FRONTIER MODELS FOR ROBOTIC MANIPULATION: RETHINKING INTERACTION AND SCAFFOLDING

Anonymous authors
Paper under double-blind review

ABSTRACT

Frontier foundation models can reason over images, write programs, and interpret execution feedback. We ask how these capabilities should be exposed to a robot, comparing two interaction paradigms: execution-time decision making, where the model chooses each next action, and executable-policy synthesis, where the model writes a persistent robot program and revises it from execution feedback. In our instantiation of the latter, GPT-6 works in a coding session with a minimal robot interface and repeatedly writes, executes, observes, and revises a policy. We evaluate on LIBERO, LIBERO-Pro, and the RoboCasa365 Target50 subset, then vary the refinement budget and the amount of human-designed scaffolding with the model, interface, and environment fixed. We conclude the following: First, stronger models improve both interaction paradigms: GPT-6 achieves 100% success on LIBERO, 95.0% on LIBERO-Pro, and 88.0% on RoboCasa365 Target50, including 75.0% on unseen composite tasks. Second, interaction remains important even for frontier models: increasing the refinement budget from one to eight turns raises LIBERO-Pro success from 45% to 94%, showing that executable-policy synthesis remains an iterative rather than one-shot generation problem. Third, additional imposed structure provides no consistent benefit: predefined skills, abstractions, and heuristic scaffolding do not reliably outperform a minimal interface and can restrict how the model discovers and revises solutions. Together, these results suggest that realizing frontier-model capabilities in robotics depends not only on model strength, but also on interaction design: effective interfaces should provide sufficient opportunities for execution feedback and revision without unnecessarily constraining the model's reasoning process.

1 INTRODUCTION

Frontier foundation models have recently made rapid progress in multimodal reasoning, programming, and long-context problem solving (Anthropic, 2026; OpenAI, 2026; Google Gemini Team, 2026). At the same time, robotic systems are increasingly incorporating such models not only as language interfaces, but as active components of robot reasoning and execution (Ahn et al., 2023; Driess et al., 2023; Liang et al., 2023; Zitkovich et al., 2023). Existing systems connect foundation-model reasoning to robot execution in different ways. We organize these approaches into two common interaction paradigms (Fig. 2). The first one is **execution-time decision making**, where the foundation model is used directly inside the execution loop and repeatedly determines what should happen next, for example by selecting a subgoal, skill, action, or tool invocation from the current observation and execution history (Zhang et al., 2026c; Sun et al., 2026; Robocurve, 2026). The second paradigm is **executable-policy synthesis**, where the foundation model instead produces an explicit robot policy, typically an executable program, that governs behavior over a longer execution horizon (Liang et al., 2023; Fu et al., 2026; Lu et al., 2026; Xiao et al., 2026). The policy can be executed, evaluated through the resulting observations and trajectories, and subsequently revised using execution feedback. As foundation-model capabilities continue to improve, one question becomes increasingly important: *how can frontier models facilitate robotic manipulation tasks through different interaction paradigms?*

We begin approaching this question with the LIBERO (Liu et al., 2023) and LIBERO-Pro (Zhou et al., 2025) benchmarks, which provide controlled settings for studying language grounding, spatial reasoning, and manipulation robustness on relatively compact tasks. For execution-time decision-

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.

making, we use the HarnessVLA system (Zhang et al., 2026c), where the model repeatedly determines the next operation during execution. For executable-policy synthesis, we use a deliberately minimal setup in which GPT-6 Astra, running in a persistent Codex session, repeatedly writes, executes, observes, and revises an executable robot policy using environment feedback. Only observations and a low-level robot interface are exposed, leaving policy construction and refinement to the model itself. Under the HarnessVLA interface, replacing GPT-5.5 with GPT-6 increases overall LIBERO-Pro success from 72.1% to 92.6%, while GPT-6 with the iterative refinement reaches 100% on LIBERO and 95.0% on LIBERO-Pro. These results show that recent frontier models exhibit strong grounding, spatial reasoning, and manipulation capabilities, and that these capabilities can be expressed effectively through both interaction paradigms on relatively compact tasks.

This strong performance naturally raises a broader question: *how far can these capabilities extend as manipulation becomes more compositional and long-horizon?* RoboCasa365 (Nasiriany et al., 2026) provides such a setting, with household tasks that require reasoning over multiple objects, intermediate states, and dependent manipulation stages. On its Target50 subset, GPT-6 with iterative refinement achieves 88.0% overall success, with 100.0% on Atomic-Seen, 87.5% on Composite-Seen, and 75.0% on Composite-Unseen tasks. Inspection of the policy-revision process illustrates how the model constructs and refines multi-stage solutions across revisions. The model maintains the task-level solution as an explicit executable program, organizes manipulation into stages, and reuses shared routines. Execution feedback guides revisions to the relevant parts of the policy while the overall task structure is preserved. These observations illustrate how frontier models express spatial reasoning, multi-stage planning, and feedback-driven policy revision through executable-policy synthesis.

These observations motivate a closer examination of the design choices within executable-policy synthesis. Therefore, we study how much iterative refinement is required, how different foundation models use the same refinement loop in terms of interaction and token cost, and whether additional human-designed structure remains beneficial. Varying only the maximum number of *write–execute–observe–revise* turns available to GPT-6 produces a clear effect: across the four LIBERO-Pro suites, overall success increases from 45% with one turn to 78%, 88%, and 94% with two, four, and eight turns, respectively. The trend is consistent across all suites, showing that execution feedback remains an integral part of policy synthesis rather than merely a mechanism for one-shot program generation.

We further evaluate different foundation models, measuring final success together with the number of refinement turns and token consumption required to solve each task. This characterizes not only whether a model can construct a successful policy, but how efficiently it can use execution feedback to reach one. Finally, we examine additional structures commonly introduced by Code-as-Policy systems, including predefined primitives, reusable skills, task decomposition, memory, and heuristic guidance (Fu et al., 2026; Zhang et al., 2026a; Lu et al., 2026). In contrast to iterative refinement, increasingly prescriptive human-designed structure does not consistently improve performance and can restrict the model's own solution process. These results suggest a simple design principle: frontier models continue to benefit substantially from interaction, while increasingly elaborate scaffolding is not necessarily required. A minimal interface that exposes observations, execution feedback, and low-level robot capabilities can leave spatial reasoning, planning, and policy construction to the model itself.

Overall, our findings are threefold. First, on LIBERO and LIBERO-Pro, frontier models perform strongly under both interaction paradigms, demonstrating substantial grounding and spatial reasoning capabilities. Second, on the more compositional RoboCasa365 tasks, the iterative refinement turns demonstrate how frontier models construct multi-stage executable solutions and revise them using execution feedback. Third, controlled ablations show that iterative execution feedback remains important, while additional human-designed primitives and heuristics provide limited benefit and can become restrictive.

2 Related Work

Generalist robot policies and frontier models. Vision-Language-Action (VLA) models learn visuomotor policies that map multimodal observations and language instructions directly to robot actions. Building on large-scale robot data, multimodal pretraining, and improved policy architectures, systems such as RT-2, OpenVLA, π_0, $\pi_{0.5}$, NORA, and AtomVLA have steadily improved

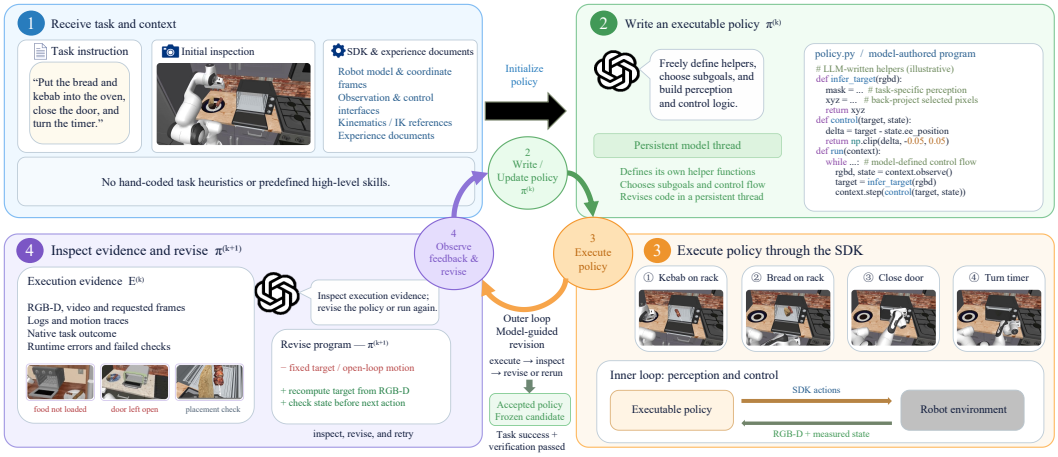

Figure 1: A foundation model freely writes executable robot policies and refines them using execution feedback. Each policy performs closed-loop perception and control, while the model revises the program across attempts until success is verified. Images show RoboCasa Heat Kebab Sandwich demonstrations.

manipulation capability (Zitkovich et al., 2023; Kim et al., 2025; Black et al., 2025b;a; Hung et al., 2025; Sun et al., 2026). A parallel development is that general-purpose frontier models, trained without any robot-specific policy learning, are beginning to act as robot policies themselves. Early attempts used language models to emit dense end-effector trajectories or showed only partial subgoal progress (Kwon et al., 2024; Berman et al., 2026), but recent frontier models can now complete a non-trivial fraction of standard manipulation benchmarks and real-world tasks under a generic action interface, either by directly commanding the robot or by writing and revising programs at runtime (Zhang et al., 2026b; Jia et al., 2026; Su et al., 2026). These developments raise a question beyond whether frontier models can control robots: *how should their reasoning capabilities be integrated into the robot execution process?*

Evaluations of frontier models as manipulation policies. A growing body of work evaluates general-purpose frontier models, including GPT-6 Astra and Claude, as robot policies without task-specific training, both in simulation and on physical hardware (Zhang et al., 2026b; Berman et al., 2026; Menon et al., 2026; Pantograph, 2026; Jia et al., 2026). These evaluations consistently report a markedly non-uniform capability profile: performance is competitive with trained policies on tasks whose difficulty lies primarily in semantic understanding and spatial reasoning, but substantially lower on tasks that demand fine geometric precision, dynamic control, or bimanual coordination (Zhang et al., 2026b; Menon et al., 2026). Outcomes further depend strongly on the specific model, embodiment, and action interface (Zhang et al., 2026b; Pantograph, 2026). Notably, several studies observe that the same model attains considerably higher success when its reasoning is coupled to the robot through an alternative mechanism, either by supervising the outputs of a pretrained VLA or by synthesizing and revising executable programs (Su et al., 2026; Berman et al., 2026; Jia et al., 2026). These findings indicate that embodied reasoning capability alone does not determine manipulation performance; the mechanism by which it is integrated with perception, action, and environment interaction is itself a critical design dimension, which we examine next.

Foundation models for execution-time decision making. A long line of work places a language or vision-language model in the robot execution loop as a high-level decision maker. SayCan grounds language-model plans in the affordances of a fixed skill library, and Inner Monologue closes the loop by feeding environment feedback back into planning (Ahn et al., 2023; Huang et al., 2023); Hi Robot extends this pattern to VLA-based execution, with a reasoning model issuing subinstructions to a low-level policy (Shi et al., 2025). Recent agentic harnesses build on the same structure with richer machinery. RoboClaw and RoboHarness let a VLM orchestrate heterogeneous controllers, including VLAs, RL policies, and TAMP systems, using structured execution memory to monitor progress, characterize policy capability boundaries, and trigger recovery (Li et al., 2026;

Huang et al., 2026); HarnessVLA uses a memory-guided planner to compose a frozen VLA with analytic primitives and refines primitive selection from execution experience (Zhang et al., 2026c). Systematic studies confirm that planner choice, controller choice, observation representation, and the planner-controller interface all substantially affect manipulation performance (Hu et al., 2026). Despite differences in controllers and memory, these systems share one interaction pattern: the foundation model stays in the loop and, at each step, selects what to do next from a set of existing behaviors based on the current observation and execution history. We refer to this family as *execution-time decision making*.

Executable policy synthesis and Code-as-Policy. An alternative line of work represents robot behavior as executable programs rather than as choices among existing skills. Code as Policies showed that a code-generating language model can synthesize policies that combine perception calls, geometric computation, control primitives, and feedback loops (Liang et al., 2023). CaP-X systematizes this paradigm, benchmarking coding agents across levels of action abstraction, perceptual grounding, and interaction, and showing that human-designed abstractions, structured execution feedback, and increased test-time computation improve robustness at lower-level interfaces (Fu et al., 2026). Subsequent work uses execution experience to grow a reusable code base: RATs acquires code skills through self-directed play and retrieves them for downstream tasks (Zhang et al., 2026a), and ASPIRE accumulates validated solutions into a persistent skill library that transfers across tasks and embodiments (Lu et al., 2026). Most recently, Agent as Policy applies this representation at execution time, with a frontier agent writing and revising programs in the loop and reusing saved procedures across trials (Jia et al., 2026). Across these systems, executable policies serve as a natural substrate for incorporating environment feedback and accumulating robot experience.

3 How Do Frontier Models Interact with Robot Execution?

As frontier foundation models (OpenAI, 2026; Anthropic, 2026; Google Gemini Team, 2026) become increasingly capable of reasoning, programming, and interpreting multimodal feedback, an important question is how these capabilities should be incorporated into robotic systems. In this work, we study how frontier-model capabilities are expressed through these interaction paradigms in representative robotic systems.

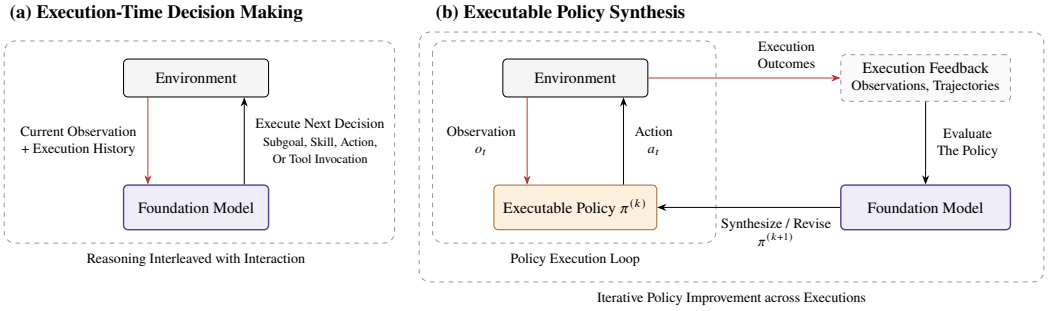

Figure 2: Two common interaction paradigms for integrating frontier foundation-model reasoning with robot execution.

As illustrated in Figure 2, we describe two common interaction paradigms through which foundation models participate in robot execution.

The first interaction paradigm uses the foundation model as an execution-time decision maker (Hu et al., 2026; Zhang et al., 2026c; Huang et al., 2023). Given the current observation, task specification, and execution history, the model repeatedly determines what should be executed next, such as a subgoal, skill, action, or tool invocation. Reasoning is therefore interleaved with environment interaction: each decision is executed, its outcome becomes part of the subsequent context, and the model reasons again to determine the next step.

The second interaction paradigm represents robot behavior as an executable policy (Liang et al., 2023; Fu et al., 2026; Lu et al., 2026; Xiao et al., 2026). Rather than repeatedly deciding what to

execute next, the model synthesizes a program that specifies the robot's behavior over an extended execution horizon. This policy can be executed in the environment, evaluated through the resulting observations and trajectories, and subsequently revised based on execution feedback. Reasoning is therefore applied to the executable policy itself, with interaction serving to iteratively improve the policy across successive executions.

Using these two interaction paradigms to organize existing systems, we investigate three questions: (1) How are frontier-model capabilities expressed in manipulation performance under these interaction paradigms? (2) How do these capabilities extend to compositional and long-horizon manipulation, and how does the model construct and revise multi-stage solutions? (3) How do iterative execution feedback and external task scaffolding affect policy synthesis?

3.1 Frontier Model Capabilities under Both Interaction Paradigms

To investigate how frontier models behave under the two interaction paradigms, we firstly evaluate them on LIBERO (Liu et al., 2023) and its perturbed extension, LIBERO-Pro (Zhou et al., 2025). Standard LIBERO spans the Spatial, Object, Goal, and LIBERO-10 suites, providing a canonical evaluation of language-conditioned manipulation. LIBERO-Pro extends this setting with controlled perturbations to task and scene configurations, including changes to object properties, spatial arrangements, instructions, and environments. Together, the two benchmarks allow us to examine how the interaction paradigms behave both in the standard benchmark setting and when the task or environment departs from its canonical configuration.

Compared Methods The comparisons include three families of approaches. OpenVLA (Kim et al., 2025), NORA (Hung et al., 2025), π_0 (Black et al., 2025b), the $\pi_{0.5}$-SFT policy evaluated through RLinf (Black et al., 2025a; Yu et al., 2026), and AtomVLA (Sun et al., 2026) serve as learned VLA baselines. HarnessVLA (Zhang et al., 2026c) is included as a representative execution-time decision-making approach, while CaP-X (Fu et al., 2026), RATS (Zhang et al., 2026a), ASPIRE (Lu et al., 2026), and our setup operates over executable robot programs.

As shown in Fig 1, the model writes a robot policy in plain Python against a minimal control interface, executes it in simulation, and receives the resulting environment feedback, including rollout outcomes, logs, observations, and video. It then revises the same policy and repeats this *write–execute–observe–revise* cycle until the task succeeds or the time budget is exhausted. The loop is deliberately minimal in externally supplied task structure: we provide no task-specific heuristic guidance, predefined multi-step manipulation skills, or task-decomposition templates. The model remains free to construct and reuse its own helper functions and subroutines during interaction. Policy execution is sandboxed from the simulator source code and success function, so task success is determined by the environment rather than asserted by the model (Appendix A).

As shown in Table 1, GPT-6 with our designed iterative refinement achieves 100% success across all four standard LIBERO suites, indicating that executable-policy refinement with a frontier model already reaches saturation on the canonical benchmark. LIBERO-Pro provides a more discriminative setting. With the HarnessVLA configuration and low-level VLA/primitive stack held fixed, replacing GPT-5.5 with GPT-6 increases overall success from 72.1% to 92.6%, with consistent gains across both spatial and task perturbations. This controlled comparison shows that stronger frontier-model capabilities translate directly into more effective execution-time decision-making when the task or scene departs from its canonical configuration.

The executable-policy paradigm shows a similar trend. GPT-6 with our designed iterative refinement reaches 95.0% overall success on LIBERO-Pro, substantially outperforming earlier program-based systems in Table 2. Although this comparison does not isolate model capability from system design, it shows that frontier-model capabilities can also be effectively expressed through iterative policy synthesis and refinement. Together, these results suggest that advances in frontier models benefit both interaction paradigms: one through stronger decisions during execution, and the other through stronger reasoning over the executable policy itself.

Figure 3 illustrates what this improvement looks like in practice. After a direct attempt to grasp the drawer handle fails because a bowl blocks access, the model does not simply retry the same motion. Instead, it identifies the obstruction, rewrites the policy to move the bowl away, and then replans the grasp. Execution feedback therefore serves not only as a failure signal, but as evidence for revising

Table 1: Success rate (%) on standard LIBERO. Bold marks the best performance in each column.

Method	Spatial	Object	Goal	LIBERO-10	Overall
OpenVLA (Kim et al., 2025)	84.7	88.4	79.2	53.7	76.5
NORA (Hung et al., 2025)	85.6	89.4	80.0	63.0	79.5
π_0 (Black et al., 2025b)	96.8	98.8	95.8	85.2	94.2
RLinf ($\pi_{0.5}$-SFT) (Black et al., 2025a; Yu et al., 2026)	99.0	96.0	97.0	89.0	95.3
AtomVLA (Sun et al., 2026)	96.4	99.6	97.6	94.4	97.0
HarnessVLA (CC) (Zhang et al., 2026c)	97.0	100.0	94.0	93.0	96.0
Iterative GPT-6 + Codex	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**

Table 2: Comparison of different methods on LIBERO-Pro. Results are reported as success rates (%). T and S denote task/instruction perturbation and spatial/position perturbation, respectively.

Method	Spat-T	Spat-S	Obj-T	Obj-S	Goal-T	Goal-S	L10-T	L10-S	Overall
CaP-X (Fu et al., 2026)	14.0	12.0	18.0	22.0	17.0	26.0	/	/	18.2
RATS (Zhang et al., 2026a)	31.0	29.0	63.0	61.0	36.0	43.0	/	/	43.8
ASPIRE (Lu et al., 2026)	60.0	51.0	95.0	98.0	45.0	81.0	38.3†	22.6†	–
Iterative GPT-6 + Codex	**100.0**	90.0	**100.0**	**100.0**	**90.0**	**100.0**	80.0	**100.0**	**95.0**
RLinf ($\pi_{0.5}$-SFT) (Black et al., 2025a; Yu et al., 2026)	42.0	59.0	71.0	78.0	45.0	42.0	49.0	14.0	50.0
HarnessVLA (GPT-5.5) (Zhang et al., 2026c)	81.0	69.0	94.0	91.0	75.0	66.0	52.0	49.0	72.1
HarnessVLA (GPT-6) (Zhang et al., 2026c)	**100.0**	**98.0**	**100.0**	99.0	88.0	99.0	**85.0**	72.0	92.6

the model's interpretation of the scene and the policy that follows from it. This ability to re-ground behavior from execution feedback provides a concrete explanation for the strong performance under LIBERO-Pro perturbations, and motivates our next question: whether the same process remains effective when reasoning and execution must unfold over substantially longer horizons.

3.2 When Manipulation Becomes Long-Horizon

To study how the two paradigms behave as manipulation becomes more compositional, we turn to RoboCasa365 (Nasiriany et al., 2026), a household manipulation benchmark with 365 tasks across 2,500 kitchen environments. We use its 50-task target suite: 18 Atomic-Seen tasks that exercise individual skills, 16 Composite-Seen tasks that chain multiple stages so that earlier actions change the state faced by later decisions, and 16 Composite-Unseen tasks whose compositions are held out from pretraining. Following Harness VLA (Zhang et al., 2026c), a session ends at the first success or after 30 minutes for atomic and 60 minutes for composite tasks. Compared to LIBERO, these splits move the evaluation from adapting to perturbed configurations toward sustaining reasoning and execution over longer, compositional horizons.

Atomic manipulation. The Atomic-Seen split largely continues the pattern observed on LIBERO. HarnessVLA achieves 92.0% with GPT-5.5 and 87.8% with GPT-6 Astra, while iterative Codex refinement with GPT-6 Astra reaches 100.0%. All three model–system configurations perform strongly when the task remains a relatively compact manipulation problem. Together with the large gains under spatial perturbations on LIBERO-Pro, this suggests that frontier models have already become highly capable at the local reasoning required for atomic manipulation: grounding the current scene, understanding object and spatial relations, and determining how the immediate manipulation should be executed. These results extend the strong performance observed on relatively compact LIBERO tasks to atomic household manipulation.

Compositional manipulation. Compositional manipulation requires coordinating multiple dependent stages. The HarnessVLA results show that model upgrades have task-dependent effects: replacing GPT-5.5 with GPT-6 Astra changes Composite-Seen success from 61.0% to 43.8% and Composite-Unseen success from 13.8% to 42.5%. With the Codex iterative refinement, GPT-6 achieves 87.5% on Composite-Seen and 75.0% on Composite-Unseen. To understand how the model constructs and refines multi-stage solutions within the refinement loop, we examine the policy-revision process in Figure 3(b). For the RoboCasa *Heat Kebab Sandwich* task, the policy explicitly decomposes the task into placing the bread, placing the kebab, closing the oven door, and activating the timer, while reusing a shared *localize–grasp–insert* routine for the two place-

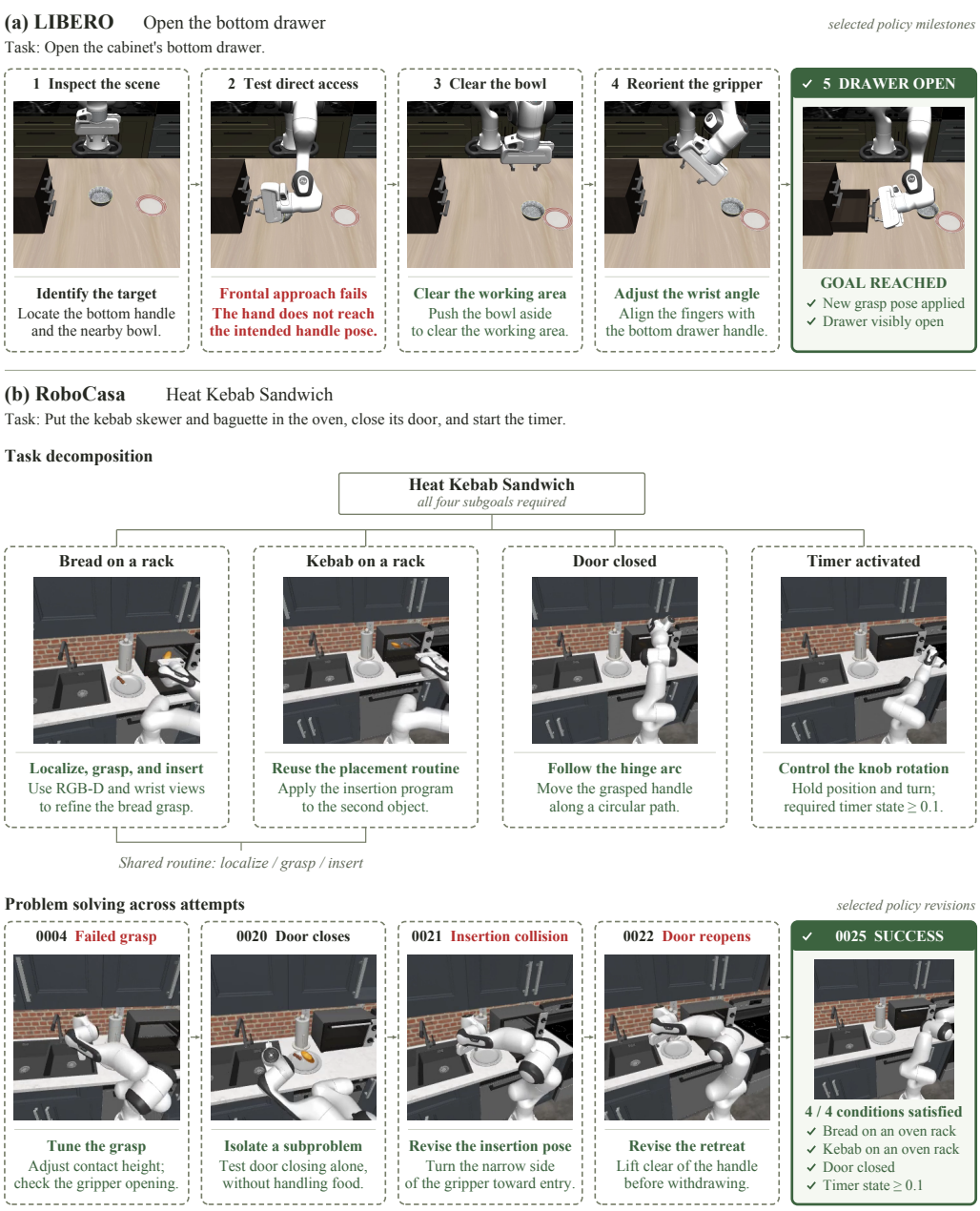

Figure 3: Examples of iterative policy refinement on LIBERO and RoboCasa. (a) On LIBERO, execution feedback allows the model to revise a failed local manipulation strategy by clearing an obstruction and adjusting the grasp before opening the drawer. (b) On the longer-horizon RoboCasa *Heat Kebab Sandwich* task, the model decomposes the task into multiple subgoals and repeatedly revises the policy across attempts to resolve grasping, insertion, and door-manipulation failures. You can check more in the Appendix html.

ment stages. Across execution attempts, failures in grasping, insertion, and retreat lead to targeted revisions of the corresponding parts of the policy while the overall task structure is preserved.

This example illustrates how an explicit executable policy enables the model to organize a multi-stage solution. Across attempts, the model revises failed components while retaining the overall task structure and reusing shared routines. Together with the LIBERO examples, these observations

Table 3: Success rates (%) on the RoboCasa365 target-task evaluation.

Method	Atomic-Seen	Composite-Seen	Composite-Unseen	Overall
π_0 (Black et al., 2025b)	34.6	6.1	1.1	14.8
$\pi_{0.5}$ (Black et al., 2025a)	39.6	7.1	1.2	16.9
RLDX-1 (Kim et al., 2026)	60.0	21.3	5.0	30.0
WorldDreamer (Wang et al., 2024)	66.3	26.7	9.0	35.3
Xiaomi-Robotics-1 (Xiaomi Robotics Team et al., 2026)	80.2	57.1	32.1	57.4
Harness VLA (GPT-5.5) (Zhang et al., 2026c)	92.0	61.0	13.8	57.1
Harness VLA (GPT-6 Astra) (Zhang et al., 2026c)	87.8	43.8	42.5	59.2
Iterative GPT-6 Astra + Codex	**100.0**	**87.5**	**75.0**	**88.0**

Table 4: Effect of refinement budget on LIBERO-Pro. The turn budget specifies the maximum number of write–execute–observe–revise iterations available to GPT-6. Each suite contains 10 tasks evaluated over three seeds, giving 30 episodes per suite and 120 episodes per budget.

Turn Budget	LIBERO-10-Swap	Object-Swap	Spatial-Swap	Goal-Swap	All
1	7/30 (23%)	16/30 (53%)	17/30 (57%)	14/30 (47%)	54/120 (45%)
2	16/30 (53%)	29/30 (97%)	27/30 (90%)	22/30 (73%)	94/120 (78%)
4	23/30 (77%)	30/30 (100%)	29/30 (97%)	24/30 (80%)	106/120 (88%)
8	26/30 (87%)	30/30 (100%)	30/30 (100%)	27/30 (90%)	113/120 (94%)

motivate a closer study of the interaction budget and external task scaffolding within the refinement loop.

4 How Much Interaction and External Structure Does a Frontier Model Need?

We now focus on design choices within the iterative policy refinement loop. Specifically, we examine how the number of execution-and-revision attempts and the external task scaffolding affect policy synthesis with frontier models.

Does a stronger model still benefit from iterative refinement? We first examine whether multi-turn execution feedback remains useful as the underlying foundation model becomes stronger. We vary only the maximum number of refinement turns, while keeping GPT-6 Astra, the robot interface, and the execution environment fixed. In each turn, the model writes or revises its current policy, executes it in the environment, observes the resulting feedback, and uses this feedback to revise the policy for the next attempt. All experiments use GPT-6 Astra with high reasoning effort and the same minimal interface with RGB-D observations. Each execution is limited to 1,000 control steps (50 s at 20 Hz), and each session terminates upon success or after 30 minutes of wall-clock time.

Table 4 shows that stronger models do not make iterative refinement redundant. With a single execution attempt, GPT-6 achieves only 45% success overall. Allowing one additional refinement turn raises success to 78%, while four and eight turns further improve performance to 88% and 94%, respectively. The gains are especially large on the harder LIBERO-10-Swap split, where success increases from 23% with one turn to 87% with eight turns.

Different task families saturate at different rates. Object-Swap reaches 97% with two turns and 100% with four, whereas LIBERO-10-Swap and Goal-Swap continue to benefit from additional refinement. These results show that execution feedback remains valuable even with GPT-6: stronger models can often solve easier tasks in only a few attempts, but more difficult tasks still benefit from repeated refinement turns.

How much external structure is necessary? We next examine how external structure interacts with increasing model capability. Here, external structure refers to task-level skills, reusable abstractions, decomposition templates, and heuristic guidance supplied by the system, rather than the program structure that the model develops by itself through interaction. Prior executable-policy systems often introduce such structure to simplify policy synthesis and guide solution discovery. An

Table 5: Detailed RATs results on the overlapping LIBERO-Pro splits. We report success rates (%) for GPT-5.5 and GPT-6 Astra from our completed trials, together with the Gemini 3.1 Pro results reported in the original RATs paper.

LIBERO-Pro Split	GPT-5.5	Gemini 3.1 Pro	GPT-6 Astra
Object-S	20.8	61.0	25.0
Object-T	16.7	63.0	50.0
Goal-S	0.0	43.0	0.0

Table 6: Interaction efficiency under different levels of external structure. Statistics are computed over the same tasks and execution budget as Table 2. Each cell reports the per-task median with the mean in parentheses. Refinement turns are policy executions per task; wall-clock time is the full session, so capped failures contribute 30 minutes. Tokens are input plus output per session as reported by the Codex CLI; † tokens are available only for successful sessions (57 of 80), since the CLI emits no usage record for a session killed at the cap.

Model	Structure	Success	Refinement Turns	Tokens	Wall-Clock Time
GPT-5.6 (xhigh)	Minimal	57/80 (71.2%)	5.0 (9.0)	0.90M (1.58M)†	10.1 min (15.3)
GPT-6	Minimal	76/80 (95.0%)	5.0 (6.5)	0.49M (0.71M)‡	4.6 min (6.4)

important question is whether scaffolding designed for earlier models continues to provide the same benefit as the underlying foundation model becomes substantially stronger.

As a representative structured system, we study RATs, which organizes policy generation around reusable robot skills acquired from prior interaction. Table 5 reports the available RATs results on the LIBERO-Pro splits shared across different model settings. The GPT-5.5 and GPT-6 Astra numbers are obtained from our completed trials, while the Gemini 3.1 Pro results are taken from the original RATs evaluation. Because the task coverage and number of completed trials differ across these settings, these results should be interpreted descriptively rather than as a controlled model comparison.

Interestingly, the available results do not show a monotonic improvement as the underlying model becomes stronger. On Object-S, Object-T, and Goal-S, Gemini 3.1 Pro achieves 61.0%, 63.0%, and 43.0%, respectively, whereas the corresponding completed GPT-6 Astra trials obtain 25.0%, 50.0%, and 0.0%. GPT-5.5 similarly achieves 20.8%, 16.7%, and 0.0%. While these partial results are insufficient to establish a general ranking between models, they suggest that simply replacing the foundation model inside an existing structured system does not automatically translate stronger model capability into better robotic performance. In particular, scaffolding that is effective for one model may not transfer unchanged as the model's own reasoning and policy-generation capabilities improve.

5 Conclusion

We presented an empirical analysis of frontier models in robotic manipulation, covering execution-time decision-making and executable-policy synthesis. Our results show strong performance in representative systems under both paradigms and highlight the value of iterative execution and revision within the iterative execution and revision.

These observations motivate exploring robotics System 2 designs that rely less on externally supplied task heuristics and better leverage frontier models' own reasoning, planning, and feedback-driven revision. Future work should investigate how this shift can support more effective task-level reasoning in robotic systems while allowing models to discover and refine solutions through interaction.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology and experimental design, refine hypotheses and conceptual framing, and assist with the interpretation of experimental results. Generative AI tools were also used to support literature search and summarization, scientific figure preparation, code development and debugging, and the drafting and editing of the manuscript.

We did not use generative AI tools to generate synthetic datasets, formulate mathematical claims, provide ingredients for mathematical proofs, or write proofs; these activities are not central to this work. All AI-assisted research decisions, analyses, code, figures, and manuscript content were reviewed by the authors. LLM-generated or LLM-assisted code used in experiments was executed and tested for correctness, and all reported experimental results were obtained from the described evaluation environments. The authors take responsibility for the final content of this work, including all text, claims, code, and artifacts produced with the aid of generative AI.

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 *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/ichter23a.pdf.

Anthropic. Claude Opus 5.5. https://www.anthropic.com/claude-opus-5-5, September 2026.

Shmuel Berman, Michael Ilie, Jia Deng, and Daniel Freeman. Claude plays robotics. https://www.anthropic.com/research/claude-plays-robotics, July 2026. Anthropic research blog, 9 July 2026.

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

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

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An Embodied Multimodal Language Model. In *Proceedings of the 40th International Conference on Machine*

Learning, volume 202 of *Proceedings of Machine Learning Research*, pp. 8469–8488. PMLR, 2023. URL https://proceedings.mlr.press/v202/driess23a.html.

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 "Jim" Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2603.22435v2.

Google Gemini Team. Gemini 3.1 Pro: A Smarter Model for Your Most Complex Tasks. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/, February 2026.

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

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

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

Chia-Yu Hung, Qi Sun, Pengfei Hong, Amir Zadeh, Chuan Li, U-Xuan Tan, Navonil Majumder, and Soujanya Poria. NORA: A small open-sourced generalist vision language action model for embodied tasks. *arXiv preprint arXiv:2504.19854*, 2025. URL https://arxiv.org/abs/2504.19854.

Mengzhao Jia, Yang Lin, Xixin Zhang, Zhihan Zhang, Xiaobai Liu, and Meng Jiang. Agent as policy for robotic manipulation. *arXiv preprint arXiv:2609.12541*, 2026. URL https://arxiv.org/abs/2609.12541.

Dongyoung Kim, Huiwon Jang, Myungkyu Koo, Suhyeok Jang, Taeyoung Kim, et al. RLDX-1 Technical Report. *arXiv preprint arXiv:2605.03269*, 2026. URL https://arxiv.org/abs/2605.03269.

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

Teyun Kwon, Norman Di Palo, and Edward Johns. Language models as zero-shot trajectory generators. *IEEE Robotics and Automation Letters*, 9(7):6728–6735, 2024. doi: 10.1109/LRA.2024.3410155.

Ruiying Li, Yunlang Zhou, Yuyao Zhu, Kylin Chen, Jingyuan Wang, Sukai Wang, Kongtao Hu, Minhui Yu, Bowen Jiang, Zhan Su, et al. RoboClaw: An agentic framework for scalable long-horizon robotic tasks. *arXiv preprint arXiv:2603.11558*, 2026. URL https://arxiv.org/abs/2603.11558.

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.

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*, volume 36, pp. 44776–44791, 2023. doi: 10.52202/075280-1939.

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.

Achu Menon, Sravanthi Machcha, Sabrina Zou, Tzu Kit Chan, and Jay Chooi. GPT-6 Astra on robotic manipulation. https://openai.robocurve.org/gpt-6-astra/, September 2026. Robocurve.

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://proceedings.iclr.cc/paper_files/paper/2026/hash/a05003fdb1e9562ab0c0a9719ea4de10-Abstract-Conference.html.

OpenAI. GPT-6 Astra: A New Generation of Intelligence. https://openai.com/index/gpt-6-astra/, September 2026.

Pantograph. Benchmarking frontier LLMs on robots. https://pantograph.com/journal/vlm-harness, September 2026. Research report.

Robocurve. Inspect Robots: The open-source evaluation framework for physical AI, 2026. URL https://github.com/robocurve/inspect-robots. Open-source software; MIT license. Accessed 26 September 2026.

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 *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 54919–54933. PMLR, 2025. URL https://proceedings.mlr.press/v267/shi25d.html.

Jiayi Su, Yixin Zheng, Mi Yan, Li Yi, Zhizheng Zhang, and He Wang. GPT 6 Astra as an embodied policy. Technical report and code, 2026. URL https://github.com/anonymous-report-421/GPT-as-Policy.

Xiaoquan Sun, Zetian Xu, Chen Cao, Zonghe Liu, Yihan Sun, Jingrui Pang, Ruijian Zhang, Zhen Yang, Kang Pang, Dingxin He, Mingqi Yuan, and Jiayu Chen. AtomVLA: Scalable post-training for robotic manipulation via predictive latent world models. *arXiv preprint arXiv:2603.08519*, 2026. URL https://arxiv.org/abs/2603.08519.

Xiaofeng Wang, Zheng Zhu, Guan Huang, Boyuan Wang, Xinze Chen, and Jiwen Lu. WorldDreamer: Towards general world models for video generation via predicting masked tokens. *arXiv preprint arXiv:2401.09985*, 2024. URL https://arxiv.org/abs/2401.09985.

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

Xiaomi Robotics Team, Jun Guo, Piaopiao Jin, Jason Li, Peiyan Li, Yingyan Li, Futeng Liu, Wanli Peng, Optimus Qin, Yifei Su, et al. Xiaomi-Robotics-1: Scaling vision-language-action models with over 100K hours of real-world trajectories. *arXiv preprint arXiv:2607.15330*, 2026. URL https://arxiv.org/abs/2607.15330.

Chao Yu, Yuanqing Wang, Zhen Guo, Hao Lin, Si Xu, Hongzhi Zang, Quanlu Zhang, Yongji Wu, Chunyang Zhu, Junhao Hu, Zixiao Huang, Mingjie Wei, Yuqing Xie, Ke Yang, Bo Dai, Zhexuan Xu, Jiakun Du, Xiangyuan Wang, Xu Fu, Letong Shi, Zhihao Liu, Kang Chen, Weilin

Liu, Gang Liu, Boxun Li, Jianlei Yang, Zhi Yang, Guohao Dai, and Yu Wang. RLinf: Flexible and efficient large-scale reinforcement learning via macro-to-micro flow transformation. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pp. 829–846. USENIX Association, 2026. URL https://www.usenix.org/conference/osdi26/presentation/yu-chao.

Junyi Zhang, Jiaxin Ge, Hanjun Yoo, Letian Fu, Zihan Yang, Yaowei Liu, Raj Saravanan, Shaofeng Yin, Justin Yu, Dantong Niu, Zirui Wang, Roei Herzig, Ken Goldberg, Yutong Bai, David M. Chan, Ion Stoica, Angjoo Kanazawa, Jiahui Lei, Haiwen Feng, and Trevor Darrell. Playful agentic robot learning. In *Conference on Robot Learning (CoRL)*, 2026a. URL https://arxiv.org/abs/2606.19419.

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*, 2026b. URL https://arxiv.org/abs/2609.24170.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Yi Nie, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Jiaji Rao, Zhengru Fang, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering frozen VLAs into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448v5*, 2026c. URL https://arxiv.org/abs/2607.08448v5.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025. URL https://arxiv.org/abs/2510.03827.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

A ITERATIVE REFINEMENT: IMPLEMENTATION AND WORKED EXAMPLES

This appendix documents the iterative policy refinement setup in enough detail to reproduce it: the complete interface the model is given, the per-cycle feedback it receives, two policies it produced, and the isolation measures that make the reported success rates attributable to the environment rather than to the model. All GPT-6 results in this paper, on both LIBERO and RoboCasa, use the RGB-D variant of the interface described here; no privileged object or fixture pose is available to the policy at any point.

A.1 WIRING

The agent is GPT-6 Astra running under the Codex command-line interface in a workspace-write sandbox. Each task is one session, started with a single prompt:

> *Read* `AGENTS.md` *and the contract documents, then solve the task.*

The workspace contains the task's natural-language instruction, the interface contract reproduced in Appendix A.2, an empty `policy.py`, and three command-line tools: one that resets the scene and records observations without moving the robot, one that executes the current `policy.py` for one episode, and one that extracts frames from a recorded episode video. No other tools, documents, code, demonstrations or prior solutions are present. Every execution creates a new attempt directory holding the simulator's verdict (`result.json`), the policy's own printed output, the initial and final robot observations, and RGB videos from both cameras. The agent reads these, revises `policy.py`, and executes again. A session ends at the first attempt reporting a clean success, or when the wall-clock cap expires.

Because the Codex sandbox denies the simulator a graphics context, policy executions are relayed through a queue directory to a runner process outside the sandbox. This is an implementation detail: the agent sees only the three tools and the attempt directories.

A.2 THE INTERFACE

The contract below is the entire control and perception interface. It is identical for LIBERO and RoboCasa, except that `drive` is present only when the robot has a mobile base (RoboCasa) and the camera names differ (`agentview`/`robot0_eye_in_hand` on LIBERO, `robot0_agentview_left`/`robot0_eye_in_hand` on RoboCasa). Explanatory prose has been shortened; every method, argument, unit and limit is as given to the model.

Listing 1: The policy contract (RGB-D variant), as given to the model.

```
# Policy contract

Your policy is a file policy.py defining run(context). It is executed in
its own sandboxed process; context is the entire interface you have, and
each of its methods is a request the simulator answers.

Permitted imports: numpy, scipy, math, typing, dataclasses, collections,
itertools, functools. Any other import is rejected before the policy runs.

## context.instruction
The task instruction, as a string.

## context.step(delta_position, delta_rotation, gripper) -> observation
Commands one control step at 20 Hz and returns the resulting observation.
* delta_position: (dx, dy, dz) in the robot base frame, each in [-1, 1],
  scaled by the controller to at most 0.05 m per step.
* delta_rotation: (drx, dry, drz) axis-angle in the base frame, each in
  [-1, 1], scaled to at most 0.5 rad per step.
* gripper: -1.0 opens, +1.0 closes.
The arm is an operational-space pose controller: each step is a delta
command relative to the current end-effector pose, not an absolute target.
context.steps_taken counts steps so far; context.max_steps is the budget.

## context.observe() -> dict
Robot state only: joint positions and velocities, robot0_eef_pos,
robot0_eef_quat, robot0_base_pos, robot0_gripper_qpos. No object or
fixture poses. Where things are in the world is for you to work out from
```

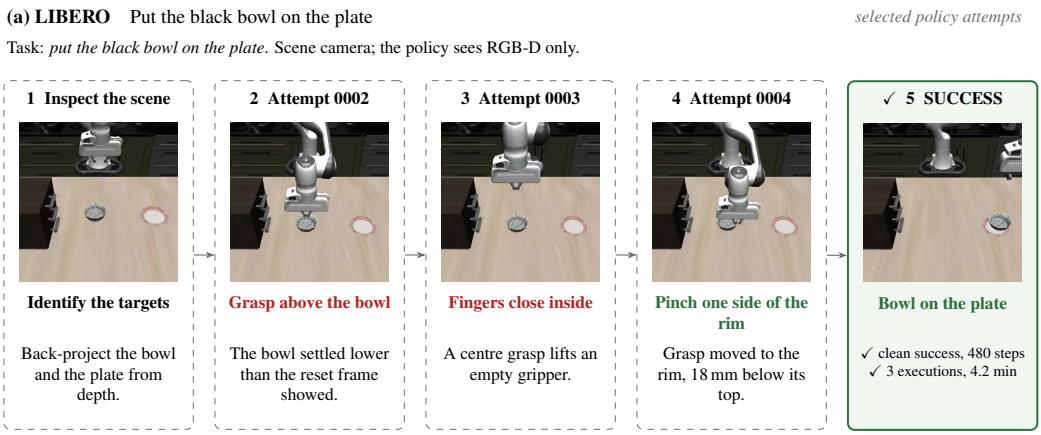

Figure 4: The bowl-on-plate session: the inspection frame, the two failed grasps, the revised grasp and the success, with the model's diagnosis under each. Red headlines are failures the model diagnosed from the returned video; the green card is the attempt the simulator graded as a clean success.

```
the cameras.

## context.camera(name) -> dict
The perception channel. Returns the current frame as a dict:
* rgb: (H, W, 3) uint8
* depth_m: (H, W) float, metres along the camera +Z axis
* intrinsics: 3x3 camera matrix
* camera_to_world: 4x4 homogeneous transform
To place a pixel in the world: back-project (u, v) with depth_m[v, u]
through intrinsics, then apply camera_to_world.

## context.drive(forward, side, yaw, torso) -> observation [mobile base only]
One step of base motion in [-1, 1] per axis; torso raises or lowers the
column. Counts against the same step budget as step. While driving, the
arm holds its desired pose and travels with the base.

## Printing
Whatever the policy prints goes to execution.log. Print the measurements
you rely on, because the log is your only way to check your own perception.

## What is not available
There is no context.scene() in this mode, and no privileged object or
fixture pose anywhere in the workspace. You cannot reach the simulator,
the task definition or the success function, and you must not try. Any
change to simulation state made outside context.step is recorded as
tampering and voids the attempt.
```

A.3 Example 1: a short LIBERO session

Task `KITCHEN_SCENE1_put_the_black_bowl_on_the_plate` (LIBERO-90), seed 0, initial state 0, 1000-step horizon, 15-minute cap. Table 7 lists the session, Figure 4 shows the attempts, and Listing 2 is the policy that succeeded, reproduced without modification.

attempt	kind	steps	outcome
0001	inspect scene	0	observations only
0002	policy	450	fail: grasp above the settled bowl
0003	policy	480	fail: fingers closed inside the bowl
0004	policy	480	**clean success**

Table 7: Bowl-on-plate session, GPT-6 Astra. Three policy executions, 4.2 minutes to the successful result.

The policy has three parts. `point` back-projects one pixel through the supplied intrinsics and extrinsics. `move` is a proportional end-effector loop, clipped to the contract's per-step limits, with

the gripper held pointing down; it is the model's own substitute for inverse kinematics and motion primitives, neither of which is provided. `run` takes one RGB-D frame, crops the point cloud to a box around the bowl, sets the grasp height from the cloud's top edge, back-projects two pixels for the bowl and plate, and executes descend, pinch, lift, carry, lower, release. The pixel coordinates, the crop box and the 48 mm lateral offset are specific to this scene and were arrived at across the three attempts.

Listing 2: The successful bowl-on-plate policy, verbatim (56 lines).

```
import numpy as np
from scipy.spatial.transform import Rotation


def point(camera, u, v):
   depth = float(camera['depth_m'][v, u])
   k = np.asarray(camera['intrinsics'])
   t = np.asarray(camera['camera_to_world'])
   p = t[:3, :3] @ (np.linalg.inv(k) @ np.array([u, v, 1]) * depth) + t[:3, 3]
   print('pixel', u, v, 'depth', depth, 'world', p.tolist())
   return p


def move(context, target, grip, count=65):
   target = np.asarray(target)
   desired = Rotation.from_euler('x', np.pi)
   for _ in range(count):
      state = context.observe()
      error = target - np.asarray(state['robot0_eef_pos'])
      rotation = (desired * Rotation.from_quat(state['robot0_eef_quat']).inv()).as_rotvec()
      context.step(np.clip(error * 7, -.5, .5).tolist(), np.clip(rotation * 1.5, -.4, .4).
          tolist(), grip)
   state = context.observe()
   print('target', target.tolist(), 'eef', state['robot0_eef_pos'], 'gripper', state['
       robot0_gripper_qpos'])


def run(context):
   for _ in range(30):
      context.step([0, 0, 0], [0, 0, 0], -1)
   camera = context.camera('agentview')
   height, width = camera['shape']
   vv, uu = np.mgrid[:height, :width]
   rays = np.stack([uu, vv, np.ones_like(uu)], axis=-1) @ np.linalg.inv(camera['intrinsics'
       ]).T
   cloud = (rays * np.asarray(camera['depth_m'])[..., None]) @ np.asarray(camera['
       camera_to_world'])[:3, :3].T + np.asarray(camera['camera_to_world'])[:3, 3]
   mask = (cloud[..., 0] > -.09) & (cloud[..., 0] < .07) & (cloud[..., 1] > -.09) & (cloud
       [..., 1] < .07) & (cloud[..., 2] > .915) & (cloud[..., 2] < 1.05)
   points = cloud[mask]
   low, high = points.min(axis=0), points.max(axis=0)
   bowl = (low + high) / 2
   print('settled bowl bounds', low.tolist(), high.tolist())
   grasp_z = high[2] - .018
   point(camera, 123, 160)
   plate = point(camera, 215, 167)
   print('bowl grasp center', bowl.tolist(), 'plate', plate.tolist())
   bowl[1] += .048
   plate[1] += .048
   move(context, [bowl[0], bowl[1], 1.16], -1)
   move(context, [bowl[0], bowl[1], grasp_z], -1)
   move(context, [bowl[0], bowl[1], grasp_z], 1, 30)
   move(context, [bowl[0], bowl[1], 1.20], 1)
   move(context, [plate[0], plate[1], 1.20], 1)
   place_z = plate[2] + grasp_z - .902 + .006
   move(context, [plate[0], plate[1], place_z], 1)
   move(context, [plate[0], plate[1], place_z], -1, 30)
   move(context, [plate[0], plate[1], 1.20], -1)
```

A.4 Example 2: a long RoboCasa composite session

Task `SteamInMicrowave` (RoboCasa composite, target split, layout 1, style 1, seed 0), mobile base unlocked, 2100-step horizon (the official RoboCasa horizon for this task), 60-minute cap. Instruction: *"Pick the squash from the sink and place it in the bowl. Then pick the bowl and place it in the microwave. Then close the microwave door and press the start button."* The task requires perception of two objects, a grasp in the sink, a rim grasp of a filled bowl, navigation to the opposite

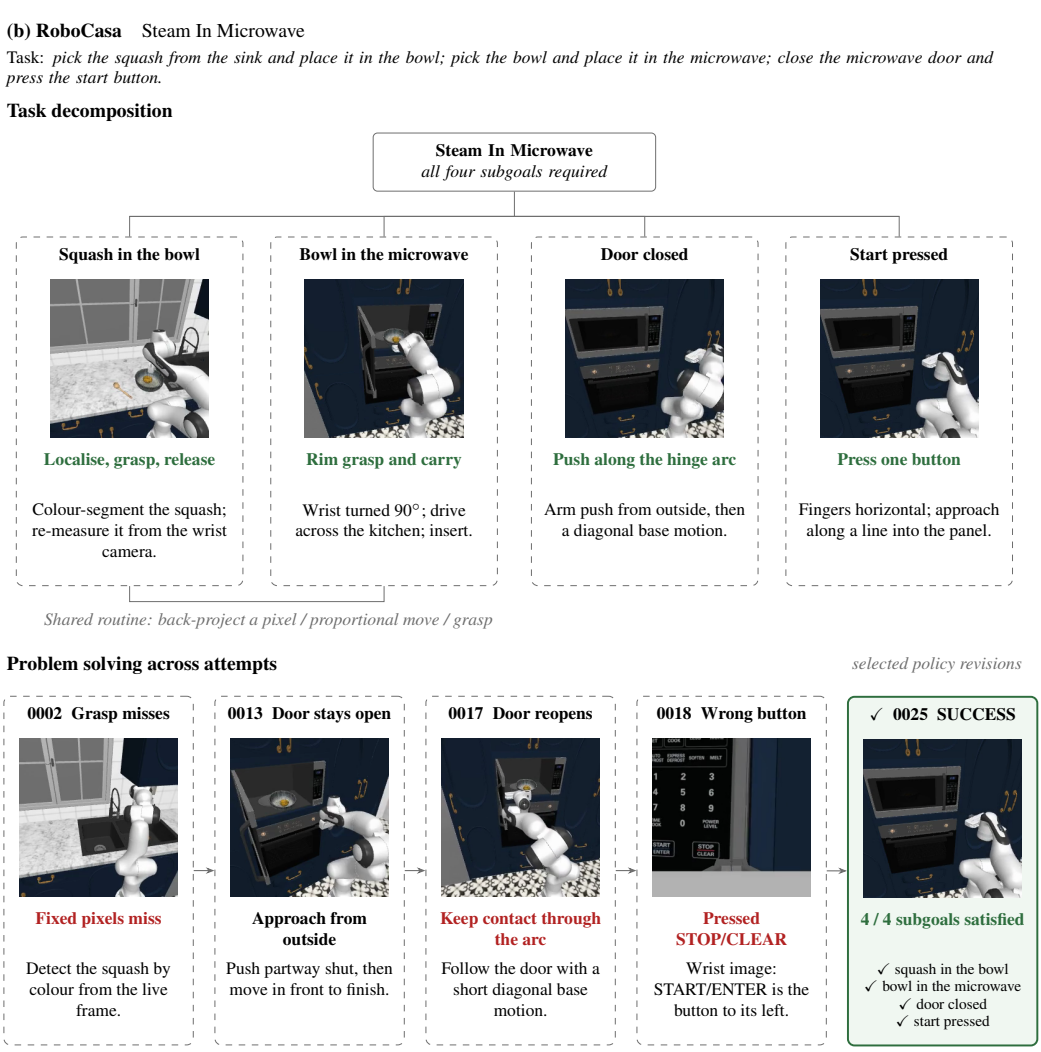

Figure 5: The SteamInMicrowave session: the four subgoals the instruction demands, each with the routine the model wrote for it, and five of the 24 policy revisions that led to the success. Red headlines are failures the model diagnosed from the returned video and wrist images; the green card is the attempt the simulator graded as a clean success.

wall, insertion into the microwave cavity, closing a hinged door, and pressing one button on a panel. The session used 24 policy executions and succeeded on the 24th, at 1807 of 2100 steps, after 60.0 minutes (Figures 5 and 6).

Course of the session. Table 8 groups the attempts into the three phases visible in the episode lengths. The model first built each sub-skill in isolation, saving each working stage to a separate file in the workspace (`policy_transfer.py`, `policy_survey.py`, `policy_carry.py`, `policy_door.py` and ten numbered full versions). It then integrated the chain and spent the second half of the session on the last two sub-goals.

The model's own account. The following are the model's messages between executions, quoted verbatim from the session transcript, chosen to show how it used the returned evidence.

> After attempt 0002: *"The first grasp missed: the robot's starting camera pose differed from the inspection frame, so the fixed pixel coordinates pointed beside the squash. I've changed the policy to detect the squash by color and measure its position from the current frame."*

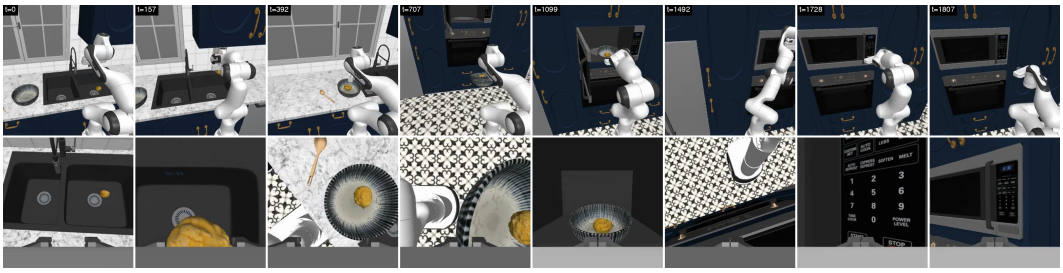

Figure 6: The successful SteamInMicrowave attempt (0025), scene camera (top) and wrist camera (bottom) at the same steps. Left to right: the reset scene with the squash in the sink and the bowl on the counter; the squash grasped from the sink; the squash released into the bowl; the filled bowl carried by its rim across the kitchen; the bowl inside the microwave cavity; the door pushed shut; the wrist camera over the control panel, from which the model identified START/ENTER after first pressing STOP/CLEAR; the final press. 1807 steps at 20 Hz.

attempts	phase	steps	what the executions established
0001	inspect	0	squash located in the sink, bowl on the counter
0002–0006	grasp	287–546	fixed pixels missed (camera pose differs from the inspection frame); colour detection and a wrist-camera remeasurement produced a stable squash grasp
0007–0011	survey	242–911	squash placed in the bowl; counter surveyed; microwave found on the opposite wall, above the oven, door open
0012–0017	integrate	1206–2097	filled bowl carried across the kitchen and inserted; door pushes from inside the arc reopened it
0018–0021	door and button	1665–1919	continuous outside push closed the door; wrist image showed the press had landed on STOP/CLEAR, target moved to START/ENTER; presses deflected off the panel
0022–0024	residual gap	1704–2100	depth snapshot showed the door still $\approx 7^\circ$ open; bowl pushed deeper; rotation stall at the step cap
0025	solve	1807	**clean success**

Table 8: SteamInMicrowave session, GPT-6 Astra, grouped by phase. Steps are the range of episode lengths within the group.

> After attempt 0013: *"The footage shows the bowl and squash resting inside the microwave, but the door stayed open. I've changed the closing motion to approach from outside the door, push it partway shut, then move in front to finish closing it."*
>
> After attempt 0017: *"The footage shows the gripper returning to the inside of the angled door and pushing it open again. I've changed the policy to maintain contact from the outside throughout closure, using a short diagonal base motion to follow the door's arc."*
>
> After attempt 0018: *"The continuous push closed the door. The close-up wrist image revealed the remaining error: I pressed 'STOP/CLEAR.' On this microwave, 'START/ENTER' is the adjacent button to its left. I'm correcting the target."*
>
> After attempt 0022: *"The depth snapshot revealed a remaining door gap: it still sits about 7° open, although it looks nearly flush in the wider view. The same snapshot confirms the start-button target. I'm extending the final closing push before repeating that press."*

The policy. The successful policy is 168 lines. Listing 3 abbreviates it: helper bodies that repeat the pattern of Listing 2 are reduced to their signatures, lists of probe pixels are elided, and comments are ours. The structure is otherwise unchanged. Three things are worth noting. The model wrote its own navigation layer (`base_y`, `base_xy`, `turn`) on top of the raw `drive` call. The door is never grasped; it is closed by pushing with the arm, then by driving the base along the door's arc. Every world coordinate in `run` was measured by back-projection during earlier attempts and then fixed.

Listing 3: The successful SteamInMicrowave policy, abbreviated from 168 lines.

```
import numpy as np
from scipy.spatial.transform import Rotation as R
```

```
def point(context, u, v): ... # back-project one pixel (as in Listing 2)
def move(context, target, grip=-1, quat=(0,1,0,0), n=100): ...
   # proportional end-effector loop, expressed in the base frame because the
   # base moves; exits early when within 8 mm and 0.06 rad
def hold(context, g, n=20): ... # n steps with the gripper command g
def drive(context, f=0, s=0, y=0, t=0, n=20): ... # n raw base steps

def yellow(context, cam='robot0_agentview_left'):
   # colour-threshold the squash, back-project the masked pixels, take the median
   c = context.camera(cam); rgb = np.asarray(c['rgb']).astype(float)
   mask = (rgb[:,:,0] > 95) & (rgb[:,:,1] > 65) & (rgb[:,:,2] < rgb[:,:,1] * .65) & ...
   ...
   return np.median(pts, axis=0)

def cloud(context):
   # dark points in a height band over the counter: the bowl
   ...
   return np.median(pp, axis=0)

def base_y(context, y): ... # drive sideways until base y matches
def turn(context, yaw): ... # rotate in place to a world heading
def base_xy(context, x, y): ... # drive to a world point, velocity in the base frame

def run(context):
   bowl = cloud(context)
   squash = yellow(context)
   drive(context, s=-.4, n=30); drive(context, f=.35, n=12)
   # --- squash: approach, re-measure from the wrist camera, grasp, lift
   move(context, [squash[0], squash[1], 1.05])
   squash = yellow(context, 'robot0_eye_in_hand')
   move(context, [squash[0], squash[1], squash[2] - .018])
   hold(context, 1)
   move(context, [squash[0], squash[1], 1.08], 1)
   # --- bowl: align the base beside it, release the squash into it
   base_y(context, bowl[1]); bowl = cloud(context)
   move(context, [bowl[0], bowl[1], 1.09], 1); hold(context, -1)
   # --- rim grasp of the filled bowl with the wrist turned 90 degrees
   rim = [bowl[0] + .135, bowl[1], 1.005]
   q = R.from_euler('z', np.pi/2) * R.from_quat([0,1,0,0])
   move(context, [rim[0], rim[1], 1.18], quat=q.as_quat())
   move(context, rim, quat=q.as_quat()); hold(context, 1, 25)
   move(context, [rim[0], rim[1], 1.2], 1, quat=q.as_quat())
   # --- cross the kitchen to the microwave and raise the torso
   drive(context, f=-.6, n=55); turn(context, 0)
   base_xy(context, 4.0, -2.92); base_xy(context, 4.7, -2.92)
   drive(context, t=1, n=50)
   # --- insert the bowl, release, push it deeper into the cavity
   qcarry = R.from_euler('z', -np.pi/2) * R.from_quat([0,1,0,0])
   move(context, [5.04, -2.91, 1.64], 1, quat=qcarry.as_quat())
   move(context, [5.39, -2.91, 1.64], 1, quat=qcarry.as_quat())
   move(context, [5.39, -2.91, 1.54], 1, quat=qcarry.as_quat(), n=70)
   hold(context, -1, 20)
   qpush = (R.from_euler('y', np.pi/2) * R.from_euler('z', np.pi)).as_quat()
   move(context, [5.10, -2.91, 1.51], 1, quat=qpush, n=60)
   move(context, [5.44, -2.91, 1.51], 1, quat=qpush, n=70)
   # --- close the door: reposition outside its arc, push with the arm,
   # then follow the arc with the base while keeping contact
   base_xy(context, 4.12, -2.91); base_xy(context, 4.12, -2.27); base_xy(context, 4.48,
       -2.27)
   move(context, [4.90, -2.39, 1.68], 1)
   move(context, [4.91, -2.82, 1.68], 1, n=80)
   drive(context, f=.4, s=-.4, n=22); drive(context, f=.4, s=-.2, n=18); drive(context, f
       =.5, n=22)
   drive(context, f=-.4, n=20)
   # --- press START/ENTER: fingers horizontal, approach along a line into the panel
   qpress = (R.from_euler('z', .44) * R.from_euler('y', np.pi/2) * R.from_euler('z', np.pi))
       .as_quat()
   base_xy(context, 4.68, -3.30)
   for x in (5.08, 5.24, 5.29, 5.33):
      move(context, [x, -3.23, 1.532], 1, quat=qpress, n=50)
   hold(context, 1, 10)
   move(context, [5.00, -3.32, 1.532], 1, quat=qpress, n=50)
```

A.5 What the sessions show

Three capabilities recur across the two sessions and, in our reading of the transcripts, across the successful sessions in the study.

Subtask identification. Neither session was given a decomposition, yet both proceeded by one. On SteamInMicrowave the model partitioned the instruction into the four subgoals of Figure 5, developed each in isolation (attempts 0002–0011 are all short episodes that end once the current subgoal is tested), saved each working stage to its own file, and only then composed the chain. When a late subgoal failed it returned to that subgoal alone rather than rewriting the whole policy, and it recognised that the squash and bowl placements shared one routine, which it reused. The same pattern appears on the short LIBERO task, where the first execution is an inspection rather than an attempt.

Spatial reasoning from raw sensing. With no object poses available, every spatial quantity the policies use was derived by the model from RGB-D geometry: back-projecting chosen pixels through the intrinsics and extrinsics; cropping a point cloud to a height band to find a bowl and taking its top edge as the grasp height; re-measuring an object from the wrist camera after the base has moved; converting world-frame targets to base-frame velocity commands for navigation; and reading a 7° residual door opening from a depth snapshot that the wide view showed as flush. Where a first estimate was wrong, the correction was geometric too: the 48 mm lateral offset on LIBERO and the choice of a rim rather than a centre grasp.

Diagnosis from returned evidence. Every revision in Table 8 follows from something the model found in the attempt directory: the empty gripper in the video, the door reopening in the footage, the wrist frame showing STOP/CLEAR under the fingers, the printed measurements in the execution log. The loop supplies no verdict finer than the binary success flag, so the model's own instrumentation, chiefly what it chose to print and which frames it chose to view, is what turned a failed attempt into a specific change.

A.6 Isolation and integrity

Each policy execution runs in a separate operating-system process under a sandbox profile that denies all file writes and all reads of the simulator source, the benchmark's task definitions and success functions, this repository, and every other experiment workspace (so that solutions to sibling tasks are unreachable). The policy's only channel to the simulator is the `context` proxy, which forwards the calls in Listing 1 over a pipe. The runner records the full simulator state after every `step` and verifies it before the next; any discrepancy is recorded as tampering in `result.json` and voids the attempt. The simulator sources are fingerprinted before and after each session. Every `result.json` carries the SHA-256 of the policy, the task definition and the initial-state file, together with the sandbox configuration that was in force, so an attempt's provenance can be audited after the fact. Across all sessions reported in this paper, no tampering was recorded and no source fingerprint changed.

A.7 Budgets and conventions

Control runs at 20 Hz on both benchmarks with the same operational-space limits (0.05 m and 0.5 rad per step). Cameras are 256×256. LIBERO attempts have a 1000-step horizon; RoboCasa composite attempts use the official per-task horizon. Wall-clock caps are per session: 15 or 30 minutes on LIBERO and 60 minutes on RoboCasa composite tasks, as stated with each result. A session ends at the first attempt reporting `clean_success`. LIBERO seed s uses initial state s of the task's initial-state file. RoboCasa target-split scenes redraw object placement at every build, so a composite result is a solve of one draw of the scene, not of the task under all placements.

B Additional Html

We also provide an additional webpage for all our experiment results and intermediate thinking process. You can check the html for more results.