

ASPIRE: Agentic /Skills Discovery for Robotics

Runyu Lu^{1 2 * †}, Yubo Wu^{1 3 *}, Ethan Kou^{1 4 *}, Haotian Lin^{1 5 *}, Letian Fu^{1 4}, Wenli Xiao^{1 5}, Ajay Mandlekar¹, Yinzhen Xu¹, Guanya Shi⁵, Ken Goldberg⁴, Ang Chen², Mosharaf Chowdhury², Jan Kautz¹, Yuke Zhu^{1 †}, Linxi “Jim” Fan^{1 †}, Guanzhi Wang^{1 †}

¹ NVIDIA, ² UMich, ³ UIUC, ⁴ UC Berkeley, ⁵ CMU

* Equal contribution, † Project leads

<https://research.nvidia.com/labs/gear/aspire/>

Abstract:

Traditional robot programming is notoriously challenging: it requires orchestrating multimodal perception, managing complex physical contact dynamics, and handling diverse environment configurations and execution failures. We introduce ASPIRE (Agentic Skill Programming through Iterative Robot Exploration), a continual learning system for robotics that autonomously writes and refines robot control programs in a code-as-policy paradigm while compounding experience into a reusable skill library. ASPIRE enables automated discovery of reusable skills that persist across multiple tasks, simulation and real-world settings, and different embodiments. Rather than relying on fixed, human-engineered pipelines, ASPIRE operates in an open-ended learning loop, consisting of three key components: (1) a closed-loop robot execution engine that exposes fine-grained multimodal traces (e.g., perception overlays, grasp candidates, motion trajectories, and collision feedback), enabling the agent to autonomously diagnose failures, synthesize repairs, and validate outcomes; (2) a continually expanding skill library that distills validated fixes into reusable, transferable robotic knowledge; and (3) an evolutionary search procedure that generates diverse programs, systematically debugging them to explore beyond single-trajectory refinement. As ASPIRE encounters more tasks, its growing skill library enables increasingly rapid adaptation. Consequently, ASPIRE surpasses prior methods by up to 77% on manipulation tasks under perturbation, 72% on Robosuite’s bimanual handover task, and up to 32% on long-horizon household tasks. The accumulated skill library further enables strong zero-shot generalization: on representative unseen long-horizon tasks, ASPIRE achieves 31% success, substantially outperforming the 4% success rate of prior methods despite their heavy reliance on test-time reasoning and retries. Performance scales with the capabilities of frontier models. With Claude Opus 5.5, ASPIRE nearly saturates these benchmarks, achieving 99% average success across all 80 LIBERO-Pro tasks and 100% success on representative Robosuite and BEHAVIOR-1K tasks. Beyond simulation, skills discovered in simulation reduce real-world robot programming effort across embodiments and robot APIs, while skills acquired through online interaction with physical robots enable ASPIRE to solve a Rubik’s Cube with progressively improving efficiency.

1. Introduction

Recent progress in software engineering agents demonstrates that language models can autonomously inspect execution traces, localize failures, revise implementations, and improve through repeated interaction with execution environments [Anthropic, 2025, OpenAI, 2025, anomalyco, 2025, Wang et al., 2025, Yang et al., 2024]. In robotics, this paradigm has inspired *code-as-policy* systems that compose perception modules, planning APIs, and control primitives into executable robot programs [Liang et al., 2023, Singh et al., 2023, Ahn et al., 2022, Huang et al., 2023, Mu et al., 2024, Fu et al., 2026]. Because robot behaviors are represented explicitly as programs, they can in principle be inspected, edited, debugged, and refined through interaction feedback.

However, existing robotic coding agents remain fundamentally limited by naive execution environments that provide only coarse task-level feedback. Debugging robot programs is intrinsically challenging because failures can arise from many interacting components, including multimodal

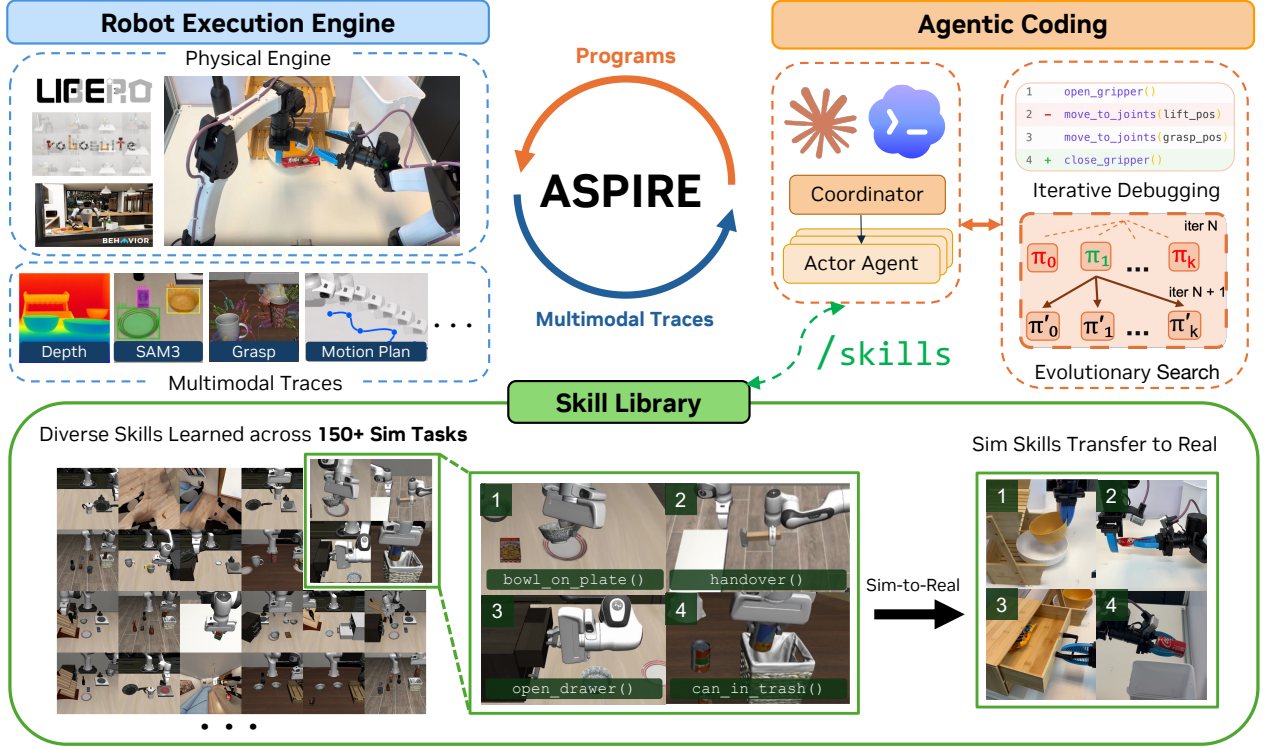


Figure 1: **ASPIRE system overview.** A *coordinator* spawns an *actor agent* (coding agent) per task, enabling parallel learning across tasks. Each actor refines and validates robot programs through *iterative debugging* with the *robot execution engine*, which exposes per-primitive multimodal traces for failure attribution and repair. *Evolutionary search* samples diverse candidate programs ($\pi_0, \dots, \pi_k$), sends each through the engine, and conditions the next generation ($\pi'_0, \dots, \pi'_k$) on surviving programs and residual failure traces. The *coordinator* writes validated repairs into a shared *skill library*, which future actors retrieve as in-context guidance; skills discovered in simulation can also be adapted as cross-embodiment guidance for real-robot programming.

perception, motion planning, grasp generation, contact dynamics, and long-horizon task coordination. A failed rollout may indicate that the task did not succeed, but not whether the root cause is incorrect perception, an unstable grasp, a planning error, or a downstream recovery failure. Without fine-grained diagnostic traces, agents have limited ability to determine what evidence to inspect, how to localize failures, or what repair strategy to attempt.

Moreover, existing systems do not accumulate experience across tasks. Once a task is completed, discovered fixes and recovery strategies are discarded rather than consolidated into reusable skills. As a result, the agent solving its hundredth task is effectively no more experienced than the agent solving its first.

Human robotics engineers take a fundamentally different approach. When a robot program fails, they replay executions, inspect perception outputs and motion trajectories, localize the failing subsystem, revise the implementation, and internalize reusable recovery strategies. Over time, debugging experience compounds into transferable knowledge, including grasp recovery heuristics, navigation strategies, prompting recipes, and procedural fixes that generalize across tasks. This accumulation of reusable knowledge is a key reason why human robot programmers become progressively more effective over time.

In this work, we introduce ASPIRE, a self-improving continual learning robotic system that autonomously writes and refines robot control programs in a code-as-policy paradigm while accumulating experience into a reusable skill library. Rather than operating within a fixed perception-plan-execute pipeline, ASPIRE runs in an open-ended learning loop, in which the agent self-determines how to inspect execution traces, diagnose failures, synthesize repairs, validate corrected behaviors, and consolidate successful recovery patterns into persistent skills that transfer across tasks.

An overview of ASPIRE is shown in Fig. 1. ASPIRE is built from three components. First, a **closed-loop robot execution engine** replaces coarse rollout-level feedback with per-primitive execution traces. For each perception, planning, grasping, and control call, the execution engine records the observations, inputs, outputs, and visual evidence if possible. These rich multimodal traces allow the agent to selectively inspect salient primitive logs, progressively localize failures, and validate repairs through re-execution. Second, ASPIRE maintains a growing **skill library** that distills validated fixes into reusable, transferable robotic knowledge retrievable as in-context guidance for future tasks. Third, ASPIRE employs an **evolutionary search procedure** that generates diverse programs, exploring beyond single-trajectory self-improvement through iterative debugging and parallel refinement. Together, these components establish a self-improving robotic system whose performance scales with experience: the more tasks ASPIRE sees, the larger its skill library grows, and the more it transfers to novel tasks, longer-horizon behaviors, and real-world robotic settings where similar failure-recovery patterns emerge under different embodiments.

Empirically, ASPIRE improves success over prior coding agents [Fu et al., 2026] by up to 77%, and with frontier models it nearly saturates standard simulation benchmarks: with Claude Opus 5.5, it reaches 99% average success across all 80 LIBERO-Pro [Zhou et al., 2025] tasks and 100% success on representative Robosuite [Zhu et al., 2020] and BEHAVIOR-1K [Li et al., 2024] tasks, and its success scales with the capability of the underlying frontier model. A skill library accumulated on LIBERO-90 [Liu et al., 2023] transfers zero-shot to LIBERO-Pro Long (31% versus 4% for prior coding agents), and stronger models discover new strategies, such as knocking a book flat to push it under a shelf. On a real bimanual robot, simulation-discovered skills reduce debugging effort, and skills learned online let ASPIRE solve a Rubik’s Cube.

2. Method

ASPIRE consists of three components that together form an open-ended learning loop: (1) a *robot execution engine* (§2.1) that exposes per-primitive multimodal traces for failure attribution and executes agent-written repairs for closed-loop validation; (2) a *skill library* (§2.2) that accumulates validated repairs as reusable skills; and (3) an *evolutionary search* procedure (§2.3) that broadens exploration beyond single-trajectory repair. A coordinator also orders tasks as a curriculum and retries low-scoring tasks with the final library (§2.4). As ASPIRE encounters more tasks, its skill library grows, allowing future tasks to inherit accumulated repairs and reusable strategies.

As shown in Fig. 1, ASPIRE adopts a coordinator-actor architecture. A central coordinator manages the shared skill library and dispatches actors (coding agents) to individual tasks, while each actor writes, executes, diagnoses, and repairs robot programs within the robot execution engine. Actors do not exchange full chat histories or raw rollout trajectories. Instead, transferable experience is distilled into the skill library, allowing each actor’s context window to remain focused on the task specification, current program, and structured execution traces associated with the current failure.

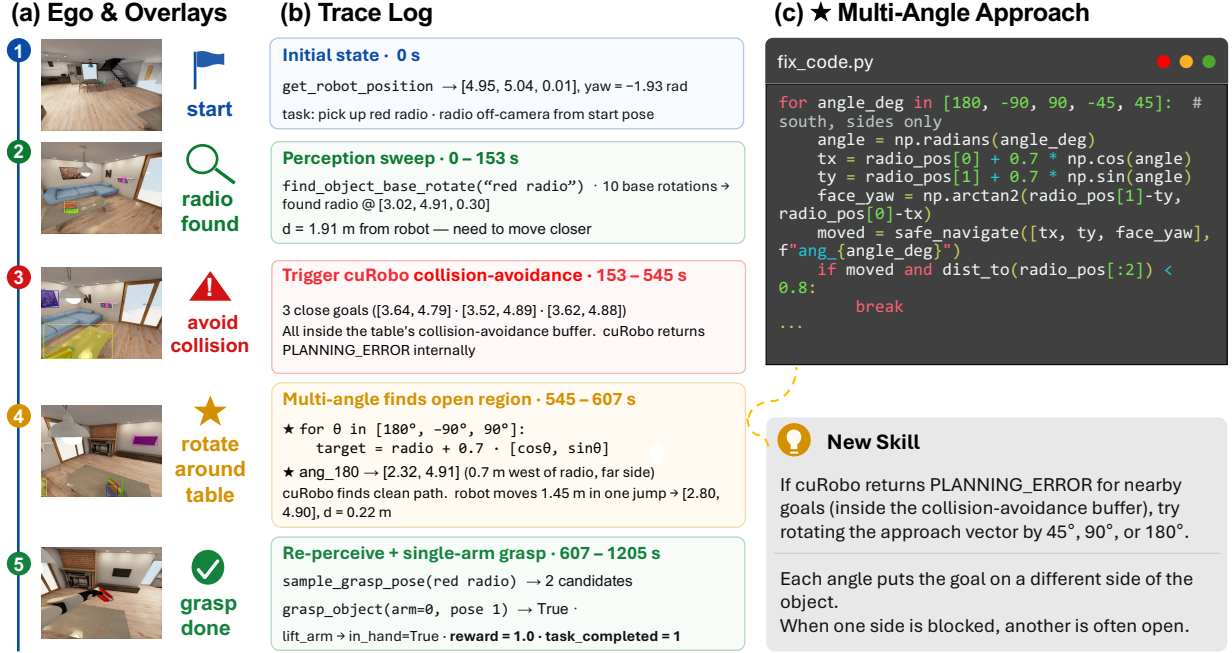


Figure 2: **Robot execution engine.** Trace-guided debugging on a BEHAVIOR-1K NAVIGATE-AND-PICK-UP-RADIO task. (a) Ego-view keyframes and overlays show the robot locating the radio but failing to approach it. (b) The primitive trace localizes the failure to repeated `PLANNING_ERRORS`: candidate navigation goals fall inside the table’s collision-avoidance buffer. (c) The agent patches the program with a multi-angle approach routine, re-perceives the radio from a reachable side, and completes the grasp. The validated repair is admitted as a reusable *Multi-Angle Approach* skill.

2.1. Robot Execution Engine

Embodied coding agents need execution evidence to debug robot programs. Prior methods expose this evidence through fixed, human-designed interfaces, typically as manually curated scene-level summaries or a set of pre-defined observations. This creates a trade-off: too little evidence can hide the failing primitive, while too much raw visual context can distract the agent from the causal chain that produced the failure.

ASPIRE turns this fixed feedback channel into an open-ended debugging environment. The robot execution engine records per-primitive multimodal traces for perception, planning, and control calls, exposes the trace to the actor, and executes actor-written repairs for closed-loop validation. We call this loop of execution, diagnosis, and repair *iterative debugging*. For each primitive call, the trace stores the invoked API, inputs and outputs, return status, and relevant multimodal evidence such as RGB keyframes, overlays, grasp candidates, object poses, and motion-planning results. The actor does not receive full video frames; the engine keeps frames immediately before and after each primitive call together with the corresponding overlays and return values, so the actor can focus on evidence around calls implicated by the failure.

Fig. 2 zooms in on one BEHAVIOR-1K NAVIGATE-AND-PICK-UP-RADIO debugging episode. The ego-view keyframes show that the robot finds the radio, repeatedly fails to approach it, and then succeeds after changing its approach direction. The primitive trace localizes the failure: perception succeeds and returns a radio pose, but repeated `navigate_to_pose` calls return `PLANNING_ERROR`. By checking the navigation return values and associated logs, the actor finds that the generated



Figure 3: **Skill library.** ASPIRE stores validated, agent-discovered repair knowledge as reusable in-context skills rather than a fixed set of human-written primitives. Top: representative entries show learned skills about localization disambiguation, motion-primitive construction, and navigation recovery. Middle: the library grows across heterogeneous categories, including localization, navigation, motion primitives, object-level grasping, scene understanding, and debugging workflows. Bottom: selected skills discovered in simulation are used as in-context guidance for real-robot programming, providing evidence that skills can transfer across embodiments. Additional skill examples and prompts are shown in Appendix A.

navigation target lies too close to the table boundary, within about 20 cm of the table edge. This triggers collision avoidance and causes the planner to fail. Thus, the failure is not due to detecting or grasping the radio, but rather to the infeasibility of the target pose under the table’s collision constraints.

The repair follows directly from this diagnosis. Rather than changing the perception prompt or grasp primitive, the actor writes a multi-angle approach routine that samples alternative navigation targets around the radio and executes an approach direction that clears the collision buffer. The execution engine exposes the evidence, validates the patched program, and enables the actor to analyze the resulting logs, form a hypothesis, and make a targeted repair decision.

2.2. Skill Library

Program failures recur across tasks, but the reusable knowledge is rarely an entire task program. ASPIRE’s skill library stores heterogeneous repair knowledge: localization heuristics, perception prompts, grasping constraints, navigation recovery strategies, motion primitives, scene-understanding routines, and debugging workflows. We do not prescribe this taxonomy in advance. Skills are induced from validated repairs: the actor diagnoses a failure from execution traces, patches the program, and validates the fix on the debug seeds.

Each skill is stored as compact in-context guidance, including the failure signature, when-to-apply condition, repair strategy, and, when useful, a representative code sketch. Fig. 3 shows representative entries and the resulting library breadth. For the radio task in Fig. 2, the admitted skill is a navigation recovery pattern rather than a complete radio-pickup program: when planner errors recur near an obstacle boundary because sampled target poses fall inside the collision buffer, sample alternative approach directions around the object before retrying perception and grasping. This representation lets future actors reuse validated repairs instead of rediscovering them through test-time reasoning, supports zero-shot transfer to harder simulated tasks, and provides the mechanism for selected simulation-discovered skills to generalize across embodiments and transfer to real robots.

Skill admission has two checks. Each actor reports structured findings: the failure mode, the validated fix, and any potentially transferable repair pattern as a proposed skill. A fresh verifier actor then receives the proposed skill and attempts the source task independently on a subset of seeds, recording which entries are sound, defective, or untested. The coordinator verifies compliance with the allowed API policy and promotes only reusable entries whose success the verifier reproduced. Appendix G.1 provides the prompts for actor reporting and skill admission.

2.3. Evolutionary Search

Iterative debugging alone can collapse into local repair loops, where the actor repeatedly patches the same failed strategy instead of exploring fundamentally different ways to solve the task. ASPIRE uses evolutionary search to broaden exploration of executable robot programs, encouraging diverse repair hypotheses and task strategies.

In each round, based on the skill library, the actor proposes a population of K candidate programs conditioned on the top-performing previous programs and failure traces from previous evaluations. Each candidate is executed in the robot execution engine, producing task outcomes together with new diagnostic traces. The next round is then conditioned on the best-performing programs together with their remaining failure modes, allowing the search to explore distinct strategies rather than repeatedly refining the same solution.

The search target is the robot program itself. Candidates are selected through closed-loop

Algorithm 1 Evolutionary search over programs

Require: task τ , program P^0 , debug/held-out seeds $S_{\text{dbg}}, S_{\text{eval}}$, skill library $\mathcal{L}$, agent M , budget (K, T) , threshold θ

Notation: $\text{EXECUTE}(P, S) = (r, Z)$ returns score r and trace bundle Z .

```

1:  $(r^*, Z^0) \leftarrow \text{EXECUTE}(P^0, S_{\text{dbg}})$ ;  $P^* \leftarrow P^0$ 
2:  $\mathcal{H} \leftarrow \{(P^0, r^*, Z^0)\}$ 
3: for  $i = 1, \dots, T$  do
4:    $\{P_i^k\}_{k=1}^K \leftarrow \text{PROPOSERPAIRS}(M, \tau, \text{Top3}(\mathcal{H}), \mathcal{L}, \mathcal{H})$ 
5:   for  $k = 1, \dots, K$  do
6:      $(r_i^k, Z_i^k) \leftarrow \text{EXECUTE}(P_i^k, S_{\text{dbg}})$ 
7:   end for
8:    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(P_i^k, r_i^k, Z_i^k)\}_{k=1}^K$ ;  $k^* \leftarrow \arg \max_k r_i^k$ 
9:   if  $r_i^{k^*} > r^*$  then
10:     $(P^*, r^*) \leftarrow (P_i^{k^*}, r_i^{k^*})$ 
11:   end if
12:   if  $r^* \geq \theta$  then
13:     break
14:   end if
15: end for
16:  $\mathcal{G} \leftarrow \text{EXTRACTVALIDATEDPATTERNS}(\mathcal{H}, P^*)$  ▷ debug history only
17:  $(r_{\text{eval}}, \cdot) \leftarrow \text{EXECUTE}(P^*, S_{\text{eval}})$  ▷ run once, after search
18: return  $(P^*, r_{\text{eval}}, \mathcal{G})$ 

```

execution on the debug seeds, and validated repairs are admitted into the skill library after search concludes, provided they generalize across environment variations and tasks. Search terminates when the best candidate reaches a success threshold θ on the debug seeds or when the budget of T rounds is exhausted; skill patterns are extracted from the debugging history, and the final program is then evaluated once on held-out evaluation seeds. Algorithm 1 summarizes the evolutionary search procedure. See Appendix G for the pipeline files.

2.4. Curriculum and Backward Transfer

The coordinator also acts as a curriculum agent. Based on the current skill library, it chooses the next task to attempt from the list. It can attempt to fill a library gap, give an existing skill a chance to work on a new task, or utilize a recent finding by picking tasks in the same suite. Because tasks are attempted sequentially, those attempted early have access to only a small library, while later tasks benefit from every skill learned before them. An early low score may reflect missing skills rather than task difficulty. Therefore, once all tasks are finished and the skill library is complete, the coordinator allows one separate rerun of low-scoring tasks that used an earlier version of the library. This backward skill transfer allows skills to be applied to tasks that came before them.

3. Experiments

3.1. Experimental Setup

For the main results, we use Claude Code [Anthropic, 2025] with Claude Opus 4.6 [Anthropic, 2026c] and a 1M-token context window as the coding agent on all simulation benchmarks. The agent writes executable Python robot programs in ASPIRE’s unified code-as-policy harness, which exposes the same robot programming APIs for perception, geometry, and motion planning across LIBERO-Pro, Robosuite, and BEHAVIOR-1K. The environment and API set are fixed across all simulation experiments. The model ablation (§3.8) compares twelve models: Claude Opus 4.6, Opus 5 [Anthropic, 2026d], Opus 5.5 [Anthropic, 2026e], Sonnet 5.5 [Anthropic, 2026f], Haiku 5.5 [Anthropic, 2026b], and Fable 5.1 [Anthropic, 2026a]; GPT-6 Astra [OpenAI, 2026c], GPT-6 Sol and GPT-6 Luna [OpenAI, 2026d], and GPT-6.1 Sol [OpenAI, 2026b]; and the open-weight Kimi K3 [Kimi Team, 2026] and

Qwen 3.8 Max [Qwen Team, 2026]. Claude and open-weight models run through Claude Code and GPT models through Codex CLI [OpenAI, 2025]. The case study (§3.9) uses Claude Opus 4.6, Opus 5, Opus 5.5, and GPT-6 Astra.

For the real-robot skill transfer study (§3.6), we use OpenAI Codex [OpenAI, 2025] with GPT-5.5 [OpenAI, 2026a] in reasoning-xhigh mode on a bimanual YAM manipulation station. We select three skills compiled by ASPIRE in Franka-based simulation: soda-can pickup, bowl-on-plate placement, and drawer push/pull. We provide these as in-context guidance to the real-robot coding agent. The real-robot setting uses a different embodiment and API from simulation, but exposes ASPIRE multimodal execution traces so the agent can autonomously run, inspect, and debug programs without task-specific human guidance during iterative debugging. We compare runs with and without the corresponding simulation-discovered skill, measuring tokens to first success and held-out real-robot success rate. For online skill learning on the real robot (§3.7), GPT-6 Astra in Codex solves a Rubik’s Cube on the same bimanual YAM station.

3.2. Benchmarks and Baselines

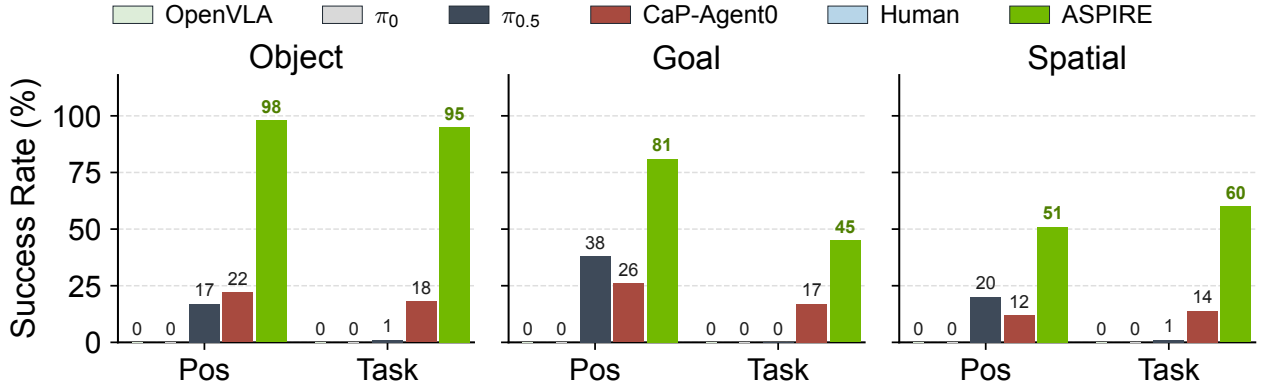
We evaluate ASPIRE on three benchmark families: LIBERO-Pro [Zhou et al., 2025] for short-horizon robustness under object, goal, and spatial perturbations; Robosuite [Zhu et al., 2020] for contact-rich single- and dual-arm manipulation; and BEHAVIOR-1K [Li et al., 2024] for long-horizon household mobile manipulation on NAVIGATE-AND-PICK-UP-SODA-CAN and NAVIGATE-AND-PICK-UP-RADIO. Our primary coding-agent baseline is CaP-Agent0 [Fu et al., 2026], which uses visual differencing, a predefined skill library, and per-episode test-time retries. We also compare with end-to-end VLA policies, including OpenVLA [Kim et al., 2024], π_0 [Black et al., 2025], and $\pi_{0.5}$ [Physical Intelligence et al., 2025]. For zero-shot transfer, we evaluate held-out LIBERO-Pro Long tasks using the skill library accumulated on LIBERO-90 [Liu et al., 2023].

3.3. Evaluation Protocol

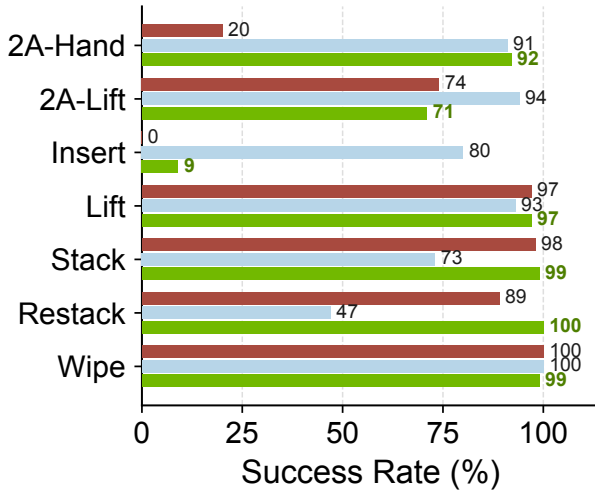
Across all benchmarks, an environment seed fixes each task instance, including object poses, distractors, and initial robot/object states. We use disjoint debug and evaluation seeds: ASPIRE learns on a small debug split, then reports success on larger held-out evaluation seeds with one generated program per LIBERO-Pro/Robosuite task, while CaP-Agent0 regenerates a separate program for each seed with test-time reasoning and retries. For BEHAVIOR-1K evaluation, ASPIRE uses incremental block execution, generating each next code block from the current multimodal trace.

3.4. Main Evaluation Results

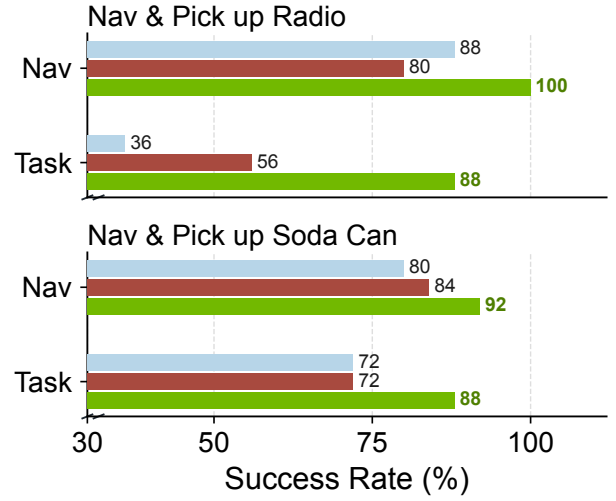
Figure 4 summarizes the main evaluation. “Human” means the programs are written by human experts. On LIBERO-Pro, ASPIRE improves over all the baselines on all three suites: averaging position and task perturbations, ASPIRE gains 77% on Object, 41.5% on Goal, and 42.5% on Spatial over the strongest baseline in each suite. $\pi_{0.5}$ is stronger than OpenVLA and π_0 on some position perturbations, but remains far below ASPIRE and largely collapses under task paraphrases. On Robosuite, ASPIRE preserves near-saturated performance on easier contact-rich tasks and substantially improves bimanual handover, from 20% to 92%. On BEHAVIOR-1K, ASPIRE outperforms both human and CaP-Agent0 on navigation and task success, with the largest task-level gain over CaP-Agent0 on NAVIGATE-AND-PICK-UP-RADIO, from 56% to 88%.



(a) **LIBERO-Pro** [Zhou et al., 2025]: macro-averaged success over 10 tasks $\times$ 50 held-out environment seeds per suite/perturbation. For each task, ASPIRE learns and collects skills on seeds 51–65 and evaluates one generated program on seeds 1–50.



(b) **Robosuite** [Zhu et al., 2020]: success over 100 held-out trials per task. ASPIRE learns on seeds 101–125 and evaluates one generated program per task on seeds 1–100; 2A tasks are bimanual.

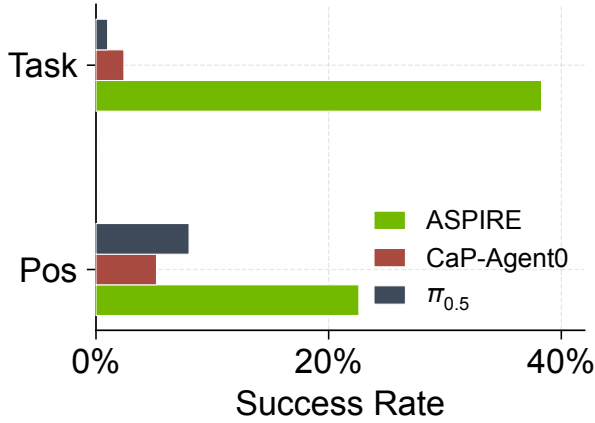


(c) **BEHAVIOR-1K** [Li et al., 2024]: long-horizon mobile manipulation on two household tasks. ASPIRE learns on seeds 26–35 and evaluates on seeds 1–25 with incremental block execution; Navigation and Task success are reported separately.

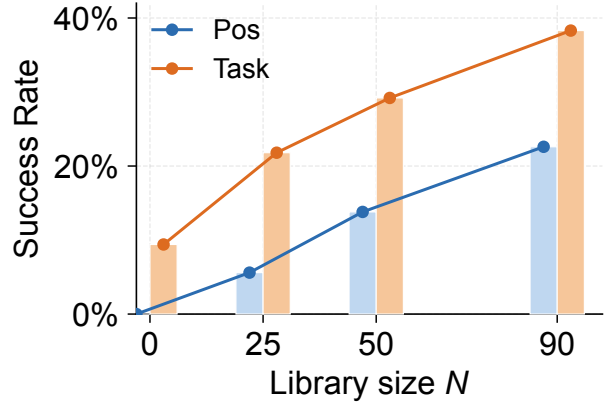
Figure 4: **ASPIRE improves over prior coding agents and end-to-end VLAs across three benchmark families.** (a) Short-horizon manipulation on LIBERO-Pro; (b) contact-rich manipulation on Robosuite; (c) long-horizon mobile manipulation on BEHAVIOR-1K. ASPIRE evaluates one generated program per task across held-out seeds, while CaP-Agent0 regenerates a separate program per seed with test-time reasoning and retries. ASPIRE improves performance across all benchmarks, with several results surpassing programs written by human experts. Detailed per-task results are in Appendices B and E.1.

3.5. Zero-Shot Transfer to Unseen Tasks

We evaluate whether repair skills accumulated on LIBERO-90 transfer zero-shot to held-out LIBERO-Pro Long tasks. ASPIRE builds library snapshots from $N \in \{0, 25, 50, 90\}$ source tasks, where $N=0$ is an empty-library setting and $N=90$ is the full suite. For each held-out task, ASPIRE generates one program and evaluates it across seeds with no additional debugging, retries, or task-specific library updates.



(a) Full $N=90$ library zero-shot results versus baselines.



(b) Zero-shot success improves as skill-library size increases.

Figure 5: **Cross-task zero-shot transfer on LIBERO-Pro Long.** Skills accumulated on LIBERO-90 improve zero-shot performance on held-out long-horizon tasks. Figure (a) compares the full $N=90$ library with baselines. Figure (b) shows success under position and task perturbations as the size of the skill library increases. All success rates are macro-averaged over 10 tasks per axis. Per-task results are in Appendix C.

Figure 5(a) compares the full $N=90$ skill library with prior baselines on the two LIBERO-Pro Long transfer axes: ASPIRE reaches 23% success on position perturbations and 38% on task perturbations, outperforming CaP-Agent0 and $\pi_{0.5}$ on both axes. Figure 5(b) shows that success increases consistently as the size of the skill library grows, indicating that validated repairs from short-horizon tasks provide reusable robotic knowledge for longer-horizon compositions.

3.6. Real-Robot Skill Transfer Across Embodiments

We evaluate whether skills learned in simulation can reduce debugging effort on a real robot with a different embodiment. The transfer is not a direct policy deployment: the real robot uses its own perception, calibration, and control stack, and the coding agent must still adapt programs through real-world execution feedback. The question is whether retrieving simulation-discovered skills provides useful in-context guidance that reduces the amount of real-world debugging needed to reach a successful program.

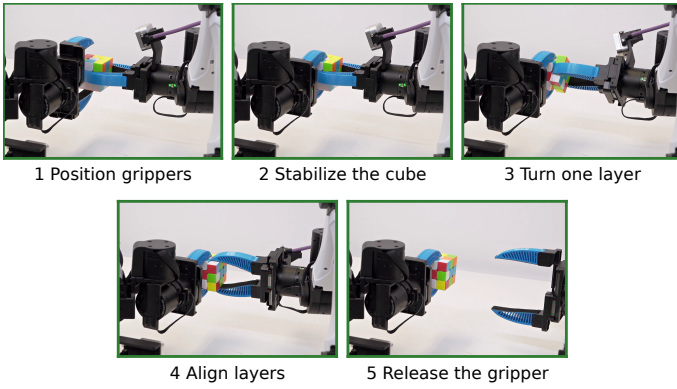
Table 1 shows that transferred skills consistently reduce debugging cost, while their effect on final success is task-dependent. Bowl placement succeeds in both settings but uses fewer tokens with skill retrieval; soda-can lifting improves from 65% to 95% while reducing total tokens by nearly an order of magnitude; and drawer manipulation reaches 55% success with skill guidance, while the no-skill baseline exhausts a larger token budget without producing a successful evaluation program. These results indicate that selected failure-derived skills can guide real-robot program synthesis across embodiment and API changes rather than merely memorizing simulator-specific code.

3.7. Online Skill Learning on a Real Robot

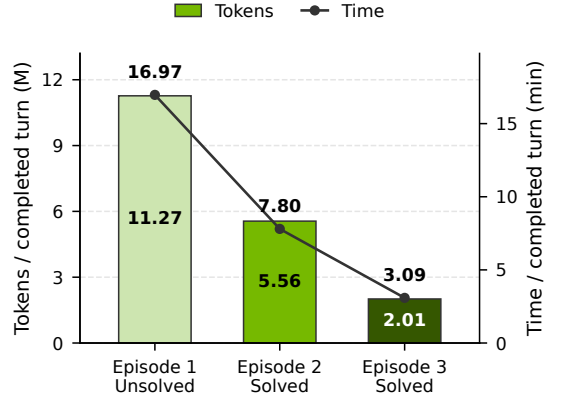
Online adaptation is an important step toward robustness in deployment. Given a fixed set of tools, ASPIRE runs on a real robot to self-improve its tool-use strategy: the coding agent acts as a **system-2** that calls executable perception and action tools and records what it learns as text skills. We show this on **Solve-Rubik’s-Cube** over three sequential real-world episodes in one GPT-6 Astra

Table 1: **Real-robot cross-embodiment skill transfer.** For each task, we compare real-robot debugging with and without retrieving a corresponding simulation-discovered skill from ASPIRE’s library. Token counts are measured until the first successful real-robot program, or until the debugging budget is exhausted for runs that do not reach success. Success rate reports evaluation trials for the generated programs.

Task	Output Tokens (M) ↓		Total Tokens (M) ↓		Success Rate ↑	
	w/o Skills	w/ Skills	w/o Skills	w/ Skills	w/o Skills	w/ Skills
Put bowl on plate	0.05	0.04	8.65	5.11	20/20	20/20
Lift soda can	0.18	0.03	61.94	6.58	13/20	19/20
Open/push drawer	1.33	0.36	334.92	81.67	0/20	11/20



(a) One face turn.



(b) Efficiency across episodes.

Figure 6: **Solving a Rubik’s Cube on the real robot.** (a) GPT-6 Astra positions both grippers, holds the cube, turns one layer, aligns the layers, and releases. (b) Time and tokens per completed turn fall across episodes as context and skills accumulate.

session, on a bimanual YAM platform with soft grippers (Appendix D). Initially, the agent is given a randomly scrambled cube and human-provided task guidance. The agent needs to identify the color formation of all faces purely from visual observation and may call a symbolic solver to provide a move sequence, then it must realize the solution and verify the physical manipulations. During the first few attempts, the agent explores strategies for a successful face turn and summarizes its experience as text-based skills. The accumulated skills cover two-layer support, single-layer isolation, face-axis alignment, collision-avoiding handovers, and state-aware slip recovery—learning through context refinement rather than model-weight updates.

The first attempt remains unsolved after 50 min 54 s; subsequent episodes solve the cube in 39 min and 77 min 10 s. Their logs report 3, 5, and 25 completed face turns, respectively, counting a half-turn as one move and excluding whole-cube reorientation. Figure 6(a) shows one face turn, and Figure 6(b) shows end-to-end time per completed turn decreasing from 16 min 58 s to 7 min 48 s and 3 min 05 s. Across the first and final episodes, tokens per turn fall from 11.27 M to 2.01 M, and estimated model cost, at OpenAI API prices [OpenAI, 2026e], from \$16.85 to \$2.89. These full-episode amortized metrics include failed attempts, recovery, and context compaction. The results demonstrate increasingly efficient execution alongside accumulated skills.

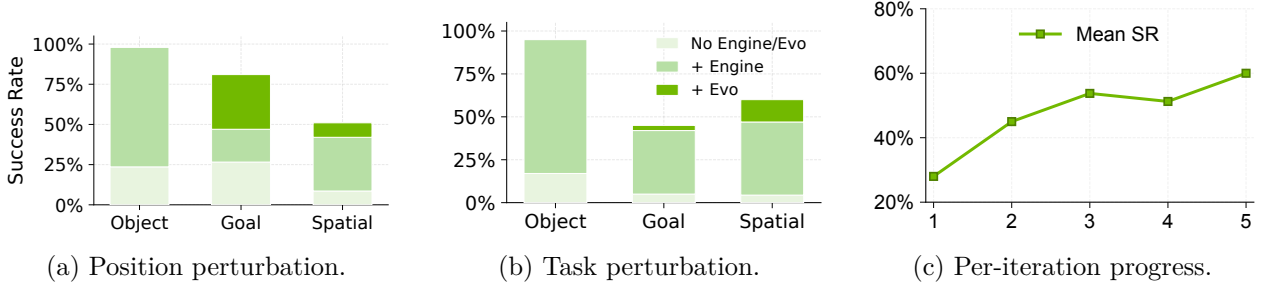


Figure 7: **Robot execution engine and evolutionary search ablations on LIBERO-Pro.**

(a), (b) Stacked bars under position and task perturbation: the base system without the robot execution engine or evolutionary search, the gain from adding the engine, and the additional gain from evolutionary search. (c) Average progress over low-performing tasks across evolutionary-search iterations. Per-task results are in Appendix E.

3.8. Ablation Studies

Robot execution engine and evolutionary search. We ablate two key components of ASPIRE on LIBERO-Pro: the robot execution engine, which exposes dense execution traces and validates task-level repairs, and evolutionary search, which explores additional repair candidates after iterative debugging with the engine. Figure 7(a) and (b) decompose success by component. The robot execution engine provides the largest average improvement, raising macro-average success from 14% to 62%; evolutionary search further improves the remaining hard tasks, reaching 72% with both components.

Evolutionary search iterations. Figure 7(c) tracks performance on low-performing tasks as the evolutionary-search budget increases. Success improves steadily in the first few rounds, suggesting that sampling multiple repair hypotheses quickly recovers alternatives missed by iterative debugging. The curve continues to increase more gradually afterward, indicating that additional rounds still help on residual hard cases, but with diminishing returns.

Backward Skill Transfer. We compare the initial and final held-out results of the five tasks selected for reruns with the full skill library in the Claude Opus 5 LIBERO-Pro Long campaign. Both evaluations use the same 50 seeded held-out layouts per task; the post-rerun column evaluates revised task programs after further debugging with the complete skill library.

Table 2: **Backward skill transfer on LIBERO-Pro Long.** Held-out Claude Opus 5 success before and after reruns. Each task has 50 held-out seeds. The five rerun tasks were selected for low initial success; task IDs give the perturbation (**pos**: position, **task**: task) and task index.

Task	Initial	Post-rerun	Δ (pp)
pos/0 Soup + sauce $\rightarrow$ basket	37/50 (74%)	47/50 (94%)	+20
pos/3 Bowl $\rightarrow$ drawer, close	0/50 (0%)	26/50 (52%)	+52
pos/9 Mug $\rightarrow$ microwave, close	0/50 (0%)	0/50 (0%)	0
task/3 Bottle $\rightarrow$ drawer, close	0/50 (0%)	36/50 (72%)	+72
task/6 Red mug $\rightarrow$ plate, pudding right	39/50 (78%)	43/50 (86%)	+8
All 20 tasks	746/1000 (74.6%)	822/1000 (82.2%)	+7.6

Four of the five selected tasks improve, raising suite-wide success from 74.6% to 82.2% (+7.6pp).

The largest gains occur on the two drawer tasks, while `pos/9` remains at 0%. The complete skill library lets the agent unlock success on two tasks and improve reliability on two more.

Model ablation. We ablate the LLM that drives the coding agent while holding the environment, robot programming API, seed splits, and execution protocol fixed. Figure 8 first compares twelve models on LIBERO-Pro Goal under the position perturbation, using the ASPIRE iterative debugging pipeline with an initially empty skill library. Success ranges from 15% with GPT-6 Luna to 99% with Claude Opus 5.5. Of the two open-weight models, Kimi K3 reaches 78% and Qwen 3.8 Max 56%, on par with Claude Opus 4.6 (56%), and Claude Haiku 5.5 reaches 68% for only \$40. Newer models are far stronger: GPT-6.1 Sol reaches 88% (up from 59% with GPT-6 Sol), Opus 5 91%, GPT-6 Astra 94%, Claude Fable 5.1 97%, and Claude Sonnet 5.5 99%. Stronger models are not necessarily more expensive: Sonnet 5.5 and Opus 5.5 reach 99% for an estimated \$121 and \$183 over the 10 tasks, against \$597 for Opus 5 at 91%. ASPIRE therefore does not depend on a particular model, and it improves as the underlying model improves.

Figure 9 and Tables 3 and 4 then compare Claude Opus 5, Claude Opus 5.5, and GPT-6 Astra, models from two families, against Opus 4.6. On LIBERO-Pro Object, Goal, and Spatial, iterative debugging alone reaches 96% average success with Opus 5, 98% with Astra, and 98.9% with Opus 5.5, against 62% for Opus 4.6 without evolutionary search and 72% with it (Figures 4 and 7). The largest gap between Opus 5 and Astra is LIBERO-Pro Goal under task perturbation, where Astra reaches 97% and Opus 5 87%. On the long-horizon LIBERO-Pro Long suite, iterative debugging alone reaches 97.9% with Opus 5.5 and 87.9% with Astra; Opus 5 reaches 74.6%, and 82.2% after targeted reruns (Table 2). On Robosuite, Opus 5 and Astra raise the seven-task mean from 81% to 99%, and Opus 5.5 to 100% (Appendix E.5), with the gain concentrated on the three hardest tasks in Table 4: `nut_assembly` rises from 9% to 98% and 99%, and `two_arm_lift` from 71% to 96% and 100%. On BEHAVIOR-1K, Opus 5.5 reaches 100% navigation and task success on both tasks, and Opus 5 and Astra 92–100% (Table 3). With these stronger models, iterative debugging alone already solves most tasks: on Object, Goal, and Spatial, evolutionary search changes success by at most 2% for Opus 5, GPT-6 Astra, and Opus 5.5, compared with up to 34% for Opus 4.6 (Goal under the position perturbation; Figure 9). On Long, it adds up to 7% for Opus 5 and 4% for Astra, and nothing for Opus 5.5. It mainly repairs individual low-success tasks, such as placing the bowl in the cabinet’s top drawer under the position perturbation, which rises from 82% to 98% for Astra, and putting the mug in the microwave under the task perturbation, which rises from 0% to 42% for Opus 5 (Tables 13–16).

3.9. Case Study of the Coding Agents

Figure 10 compares how four models and a human solve two tasks. The rows show one episode each, so they illustrate strategies, not matched trials.

Book under the shelf. The gap under the shelf is too low for a hand holding an upright book. The human holds the book by its top edge, tilts it, and slides it in. No model finds this move. Opus 4.6 carries the book above the cabinet and stalls (0%). Opus 5 rolls the book flat but inserts it into the compartment above the target (2%). The two models that succeed let go of the book: Opus 5.5 knocks it flat without grasping it and pushes it under (68%), and Astra rolls it flat, drops it, and pushes it under (22%).

Push the plate. In the original layout the plate lies in the open, and the human pushes it straight to the stove. The position perturbation puts a bowl in front of the plate and a box in its path. Opus 4.6 pushes without moving the plate (0%), and Opus 5 loses its grip and drags the bowl instead

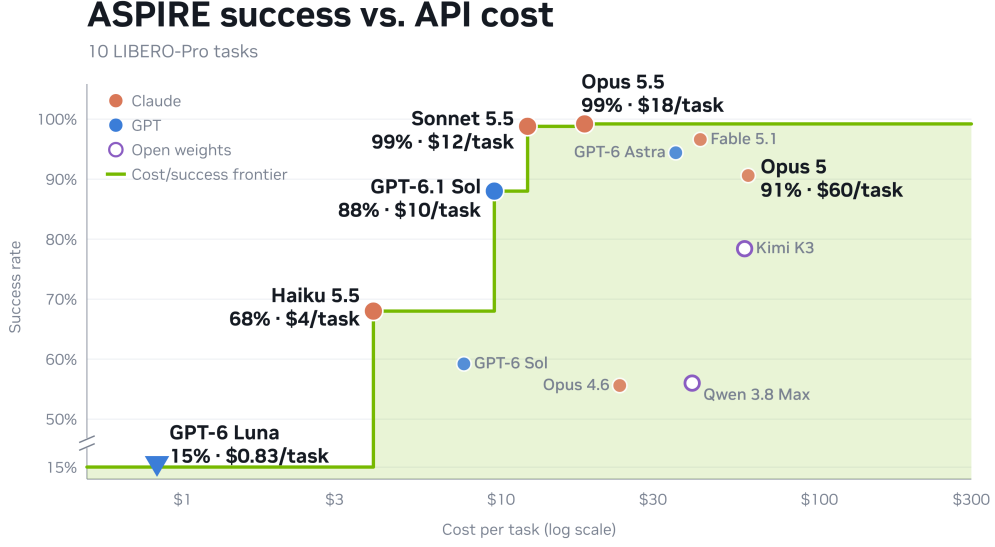


Figure 8: **Model ablation: success versus cost.** Held-out success versus estimated API cost per task (list prices, log scale) on the 10 LIBERO-Pro Goal tasks under the position perturbation, using ASPIRE iterative debugging without evolutionary search and with an initially empty skill library. Colors mark the model family; hollow markers are the open-weight Kimi K3 and Qwen 3.8 Max. The green line is the cost/success frontier: the models that no cheaper model outperforms, labeled with success and cost per task. The success axis is broken below 50% to show GPT-6 Luna (15%). All values are in Table 18; per-task results for Claude Opus 5, Opus 5.5, and GPT-6 Astra are in Appendix E.6.

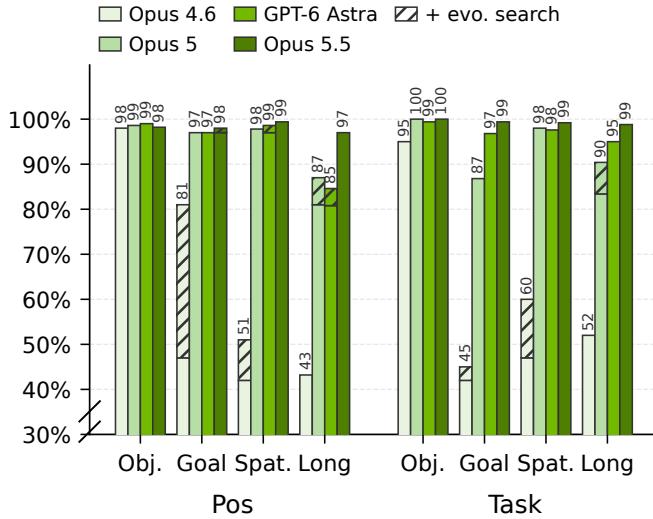


Table 3: **BEHAVIOR-1K.** Navigation and task success on held-out seeds.

Task		Claude Opus			GPT-6
		4.6	5	5.5	
SODA-CAN	Nav.	92%	100%	100%	96%
	Task	88%	100%	100%	96%
RADIO	Nav.	100%	100%	100%	96%
	Task	88%	92%	100%	96%

Table 4: **Robosuite.** Success over 100 held-out trials on the three hardest tasks (all seven: Appendix E.5).

Task	Claude Opus			GPT-6
	4.6	5	5.5	
two_arm_handover	92%	99%	100%	95%
two_arm_lift	71%	96%	100%	100%
nut_assembly	9%	98%	100%	99%

Figure 9: **Model ablation on LIBERO-Pro.** Held-out success per suite and perturbation (10 tasks $\times$ 50 seeds); hatched segments are the gain from evolutionary search (Opus 4.6 on Long without it). Unlike the empty-library runs in Figure 8, these runs carry the skill library across tasks within a suite, so their Goal values under position perturbation differ. Tables 3 and 4 (right) give the same comparison on BEHAVIOR-1K and Robosuite.

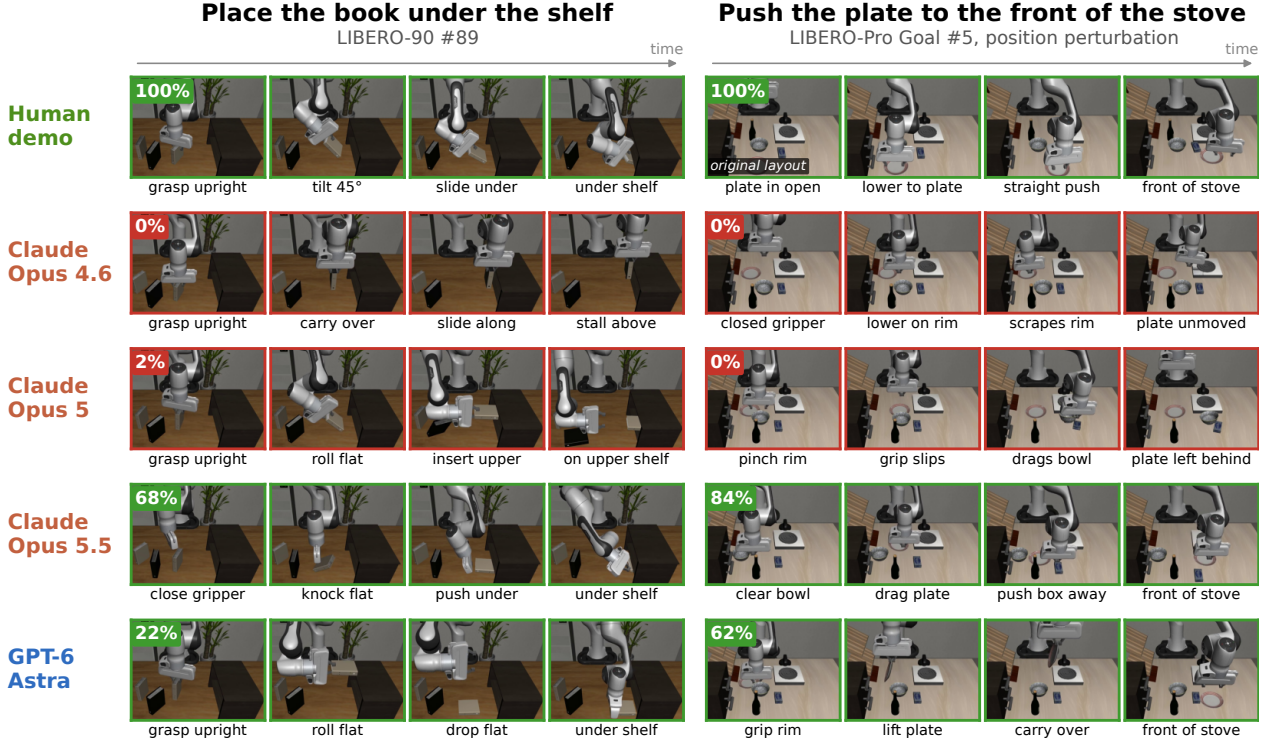


Figure 10: **Case study: five solvers, two tasks.** Each row shows one episode: the human teleoperation demonstration from the original LIBERO dataset, and one held-out evaluation episode of the ASPIRE program written by each model. Frames run left to right in time and are labelled with the step they show. Percentages are held-out success over 50 seeds (Table 26); the demonstrations all succeed. Green and red borders mark the shown episode’s success or failure. The plate demonstration uses the original, unblocked layout. Two more tasks are in Appendix F.

(0%). Opus 5.5 pushes the bowl and the box out of the way and then pushes the plate (84%). Astra does not push at all: it lifts the plate over the clutter and sets it down in front of the stove (62%).

Weaker models keep the default grasp-and-move strategy and fail at the robot’s limits. Stronger models change strategy and reach the goal in different ways. Because ASPIRE validates each program by executing it, the same loop turns a stronger model into a more capable robot program.

4. Related Work

Agentic robot control. Robot control has been studied through end-to-end vision-language-action policies and executable programs that compose perception, planning, and control APIs [Ahn et al., 2022, Brohan et al., 2023, Octo Model Team et al., 2024, Kim et al., 2024, Black et al., 2025, Mu et al., 2024, Physical Intelligence et al., 2025, NVIDIA et al., 2025, Sun et al., 2026, Li et al., 2026, Fu et al., 2026, Chen et al., 2026]. Software-engineering agents provide a related write-execute-debug loop for code [Jimenez et al., 2024, Wang et al., 2025, Yang et al., 2024, Wang et al., 2024a, Chen et al., 2024, Anthropic, 2025, OpenAI, 2025, anomalyco, 2025]. ASPIRE builds on the executable-program paradigm, but focuses on persistent embodied improvement: the agent freely selects primitive-level multimodal traces, writes repairs, validates, and preserves successful experiences across tasks.

Self-improving agents and skill libraries. LLM agents have been improved through open-ended memory, skill libraries, and self-evolving skill repositories [Wang et al., 2023, Tziafas and Kasaei, 2024, Peng et al., 2024, Fu et al., 2025, Zhang et al., 2026, Xie et al., 2026]. Other systems use LLMs to generate rewards, curricula, environments, reset, or search candidates [Ma et al., 2024a, Yu et al., 2023, Xie et al., 2024, Liang et al., 2024, Ma et al., 2024b, Du et al., 2023, Wang et al., 2024b, Romera-Paredes et al., 2024, Novikov et al., 2025, Guo et al., 2025, 2026, Cao et al., 2026, Xiao et al., 2026]. Unlike success-only memories, textual reflections, or reward functions for downstream policy training, ASPIRE stores validated repair knowledge extracted from attributed embodied failures. Its skills are open-ended in both content and admission: they emerge from the agent’s own debugging experience, span heterogeneous categories (§2.2), and are reused as in-context guidance for future program repair. Evolutionary search (§2.3) further broadens the space of executable program repairs before reusable patterns are admitted to the library.

5. Limitations

ASPIRE has several important limitations. First, while selected simulation-discovered skills support autonomous real-robot debugging on a different embodiment, our system is not yet a fully autonomous real-world lifelong learner. Unlike simulation, where success checking and scene resets are cheap and programmatic, real-world deployment still requires robust success detection, safe reset, safety monitoring, and calibration maintenance. Future work should close this evaluation-and-reset loop to scale sim-to-real skill transfer across broader real-world task suites. Second, ASPIRE relies on a frozen LLM to interpret multimodal traces, write program repairs, and propose evolutionary-search candidates, so its performance depends on that model. Weaker models such as GPT-6 Luna, Claude Opus 4.6, Qwen 3.8 Max, and GPT-6 Sol reach only 15–59% on LIBERO-Pro Goal under the position perturbation, and the best open-weight model, Kimi K3, reaches 78% (Figure 8); our best results use proprietary frontier models. Third, ASPIRE writes programs using a predefined API of perception, planning, and control primitives. This API makes debugging tractable and safe, but also bounds the behaviors the agent can express: if a task requires sensing, control, or interaction capabilities outside the exposed primitives, the agent must either approximate them inefficiently or rely on humans to extend the API. Future systems should study how agents can safely propose, validate, and incorporate new robot primitives. Fourth, the skill library currently prioritizes validated reusable repairs but does not fully solve long-term memory management. As the library grows, some entries may become stale, overly specific, redundant, or misleading for a new task, which can hurt individual tasks in zero-shot transfer. More robust retrieval, pruning, ranking, and re-validation mechanisms are needed to keep the library useful at scale. Finally, the debug and evolutionary-search loop is compute-intensive, consuming many LLM calls and simulator or robot rollouts per task; scaling to very large task suites will require either cheaper LLM inference, more sample-efficient search, or stronger mechanisms for reusing prior repairs.

6. Conclusion

We present ASPIRE, a continual learning robotic system that autonomously writes and refines robot control programs, guided by multimodal execution traces and evolutionary search over candidate programs, while compounding validated repairs into a reusable skill library. Across diverse benchmarks, ASPIRE outperforms existing VLA and coding-agent baselines, transfers zero-shot to unseen long-horizon tasks, and improves with the underlying model, reaching 99% average success across all 80 LIBERO-Pro tasks and 100% success on representative Robosuite and BEHAVIOR-1K tasks with Claude Opus 5.5. On three real-robot tasks, skills discovered in simulation reduce real-robot programming token cost despite a different embodiment and robot API, and skills learned

online let GPT-6 Astra solve a Rubik’s Cube on a real bimanual robot.

Acknowledgments

We thank Nadun Ranawaka, Jimmy Wu, Matin Furutan, Abhi Maddukuri, Yulu Gan, Matin Nikoui, and Yuqi Xie for their help with open-source support, real robot infrastructure, advice and guidance on the paper, website release, and experimental equipment. We are also grateful to the members of NVIDIA GEAR, UMich SymbioticLab, UsesysLab, UC Berkeley AUTOLab, CMU LeCAR Lab for their kind support.

References

- Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Goper, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022.
- anomalyco. Opencode: The open source ai coding agent, 2025. URL <https://github.com/anomalyco/opencode>.
- Anthropic. Claude code: Best practices for agentic coding, 2025. URL <https://www.anthropic.com/engineering/claude-code-best-practices>.
- Anthropic. Claude Fable, 2026a. URL <https://www.anthropic.com/claude/fable>.
- Anthropic. Introducing Claude Haiku 5.5, 2026b. URL <https://www.anthropic.com/claude-haiku-5-5>.
- Anthropic. Introducing Claude Opus 4.6, 2026c. URL <https://www.anthropic.com/news/claude-opus-4-6>.
- Anthropic. Introducing Claude Opus 5, 2026d. URL <https://www.anthropic.com/news/claude-opus-5>.
- Anthropic. Introducing Claude Opus 5.5, 2026e. URL <https://www.anthropic.com/claude-opus-5-5>.
- Anthropic. Introducing Claude Sonnet 5.5, 2026f. URL <https://www.anthropic.com/claude-sonnet-5-5>.
- Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael 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, 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.
- Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning (CoRL)*, 2023.
- Shiyi Cao, Ziming Mao, Joseph Gonzalez, and Ion Stoica. K-search: LLM kernel generation via co-evolving intrinsic world model. *arXiv preprint arXiv:2602.19128*, 2026.

- Kaiyuan Chen, Shuangyu Xie, Letian Fu, Justin Yu, William Pacini, Sandeep Bajamahal, Hudson Kim, Jaimyn Drake, Daehwa Kim, Haoru Xue, Jonathan Francis, Christian Juetten, Peter Schaldenbrand, Muhammet Yunus Seker, Ruwan Wickramarachchi, Uksang Yoo, Guanzhi Wang, Adithyavairavan Murali, Balakumar Sundaralingam, S. Shankar Sastry, Spencer Huang, Yuke Zhu, Linxi Fan, and Ken Goldberg. GaP: A graph-as-policy multi-agent self-learning harness for variational automation tasks. In *Conference on Robot Learning (CoRL)*, 2026.
- Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations (ICLR)*, 2024.
- Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. In *International Conference on Machine Learning (ICML)*, 2023.
- Honghao Fu, Junlong Ren, Qi Chai, Deheng Ye, Yujun Cai, and Hao Wang. Vistawise: Building cost-effective agent with cross-modal knowledge graph for minecraft. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.
- Max Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Fan. Cap-x: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026.
- Ping Guo, Chenyu Zhu, Siyuan Chen, Fei Liu, Xi Lin, Zhichao Lu, and Qingfu Zhang. Evoengineer: Mastering automated CUDA kernel code evolution with large language models. *arXiv preprint arXiv:2510.03760*, 2025.
- Ping Guo, Chao Li, Yinglan Feng, and Chaoning Zhang. Code evolution for control: Synthesizing policies via LLM-driven evolutionary search. *arXiv preprint arXiv:2601.06845*, 2026.
- Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3D value maps for robotic manipulation with language models. In *Conference on Robot Learning (CoRL)*, 2023.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.
- Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In *Conference on Robot Learning (CoRL)*, 2024.
- Kimi Team. Kimi K3: Open frontier intelligence, 2026. URL <https://arxiv.org/abs/2607.24653>.
- Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabriel Levine, Wensi Ai, Benjamin Martinez, et al. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning (CoRL)*, 2024.
- Ruiying Li, Yunlang Zhou, YuYao Zhu, Kylin Chen, Jingyuan Wang, Sukai Wang, Kongtao Hu, Minhui Yu, Bowen Jiang, Zhan Su, Jiayao Ma, Xin He, Yongjian Shen, Yang Yang, Guanghui Ren, Maoqing Yao, Wenhao Wang, and Yao Mu. Roboclaw: An agentic framework for scalable long-horizon robotic tasks. *arXiv preprint arXiv:2603.11558*, 2026.

- 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.
- William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. Eurekaverse: Environment curriculum generation via large language models. In *Conference on Robot Learning (CoRL)*, 2024.
- 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)*, 2023.
- 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)*, 2024a.
- Yecheng Jason Ma, William Liang, Hung-Ju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. Dreureka: Language model guided sim-to-real transfer. In *Robotics: Science and Systems (RSS)*, 2024b.
- Yao Mu, Juntao Chen, Qinglong Zhang, Shoufa Chen, Qiaojun Yu, Chongjian Ge, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, Peize Sun, Haibao Yu, Chao Yang, Wenqi Shao, Wenhui Wang, Jifeng Dai, Yu Qiao, Mingyu Ding, and Ping Luo. Robocodex: Multimodal code generation for robotic behavior synthesis. In *International Conference on Machine Learning (ICML)*, 2024.
- Alexander Novikov, Ngô Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.
- NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llonet, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.
- Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems (RSS)*, 2024.
- OpenAI. Codex: Ai coding partner from openai, 2025. URL <https://openai.com/codex/>.
- OpenAI. Introducing GPT-5.5, 2026a. URL <https://openai.com/index/introducing-gpt-5-5/>.
- OpenAI. Introducing GPT-6.1 Sol, 2026b. URL <https://openai.com/index/introducing-gpt-6-1-sol/>.
- OpenAI. GPT-6 Astra: A new generation of intelligence, 2026c. URL <https://openai.com/index/gpt-6-astra/>.

- OpenAI. Announcing GPT-6 Sol and GPT-6 Luna in the API, Codex and ChatGPT, 2026d. URL <https://community.openai.com/t/announcing-gpt-6-sol-and-gpt-6-luna-in-the-api-codex-and-chatgpt/1399925>.
- OpenAI. Api pricing, 2026e. URL <https://developers.openai.com/api/docs/pricing>.
- Shaohui Peng, Xing Hu, Qi Yi, Rui Zhang, Jiaming Guo, Di Huang, Zikang Tian, Ruizhi Chen, Zidong Du, Qi Guo, Yunji Chen, and Ling Li. Hypothesis, verification, and induction: grounding large language models with self-driven skill learning. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI’24/IAAI’24/EAAI’24. AAAI Press, 2024. ISBN 978-1-57735-887-9. doi: 10.1609/aaai.v38i13.29376. URL <https://doi.org/10.1609/aaai.v38i13.29376>.
- Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Conference on Robot Learning (CoRL)*, 2025.
- Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, 2026. URL <https://qwen.ai/blog?id=qwen3.8>.
- Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.
- Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.
- 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.
- Georgios Tzifas and Hamidreza Kasaei. Lifelong robot library learning: Bootstrapping composable and generalizable skills for embodied control with language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *International Conference on Machine Learning (ICML)*, 2024a.
- Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025.

- Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. In *International Conference on Machine Learning (ICML)*, 2024b.
- 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. In *Conference on Robot Learning (CoRL)*, 2026.
- Senwei Xie, Yuntian Zhang, Ruiping Wang, and Xilin Chen. Uni-skill: Building self-evolving skill repository for generalizable robotic manipulation. *arXiv preprint arXiv:2603.02623*, 2026.
- Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2024.
- John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
- Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In *Conference on Robot Learning (CoRL)*, 2023.
- Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. Skillflow: benchmarking lifelong skill discovery and evolution for autonomous agents, 2026. URL <https://arxiv.org/abs/2604.17308>.
- 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.
- Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Abhiram Maddukuri, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

A. Skill Library Details

This appendix expands the representative entries shown in Figure 3 into the skill-library taxonomy. Each entry lists: (i) the *problem* extracted from the triggering failure trace; (ii) the *when-to-apply* guard encoding the situational retrieval condition; (iii) the validated *repair snippet*; and (iv) the origin task(s) that produced the entry.

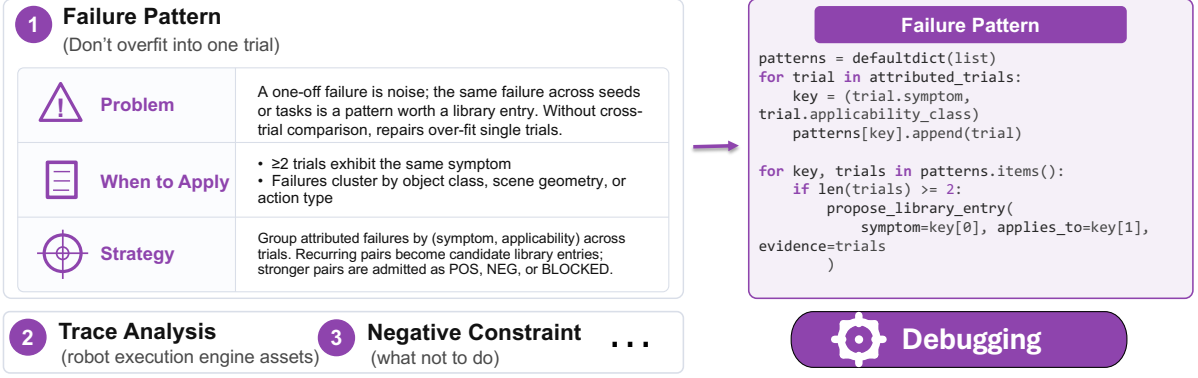


Figure 11: **Debugging skills.** Representative skill-library entries that encode reusable debugging strategies, including failure signatures, when-to-apply guards, and validated repair sketches.

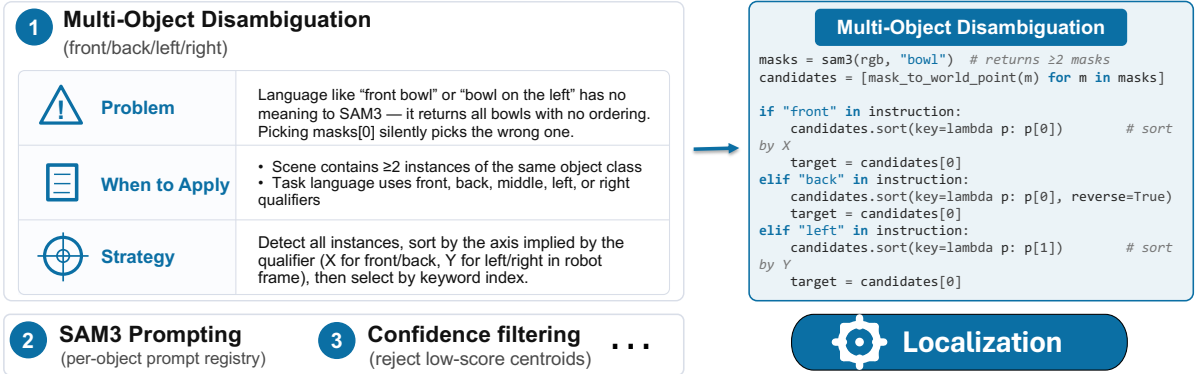


Figure 12: **Localization skills.** Representative entries for grounding ambiguous language and object references into robust perception and localization routines.

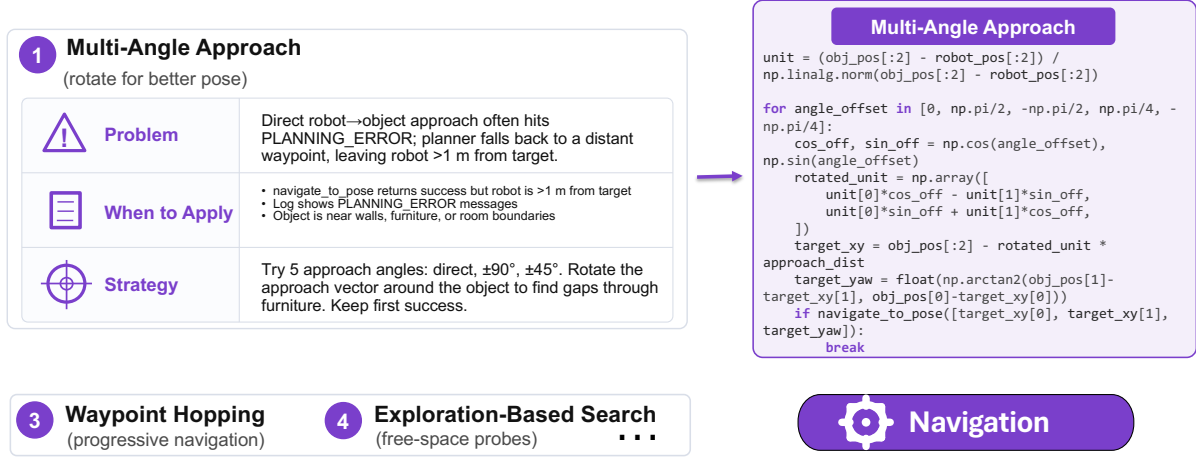


Figure 13: **Navigation skills.** Representative entries for recovering from motion-planning failures and selecting collision-aware approach poses.

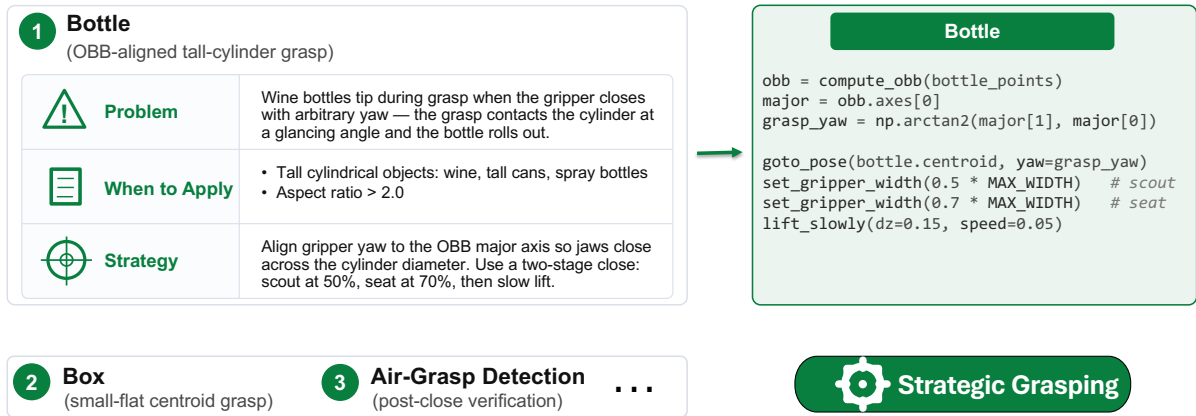


Figure 14: **Strategic grasping skills.** Representative entries for choosing task-appropriate grasp points and adapting grasp strategy to object geometry and scene context.

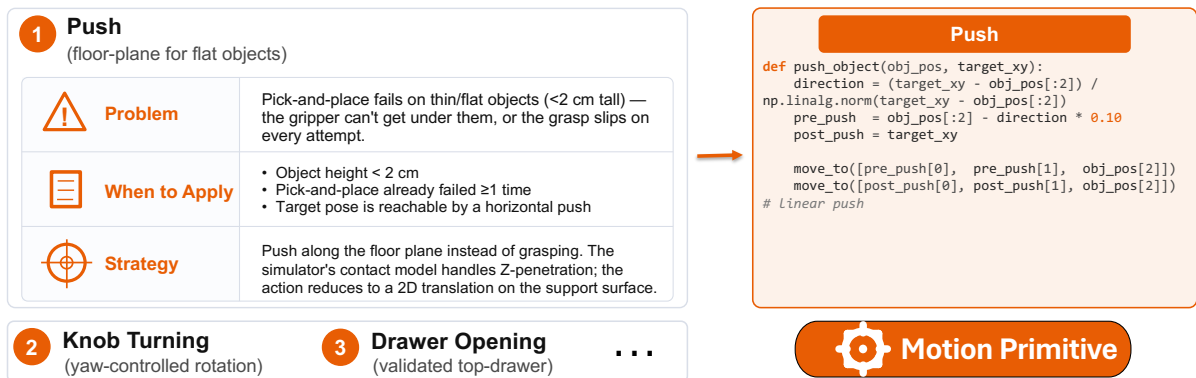


Figure 15: **Motion-primitive skills.** Representative entries for reusable low-level motion patterns, contact-rich alignment, and execution-time recovery.

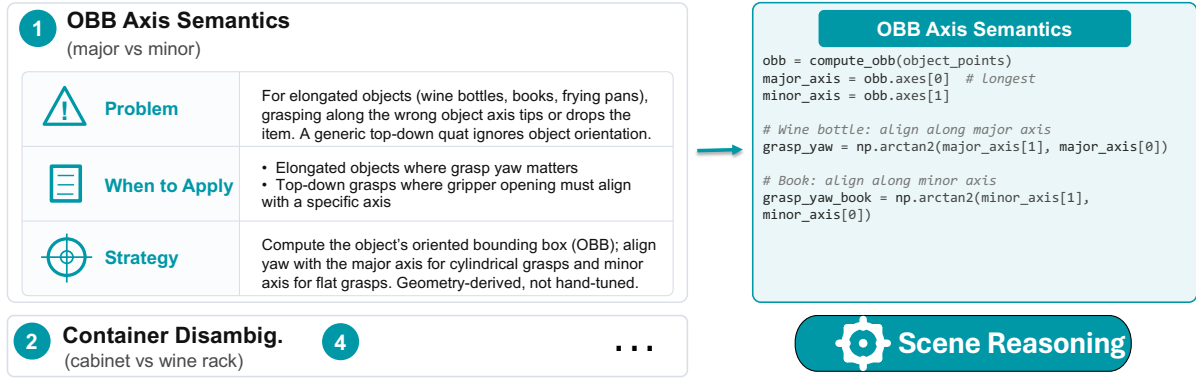


Figure 16: **Scene-reasoning skills.** Representative entries for reasoning over spatial relations, support surfaces, occlusions, and scene-level task constraints.

B. Main Benchmark Result Tables

B.1. Macro-Averaged Main Results

This section reports the exact numerical values visualized in Figure 4 of the main paper.

Table 5: **Main results on LIBERO-Pro.** Success of OpenVLA [Kim et al., 2024], π_0 [Black et al., 2025], $\pi_{0.5}$ [Physical Intelligence et al., 2025], CaP-Agent0 [Fu et al., 2026], and ASPIRE (Opus 4.6) on LIBERO-Pro [Zhou et al., 2025] Object, Goal, and Spatial under the position (Pos) and task (Task) perturbations, macro-averaged across 10 tasks per suite. **Overall** columns are macro-averaged across the three suites. All ASPIRE numbers are on the 50-seed held-out evaluation (seeds 1–50). See the per-task breakdown in Appendices E.1 and E.2.

Method	Object		Goal		Spatial		Overall		
	Pos	Task	Pos	Task	Pos	Task	Pos	Task	All
OpenVLA	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
π_0	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
$\pi_{0.5}$	17.0%	1.0%	38.0%	0.0%	20.0%	1.0%	25.0%	1.0%	13.0%
CaP-Agent0	22.0%	18.0%	26.0%	17.0%	12.0%	14.0%	20.0%	16.0%	18.0%
ASPIRE	98.0%	95.0%	81.0%	45.0%	51.0%	60.0%	77.0%	67.0%	72.0%

Table 6: **Main results on Robosuite.** Success of CaP-Agent0 [Fu et al., 2026] and ASPIRE (Opus 4.6) on 7 Robosuite [Zhu et al., 2020] manipulation tasks (100-trial held-out evaluation). **Mean** is the unweighted mean across all 7 tasks.

Task	CaP-Agent0	ASPIRE
cube_lift	97%	97%
cube_stack	98%	99%
cube_restack	89%	100%
spill_wipe	100%	99%
two_arm_handover	20%	92%
two_arm_lift	74%	71%
nut_assembly	0%	9%
<i>Mean</i>	<i>68.0%</i>	<i>81.0%</i>

Table 7: **Main results on BEHAVIOR-1K.** Navigation (Nav) and task (Task) success on two representative BEHAVIOR-1K [Li et al., 2024] household tasks, Soda Can pick-up and Radio pick-up, reported separately. All numbers are on the 25-seed held-out evaluation (seeds 1–25) using Opus 4.6. ASPIRE runs interactive block-by-block generation with the skill library accumulated on seeds 26–35.

Task	Human		CaP-Agent0		ASPIRE	
	Nav	Task	Nav	Task	Nav	Task
Soda Can pick-up	80%	72%	84%	72%	92%	88%
Radio pick-up	88%	36%	80%	56%	100%	88%

C. LIBERO-Pro Long Zero-Shot Transfer

C.1. Macro-Averaged Main Results

Table 8: **Zero-shot transfer to LIBERO-Pro Long.** Macro-averaged success of the ASPIRE skill library, accumulated on LIBERO-90, on LIBERO-Pro Long [Zhou et al., 2025] using Opus 4.6. N denotes the number of LIBERO-90 tasks whose repair skills seed the library; no additional debugging is performed at test time. Pos and Task denote the position and task perturbations, 10 tasks each.

Method	LIBERO-Pro Long Zero-Shot Transfer		
	Pos	Task	Overall
OpenVLA	0.0%	0.0%	0.0%
π_0	0.0%	0.0%	0.0%
$\pi_{0.5}$	8.0%	1.0%	5.0%
CaP-Agent0	5.2%	2.4%	3.8%
ASPIRE ($N=0$, zero-shot)	0.0%	9.4%	4.7%
ASPIRE ($N=25$, zero-shot)	5.6%	21.8%	13.7%
ASPIRE ($N=50$, zero-shot)	13.8%	29.2%	21.5%
ASPIRE ($N=90$, zero-shot)	22.6%	38.3%	30.5%

C.2. Per-Task Breakdown

Table 9: **Per-task zero-shot transfer to LIBERO-Pro Long.** As in Table 8, per task and snapshot size N (seeds 1–50, $n=50$). **Pos**: position perturbation. **Task**: task perturbation.

Task	Pos				Task			
	$N=0$	$N=25$	$N=50$	$N=90$	$N=0$	$N=25$	$N=50$	$N=90$
Stove + moka pot	0%	30%	84%	100%	0%	2%	68%	26%
Bowl in drawer	0%	0%	0%	0%	0%	0%	0%	0%
Mug in microwave	0%	0%	0%	0%	0%	0%	0%	0%
Both mokas on stove	0%	0%	0%	0%	0%	0%	84%	100%
Soup + cream cheese	0%	12%	2%	0%	0%	68%	60%	70%
Soup + tomato sauce	0%	14%	20%	0%	78%	68%	2%	70%
Cream cheese + butter	0%	0%	6%	94%	0%	0%	34%	86%
Mug on two plates	0%	0%	16%	32%	16%	28%	8%	0%
Mug + pudding	0%	0%	6%	0%	0%	0%	8%	4%
Book in caddy	0%	0%	4%	0%	0%	52%	28%	26%
<i>Mean</i>	<i>0.0%</i>	<i>6.0%</i>	<i>14.0%</i>	<i>23.0%</i>	<i>9.0%</i>	<i>22.0%</i>	<i>29.0%</i>	<i>38.0%</i>

D. Harness for an Agentic System 2 on Real Robots

Apart from the execution of the open-loop program, we consider extending the harness to support the closed-loop interaction of coding agents, as shown in Figure 17. The agent serves as a System 2 connected to a persistent Code-as-Policy (CAP) runtime through the Model Context Protocol (MCP). The agent interprets tasks, plans, selects actions, and evaluates outcomes. The runtime comprises an MCP **bridge**, a shared **session manager**, CAP API with perception and planning providers, and the real YAM environment that owns arm services. The MCP **bridge** serves a Streamable HTTP endpoint with an explicit allowlist of tools. Requests are validated against JSON schemas before dispatch. Tool availability reflects the configured providers; discovery also reports provider identifiers and source hashes for packaged CAP programs. The interface exposes the following classes of capabilities:

- **Discovery and monitoring.** `get_capability_graph` describes capabilities; `get_job` reports the current or most recent job; and `read_operator_messages` retrieves queued instructions.
- **Observation and perception.** `observe` returns state, camera images, and freshness metadata. `cap_get_robot_state` returns joint, gripper, and end-effector state. `cap_localize_object` estimates geometry from a text query and calibrated RGB-D.
- **Motion and grasping.** `cap_move_to_pose` commands an absolute end-effector pose that the arm will be tracking; `cap_move_to_joints` directly commands six joint angles in radians; `cap_move_synchronized` coordinates targets for both arms; and `cap_set_gripper` commands one gripper with a normalized opening in $[0, 1]$, from closed to open.

For the basic action tool `cap_move_to_pose`, the agent must choose the Cartesian target position. Internally, the tool uses Mink for inverse kinematics and cuRobo for trajectory planning. Physical operations run as serialized background jobs, with concurrent actuation requests rejected. MCP action calls wait for their job to finish and return structured status, measured terminal state, errors and diagnostics, and camera images as native MCP image content. Observations include timestamps, frame identifiers, acquisition age, and stale or missing flags. Control-level success only confirms the commanded motion; the agent judges task completion from physical evidence.

On the deployment end, `RealYamEnv` owns camera acquisition and connects to the arm services through RPC. The CAP adapter routes actuation through one control worker. Trajectories are interpolated in time and streamed as joint and gripper position targets to the arm services and CAN motor drivers. Station profiles supply hardware endpoints, camera identities and calibration, robot models, joint limits, and home targets. Actuation requires explicit enablement at server startup.

Agent Sessions and Closed-Loop Workflow A fresh agent session is launched through the Codex CLI with the agent operating guide and operator task. Repository instructions define execution rules; tool-use and task-specific skills supply action semantics and manipulation strategies. The conversation retains the goal, plan, observations, and unresolved subgoals. The robot runtime persists independently, allowing a fresh agent to attach without recreating the environment or reopening cameras. The session instructions specify the agent loop: 1) **Initialize.** Read the applicable instructions and discover the available tools. A motion-enabled fresh session invokes Home once, checks its terminal result, measured joint residuals, gripper state, and camera freshness, and reports readiness. Observation-only startup performs readiness checks without motion; 2) **Observe and reason.** Inspect the scene, obtain calibrated object geometry when needed, and identify the next subgoal and its completion evidence. 3) **Act.** Select a registered motion or gripper tool, ground its targets in the available observations, and wait for its terminal result before issuing a dependent action.

4) **Verify.** Evaluate returned state and images against the subgoal. Reuse sufficient fresh evidence; request another observation or localization when evidence is stale, ambiguous, or invalidated by motion.

With this System 2 harness, we show that **GPT-6 Astra** can perform contact-rich, long-horizon manipulation tasks as demonstrated in 18. Specifically, we task solving **Rubik’s Cube** from randomly initialized cube states. Human operator would provide an heuristic context documents (general rules and potential failure cases, in text form), and the agent explores reliable skills or motion patterns such as cube re-orientation and face turning. The agent first rotates the cube to expose all faces and perceive its state, then calls the Kociemba solver to obtain a solution, and then executes the plan while updating its memory of the cube state.

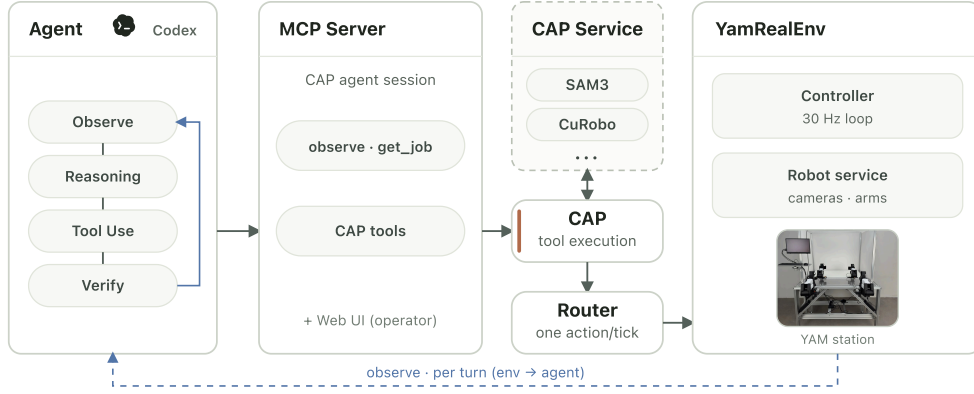
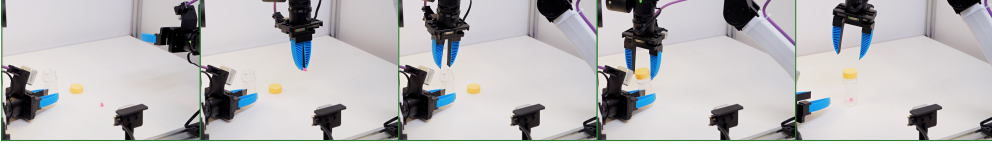


Figure 17: **System design for the agentic workflow.** The coding agent runs an observe–reason–act–verify loop. It may call observation tools to acquire four-view images or target object pose estimates. CaP tools provide action primitives such as end-effector control.

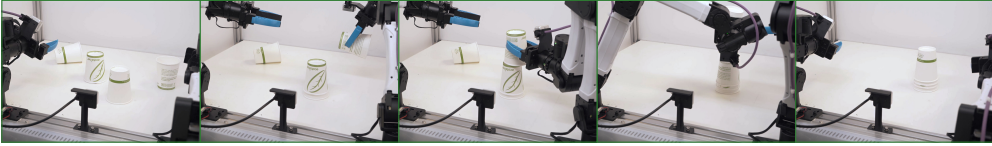
(a) Unscrew the bottle cap.



(b) Put the pill into the bottle and tighten the cap.



(c) Stack the cups upside down.



(d) Solve the Rubik's Cube.

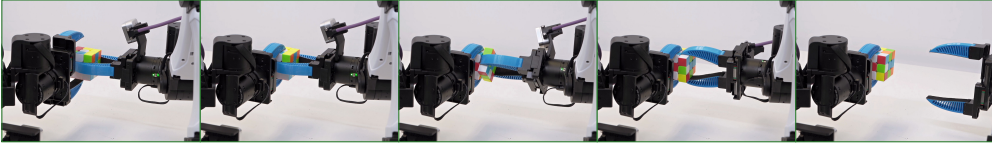


Figure 18: **Real robot experiments on system-2.** We provide example for real robot tasks on a bi-manual YAM platform that require atomic manipulation skills including pick-and-place, handover, re-orientation and fine-grained adjustments.

E. Ablation Details

E.1. LIBERO-Pro Position Perturbation Per-Task Breakdown

Table 10: **Per-task ablation on LIBERO-Pro, position perturbation, Opus 4.6.** Success on seeds 1–50 ($n=50$). **w/o Engine & Evo:** ASPIRE without the robot execution engine and evolutionary search, i.e., zero-shot Claude Opus 4.6 with 15 example programs. **w/o Evo:** ASPIRE without evolutionary search, i.e., the program repaired with the robot execution engine and skill library. **Evo. search:** raw evolutionary search result (best validated candidate), or “–” if evolutionary search was not run for that task. **ASPIRE:** per-task winner between the repaired program and evolutionary search, selected on a held-out validation set (seeds 66–80).

Task	w/o Engine & Evo	w/o Evo	Evo. search	ASPIRE
<i>LIBERO-Pro Object</i>				
Alphabet soup	0%	100%	–	100%
BBQ sauce	6%	100%	–	100%
Butter	0%	100%	–	100%
Chocolate pudding	0%	100%	–	100%
Cream cheese	92%	100%	–	100%
Ketchup	2%	92%	–	92%
Milk	6%	92%	–	92%
Orange juice	46%	100%	–	100%
Salad dressing	18%	96%	–	96%
Tomato sauce	66%	98%	–	98%
Mean	24.0%	98.0%	–	98.0%
<i>LIBERO-Pro Goal</i>				
Open middle drawer	28%	2%	72%	72%
Open top drawer+bowl	0%	86%	–	86%
Push plate to stove	0%	0%	0%	0%
Bowl on plate	4%	36%	92%	92%
Bowl on stove	16%	56%	96%	96%
Bowl on cabinet top	72%	74%	100%	100%
Cream cheese in bowl	0%	74%	92%	74%
Wine on rack	6%	46%	98%	98%
Wine on cabinet top	100%	24%	100%	100%
Turn on stove	40%	68%	90%	90%
Mean	27.0%	47.0%	82.0% ⁽ⁿ⁼⁹⁾	81.0%
<i>LIBERO-Pro Spatial</i>				
Betw. plate+ramekin	2%	10%	94%	94%
From table center	0%	32%	82%	32%
In top drawer	2%	22%	10%	10%
Next to cookie box	2%	16%	12%	16%
Next to plate	8%	100%	100%	100%
Next to ramekin	0%	20%	40%	20%
On cookie box	0%	32%	24%	24%
On ramekin	58%	42%	94%	94%
On stove	4%	86%	–	86%
On wooden cabinet	10%	56%	34%	34%
Mean	9.0%	42.0%	54.0% ⁽ⁿ⁼⁹⁾	51.0%
Overall mean	20.0%	62.0%	68.0% ⁽ⁿ⁼¹⁸⁾	77.0%

E.2. LIBERO-Pro Task Perturbation Per-Task Breakdown

Table 11: **Per-task ablation on LIBERO-Pro, task perturbation, Opus 4.6.** As in Table 10, for the task perturbation.

Task	w/o Engine & Evo	w/o Evo	Evo. search	ASPIRE
<i>LIBERO-Pro Object</i>				
Alphabet soup	0%	100%	–	100%
BBQ sauce	88%	100%	–	100%
Butter	78%	98%	–	98%
Chocolate pudding	0%	100%	–	100%
Cream cheese	0%	72%	74%	72%
Ketchup	0%	96%	–	96%
Milk	0%	100%	–	100%
Orange juice	0%	100%	–	100%
Salad dressing	0%	92%	–	92%
Tomato sauce	4%	96%	–	96%
<i>Mean</i>	<i>17.0%</i>	<i>95.0%</i>	–	<i>95.0%</i>
<i>LIBERO-Pro Goal</i>				
Open middle drawer	0%	0%	0%	0%
Open top drawer+bowl	0%	50%	46%	46%
Push plate to stove	26%	58%	76%	76%
Bowl on plate	0%	88%	82%	88%
Bowl on stove	0%	0%	0%	0%
Bowl on cabinet top	0%	0%	4%	4%
Cream cheese in bowl	0%	82%	74%	74%
Wine on rack	0%	74%	86%	86%
Wine on cabinet top	0%	14%	72%	14%
Turn on stove	24%	52%	66%	66%
<i>Mean</i>	<i>5.0%</i>	<i>42.0%</i>	<i>51.0%⁽ⁿ⁼¹⁰⁾</i>	<i>45.0%</i>
<i>LIBERO-Pro Spatial</i>				
Betw. plate+ramekin	14%	8%	84%	8%
From table center	26%	46%	64%	64%
In top drawer	4%	96%	–	96%
Next to cookie box	0%	94%	–	94%
Next to plate	8%	70%	92%	70%
Next to ramekin	0%	60%	42%	60%
On cookie box	0%	6%	70%	70%
On ramekin	20%	10%	46%	46%
On stove	0%	32%	42%	42%
On wooden cabinet	13%	46%	98%	46%
<i>Mean</i>	<i>4.0%</i>	<i>47.0%</i>	<i>67.0%⁽ⁿ⁼⁸⁾</i>	<i>60.0%</i>
<i>Overall mean</i>	<i>9.0%</i>	<i>61.0%</i>	<i>59.0%⁽ⁿ⁼¹⁹⁾</i>	<i>67.0%</i>

E.3. Evolutionary Search Progress Per-Task Breakdown

Table 12: **Evolutionary search progress.** Held-out success (seeds 1–50) of the best candidate at each iteration on selected LIBERO-Pro tasks. Opus 4.6. Blank cells indicate that the search terminated early.

Suite	Task	Iter 0	Iter 1	Iter 2	Iter 3	Iter 4
Goal (Pos)	Bowl → plate	62%	60%	60%	18%	86%
Goal (Pos)	Wine bottle → rack	40%	76%	74%		
Goal (Pos)	Bowl → stove	62%	82%			
Goal (Task)	Push plate → stove	0%	80%			
Spatial (Pos)	Bowl on cabinet → plate	16%	38%	38%	46%	30%
Spatial (Pos)	Bowl next to cookie → plate	16%	20%	4%	4%	40%
Spatial (Task)	Bowl on cookie → plate	26%	4%	70%		
Spatial (Task)	Bowl on ramekin → plate	2%	0%	22%	36%	18%

E.4. Model Ablation on LIBERO-Pro With and Without Evolutionary Search

Table 13: **Per-task model ablation on LIBERO-Pro Object.** Success over 50 held-out seeds with (w/) and without (w/o) evolutionary search (evo). Tasks are labelled by a shortened form of the instruction the policy receives (the perturbed one under the task perturbation). Object tasks pick the named item and place it in the basket. Tasks not selected for evolutionary search keep their w/o evo value in the w/ evo column. Bold marks tasks where evolutionary search changed the held-out result.

Task	Claude Opus 5		Claude Opus 5.5		GPT-6 Astra	
	w/o evo	w/ evo	w/o evo	w/ evo	w/o evo	w/ evo
<i>Position perturbation</i>						
0 Alphabet soup	100%	100%	100%	100%	100%	100%
1 Cream cheese	100%	100%	100%	100%	100%	100%
2 Salad dressing	100%	100%	100%	100%	100%	100%
3 BBQ sauce	92%	92%	88%	88%	90%	90%
4 Ketchup	100%	100%	100%	100%	100%	100%
5 Tomato sauce	100%	100%	100%	100%	100%	100%
6 Butter	100%	100%	100%	100%	100%	100%
7 Milk	94%	94%	94%	94%	100%	100%
8 Chocolate pudding	100%	100%	100%	100%	100%	100%
9 Orange juice	100%	100%	100%	100%	100%	100%
<i>Average</i>	98.6%	98.6%	98.2%	98.2%	99.0%	99.0%
<i>Task perturbation</i>						
0 Cream cheese	100%	100%	100%	100%	100%	100%
1 Alphabet soup	100%	100%	100%	100%	100%	100%
2 Tomato sauce	100%	100%	100%	100%	100%	100%
3 Ketchup	100%	100%	100%	100%	100%	100%
4 Milk	100%	100%	100%	100%	94%	94%
5 BBQ sauce	100%	100%	100%	100%	100%	100%
6 Orange juice	100%	100%	100%	100%	100%	100%
7 Butter	100%	100%	100%	100%	100%	100%
8 Salad dressing	100%	100%	100%	100%	100%	100%
9 Chocolate pudding	100%	100%	100%	100%	100%	100%
<i>Average</i>	100.0%	100.0%	100.0%	100.0%	99.4%	99.4%

Table 14: **Per-task model ablation on LIBERO-Pro Goal.** As in Table 13, for LIBERO-Pro Goal.

Task	Claude Opus 5		Claude Opus 5.5		GPT-6 Astra	
	w/o evo	w/ evo	w/o evo	w/ evo	w/o evo	w/ evo
<i>Position perturbation</i>						
0 Open middle drawer	90%	90%	100%	100%	96%	96%
1 Bowl → stove	94%	94%	90%	100%	92%	92%
2 Wine bottle → drawer top	98%	98%	100%	100%	100%	100%
3 Open drawer, bowl inside	100%	100%	100%	100%	100%	100%
4 Bowl → drawer top	100%	100%	100%	100%	92%	92%
5 Push plate to stove	88%	88%	84%	84%	98%	98%
6 Cream cheese → bowl	100%	100%	100%	100%	92%	92%
7 Turn on stove	100%	100%	100%	100%	100%	100%
8 Bowl → plate	100%	100%	96%	96%	100%	100%
9 Wine bottle → rack	100%	100%	100%	100%	100%	100%
<i>Average</i>	97.0%	97.0%	97.0%	98.0%	97.0%	97.0%
<i>Task perturbation</i>						
0 Open bottom drawer	0%	0%	100%	100%	82%	82%
1 Plate → stove	100%	100%	100%	100%	100%	100%
2 Wine bottle → bowl	96%	96%	100%	100%	100%	100%
3 Open drawer, cheese inside	98%	98%	100%	100%	100%	100%
4 Plate → drawer top	88%	88%	100%	100%	100%	100%
5 Push cheese to stove	98%	98%	100%	100%	98%	98%
6 Wine bottle → bowl	94%	94%	100%	100%	100%	100%
7 Turn off stove	100%	100%	94%	94%	92%	92%
8 Wine bottle → plate	94%	94%	100%	100%	100%	100%
9 Cream cheese → rack	100%	100%	100%	100%	96%	96%
<i>Average</i>	86.8%	86.8%	99.4%	99.4%	96.8%	96.8%

Table 15: **Per-task model ablation on LIBERO-Pro Spatial.** As in Table 13, for LIBERO-Pro Spatial. Spatial tasks pick the black bowl at the listed location and place it on the plate.

Task	Claude Opus 5		Claude Opus 5.5		GPT-6 Astra	
	w/o evo	w/ evo	w/o evo	w/ evo	w/o evo	w/ evo
<i>Position perturbation</i>						
0 Between plate & ramekin	98%	98%	100%	100%	98%	98%
1 Next to ramekin	96%	96%	100%	100%	100%	100%
2 Table center	100%	100%	100%	100%	96%	96%
3 On cookie box	98%	98%	100%	100%	96%	96%
4 In cabinet top drawer	96%	96%	100%	100%	82%	98%
5 On ramekin	90%	90%	100%	100%	100%	100%
6 Next to cookie box	100%	100%	100%	100%	100%	100%
7 On stove	100%	100%	100%	100%	98%	98%
8 Next to plate	100%	100%	94%	94%	100%	100%
9 On wooden cabinet	100%	100%	100%	100%	100%	100%
<i>Average</i>	97.8%	97.8%	99.4%	99.4%	97.0%	98.6%
<i>Task perturbation</i>						
0 Not btw. plate & ramekin	98%	98%	100%	100%	92%	92%
1 Next to cookie box	92%	92%	100%	100%	100%	100%
2 Next to plate	90%	90%	92%	92%	92%	92%
3 On cabinet top	100%	100%	100%	100%	100%	100%
4 On wooden cabinet top	100%	100%	100%	100%	100%	100%
5 On cookie box	100%	100%	100%	100%	100%	100%
6 On stove	100%	100%	100%	100%	100%	100%
7 On cabinet top	100%	100%	100%	100%	100%	100%
8 Next to ramekin	100%	100%	100%	100%	96%	96%
9 On stove	100%	100%	100%	100%	96%	96%
<i>Average</i>	98.0%	98.0%	99.2%	99.2%	97.6%	97.6%

Table 16: **Per-task model ablation on LIBERO-Pro Long.** As in Table 13, for LIBERO-Pro Long.

Task	Claude Opus 5		Claude Opus 5.5		GPT-6 Astra	
	w/o evo	w/ evo	w/o evo	w/ evo	w/o evo	w/ evo
<i>Position perturbation</i>						
0 Soup + sauce → basket	94%	94%	98%	98%	64%	76%
1 Cheese + butter → basket	98%	98%	100%	100%	94%	94%
2 Stove on + moka pot	98%	98%	98%	98%	96%	96%
3 Bowl → drawer, close	52%	78%	98%	98%	90%	90%
4 Mugs → L/R plates	98%	98%	100%	100%	90%	90%
5 Book → caddy back	96%	96%	84%	84%	96%	96%
6 Mug → plate, pudding right	98%	98%	98%	98%	84%	84%
7 Soup + cheese → basket	88%	92%	98%	98%	82%	84%
8 Both moka pots → stove	88%	92%	100%	100%	82%	82%
9 Mug → microwave, close	0%	26%	96%	96%	30%	54%
<i>Average</i>	81.0%	87.0%	97.0%	97.0%	80.8%	84.6%
<i>Task perturbation</i>						
0 Cheese + sauce → basket	98%	98%	100%	100%	94%	94%
1 Soup + butter → basket	98%	98%	88%	88%	92%	92%
2 Stove on + pan	94%	94%	100%	100%	100%	100%
3 Bottle → drawer, close	72%	100%	100%	100%	96%	96%
4 Mugs → L/R plates, swapped	100%	100%	100%	100%	100%	100%
5 Cup → caddy back	86%	86%	100%	100%	100%	100%
6 Red mug → plate, pudding	86%	86%	100%	100%	100%	100%
7 Ketchup + cheese → basket	100%	100%	100%	100%	100%	100%
8 Left moka pot → stove	100%	100%	100%	100%	100%	100%
9 Mug → microwave, close	0%	42%	100%	100%	68%	68%
<i>Average</i>	83.4%	90.4%	98.8%	98.8%	95.0%	95.0%

E.5. Model Ablation on Robosuite

 Table 17: **Model ablation on Robosuite.** Success on all seven Robosuite tasks over 100 held-out trials per task. Table 4 shows the three hardest tasks.

Task	Claude Opus 4.6	Claude Opus 5	Claude Opus 5.5	GPT-6 Astra
cube_lift	97%	100%	100%	100%
cube_stack	99%	100%	100%	99%
cube_restack	100%	100%	100%	100%
spill_wipe	99%	99%	100%	99%
two_arm_handover	92%	99%	100%	95%
two_arm_lift	71%	96%	100%	100%
nut_assembly	9%	98%	100%	99%
<i>Mean</i>	<i>81%</i>	<i>99%</i>	<i>100%</i>	<i>99%</i>

E.6. Model Ablation on LIBERO-Pro Goal under Position Perturbation

Table 18: **Success and cost of all models in Figure 8.** Held-out success on the 10 LIBERO-Pro Goal tasks under the position perturbation (50 seeds each), using ASPIRE iterative debugging without evolutionary search and with an initially empty skill library. Cost is estimated at list API prices; agent time is the mean time per task.

Model	Open weights	Success	Cost / task	Total cost	Agent time / task (min)
Claude Opus 5.5		99.2%	\$18.29	\$183	51
Claude Sonnet 5.5		98.8%	\$12.11	\$121	56
Claude Fable 5.1		96.6%	\$42.24	\$422	68
Claude Opus 5		90.6%	\$59.72	\$597	76
Claude Haiku 5.5		68.0%	\$3.97	\$40	54
Claude Opus 4.6		55.6%	\$23.58	\$236	61
GPT-6 Astra		94.4%	\$35.37	\$354	31
GPT-6.1 Sol		88.0%	\$9.52	\$95	49
GPT-6 Sol		59.2%	\$7.64	\$76	42
GPT-6 Luna		14.8%	\$0.83	\$8	47
Kimi K3	✓	78.4%	\$58.19	\$582	139
Qwen 3.8 Max	✓	56.0%	\$39.82	\$398	103

Table 19: **Per-task model ablation on LIBERO-Pro Goal, position perturbation.** Iterative debugging results without evolutionary search, as in Figure 8. Stage 1 iterative debugging on seeds 51–65; Stage 2 held-out evaluation on seeds 1–50 is reported below. Claude Opus 4.6 (max effort), Claude Opus 5 and Claude Opus 5.5 run through Claude Code with a 1M-token context; GPT-6 Astra runs through Codex with a 258k-token context. The skill library is initially empty.

Task	Claude Opus 4.6	Claude Opus 5	Claude Opus 5.5	GPT-6 Astra
Open middle drawer	0%	100%	100%	98%
Bowl on stove	0%	100%	100%	92%
Wine bottle on cabinet	94%	100%	100%	100%
Bowl inside top drawer	0%	100%	100%	100%
Bowl on cabinet top	96%	100%	100%	100%
Push plate to stove	0%	14%	92%	54%
Cream cheese in bowl	82%	100%	100%	100%
Turn on stove	100%	100%	100%	100%
Bowl on plate	100%	94%	100%	100%
Wine bottle on rack	84%	98%	100%	100%
<i>Mean</i>	<i>55.6%</i>	<i>90.6%</i>	<i>99.2%</i>	<i>94.4%</i>

E.7. Model Ablation Token Usage on LIBERO-Pro Object, Goal, and Spatial

Table 20: **Token usage and cost on LIBERO-Pro Object, Goal, and Spatial.** Token usage, estimated API cost, and compute for the 60-task LIBERO-Pro runs. Total tokens include all input and output. Evolutionary search was triggered on tasks below 90% success. GPT-6 Astra (xhigh, Codex) is the Object, Goal, and Spatial phase of a run that continues on LIBERO-Pro Long; its coordinator is split at the phase boundary, and Codex’s approval-reviewer calls are excluded. Claude Opus 4.6 (max effort) is likewise the first phase of a run that continues on LIBERO-Pro Long; its coordinator is allocated by agent cost, and its token counts are exact, including Claude Code background calls. Claude Opus 4.6 costs are from a re-run with the current pipeline; its success rates are reported from the original run (Tables 10 and 11).

Metric	Claude Opus 4.6	Claude Opus 5	Claude Opus 5.5	GPT-6 Astra
<i>Without evolutionary search</i>				
Total tokens	2,009.3M	2,542.9M	1,538.3M	522.1M
Cache-read tokens	1,966.5M	2,471.7M	1,493.5M	505.4M
Output tokens	8.18M	13.44M	8.02M	2.57M
Est. cost (USD)	\$1,376	\$1,925	\$650	\$776
Cost per task (USD)	\$22.9	\$32.1	\$10.8	\$12.9
Sim rollouts	12,214	10,941	10,555	7,905
LLM calls	9,405	8,535	7,896	5,950
Actor time per task (min)	79	60	46	27
<i>Evolutionary search only, additional usage</i>				
Tasks searched	31	3	7	2
Total tokens	4,937.7M	54.7M	246.7M	17.4M
Cache-read tokens	4,908.6M	52.7M	227.0M	17.1M
Output tokens	5.17M	0.24M	1.12M	0.06M
Est. cost (USD)	\$2,761	\$43	\$160	\$23
Cost per searched task (USD)	\$89.1	\$14.4	\$22.9	\$11.4
Sim rollouts	9,529	454	1,359	344
LLM calls	17,037	352	746	236
Actor time per task (min)	147	43	32	39

Table 21: **Per-suite success and cost on LIBERO-Pro Object, Goal, and Spatial.** Held-out success (SR), list-price cost per task, and actor time per task for each suite and perturbation in the 60-task LIBERO-Pro runs, without evolutionary search. Actor, verifier, and rerun usage is attributed to its task; the coordinator is allocated to tasks in proportion to their agent cost.

Suite · perturbation	Claude Opus 4.6			Claude Opus 5			Claude Opus 5.5			GPT-6 Astra		
	SR	\$/task	min/task	SR	\$/task	min/task	SR	\$/task	min/task	SR	\$/task	min/task
Goal · Position	–	\$35.0	112	97.0%	\$35.1	76	97.0%	\$13.3	56	97.0%	\$13.7	27
Goal · Task	–	\$28.1	100	86.8%	\$39.3	78	99.4%	\$11.5	46	96.8%	\$12.0	25
Object · Position	–	\$20.1	68	98.6%	\$29.2	48	98.2%	\$9.2	41	99.0%	\$14.8	27
Object · Task	–	\$11.0	36	100.0%	\$24.7	46	100.0%	\$9.4	40	99.4%	\$13.3	28
Spatial · Position	–	\$24.7	90	97.8%	\$37.1	57	99.4%	\$11.5	45	97.0%	\$13.0	29
Spatial · Task	–	\$18.8	70	98.0%	\$27.2	52	99.2%	\$10.1	45	97.6%	\$10.8	25
<i>Position (avg)</i>	–	\$26.6	90	97.8%	\$33.8	60	98.2%	\$11.3	47	97.7%	\$13.8	28
<i>Task (avg)</i>	–	\$19.3	69	94.9%	\$30.4	59	99.5%	\$10.4	44	97.9%	\$12.0	26
<i>Goal (avg)</i>	–	\$31.6	106	91.9%	\$37.2	77	98.2%	\$12.4	51	96.9%	\$12.8	26
<i>Object (avg)</i>	–	\$15.5	52	99.3%	\$26.9	47	99.1%	\$9.3	41	99.2%	\$14.1	28
<i>Spatial (avg)</i>	–	\$21.7	80	97.9%	\$32.1	55	99.3%	\$10.8	45	97.3%	\$11.9	27
<i>All 60</i>	–	\$22.9	79	96.4%	\$32.1	60	98.9%	\$10.8	46	97.8%	\$12.9	27

E.8. Model Ablation Token Usage on LIBERO-Pro Long

Table 22: **Token usage and cost on LIBERO-Pro Long.** As in Table 20, for the 20-task LIBERO-Pro Long runs. Token usage is transcript-derived, and total tokens include input and estimated output; Claude Opus 4.6 token counts are exact. All four runs used the skill library from their LIBERO-Pro phase.

Metric	Claude Opus 4.6	Claude Opus 5	Claude Opus 5.5	GPT-6 Astra
<i>Without evolutionary search</i>				
Total tokens	550.5M	1,544.2M	800.2M	490.5M
Cache-read tokens	539.1M	1,487.6M	774.6M	479.3M
Output tokens	1.98M	5.28M	3.56M	2.03M
Est. cost (USD)	\$411	\$1,206	\$340	\$672
Cost per task (USD)	\$20.6	\$60.3	\$17.0	\$33.6
Sim rollouts	3,169	4,722	3,870	3,826
LLM calls	2,207	5,451	3,403	4,413
Actor time per task (min)	73	137	70	73
<i>Evolutionary search only, additional usage</i>				
Tasks searched	19	8	2	6
Total tokens	5,119.9M	1,111.1M	72.5M	124.4M
Cache-read tokens	5,090.3M	1,070.3M	66.7M	122.9M
Output tokens	4.68M	1.78M	0.33M	0.20M
Est. cost (USD)	\$2,792	\$826	\$47	\$146
Cost per searched task (USD)	\$146.9	\$103.2	\$23.6	\$24.4
Sim rollouts	7,052	2,570	415	1,274
LLM calls	16,949	2,427	214	1,163
Actor time per searched task (min)	256	291	114	143

Table 23: **Per-perturbation success and cost on LIBERO-Pro Long.** As in Table 21, for the 20-task LIBERO-Pro Long runs. All four runs used the skill library from their LIBERO-Pro phase.

Perturbation	Claude Opus 4.6			Claude Opus 5			Claude Opus 5.5			GPT-6 Astra		
	SR	\$/task	min/task	SR	\$/task	min/task	SR	\$/task	min/task	SR	\$/task	min/task
Position	–	\$20.3	73	81.0%	\$73.7	154	97.0%	\$17.7	74	80.8%	\$37.4	84
Task	–	\$20.8	72	83.4%	\$46.9	119	98.8%	\$16.3	66	95.0%	\$29.8	63
<i>All 20</i>	–	\$20.6	73	82.2%	\$60.3	137	97.9%	\$17.0	70	87.9%	\$33.6	73

E.9. LIBERO-Pro Long Iterative Debugging

Table 24: **Iterative debugging on LIBERO-Pro Long.** Stage 2 held-out success on seeds 1–50 under the position (Pos) and task (Task) perturbations, without evolutionary search, using Claude Opus 4.6 in Claude Code. **ZS**: zero-shot result; **ID**: result after iterative debugging.

Task	Pos		Task	
	ZS	ID	ZS	ID
Soup + sauce to basket (S2)	0%	22%	70%	98%
Cheese + butter to basket (S2)	94%	94%	86%	86%
Stove knob + moka pot (S3)	100%	100%	26%	68%
Bowl/bottle to drawer (S4)	0%	0%	0%	0%
Mugs to plates L/R (S5)	32%	32%	0%	0%
Book/cup to caddy back (Study1)	0%	0%	26%	26%
Mug to plate + pudding (S6)	0%	36%	4%	34%
Soup/ketchup + cheese (S1)	0%	94%	70%	94%
Moka pot(s) to stove (S8)	0%	0%	100%	100%
Mug to microwave + close (S6)	0%	54%	0%	14%
<i>Mean</i>	<i>22.6%</i>	<i>43.2%</i>	<i>38.3%</i>	<i>52.0%</i>

E.10. Iterative Debugging Variance on LIBERO-Pro Goal under Position Perturbation

Table 25: **Iterative debugging variance on LIBERO-Pro Goal, position perturbation.** Stage 2 held-out success on seeds 1–50 for three independent iterative debugging runs with Claude Opus 4.6 in Claude Code. Each run starts from a clean skill library with no prior knowledge carried over.

Task	Held-Out Success Rate						
	Run 1	Run 3	Run 4	Mean	Std	Min	Max
Open middle drawer	0%	100%	0%	33.3%	47.1%	0%	100%
Bowl on stove	8%	0%	0%	2.7%	3.8%	0%	8%
Wine bottle on cabinet	100%	96%	96%	97.3%	1.9%	96%	100%
Open drawer + bowl inside	62%	0%	0%	20.7%	29.2%	0%	62%
Bowl on cabinet top	96%	80%	96%	90.7%	7.5%	80%	96%
Push plate to stove	0%	0%	0%	0.0%	0.0%	0%	0%
Cream cheese in bowl	100%	100%	100%	100.0%	0.0%	100%	100%
Turn on stove	100%	100%	100%	100.0%	0.0%	100%	100%
Bowl on plate	92%	100%	100%	97.3%	3.8%	92%	100%
Wine bottle on rack	70%	98%	100%	89.3%	13.7%	70%	100%
<i>Mean</i>	<i>62.8%</i>	<i>67.4%</i>	<i>59.2%</i>	<i>63.1%</i>	<i>3.4%</i>	<i>59.2%</i>	<i>67.4%</i>

F. Case Study Details

Table 26: **Case-study success rates.** Success over 50 held-out seeds for each model on the case-study tasks, from the case-study runs (separate from the per-task tables in Appendix E). L90 denotes LIBERO-90; Goal, Spatial, and Long denote LIBERO-Pro suites, with Pos and Task denoting the position and task perturbations. Opus refers to Claude Opus, Astra refers to GPT-6 Astra. Evo refers to evolutionary search, run on tasks below 90% success rate.

Task	Suite	Opus 4.6	Opus 5	Opus 5.5	Opus 5.5 +Evo	Astra	Astra +Evo
Book under cabinet shelf	L90 #89	0%	2%	68%	100%	22%	96%
Open bottom drawer	L90 #6	0%	0%	90%	90%	92%	100%
Push plate to stove front	Goal #5, Pos	0%	0%	84%	94%	62%	92%
Bowl on stove	Goal #1, Pos	56%	64%	90%	90%	90%	90%
Mug in microwave, close	Long #9, Pos	54%	16%	96%	96%	98%	98%
Plate on stove	Goal #1, Task	0%	54%	100%	100%	100%	100%
Akita bowl on plate	Spatial #2, Task	46%	62%	92%	92%	92%	92%

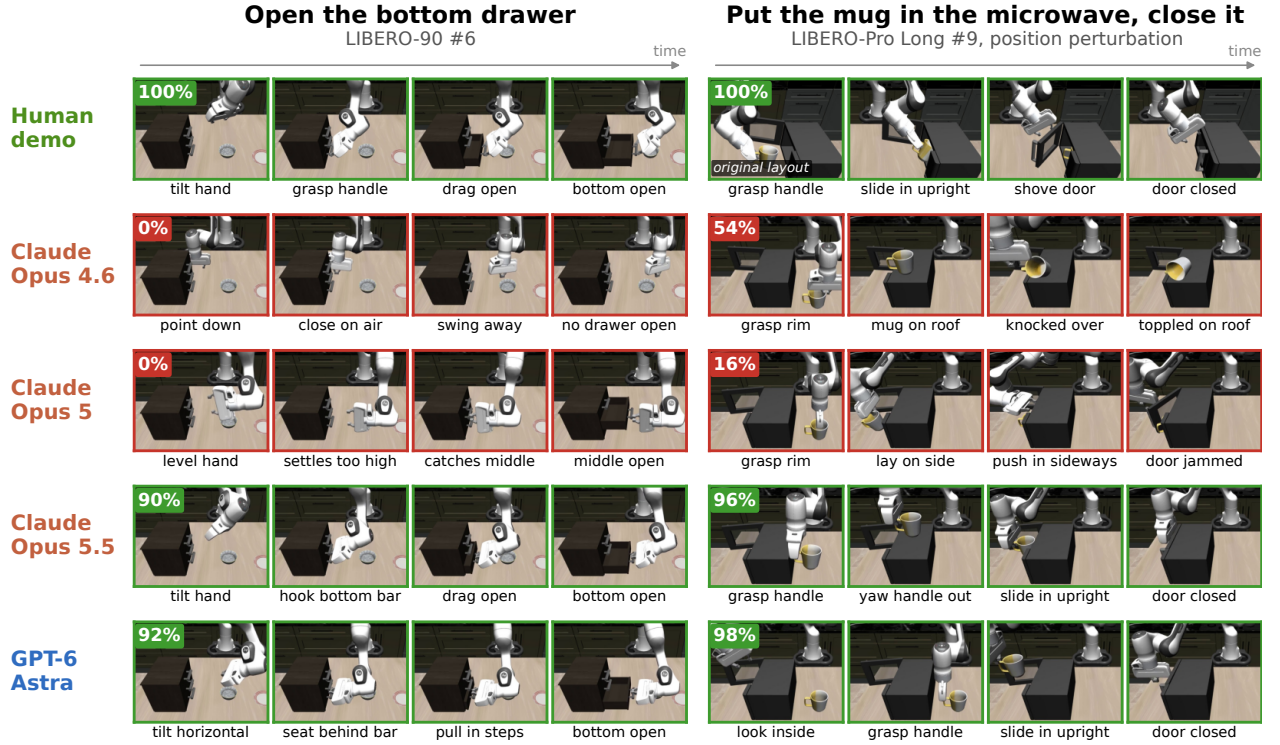


Figure 19: **Case study: two more tasks.** Same layout as Figure 10. The mug demonstration is from the original LIBERO-Long layout, with the microwave on the other side.

Open the bottom drawer. The bottom handle is too low for a level hand. The human tilts the hand and pulls the handle. Opus 5.5 and Astra do the same: they close the gripper, tilt it, hook the fingertips behind the handle, and drag the drawer open (90% and 92%). Opus 5 keeps the hand level, catches the middle handle, and opens the middle drawer (0%). Opus 4.6 never reaches a handle (0%).

Mug in the microwave. Opus 5.5 and Astra grasp the handle and slide the mug in upright, as the human does (96% and 98%); Astra first looks into the microwave with the wrist camera. Opus 5 grasps the rim, so it must lay the mug on its side to fit it through the opening; the handle then sticks out and the door jams (16%). The Opus 4.6 episode treats the microwave as top-loading and leaves the mug on its roof.

G. Agent Pipeline Skills

G.1. Codebase Overview and Skill File Locations

The code is publicly available at <https://github.com/NVlabs/ASPIRE>.

```
ASPIRE/
├── src/cap_harness/
│   ├── api.py, contracts.py ..... public CaP program API
│   ├── run.py, batch.py ..... single-run and batched runners
│   ├── providers/ ..... sam3, graspnet, curobo, pyroki
│   ├── libero/ robosuite/ behavior/ yam_real/ ..... benchmark adapters
│   └── validation/ ..... held-out evaluators
├── docs/ ..... api-reference.md, run-artifacts.md, setup
├── configs/ ..... environment pins, validation profiles
├── scripts/ ..... bootstrap_*.sh, supervise_services.sh
├── .claude/skills/
│   ├── iterative-debugging/ ..... fix loop + skill library
│   └── evosearch/ ..... evolutionary search
├── .codex/skills/ ..... Codex entry points
├── .agents/skills/ ..... System 2 skills
└── AGENTS.md ..... System 2 execution rules
```

Pipeline files. Each pipeline skill is a directory with the same four prompt files, entered through SKILL.md: `main-agent-prompt.md` (coordinator), `subagent-prompt.md` (actor agent: the forbidden-API rules, findings schema, and debugging protocol), `verifier-prompt.md` (iterative-debugging only), and `clean-task-slate.md`. The orchestration code is in `scripts/` (`dispatch.py` fills the templates per task, `run_validation.py` runs the held-out partition, `lint_skills.py` checks a proposal before a verifier is spent on it, `campaign.py` holds campaign settings). BEHAVIOR-1K uses a different protocol, in `iterative-debugging/behavior/`.

Skill library. The skill library is stored in `.claude/skills/iterative-debugging/skills/`: `grasp.md`, `localize.md`, `transport.md`, `manipulation.md`, and `task-exploration.md`, indexed by `skills/README.md`. Actor agents report candidate repairs in a structured findings file (observed failure mode, validated repair, transferable patterns, task-specific quirks, and success rate on debug configurations). A separate verifier agent must reproduce the fix from the proposal alone, and only then does the coordinator promote it. Entries carry provenance (source task, development seeds, held-out result).