arm="left" or
↪  arm="right". These sides match the default scene view (right blue, left orange);
↪  observation metadata maps sides to robot arm names and wrist cameras. Both arms use the
↪  same simulator world coordinate frame, not separate base frames. For example,
↪  set_pose(arm="right", d_z=0.01) lifts the right arm by 1 cm from its actual pose. The
↪  other arm keeps its existing controller command; physical contact may still move it.
↪  Actions execute one at a time and wait for completion; alternate arms for handoffs. Omit
↪  arm on single-arm tasks.

get_proprio reports actual fingertip-midpoint position, orientation, and the commanded
↪  open/closed state, not measured jaw width or grasp success. set_pose immediately executes
↪  translations in metres and rotations in degrees around fixed robot axes, pivoting at the
↪  fingertip midpoint. Omitted deltas are zero. Rotations compose onto the current rotation;
↪  they are not additions to Euler readouts. set_gripper separately opens/closes. Your
↪  existing controller handles interpolation and contact; blocked moves may stop short. Each
↪  action returns actual pose/displacement plus fresh selected RGB-D views. Inspect feedback
↪  and correct incrementally.

There is no target preview, hover, point-cloud screenshot, camera-motion tool or separate
↪  execute action. Use native camera_raw only when224px images lack needed detail. Reuse
↪  useful observation views rather than repeatedly requesting every camera. End with
↪  end_episode when finished. Stop on EPISODE OVER; no grading feedback is available.
```

A.3 WHERE A RETRY LANDS

Table 4 follows each retry to the outcome of the next try, grouped by why the previous try failed. It supports the recovery analysis in Q2.

Previous try failed by, then next try	Sol VIA	Opus VIA	Fable RGBD	Fable VIA	Astra RGBD	Astra VIA
After a precision miss (share of retries)	38%	46%	51%	50%	48%	46%
then done	12%	31%	22%	39%	17%	36%
then precision miss	42%	44%	50%	48%	44%	42%
then wrong approach	43%	25%	28%	12%	37%	22%
then wrong object	3%	0%	0%	0%	2%	0%
After a wrong approach (share of retries)	56%	52%	43%	44%	50%	52%
then done	13%	16%	15%	28%	31%	37%
then precision miss	12%	36%	31%	24%	30%	26%
then wrong approach	75%	48%	54%	48%	38%	37%
then wrong object	1%	0%	0%	1%	0%	0%
After a wrong object (share of retries)	6%	1%	6%	5%	3%	2%
then done	0%	0%	0%	0%	0%	0%
then precision miss	7%	50%	0%	0%	45%	67%
then wrong approach	50%	0%	25%	0%	20%	0%
then wrong object	43%	50%	75%	100%	35%	33%

Table 4: **Where a retry lands.** Each retry is a try at a subtask that follows a failed try at the same subtask, grouped by why the previous try failed. For each group, the share of retries that ended done, in a precision miss, in a wrong approach or on the wrong object. Shares are computed per task, then averaged over tasks. After a wrong object, only 3 to 13 retries per model exist, so those shares are noisy.

A.4 SUBTASK VERIFICATION

Table 5 counts the decisions where an agent can be wrong about a subtask being done: ending the episode believing the task is complete, and moving on from a named single-object subtask to another subtask. It supports the analysis in Q3.

	Sol VIA	Opus VIA	Fable RGBD	Fable VIA	Astra RGBD	Astra VIA
Ended the episode believing it was done, but not solved	20/30 (67%)	39/63 (62%)	28/57 (49%)	30/66 (45%)	26/68 (38%)	20/61 (33%)
Left a named single-object subtask undone for another subtask	51/66 (77%)	31/52 (60%)	45/74 (61%)	16/63 (25%)	60/107 (56%)	39/90 (43%)
Ended the episode saying it could not do it (not counted)	3	0	0	1	5	0
Subtask verification accuracy	26.0%	39.1%	44.3%	64.3%	50.9%	60.9%

Table 5: **Subtask verification.** Wrong decisions over all decisions of each kind. Subtask verification accuracy is the share of right decisions over both kinds together. Budget cut-offs and episodes the agent ended saying it could not finish are left out, since they claim nothing.

A.5 ZERO-SHOT EVALUATION, ALL METRICS

Success only trials

Model	Interface	Success (%)	Cost ($)	EEF cmds	Turns/EEF	Calls/EEF	Wall/EEF (min)
GPT-5.6-Sol	VIA	16.9	16.61	111	2.5	2.5	0.46
Claude Opus 5	VIA	30.9	35.15	70	4.3	4.6	1.36
Claude Fable 5.1	Multi-view RGBD	36.5	15.63	93	1.1	1.6	0.79
	VIA	41.5	31.36	71	2.0	5.6	1.43
GPT-6-Astra	Multi-view RGBD	47.3	7.20	64	1.3	1.4	0.41
	VIA	49.2	20.37	74	2.4	2.9	0.58

All trials

Model	Interface	Cost ($)	EEF cmds	Turns/EEF	Calls/EEF	Wall/EEF (min)
GPT-5.6-Sol	VIA	25.13	128	3.7	3.3	0.91
Claude Opus 5	VIA	56.86	92	5.0	5.7	2.13
Claude Fable 5.1	Multi-view RGBD	30.40	115	1.0	1.7	1.11
	VIA	51.04	88	2.2	6.9	1.98
GPT-6-Astra	Multi-view RGBD	13.39	93	1.3	1.3	0.44
	VIA	32.47	105	2.3	3.0	0.70

Table 6: **Zero-shot evaluation, all metrics.** The agents and interfaces of Table 1 with success-only and all-trial metrics, averaged over 22 tasks with 5 seeds each. "Success only" excludes trials with any failed subtasks. EEF cmds counts the distinct end-effector commands the agent gives the simulator; Turns/EEF, Calls/EEF and Wall/EEF divide a trial's model turns (requests to the model, each of which can issue several tool calls), tool calls and wall time (excluding offline Blender rendering) by that count.

A.6 Autonomous improvement, all metrics

Success only trials

Method	Success (%)	Cost ($)	EEF cmds	Turns/EEF	Calls/EEF	Wall/EEF (min)
GPT-6-Astra baseline	54.0	18.24	67	2.5	3.5	0.63
In-context success demo (ICL)	72.7	26.97	52	3.3	4.4	0.60
Text prompt, round 1 (T1)	62.5	27.78	68	3.1	3.4	0.67
Text prompt, round 2 (T2)	72.0	20.13	73	2.3	3.2	0.54

All trials

Method	Cost ($)	EEF cmds	Turns/EEF	Calls/EEF	Wall/EEF (min)
GPT-6-Astra baseline	28.23	87	2.5	3.4	0.65
In-context success demo (ICL)	30.64	64	3.1	4.2	0.62
Text prompt, round 1 (T1)	30.78	75	3.0	3.3	0.66
Text prompt, round 2 (T2)	22.77	79	2.3	3.2	0.54

Table 7: **Autonomous improvement, all metrics.** The arms of Table 2 with success-only and all-trial metrics. EEF cmds counts the end-effector commands the simulator executed; Turns/EEF, Calls/EEF and Wall/EEF divide a trial's model turns, tool calls and wall time (excluding offline Blender rendering) by that count.

A.7 Autonomous improvement by task

Task	Baseline	ICL	T1	T2
Beam	5 / 2	8 / 1	8 / 1	9 / 0
PCR setup	6 / 2	8 / 0	5 / 1	5 / 0
Cup pyramid	3 / 7	5 / 4	5 / 3	7 / 3
Stool	3 / 3	4 / 1	3 / 0	4 / 1
Handover dominos	3 / 6	6 / 0	5 / 4	8 / 1
All (of 50)	20 / 20	31 / 6	26 / 9	33 / 5
failures ended by a success claim	10/30 (33%)	13/19 (68%)	15/24 (63%)	12/17 (71%)
Gamepad (of 5, no successful episode)	0 / 3	–	3 / 2	5 / 0
failures ended by a success claim	2/5 (40%)	–	0/2 (0%)	–

Table 8: **Autonomous improvement by task.** Trials solved / trials that exhausted their simulation-step or wall-clock budget, out of 10 per task and 50 in total. Failures not ended by the budget were ended by the agent itself, declaring the task done: the "success claim" rows give their number and share of that arm's failures. Gamepad, run separately with 5 trials per arm and no ICL arm, has no successful Astra episode: its prompts are written from failed attempts.

A.8 Subtask levels as stacked bars

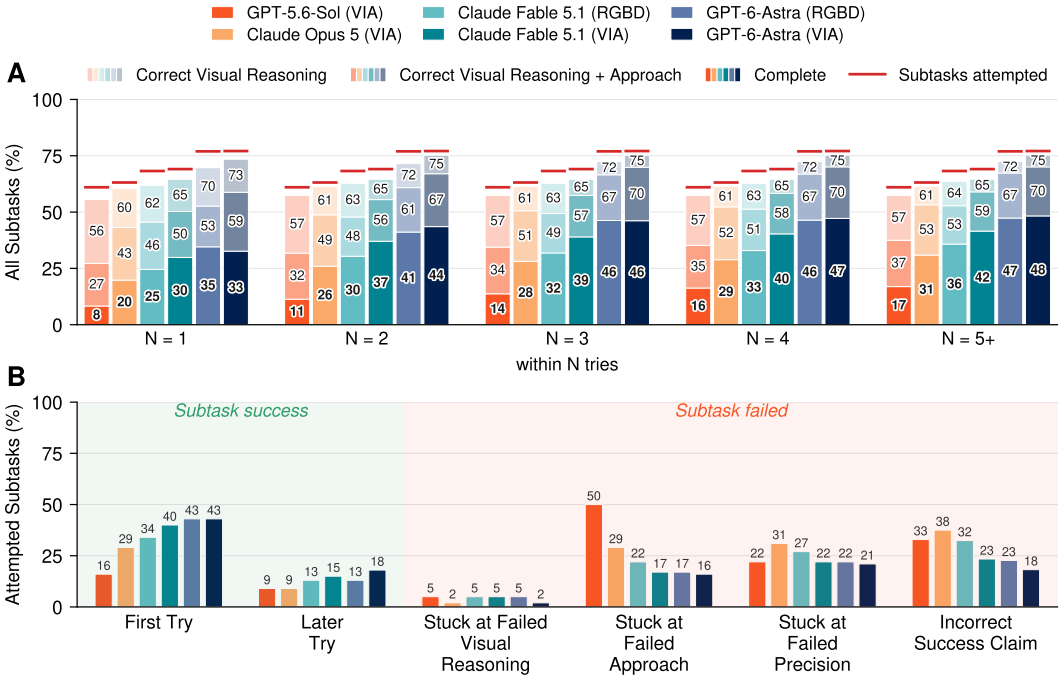

Figure 4: **Top:** This plot includes all rubric subtasks, even those the agent never reached. It shows the share with correct visual reasoning, the share with both correct visual reasoning and approach, and the share succeeded (i.e. correct visual understanding + approach + precision) in at least one of the first N tries, so every level grows with N. GPT-6 Astra generally performs best. More tries mainly improve likelihood of correct approach and success. Visual reasoning changes little. **Bottom:** This plot breaks down agent behavior on subtasks it attempted. Stronger models succeed at more subtasks on the first try or through monotonic progress, without reverting to an incorrect approach after finding the right one. When they get stuck, precision errors are more common than visual reasoning or approach errors. Stronger models also make fewer incorrect success claims, though GPT-6 Astra still claims success on 18% of subtasks it did not succeed at.