STEPS: SCENE-GROUNDED TASK ESCALATION THROUGH PROGRESSIVE SKILL LEARNING TOWARD OPEN-ENDED GOALS

Anonymous authors
Paper under double-blind review

ABSTRACT

Code-as-Policy robot agents increasingly improve through practice, yet evaluation on predefined tasks makes it difficult to distinguish genuine capability acquisition from adaptation to practiced instances. We present STEPS, a framework for evaluating self-improvement on human-specified goals such as *"tidy the office desk"*. STEPS generates physically valid, oracle-solvable scenes; defines the full goal as an L5 task and derives L1-L4 diagnostic tasks probing subtasks, component capabilities, and constraints; and evaluates a frozen agent on the practiced L5 task, pose-perturbed L5 instances, and held-out lower-level tasks. Across five goals and 105 tasks, a skill library developed with human assistance achieves over 90% success on practiced L5 tasks but performs substantially worse on regenerated instances and held-out L3-L4 tasks. On one evolution route, skills distilled from L1-L4 practice usually pass isolated validation but are rarely invoked when solving the full goal. Despite this fitting, some learned capabilities transfer beyond the development scenes to external manipulation tasks, and weaknesses exposed by STEPS provide actionable targets for capability updates that further improve cross-domain performance. These results show that practiced-task success alone can overstate capability acquisition and should be complemented with goal-derived diagnostics that reveal what transfers, where improvement breaks down, and what capability to target next.

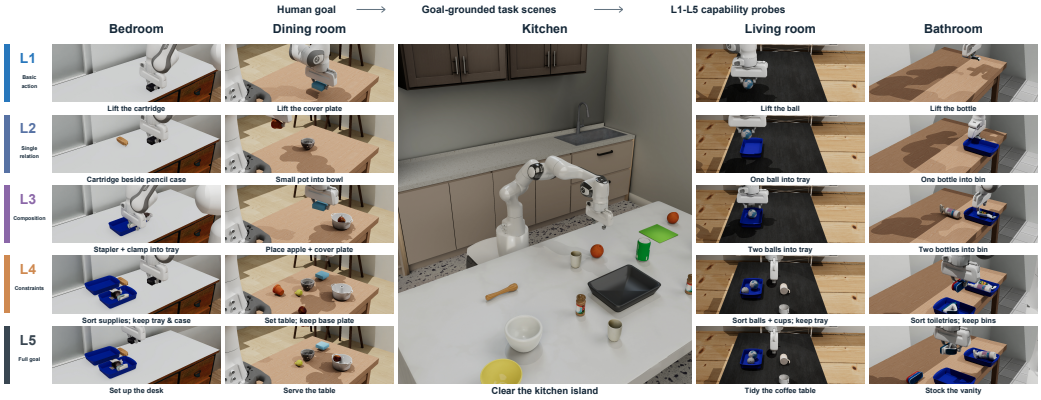

Figure 1: From open-ended human goals, the framework constructs a L5 task scene and derives L1–L4 variants that probe individual skills, compositional relations, and constraints. Skills learned on the L5 task are then evaluated across regenerated L5 instances and lower-level tasks to assess generalization.

1 INTRODUCTION

Self-improvement has become a central recipe for language-model agents, which attempt tasks, learn from execution feedback, and keep what works as reusable skills Wang et al. (2023a). The

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

same recipe now drives robot manipulation, where Code-as-Policy agents write programs against perception and control interfaces Liang et al. (2023); Singh et al. (2023); Fu et al. (2026) and improve by repairing them from grounded feedback and distilling skill libraries Zhang et al. (2026a); Lu et al. (2026). However, evaluating such improvement is a unique challenge. Humans give robots goals such as *"tidy the office desk"*, which specify an outcome over objects and their relations rather than a layout or a success program, whereas benchmarks fix their tasks in advance and rarely contain the goal a user cares about Liu et al. (2023); Nasiriany et al. (2024); Mu et al. (2025). Even a suitable task turns into practice: the programs and skills of the agent accumulate around the scored instance, and the gains of self-evolving agents are often fragile beyond the conditions under which they emerged Wu et al. (2026b); Shi et al. (2026); Wu et al. (2026a). As a result, while self-improving agents report steady gains on the tasks they practice, much less is understood about whether these gains reflect the capabilities a goal requires.

These challenges raise the following question: does practice on a human goal improve an agent at the goal, or only at the practiced instance? Our central observation is that the goal, not a single task, should be the unit of evaluation: a goal induces tasks from lifting one of its objects to satisfying all of its relations (Fig. 1), and improvement on the goal should widen the range of tasks the agent solves. A natural way to instantiate this idea is to generate task scenes from the goal in simulation, develop the agent on the complete goal, and hold out the tasks derived from it. However, generating such tests is hindered by two key challenges. The task scenes must suit the goal: its objects and their supports must be present, physically valid, and reachable, and a success condition written by a language model may not measure what the goal asks for. Moreover, improvement must be told apart from fitting to the practiced instance, which requires held-out tasks traced to the goal and skills traced from proposal to execution.

Motivated by these challenges, we introduce STEPS as a framework for evaluating robot self-improvement on human goals. **Human-Directed Task-Scene Generation** plans from the goal the support surfaces and the objects bound to them, lays out the supports in functional zones and wall runs that keep the floor clear, and accepts the scene only if it passes layered physical checks and a privileged oracle solves the goal; accepted scenes are regenerated under controlled pose perturbations. **L5-Centered Hierarchical Task Design** compiles the goal relations into the success condition of an L5 task from verified templates and derives from the same relations L1-L4 tasks, each labeled a goal subtask, a component-capability probe, or a constraint probe. **Self-Improving Code-as-Policy Evaluation** lets an agent develop its skill library on the L5 task without restriction, then evaluates the frozen agent on the L5 task, its regenerated instances, and the held-out L1-L4 tasks, which serve as a measurement axis rather than a curriculum (Fig. 2). As Table 1 shows, only STEPS both derives its tasks from a human goal and holds out tasks derived from the same goal. Applied to one Code-as-Policy system, it reveals that a skill library nearly saturating the L5 score of each goal still fails many held-out tasks and regenerated instances of that goal.

This paper makes the following contributions: 1) **Goal-conditioned task scenes.** We convert an open-ended human goal into a validated and regenerable task scene and an L5-centered task hierarchy whose success conditions are compiled from the goal relations. 2) **Evaluation protocol.** We separate unrestricted development on the L5 task from frozen evaluation on it, its regenerated instances, and its held-out L1-L4 tasks, and trace each learned skill from proposal to execution with stage records. 3) **Empirical findings.** Across five scenes and 105 tasks, the library, developed with human assistance on the L5 goals, solves 24 of 25 L5 trials but only 61.6% and 40.8% of the held-out L3 and L4 trials (Table 3), and 58% and 31% of L5 trials under small and medium pose perturbations; on the evolution routes we tested, skills distilled from L1-L4 practice mostly pass validation but are rarely executed at the goal. Beyond diagnosis, we further test whether the learned capabilities transfer beyond the development scenes and whether the weaknesses exposed by STEPS provide actionable targets for capability updates that improve performance on external tasks.

2 Related Work

2.1 Scene and Task Generation for Embodied Simulation

Language-guided and agentic methods generate indoor scenes with semantic control, tool use, and iterative refinement (Yang et al., 2024; Sun et al., 2025), with recent systems extending generation toward simulation-ready scenes and task-driven worlds (Yang et al., 2025; Xia et al., 2026; Pfaff

Table 1: **Comparison of capabilities for goal-driven self-improvement.** ✓: provided; ✗: not provided.

	Benchmarks		Scene generation		Task generation		Self-improving agents		
	LIBERO-PRO	RoboCasa	Holodeck	SceneWeaver	RoboGen	GenSim	RATs	ASPIRE	**STEPS**
Task scenes that suit the goal									
Room-scale scenes	✗	✓	✓	✓	✗	✗	✗	✗	✓
Scenes generated from language	✗	✗	✓	✓	✓	✓	✗	✗	✓
Tasks derived from a human goal	✗	✗	✗	✗	✗	✓	✗	✗	✓
Solvability checked by an oracle	✗	✗	✗	✗	✗	✓	✗	✗	✓
Improvement told apart from fitting to the practiced instance									
Perturbed instances of each task	✓	✓	✗	✗	✗	✓	✓	✓	✓
Held-out tasks derived from the goal	✓	✗	✗	✗	✗	✗	✗	✗	✓
Skills traced from proposal to execution	✗	✗	✗	✗	✗	✗	✗	✓	✓

et al., 2026; Wang et al., 2025). Task-generation methods synthesize executable tasks, decompositions, rewards, and demonstrations (Wang et al., 2023b; Hua et al., 2025; Katara et al., 2023; Wang et al., 2023c). GenSim (Wang et al., 2023b) also supports goal-directed curricula that connect generated practice to a target task. Adaptive curricula organize experience according to current competence through initial-state expansion, intermediate-difficulty goals, or generated environments (Ma et al., 2024). Benchmarks also specify goals as predicates over object states (Li et al., 2024a) and probe generalization in tiers (Jiang et al., 2023; Garcia et al., 2025), under systematic perturbations (Pumacay et al., 2024), or through decomposed diagnostic evaluation (Li et al., 2024b), but they fix these tests in advance. STEPS also generates scenes and tasks, but from a fixed human goal, and uses the derived tasks to measure what practice on the goal produces rather than to direct it.

2.2 Skill Acquisition and Agentic Robot Execution

Language-conditioned systems ground plans in skill affordances (Ahn et al., 2022), programs over predefined APIs (Liang et al., 2023; Singh et al., 2023), or spatial value maps (Huang et al., 2023), and agentic frameworks add execution feedback, verification, and recovery (Fu et al., 2026; Fang et al., 2026). Methods that learn from robot experience differ mainly in where knowledge is retained. Test-time methods adapt models or plans during deployment (Jiang et al., 2026; Li et al., 2026a). Parametric methods update policy weights from self-generated data (Bousmalis et al., 2023), learned rewards (Seyed Ghasemipour et al., 2025), fleet-scale RL (Wang et al., 2026b), or world-model rollouts (Guo et al., 2026); RoboClaw (Li et al., 2026b) automates self-resetting practice for developer-specified skills, and ENPIRE (Xiao et al., 2026) lets a coding agent rewrite per-task training pipelines. Non-parametric methods accumulate textual feedback, skill documentation, context harnesses, or corrective lessons around a frozen model (Madaan et al., 2023; Shinn et al., 2023; Ju et al., 2026; Wang et al., 2026a; Liu et al., 2026). Others maintain executable skill libraries: Voyager (Wang et al., 2023a) expands skills through an exploration-maximizing curriculum, RATs (Zhang et al., 2026a) through self-directed play, and ASPIRE (Lu et al., 2026) by distilling validated repairs from diversity-driven searches. Execution harnesses improve target tasks with given capabilities: Harness VLA (Zhang et al., 2026b) learns primitive success boundaries, GaP (Chen et al., 2026) rewires computation graphs over a skill library, and Zetta (Ding et al., 2026) evolves critics and recovery skills around a frozen policy. These methods report gains on the tasks they practice or on fixed benchmarks, and open-ended self-evolution often yields fragile gains (Wu et al., 2026b; Shi et al., 2026; Wu et al., 2026a). STEPS is complementary: rather than another acquisition method, it provides goal-derived tests on which any of them can be developed, validated, and analyzed (Sec. 3.3).

3 Method

Method Overview. A self-improving agent turns any fixed test into practice, after which success on the test no longer separates what the agent has acquired from what it has fitted to that instance. STEPS therefore keeps the goal fixed and regenerates its tests. Given a goal, it *generates* a validated task scene (Sec. 3.1), *compiles* the goal into an L5 task and *derives* held-out L1-L4 tasks from it (Sec. 3.2), and *evaluates* a Code-as-Policy agent whose skill library is frozen after development (Sec. 3.3; Fig. 2).

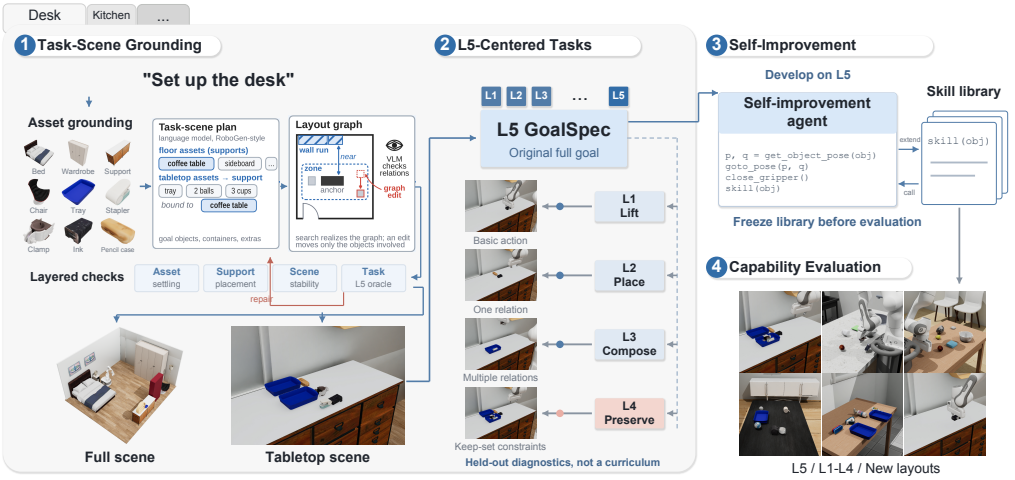

Figure 2: **Overview of STEPS on the office goal. (1)** A language model plans floor assets and the tabletop assets bound to them; the floor layout follows a graph of zones, wall runs, and relations that a vision-language model corrects by graph edits, and the scene is accepted after layered validation and regenerated under pose perturbations. **(2)** The goal relations define the L5 success condition, from which labeled L1-L4 tasks are derived. **(3)** The library is developed on the L5 task only and frozen before evaluation on all tasks and instances. Numbers are from Sec. 4 (86%: L3 goal subtasks).

3.1 Human-Directed Task-Scene Generation

Building upon RoboGen Wang et al. (2023c), whose language model proposes a task before generating its scene, STEPS takes the goal and the room type from a human and lets the language model interpret the goal instead. The resulting plan separates floor assets, which provide the support surfaces, from tabletop assets, namely the goal objects, their containers, and other goal-relevant objects, and binds each tabletop asset to its support.

The floor assets are laid out first, in functional zones and continuous wall runs that keep the floor clear for the robot to reach the supports. Zones, wall runs, and relations such as *near*, *side of*, or *around* form a layout graph, which a search in the style of Holodeck Yang et al. (2024) realizes while keeping doors and windows clear. A vision-language model checks the relations before placement and on rendered views afterward, and each of its corrections is applied as an edit of named objects in the graph, after which only those objects are placed again. We refer to this procedure as **graph-edit correction**; it keeps the parts of the layout that already pass. The tabletop assets are then placed on their bound supports.

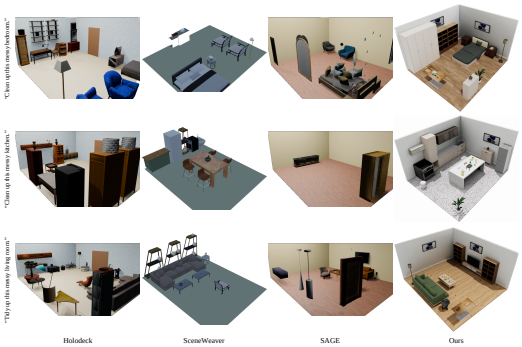

Figure 3: **Scenes generated from the same instruction.** One scene per method and room type, generated from the instruction on the left. Compared with the baselines, STEPS generates scenes with more realistic layouts and more complete content, and follows the instructions more faithfully.

Agentic generators refine scenes with critics of semantics, appearance, and stability Yang et al. (2025); Xia et al. (2026). Because the scenes of STEPS must also serve as tasks, its **layered validation** extends such checks from the assets to the task: each movable asset must settle upright on a plane, each placed object must rest on its support without overlap, the assembled scene must stay at rest without interpenetration (tol-

erances in Appendix A), and a privileged oracle must solve the L5 task, which shows that the robot can reach every goal object and target. The accepted scene is frozen and regenerated by perturbing the poses of its movable objects (Sec. 4.4).

3.2 L5-Centered Hierarchical Task Design

An open-ended goal specifies an outcome rather than a success program, and a success condition written by a language model may not measure what the goal asks for. STEPS therefore compiles every success condition from verified templates of atomic verbs, such as lifting, placing, stacking, and pushing, each of which also provides an oracle procedure. The goal is formalized as groups of goal objects, each assigned a template relation such as *inside*, *near*, *covering*, or *in a row*, and the L5 task succeeds when all relations hold and the gripper is empty.

From the same relations, STEPS derives tasks in which every object starts from its L5 pose: L1 lifts or pushes one goal object and L2 and L3 satisfy the relations of one or several objects, all in reduced scenes, while L4 combines groups in the full scene, where targets and unselected goal objects must stay in place. A language model only selects five tasks per level, and tasks that the privileged oracle fails to solve are flagged rather than removed. Each task is traceable to the goal as a *goal subtask* that every L5 solution also completes, a *component-capability probe* of a capability that L5 solutions exercise but L5 success does not reveal, or a *constraint probe* that adds a requirement the goal does not impose.

The hierarchy is thus a diagnostic interface rather than a curriculum. Because every derived task starts from the L5 poses, failing it cannot be blamed on a harder layout, and its label states what the failure means: an agent that solves the L5 task but fails one of its goal subtasks lacks a capability that the L5 score does not certify.

3.3 Self-Improving Code-as-Policy Evaluation

Following CaP-X Fu et al. (2026), the agent writes programs against a primitive-level interface for perception, grasping, motion, and navigation, without task-level verbs such as *place* or *stack*, and each program is scored by the success condition of its task. The agent improves by extending a skill library of parameterized functions, each with a short usage description, that call only interface and library functions. To keep evaluation from turning into practice, STEPS adopts a **develop-then-freeze protocol**: the library is developed on the L5 task without restriction while the L1-L4 tasks are held out, and the frozen agent is then run on the L5 task, its regenerated instances, and its L1-L4 tasks. The L1-L4 tasks attribute the L5 score to the capabilities that the goal requires, and the regenerated instances test whether it survives perturbed object poses.

When an evolution route instead learns skills on some L1-L4 tasks, STEPS keeps **stage records** of whether each skill is proposed from a successful episode, validated on held-out episodes of its source task, offered at L5, and executed there, so that a change at L5 can be traced to the stage that produced it.

4 Experiments

We first compare the task scenes of STEPS with those of three scene generators. We then use STEPS to evaluate one system, a Code-as-Policy agent with a skill library, and ask three questions. **Q1:** Does the L5 score hold on regenerated instances of the goal? **Q2:** What do the held-out L1-L4 tasks add to the L5 score? **Q3:** Can the stage records locate where an evolution route stops contributing at the goal? We also report the same system on RoboSuite and LIBERO-PRO to situate it.

4.1 Task-Scene Generation

We compare STEPS with Holodeck Yang et al. (2024), SceneWeaver Yang et al. (2025), and SAGE Xia et al. (2026) on three common room types, a bedroom, a kitchen, and a living room, each generated from a short clean-up instruction (Fig. 3). For each method and each room type, we generate 10 independent scenes and report metrics averaged over these 10 runs. To ensure a fair comparison, all methods retrieve 3D assets from the same standardized asset library. Following the

Table 2: Scene-generation evaluation on common scene types. Scores are averaged across scenes for each room type and reported as mean ± sample standard deviation.

Room Type	Method	Visual					Physics	
		Real. ↑	Func. ↑	TF ↑	Lay. ↑	Comp. ↑	Coll. % ↓	Stab. % ↑
Bedroom	Holodeck Yang et al. (2024)	5.44 ± 0.53	6.33 ± 0.71	7.11 ± 3.37	6.44 ± 0.53	7.11 ± 0.60	13.48 ± 8.93	73.72 ± 13.42
	SceneWeaver Yang et al. (2025)	5.20 ± 0.63	6.10 ± 0.74	5.70 ± 1.64	6.60 ± 0.52	5.80 ± 1.23	15.28 ± 20.79	68.32 ± 12.90
	SAGE Xia et al. (2026)	6.83 ± 0.41	7.33 ± 1.03	7.11 ± 3.26	5.17 ± 0.41	6.00 ± 1.63	6.20 ± 10.51	75.18 ± 19.70
	STEPS (Ours)	7.00 ± 0.32	8.00 ± 0.21	10.00 ± 0.00	7.90 ± 0.32	8.10 ± 0.57	0.00 ± 0.00	90.00 ± 1.25
Kitchen	Holodeck Yang et al. (2024)	4.00 ± 0.63	3.00 ± 1.10	5.50 ± 3.73	4.83 ± 0.41	3.67 ± 1.37	14.00 ± 8.63	74.35 ± 12.96
	SceneWeaver Yang et al. (2025)	3.90 ± 0.99	3.70 ± 1.77	4.30 ± 2.16	5.10 ± 1.45	3.30 ± 1.77	1.25 ± 3.95	68.35 ± 23.85
	SAGE Xia et al. (2026)	4.38 ± 0.18	5.00 ± 0.63	7.00 ± 3.46	4.00 ± 0.63	4.50 ± 0.71	0.00 ± 0.00	96.67 ± 8.16
	STEPS (Ours)	6.00 ± 0.56	6.90 ± 0.32	9.85 ± 0.34	7.10 ± 0.32	7.70 ± 0.48	0.00 ± 0.00	90.83 ± 6.69
Living Room	Holodeck Yang et al. (2024)	5.33 ± 0.50	6.56 ± 0.53	9.22 ± 2.33	6.33 ± 0.71	7.00 ± 1.00	22.02 ± 14.88	71.98 ± 15.43
	SceneWeaver Yang et al. (2025)	3.90 ± 0.99	5.30 ± 1.06	7.00 ± 2.59	6.20 ± 0.92	4.70 ± 0.95	8.22 ± 16.51	65.19 ± 20.21
	SAGE Xia et al. (2026)	4.90 ± 1.45	6.00 ± 0.82	8.44 ± 2.79	4.20 ± 1.03	5.00 ± 1.33	0.00 ± 0.00	90.37 ± 12.41
	STEPS (Ours)	7.00 ± 0.36	7.80 ± 0.42	10.00 ± 0.00	8.00 ± 0.00	7.80 ± 0.42	0.00 ± 0.00	92.14 ± 2.67

evaluation protocol of Yang et al. (2025), we assess both visual quality and physical plausibility. Visual quality is evaluated from rendered scene views using GPT-5.5, which assigns scores from 1 to 10 for Realism (Real.), Functionality (Func.), Text Fidelity (TF), Layout (Lay.), and Completeness (Comp.). Physical plausibility is measured by the collision ratio (Coll.), defined as the percentage of object meshes involved in collisions, and the stability ratio (Stab.), defined as the percentage of objects that remain stable in simulation. As in prior work, an object is considered unstable if, after 120 simulation steps, its relative translation exceeds 0.2 m or its rotation exceeds 8°. For STEPS, the vision-language model used during scene generation is GPT-5.6 Sol; this model is separate from the GPT-5.5 evaluator used for metric scoring.

As Table 2 shows, STEPS scores highest on every visual criterion in all three room types, and its visual scores vary little across scenes, with standard deviations of at most 0.57. Averaged over the room types, its margin over the best baseline is largest on text fidelity (2.2 points), followed by layout and completeness (1.7 each), which the goal-conditioned asset plan and the zones and wall runs of Sec. 3.1 address; in the kitchen, it reaches 7.10 on layout and 7.70 on completeness, against at most 5.10 and 4.50 for the baselines. Fig. 3 illustrates the difference: the STEPS scenes place large furniture along the walls with open floor around it and set small objects on tables and counters, the objects that a clean-up instruction refers to, whereas the SAGE kitchen, for example, contains only two pieces of furniture. On the physical criteria, STEPS is the only method without colliding objects in all three room types, and 90–92% of its objects remain stable, against 65-74% for Holodeck and SceneWeaver. SAGE, which also checks its scenes in a physics simulator, has no collisions in the kitchen and the living room and is the most stable in the kitchen (96.67% against 90.83%), but scores lower on completeness there and in the living room (4.50 and 5.00, against 7.70 and 7.80). STEPS thus pairs collision-free scenes with the most complete content in every room type.

4.2 Evaluation Setup

Benchmark. The five goals are set in an office, a kitchen, a dining room, a living room, and a bathroom (Fig. 4). Each goal yields one L5 task and five tasks at each of L1-L4, or 105 tasks. Of the 100 L1-L4 tasks, 46 are goal subtasks, 29 are component-capability probes, and 25 are constraint probes. Physics is simulated in MuJoCo Todorov et al. (2012), and Figs. 2 and 4 are rendered with the Nyx path tracer from Genesis Team (2025).

Metrics. We report success rates with 95% Wilson intervals. Trials of a task share its initial layout, so these intervals understate the uncertainty across tasks, and we read differences between single tasks as observations rather than estimates. Table 3 reports our first run of all 105 tasks, five trials each; the reruns in Q1 and Q2 use a runtime in which faults identified in the first run are fixed (Sec. 4.4).

4.3 Development Saturates the L5 Score

Our first question is whether development can saturate the L5 score (Q1). STEPS places no restriction on development, since a self-improving agent, or its developers, can practice a fixed task until

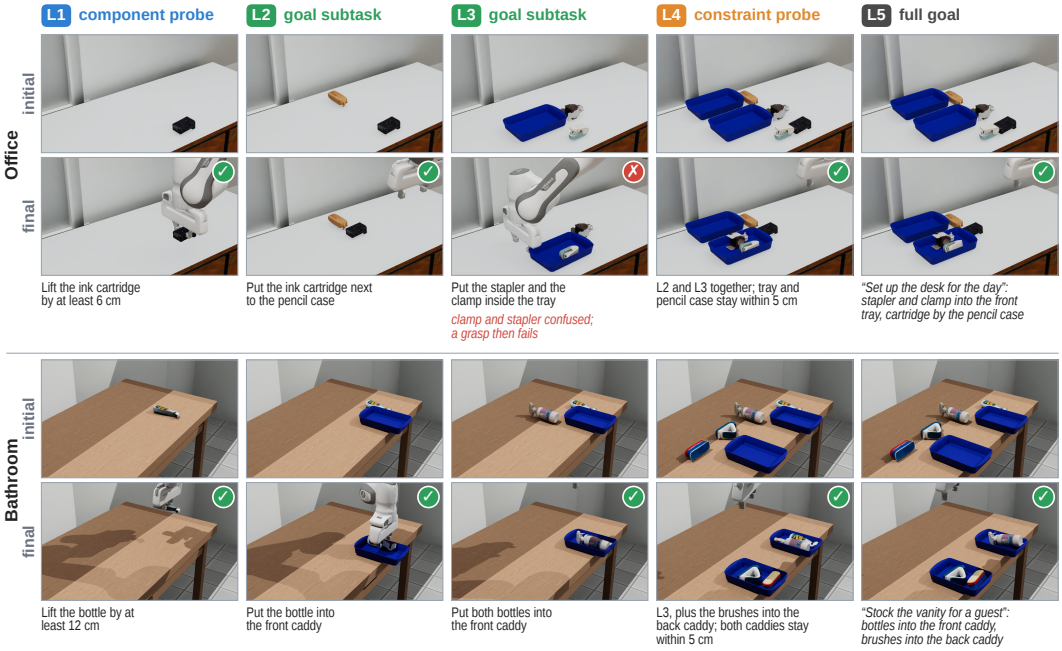

Figure 4: **Two goals across five levels.** One trial per level for the office (top) and bathroom (bottom) goals, showing task instructions and initial/final states. L1-L3 use reduced scenes and L4-L5 the full scene, all from the L5 poses. The office L3 subtask fails although L4 and L5 succeed; all bathroom trials succeed. Labels indicate task type, and marks indicate success. Appendix A.2 shows all five rooms.

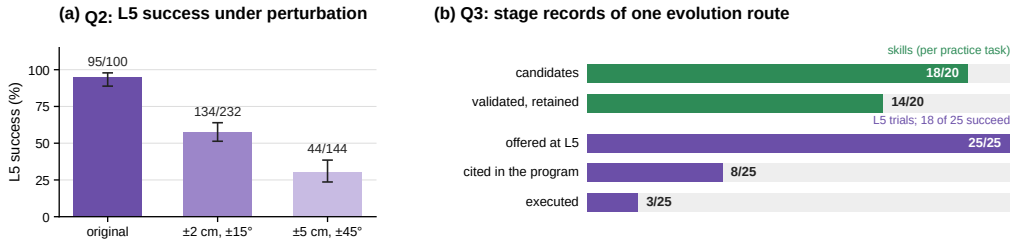

Figure 5: **(a) Q2:** L5 success of the frozen agent on the original L5 tasks and on their regenerated instances, pooled over the five rooms; labels give successes over trials, and error bars are 95% Wilson intervals. **(b) Q3:** Stage records of one evolution route. Green bars trace practice tasks from candidate generation to validation; purple bars trace retained skills from being offered at L5 to citation and unmodified execution.

it is solved. The harness agent developed the skill library in this way on the five goals (Sec. 4.2). As shown in Table 3, the frozen library solves the L5 tasks in 24 of 25 trials, and in 185 of 200 trials (92.5%) over two reruns with the fixed runtime, with at least 35 of 40 in every room (Appendix A.1). By this score, development has solved the benchmark.

Inspecting the library, however, shows that many of its helpers implement complete goal-specific routines and retain bindings to the objects used during development. A saturated L5 score therefore certifies a solution to the developed instance of each goal, but not necessarily the underlying capabilities. The question is whether STEPS can reveal this from the behavior of the frozen agent alone.

Table 3: **Scene-wise in-domain performance on STEPS.** Each cell reports successes/trials (success rate). Best results within each scene are shown in bold.

Scene	Method	L1	L2	L3	L4	L5	Overall
Office	STEPS	24/25 (96%)	**25/25 (100%)**	**15/25 (60%)**	11/25 (44%)	**5/5 (100%)**	80/105 (76.2%)
	ASPIRE	**25/25 (100%)**	24/25 (96%)	**15/25 (60%)**	**18/25 (72%)**	3/5 (60%)	**85/105 (81.0%)**
	CaP-Agent0	**25/25 (100%)**	16/25 (64%)	4/25 (16%)	0/25 (0%)	0/5 (0%)	45/105 (42.9%)
Kitchen	STEPS	**15/25 (60%)**	**20/25 (80%)**	**18/25 (72%)**	**13/25 (52%)**	**5/5 (100%)**	**71/105 (67.6%)**
	ASPIRE	10/25 (40%)	13/25 (52%)	3/25 (12%)	3/25 (12%)	1/5 (20%)	30/105 (28.6%)
	CaP-Agent0	9/25 (36%)	6/25 (24%)	1/25 (4%)	0/25 (0%)	0/5 (0%)	16/105 (15.2%)
Dining	STEPS	17/25 (68%)	**24/25 (96%)**	5/25 (20%)	**5/25 (20%)**	**5/5 (100%)**	**56/105 (53.3%)**
	ASPIRE	**24/25 (96%)**	19/25 (76%)	**8/25 (32%)**	0/25 (0%)	2/5 (40%)	53/105 (50.5%)
	CaP-Agent0	22/25 (88%)	9/25 (36%)	0/25 (0%)	0/25 (0%)	0/5 (0%)	31/105 (29.5%)
Living	STEPS	22/25 (88%)	13/25 (52%)	14/25 (56%)	8/25 (32%)	**5/5 (100%)**	62/105 (59.0%)
	ASPIRE	**24/25 (96%)**	**19/25 (76%)**	**20/25 (80%)**	**13/25 (52%)**	3/5 (60%)	**79/105 (75.2%)**
	CaP-Agent0	**24/25 (96%)**	6/25 (24%)	0/25 (0%)	0/25 (0%)	0/5 (0%)	30/105 (28.6%)
Bathroom	STEPS	**24/25 (96%)**	**25/25 (100%)**	**25/25 (100%)**	**14/25 (56%)**	**4/5 (80%)**	**92/105 (87.6%)**
	ASPIRE	11/25 (44%)	10/25 (40%)	2/25 (8%)	3/25 (12%)	0/5 (0%)	26/105 (24.8%)
	CaP-Agent0	14/25 (56%)	10/25 (40%)	2/25 (8%)	1/25 (4%)	0/5 (0%)	27/105 (25.7%)
Total	STEPS	**102/125 (81.6%)**	**107/125 (85.6%)**	**77/125 (61.6%)**	**51/125 (40.8%)**	**24/25 (96.0%)**	**361/525 (68.8%)**
	ASPIRE	94/125 (75.2%)	85/125 (68.0%)	48/125 (38.4%)	37/125 (29.6%)	9/25 (36.0%)	273/525 (52.0%)
	CaP-Agent0	94/125 (75.2%)	47/125 (37.6%)	7/125 (5.6%)	1/125 (0.8%)	0/25 (0.0%)	149/525 (28.4%)

4.4 Held-Out Tasks Reveal the Fitting

We ask whether held-out tasks and regenerated instances can distinguish capability from fitting (Q2).

Held-out tasks. A goal subtask starts from the L5 poses and is satisfied by every successful solution of its L5 task (Sec. 3.2), so it is feasible by construction rather than defined by our learned skills. The skill-free baseline also solves many lower-level probes (Table 3), confirming that their solvability does not depend on the developed library. Yet, against 24/25 successes at L5, L2 goal subtasks succeed in 104/110 trials, while L3 subtasks that combine several object relations succeed in only 74/115. L4 constraint probes are lower still at 40.8% overall. Thus, high L5 success can coexist with unreliable constituent capabilities.

The held-out tasks also expose runtime faults hidden by the developed L5 instances. After fixing them, the gap narrows but remains: L3 goal subtasks succeed in 138/161 trials versus 185/200 at L5 (Table 5). Restating the full goals in the same template gives 32/35 successes, ruling out the task template as the main explanation.

Regenerated instances. Held-out tasks keep the poses and change what is asked; regenerated instances keep the goal and change the poses. After freezing the library, we perturb movable objects by up to ± 2 cm and $\pm 15^\circ$ (small), or ± 5 cm and $\pm 45^\circ$ (medium), rejecting invalid or already-solved instances. Success falls from 95% on the original L5 tasks to 58% and 31% under small and medium perturbations, respectively (Fig. 5a). Together, these tests show that an L5 score above 90% certifies the developed instances, but not by itself the capabilities required by the goal.

4.5 The Task Scenes Yield Transferable Skills

We next ask whether the task scenes, despite this fitting, yield reusable skills (Q3). Beyond object-specific bindings, the learned routines

Table 4: **RoboSuite evaluation.** Entries report successes/planned trials (success rate). The seven-task RATs subset comprises the three Cube tasks, Nut assembly, Spill wiping, Two-arm handover, and Two-arm lifting; the three NutAssembly variants and the lower Pick-and-Place block are additional evaluations.

Task	RATs	STEPS	STEPS (Update)
Standard manipulation tasks			
Cube lifting	42/50 (84%)	49/50 (98%)	50/50 (100%)
Cube restacking	23/50 (46%)	34/50 (68%)	34/50 (68%)
Cube stacking	30/50 (60%)	48/50 (96%)	48/50 (96%)
Nut assembly	0/50 (0%)	0/50 (0%)	0/50 (0%)
Nut Assembly Square	0/50 (0%)	15/50 (30%)	15/50 (30%)
Nut Assembly Round	0/50 (0%)	0/50 (0%)	0/50 (0%)
Nut Assembly Single	0/50 (0%)	0/50 (0%)	0/50 (0%)
Spill wiping	50/50 (100%)	34/50 (68%)	40/50 (80%)
Two-arm handover	10/50 (20%)	27/50 (54%)	27/50 (54%)
Two-arm lifting	17/50 (34%)	28/50 (56%)	28/50 (56%)
Pick-and-Place tasks			
PickPlaceMilk	12/50 (24%)	35/50 (70%)	35/50 (70%)
PickPlaceCan	33/50 (66%)	27/50 (54%)	40/50 (80%)
PickPlaceBread	14/50 (28%)	42/50 (84%)	42/50 (84%)
PickPlaceCereal	1/50 (2%)	39/50 (78%)	42/50 (84%)
PickPlaceSingle	16/50 (32%)	35/50 (70%)	36/50 (72%)
PickPlace (four objects)	0/50 (0%)	11/50 (22%)	11/50 (22%)
PickPlace subtotal	76/300 (25.3%)	189/300 (63.0%)	206/300 (68.7%)

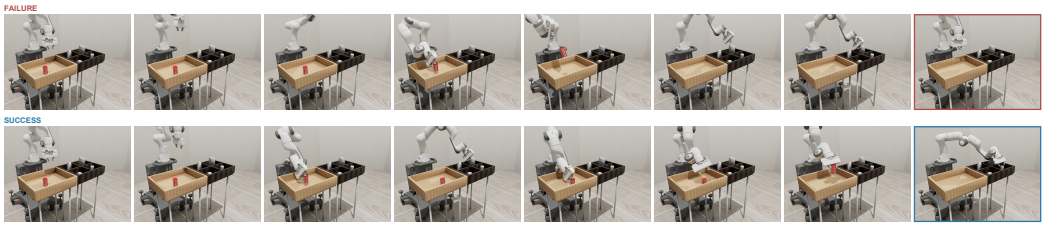

Figure 6: **A capability update recovers a RoboSuite execution.** From the same PickPlaceCan initial state, the original strategy fails, whereas the updated old–new composition succeeds. This is a qualitative recovery example; aggregate gains are reported in Table 4.

encode geometric knowledge such as collision-free lifting, container placement, grasp orientation, and supported pushing. The harness agent abstracts these routines into skill cards over the primitive interface.

Cross-domain transfer. We evaluate a Code-as-Policy agent with the frozen cards on RoboSuite Zhu et al. (2020), whose tasks and objects differ from the development scenes, and compare against RATs Zhang et al. (2026a). On the seven RoboSuite tasks reported by RATs, our agent succeeds in 220/350 trials versus 172/350 for RATs; on six Pick-and-Place tasks, it achieves 189/300 versus 76/300 (Table 4). In a matched pilot on two RoboSuite and two LIBERO tasks, adding the cards raises success from 19/72 to 34/72. Thus, fitting to the developed L5 instances does not imply that the library contains no reusable capability: part of the learned knowledge transfers to tasks on which it was not developed.

4.6 Stage Records Guide Capability Updates

Finally, we ask whether the weaknesses exposed by stage records can guide further capability updates (Q4). We identify lower-level predicates that were unreliable in the held-out evaluation and construct derived long-horizon tasks that preserve each weak predicate as a mandatory first stage followed by two ordered extensions. Of 37 statically accepted derived L5 tasks, 35 yield valid baseline executions, of which the frozen library succeeds on only 2. The stage records therefore turn hidden lower-level weaknesses into targeted long-horizon probes.

We use failures on these probes to repair and extend the library, then evaluate the updates on RoboSuite without further adaptation. Pick-and-Place success rises from 189/300 (63.0%) to 206/300 (68.7%), including PickPlaceCan from 27/50 (54%) to 40/50 (80%); Spill Wiping also improves from 34/50 (68%) to 40/50 (80%) (Table 4). Thus, stage records not only reveal where capability is missing but also provide concrete targets whose repair can improve performance beyond the tasks used for development.

5 Conclusion

We presented STEPS, a framework for evaluating robot self-improvement by separating development on a human goal from frozen evaluation on held-out tasks, regenerated instances, and skill-execution traces. Across five goals and 105 tasks, a Code-as-Policy system exceeds 90% success on developed L5 tasks but generalizes less reliably to held-out L3-L4 tasks and regenerated instances. Its learned skills nevertheless transfer to RoboSuite, where it outperforms RATs on most evaluated tasks, while stage records reveal where apparent self-improvement fails to become executable capability. Together, these results show that practiced-goal success should be interpreted alongside held-out behavior and execution-level evidence.

Limitations. STEPS diagnoses self-improvement but does not solve it: we have not yet found an evolution route that reliably improves goal-level performance. We also focus on interpretable Code-as-Policy agents; extending the protocol to learned real-world visuomotor policies remains future work.

AI USE STATEMENT

Generative AI systems were employed at several stages of this study as supporting research tools. Their use included assistance with implementing methods, refining research questions and conceptual ideas, reviewing aspects of the proposed methodology and experimental setup, translating text, and restructuring or cleaning data. We also used these tools for a number of auxiliary tasks that are commonly recommended for disclosure, such as exploring possible experimental settings, generating and revising code or other research materials, brainstorming ideas, locating and summarizing relevant literature, identifying potentially useful research directions and references, improving manuscript drafts, enhancing clarity and readability, and standardizing citation or reference formatting.

Generative AI was not used to carry out qualitative or thematic analysis, nor was it used to interpret the outcomes of the experiments. Mathematical theorem formulation or proof generation was outside the scope of this work.

Any material produced with AI assistance was subsequently examined by the authors before being incorporated into the study. Suggestions related to hypotheses, methodological choices, or experimental configurations were cross-checked with prior literature and, when relevant, further tested through empirical evaluation. Code and implementation details developed with AI assistance were manually inspected and verified using the testing procedures appropriate to the corresponding components, including unit-level, integration, simulation, and end-to-end validation.

For the agent-based data-generation framework introduced in this work, outputs such as synthetic scenes, assets, task specifications, and robot demonstrations were not accepted directly from generative models. Instead, they were evaluated using the geometric, physical, kinematic, task-completion, and data-consistency checks described in the paper. Data preprocessing, cleaning, and reformatting were likewise performed through transparent and inspectable scripts or structured transformation procedures.

When generative AI was used during literature exploration or manuscript preparation, its outputs were treated only as preliminary assistance. Literature summaries, brainstorming results, proposed research gaps, and suggested references were used to support discovery rather than as authoritative sources. All scientific statements and citations ultimately included in the manuscript were verified against the corresponding original publications or other primary materials. Likewise, AI-assisted manuscript text, code, bibliographic entries, and related research artifacts were reviewed, corrected, and revised by the authors prior to inclusion.

The interpretation of experimental findings and the scientific conclusions of this study were determined by the authors rather than by generative AI systems. All reported quantitative results were produced by the implemented experimental pipeline and were not generated, modified, or fabricated by an AI assistant. The authors take full responsibility for the final manuscript, including its claims, text, code, references, and all other research outputs developed with the assistance of generative AI.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gober, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022.

Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X. Lee, Maria Bauza, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, Antoine Laurens, Claudio Fantacci, Valentin Dalibard, Martina Zambelli, Murilo Martins, Rugile Pevceviciute, Michiel Blokzijl, Misha Denil, Nathan Batchelor, Thomas Lampe, Emilio Parisotto, Konrad Żołna, Scott Reed, Sergio Gómez Colmenarejo, Jon Scholz, Abbas Abdolmaleki, Oliver Groth, Jean-Baptiste Regli, Oleg Sushkov, Tom Rothörl, José Enrique Chen, Yusuf Aytar, Dave Barker, Joy Ortiz, Martin Riedmiller, Jost Tobias Springenberg, Raia Hadsell, Francesco Nori, and Nicolas Heess. Robocat: A self-improving generalist agent for robotic manipulation, 2023. URL https://arxiv.org/abs/2306.11706.

Kaiyuan Chen, Shuangyu Xie, Letian Fu, Justin Yu, William Pacini, Sandeep Bajamahal, Hudson Kim, Jaimyn Drake, Daehwa Kim, Haoru Xue, Jonathan Francis, Christian Juette, Pe-

ter Schaldenbrand, Muhammet Yunus Seker, Ruwan Wickramarachchi, Uksang Yoo, Guanzhi Wang, Adithyavairavan Murali, Balakumar Sundaralingam, S. Shankar Sastry, Spencer Huang, Yuke Zhu, Linxi Fan, and Ken Goldberg. Gap: A graph-as-policy multi-agent self-learning harness for variational automation tasks. *arXiv preprint arXiv:2607.05369*, 2026. URL https://arxiv.org/abs/2607.05369.

Xin Ding, Liang Mi, Mingzhe Huang, Zixuan Wang, Chao Zhang, Zixu Hao, Fu Chen, Xiangyu Li, Yikai Zheng, Yaoyu Guo, Weijun Wang, Kun Li, Hao Wu, Yunxin Liu, and Ting Cao. Zetta ζ: An efficient closed-loop embodied harness for self-evolving physical intelligence, 2026. URL https://arxiv.org/abs/2608.16590.

Zhirui Fang, Qingchi Yu, Ziyang Chen, Longfei Li, Haoran Ma, Keru Zhou, Xinrun Xu, Samith Va, Yuxuan Hu, Peixuan Song, Qiang Du, Bin Qian, Yongkang Deng, Xin Li, Yezhen Wang, Zhe Li, Hao Luo, Shuyan Li, Ziwei Wang, Weijian Deng, and Xiu Li. Emerge-policy: A robot mind emerges beyond a single policy, 2026. URL https://arxiv.org/abs/2608.29896.

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.

Ricardo Garcia, Shizhe Chen, and Cordelia Schmid. Towards generalizable vision-language robotic manipulation: A benchmark and llm-guided 3d policy. In *International Conference on Robotics and Automation (ICRA)*, 2025.

Yanjiang Guo, Tony Lee, Lucy Xiaoyang Shi, Jianyu Chen, Percy Liang, and Chelsea Finn. Vlaw: Iterative co-improvement of vision-language-action policy and world model, 2026. URL https://arxiv.org/abs/2602.12063.

Pu Hua, Minghuan Liu, Annabella Macaluso, Yunfeng Lin, Weinan Zhang, Huazhe Xu, and Lirui Wang. Gensim2: Scaling robot data generation with multi-modal and reasoning llms. In *8th Annual Conference on Robot Learning*, 2025.

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. *arXiv preprint arXiv:2307.05973*, 2023.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. In *Fortieth International Conference on Machine Learning*, 2023.

Yunfan Jiang, Yevgen Chebotar, Ruijie Zheng, Fengyuan Hu, Yunhao Ge, Jimmy Wu, Tianyuan Dai, Scott Reed, Li Fei-Fei, Yuke Zhu, and Linxi "Jim" Fan. Robottt: Context scaling for robot policies. *arXiv preprint arXiv: 2607.15275*, 2026.

Ruofei Ju, Xinrui Wang, Xin Ding, Yifan Yang, Hao Wu, Shiqi Jiang, Qianxi Zhang, Hao Wen, Xiangyu Li, Weijun Wang, Kun Li, Yunxin Liu, Haipeng Dai, Wei Wang, and Ting Cao. Embodiskill: Skill-aware reflection for self-evolving embodied agents, 2026.

Pushkal Katara, Zhou Xian, and Katerina Fragkiadaki. Gen2sim: Scaling up robot learning in simulation with generative models, 2023.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael 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)*, 2024a.

Guoliang Li, Ruihua Han, Chengyang Li, He Li, Shuai Wang, Wenchao Ding, Hong Zhang, and Chengzhong Xu. Agentic self-evolutionary replanning for embodied navigation, 2026a. URL https://arxiv.org/abs/2603.02772.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. In *NeurIPS 2024*, 2024b.

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, Yangyang, Guanghui Ren, Maoqing Yao, Wenhao Wang, and Yao Mu. RoboClaw: An agentic framework for scalable long-horizon robotic tasks. *arXiv preprint arXiv:2603.11558*, 2026b.

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.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

Yang Liu, Weixing Chen, Xinshuai Song, Tao Pu, Siwen Mo, Yongjie Bai, Zihao Chen, Qianran Sun, Liruo Zhong, Ying Shen, and Liang Lin. Phyagentos: A self-evolving operating system for embodied agents with decoupled cognitive planning and physical execution, 2026. URL https://arxiv.org/abs/2607.16636.

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

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023.

Yao Mu, Tianxing Chen, Zanxin Chen, Shijia Peng, Zhiqian Lan, Zeyu Gao, Zhixuan Liang, Qiaojun Yu, Yude Zou, Mingkun Xu, Lunkai Lin, Zhiqiang Xie, Mingyu Ding, and Ping Luo. Robotwin: Dual-arm robot benchmark with generative digital twins. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, pp. 27649–27660, June 2025.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems (RSS)*, 2024.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. Scenesmith: Agentic generation of simulation-ready indoor scenes, 2026. URL https://arxiv.org/abs/2602.09153.

Wilbert Pumacay, Ishika Singh, Jiafei Duan, Ranjay Krishna, Jesse Thomason, and Dieter Fox. The colosseum: A benchmark for evaluating generalization for robotic manipulation. 2024.

Seyed Kamyar Seyed Ghasemipour, Ayzaan Wahid, Jonathan Tompson, Pannag Sanketi, and Igor Mordatch. Self-improving embodied foundation models. In *NeurIPS*, 2025. URL https://arxiv.org/abs/2509.15155. Appearing in NeurIPS 2025.

Jiajun Shi, Siyuan Tao, Yuhao Wu, Zexuan Wang, Jingyuan Zhang, Jiaheng Liu, Xinping Lei, Xinrong Zhang, Siyuan Fang, Zhewen Tan, Tianle Cai, Junhao Fang, Jiameng Huang, Yueyang Wang, Jinkai Liu, Yuxuan Zhang, Jian Yang, Zhoujun Li, Shen Yan, Wenhao Huang, and Ge Zhang. S3gym: Can llms turn self-testing and self-judging into self-improvement?, 2026. URL https://arxiv.org/abs/2608.31100.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

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.

Fan-Yun Sun, Weiyu Liu, Siyi Gu, Dylan Lim, Goutam Bhat, Federico Tombari, Manling Li, Nick Haber, and Jiajun Wu. Layoutvlm: Differentiable optimization of 3d layout via vision-language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 29469–29478, 2025.

Genesis AI Team. The role of simulation in scalable robotics: Genesis world 1.0 and the path forward, 2025. URL https://www.genesis.ai/blog/the-role-of-simulation-in-scalable-robotics-genesis-world-10-and-the-path-forward. Accessed: 2026-09-26.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. In *Arxiv*, 2023b.

Peidong Wang, Zhiming Ma, Ying Chang, Xufang Luo, Yiqun Zhang, Zihan Wang, Xiaocui Yang, Shi Feng, Yuqing Yang, and Dongsheng Li. Self-evolving embodied agents via skill-harness evolution, 2026a. URL https://arxiv.org/abs/2608.11350.

Xinjie Wang, Liu Liu, Yu Cao, Ruiqi Wu, Wenkang Qin, Dehui Wang, Wei Sui, and Zhizhong Su. Embodiedgen: Towards a generative 3d world engine for embodied intelligence, 2025. URL https://arxiv.org/abs/2506.10600.

Yi Wang, Xinchen Li, Pengwei Xie, Pu Yang, Buqing Nie, Yunuo Cai, Qinglin Zhang, Chendi Qu, Jeffrey Wu, Jianheng Song, Xinlin Ren, Jingshun Huang, Mingjie Pan, Siyuan Feng, Zhi Chen, and Jianlan Luo. Learning while deploying: Fleet-scale reinforcement learning for generalist robot policies, 2026b. URL https://arxiv.org/abs/2605.00416.

Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023c.

Yuhao Wu, Jingyuan Zhang, Jiajun Shi, Xinping Lei, Qingshui Gu, Yuxuan Zhang, Zexuan Wang, Chen He, Chen Huang, Maojia Song, Zhiyuan Zeng, Shaowen Wang, Jinkai Liu, Yunfeng Shi, Jiaheng Liu, Shen Yan, Wenhao Huang, Ge Zhang, and Wenxuan Zhang. Harnessdev: Can llms create and evolve their own agent harness?, 2026a. URL https://arxiv.org/abs/2609.01437.

Yuhao Wu, Jingyuan Zhang, Jiajun Shi, Yuxuan Zhang, Xinping Lei, Junting Zhou, Zexuan Wang, Yuchen Wu, Huan Zhou, Duo Wang, Yinzhu Piao, Yongchang Peng, Yunfeng Shi, Jin Chen, Zuo Wang, Jinkai Liu, Jiaheng Liu, Wenxuan Zhang, Shen Yan, Wenhao Huang, and Ge Zhang. Aspire: Can models self-evolve from vague goals?, 2026b. URL https://arxiv.org/abs/2608.31111.

Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, Shuran Song, and Fangyin Wei. Sage: Scalable agentic 3d scene generation for embodied ai. *arXiv preprint arXiv:2602.10116*, 2026.

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

Yandan Yang, Baoxiong Jia, Shujie Zhang, and Siyuan Huang. Sceneweaver: All-in-one 3d scene synthesis with an extensible and self-reflective agent. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16227–16237, 2024.

Junyi Zhang, Jiaxin Ge, Hanjun Yoo, Letian Fu, Zihan Yang, Yaowei Liu, Raj Saravanan, Shaofeng Yin, Justin Yu, Dantong Niu, Zirui Wang, Roei Herzig, Ken Goldberg, Yutong Bai, David M. Chan, Ion Stoica, Angjoo Kanazawa, Jiahui Lei, Haiwen Feng, and Trevor Darrell. Playful agentic robot learning. *arXiv preprint arXiv:2606.19419*, 2026a.

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

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.

APPENDIX

TABLE OF CONTENTS

A Implementation Details **16**

A.1 Runs of the Frozen Library 16

A.2 Qualitative Trials 16

B Details of Asset and Scene Construction **20**

B.1 A Common Object Representation 20

B.2 Scene Requirements and Layout Review. 20

C Details of Task Generation and Validation **21**

C.1 Scene-Conditioned Task Proposals 21

C.2 Nontrivial Initial States and Oracle Checks 21

C.3 Five-Room Task Inventory 21

D Details of Atomic Actions and Composition **22**

D.1 Shared Control Functions 22

D.2 Four Execution Patterns 22

D.3 Success Predicates and Capability Boundaries 23

E Details of Policy Execution and Evaluation **23**

E.1 Policy Interface and Skill Usage 23

E.2 Execution Failures and Error Attribution. 23

E.3 Frozen Programs and Cross-Benchmark Protocols 24

F Reproducibility and Limitations **24**

F.1 Reproducibility Records 24

F.2 Evidence Scope and Capability Limitations 24

A Implementation Details

This section describes scene validation, the first run of the frozen library, and qualitative trials across the five rooms and five task levels.

Scene validation. Each movable asset is dropped from $2\,\mathrm{mm}$ above a plane and must settle with at most 15° of orientation drift and $5\,\mathrm{cm}$ of horizontal drift. The assembled scene is simulated for $2\,\mathrm{s}$ and accepted only if no object moves more than $5\,\mathrm{cm}$ or tilts more than 10° and no two objects interpenetrate by more than $5\,\mathrm{mm}$. A derived task is flagged unless the privileged oracle solves it in at least two of three seeds, from initial states that do not already satisfy its success condition. Appendix C.2 specifies the scope of these checks in the historical evaluation.

A.1 Runs of the Frozen Library

Table 5: **L5 tasks and their L3 goal subtasks under the fixed runtime.** Successes over trials. L5 pools two reruns of the five L5 tasks, 20 trials per room each; each L3 task has 7 trials so far. The restated goal is the L3 task whose relations are exactly those of the L5 task, phrased by the task template; each other goal subtask requires part of the goal. ‡Excludes two tasks that place a cup at given table coordinates; of the other two subtasks, the three cups in a row succeed in 1/7 and the two balls in the tray in 7/7. Intervals are 95% Wilson intervals over trials.

Room	**L5** full goal	**L3** restated goal	**L3** other goal subtasks	**L3** all goal subtasks
Office	35/40	4/7	23/28	27/35
Kitchen	35/40	7/7	19/28	26/35
Dining room	38/40	7/7	28/28	35/35
Living room‡	40/40	7/7	8/14	15/21
Bathroom	37/40	7/7	28/28	35/35
All five rooms	185/200	32/35	106/126	138/161
Rate (%)	**92.5**	**91.4**	**84.1**	**85.7**
95% interval	[88.0, 95.4]	[77.6, 97.0]	[76.8, 89.5]	[79.5, 90.3]

Table 5 lists the L5 tasks and their L3 goal subtasks under the fixed runtime, discussed in Sec. 4.4. In the first run (Table 3), the goal subtasks succeed in 104 of 110 trials at L2 and 74 of 115 at L3, and the living-room coordinate probes in 3 of 15 and 3 of 10. The parsing error of that run arises only after an attempt that the agent itself judged unfinished, so it weighs on the tasks that the library does not solve at once, and some of the trials it ended had already gone wrong through action or perception errors. A controller-state error also failed every dining-room L4 and L5 trial; Table 3 reports the rerun after its fix (5 of 25 and 5 of 5), without which the run totals 351 of 525 trials.

A.2 Qualitative Trials

Figs. 7–11 show one trial of the evolution route of Sec. 4.6 at each of the five levels in each of the five rooms, in six frames from the initial to the final state.

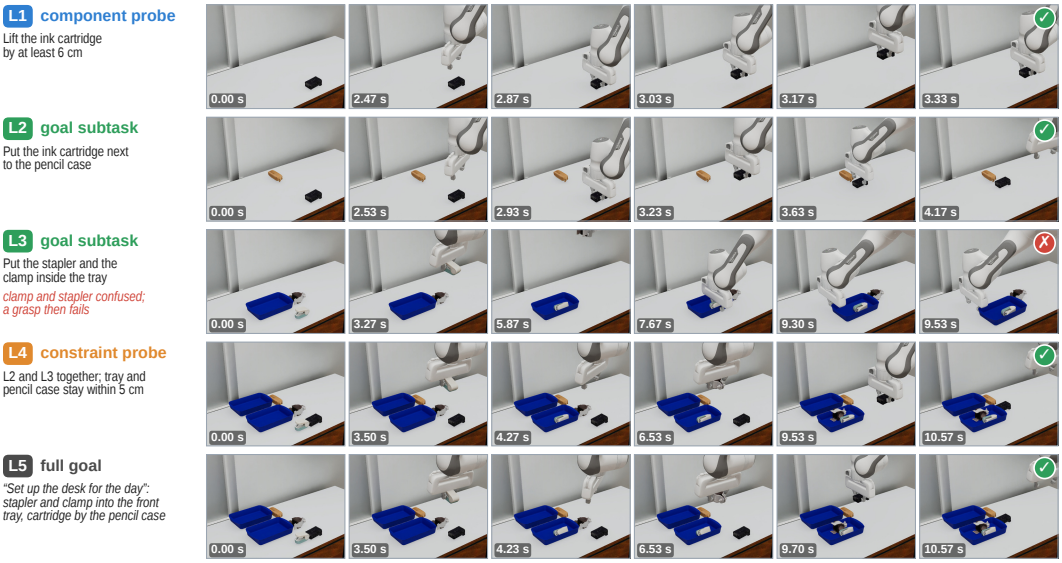

Figure 7: **Office trials at all five levels.** One trial per level for the office goal ("Set up the desk for the day"), L1 (top) to L5 (bottom), each in six chronological frames from the initial (left) to the final state (right); time stamps are those of the rendered replay at 30 frames per second.

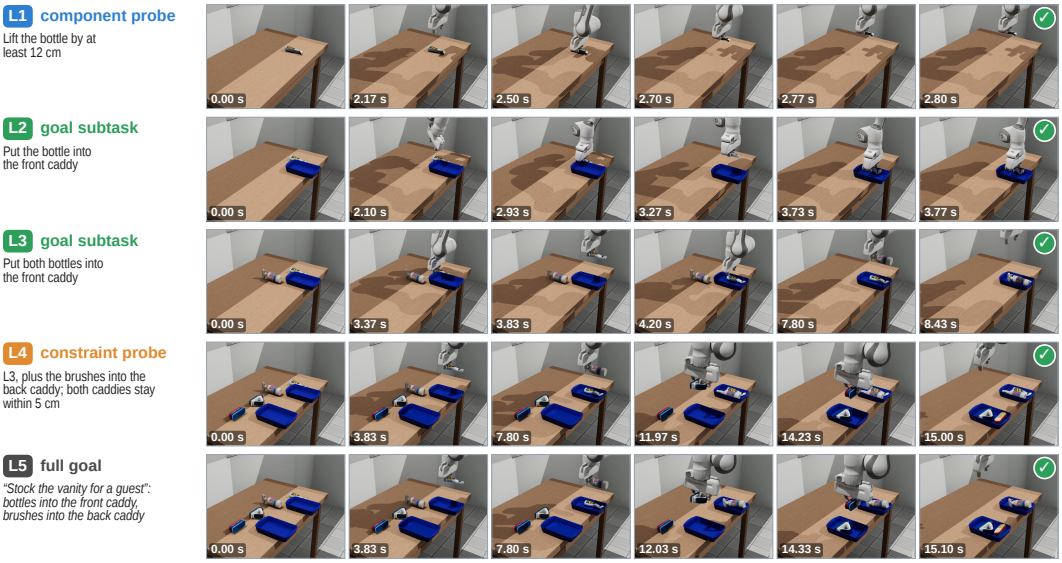

Figure 8: **Bathroom trials at all five levels.** One trial per level for the bathroom goal ("Stock the vanity for a guest"), L1 (top) to L5 (bottom), each in six chronological frames from the initial (left) to the final state (right), shown and rendered as in Fig. 7. These are the bathroom trials of Fig. 4; all five succeed.

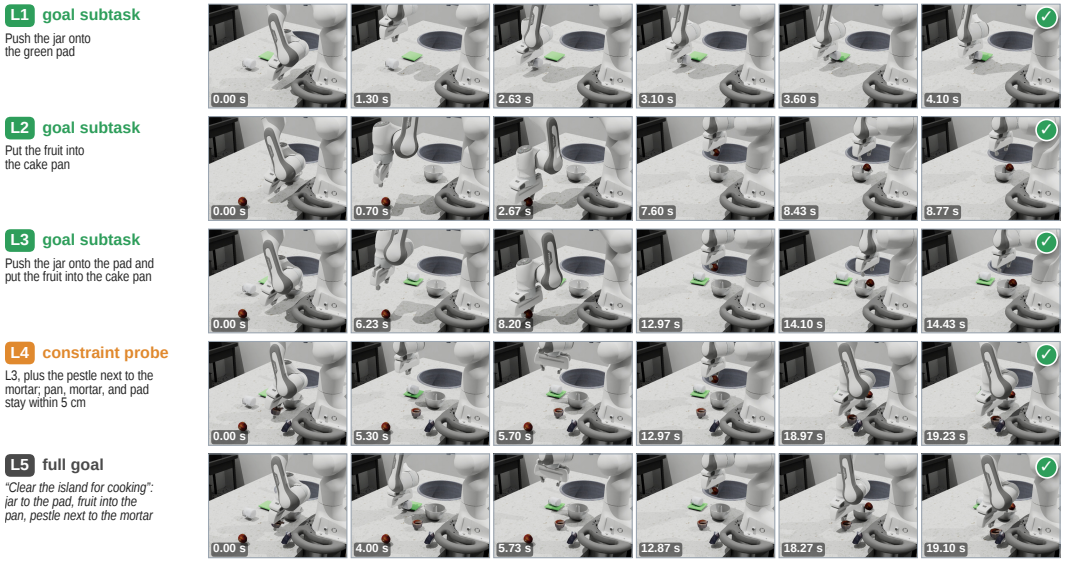

Figure 9: **Kitchen trials at all five levels.** One trial per level for the kitchen goal ("Clear the island for cooking"), L1 (top) to L5 (bottom), each in six chronological frames from the initial (left) to the final state (right), shown and rendered as in Fig. 7. The L1 push is the one goal subtask at L1 (Sec. 4.2). The replay of the L3 trial succeeds, whereas the original trial failed; its mark shows the replay.

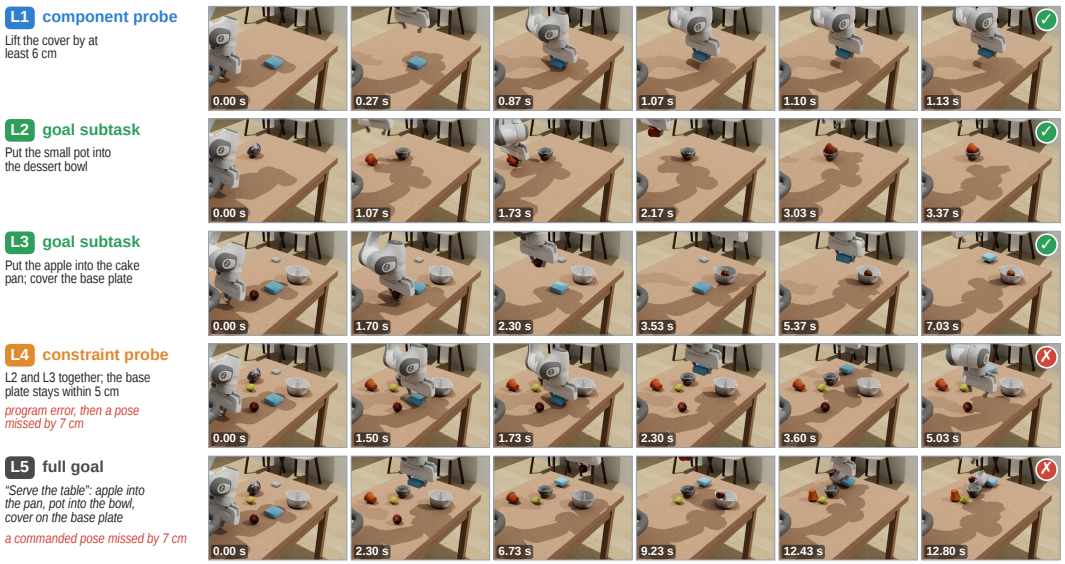

Figure 10: **Dining-room trials at all five levels.** One trial per level for the dining-room goal ("Serve the table"), L1 (top) to L5 (bottom), each in six chronological frames from the initial (left) to the final state (right), shown and rendered as in Fig. 7. The L4 constraint probe and the L5 task fail; red text gives the causes from their records.

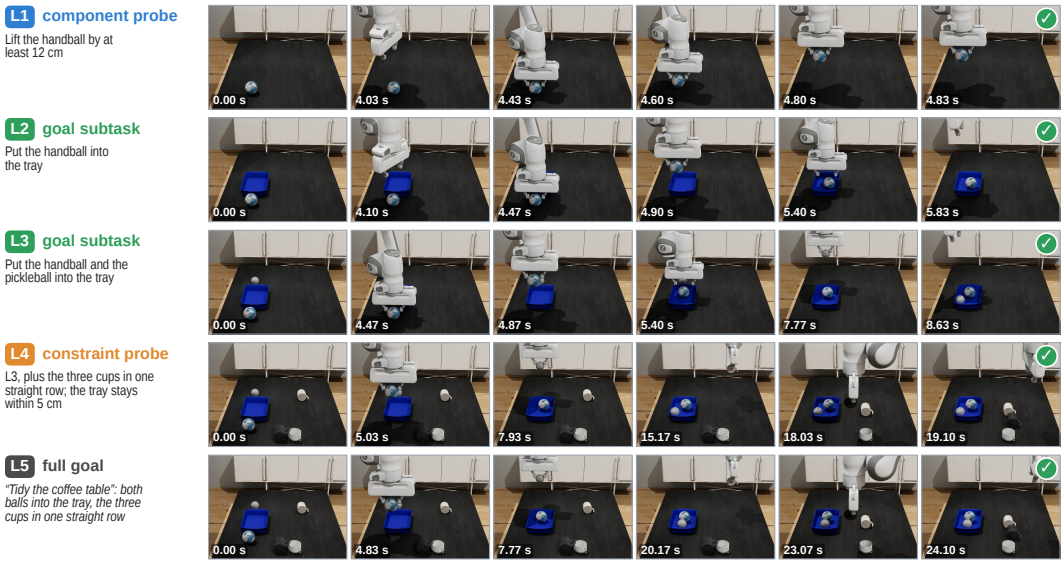

Figure 11: **Living-room trials at all five levels.** One trial per level for the living-room goal ("Tidy the coffee table"), L1 (top) to L5 (bottom), each in six chronological frames from the initial (left) to the final state (right), shown and rendered as in Fig. 7; all five succeed.

B Details of Asset and Scene Construction

The construction pipeline separates asset conversion, scene assembly, task specification, and execution. Assets from different sources are converted into a common object representation; scenes specify their layout and support surfaces; task templates define the success predicates and privileged oracle programs. The language model selects task concepts or generates policy code, while the runtime determines physical initialization and task success. The scene-validation thresholds are given in Appendix A; here we describe the representations used before validation.

B.1 A Common Object Representation

Object contract. The asset converters produce objects compatible with the robosuite `MujocoXMLObject` interface. Each converted asset contains an XML body named `object`, its visual and collision geometries, and three reference sites: `bottom_site`, `top_site`, and `horizontal_radius_site`. An accompanying manifest records conversion metadata. This shared representation allows scene assembly, pose queries, grasp selection, and placement checks to use the same interface across asset sources. Generated task artifacts also retain object names, scaled bounding boxes, arena XML, and a robot workspace reference.

Visual and collision geometry. Visual geometries use `group=1` and `contype=0`; collision geometries use `group=0` and retain collision participation. Experimental configurations typically use `condim=4` or higher to include torsional friction and reduce rotation of small objects within the gripper. The reference sites supply object bounds and horizontal extent for support-height estimates and grasp or placement candidates.

Scale and contact geometry. URDF- and RoboCasa-derived assets are scaled to a reachable working size of approximately 0.4–$0.5\,\mathrm{m}$, with the resulting bounding box stored for each instance. Task construction uses these recorded dimensions rather than inferring dimensions from the language description. Convex decomposition and local collision processing are applied selectively to articulated assets or assets requiring detailed contact geometry. A single convex hull can fill a door gap or obscure a handle, preventing otherwise plausible contact with the gripper.

Inventory and evaluation scope. The construction snapshot contains approximately 2,260 `object.xml` files and 23 manifests. These counts describe generated asset artifacts, not distinct evaluation objects or validated skills. A separate perception and grasp-screening batch considered approximately 300 rigid assets, of which approximately 176 met its perception-candidate criterion. The five-room evaluation uses a small subset of object instances. Asset replacement, collision inspection, and grasp screening are separate diagnostics and do not contribute trials to the 525-trial evaluation.

B.2 Scene Requirements and Layout Review

Scene requirements. The scene-requirement generator maps a room description to a room type, dimensions, style, functional regions, and proposed floor and wall objects. It requests 6–12 essential objects, uses lowercase `snake_case` object types, and separates the primary activity, support furniture, storage, decoration, and circulation space. Tabletop manipulation objects, doors, windows, rugs, and ceiling objects are excluded from this floor-and-wall planning stage. A scene intended for manipulation must identify at least one support surface, such as a desk, dining table, or kitchen island.

Object recommendation and semantic review. The object recommender returns JSON records that distinguish floor and wall placement, give dimensions as length, width, and height in centimeters, and explicitly indicate whether a support surface is required. A semantic reviewer then inspects top-down and perspective renderings with bounding boxes. It checks the overall room, proposes missing objects, and suggests moving, removing, or replacing existing objects. This review establishes a usable layout and support structure; task success is evaluated separately during execution.

C Details of Task Generation and Validation

This section describes scene-conditioned task proposals, the template-based success predicates and oracle programs, and the validation scope of the five-room evaluation. We distinguish the task-generation procedure from the historical task set retained for evaluation.

C.1 Scene-Conditioned Task Proposals

Candidate task generation. For generated scenes, the task proposer receives the objects already present, their roles and dimensions, and the available targets. It selects a subject, a target, and one of three relations: `inside`, `on`, or `near`. It may not introduce absent objects. Each proposal has a unique `snake_case` name and a natural-language instruction. A minimal record is:

```
{
  "task_name": "place_cup_on_tray",
  "prompt": "Place the cup onto the tray.",
  "subject": "cup",
  "target": "tray",
  "relation": "on"
}
```

Predicate and oracle construction. The runtime instantiates `check_success` and `oracle_code` from the selected relation. The corresponding predicates check containment, support, or proximity; placement success additionally requires that the subject is no longer held by the gripper. The oracle uses a prescribed grasp, lift, transport, release, and retreat sequence. Thus, the task proposer chooses the objects and relation, but does not write the success predicate. This relation-based generator was used for scene-driven tasks and oracle or pilot experiments; it does not by itself describe every task in the five-level evaluation.

C.2 Nontrivial Initial States and Oracle Checks

Validation criterion. Initial-state validity, oracle execution, and policy evaluation are recorded separately. The validation criterion requires that none of three reset states already satisfies the task and that the privileged oracle succeeds in at least two of the three seeds. A failed oracle check flags a task for review; failure of this particular program does not establish physical infeasibility.

Historical validation. The asset-v4 validation on September 13, 2026 tested 105 task specifications over three seeds each. Of these, 87 met both criteria. The remaining 18 were retained under the exploratory evaluation protocol, including three kitchen L2 tasks with a risk of initially satisfied success conditions. Consequently, the historical evaluation should not be interpreted as containing only oracle-validated, initially unsatisfied tasks.

Version scope. The 105 specifications used for the 525-trial evaluation match the integrated suite, but a complete, independent oracle-validation record for that integrated version was not verified in the appendix audit. Compared with asset v4, all 105 retain the same object configurations and oracle code, whereas only 81 retain identical success-predicate abstract syntax trees. Runtime conditions also require a separate check. The historical 87/105 validation result therefore cannot be assigned directly to the integrated suite. The 525-trial batch is reported as an exploratory evaluation over the full task set, with historical validation labels usable only in analyses that identify their version explicitly.

C.3 Five-Room Task Inventory

Task allocation. Each room contains five specifications at each of L1–L4 and one complete-room task at L5. Evaluating each specification in five trials gives 105 trials per room and 525 in total, as shown in Table 6. The levels organize task composition and state-preservation requirements. The exact objects, actions, constraints, and success conditions are defined by each specification; they cannot be inferred from the level label alone.

Table 6: Task-specification counts in the five-room evaluation. The final column counts trials, with five trials per specification.

Room	L1	L2	L3	L4	L5	Trials
Office	5	5	5	5	1	105
Kitchen	5	5	5	5	1	105
Dining room	5	5	5	5	1	105
Living room	5	5	5	5	1	105
Bathroom	5	5	5	5	1	105
Total	25	25	25	25	5	525

Table 7: Atomic-action families and their execution patterns. The lift action stops after lifting; the other actions in its family specialize transport and placement. Joint actuation uses privileged geometry in the oracle.

Pattern	Action labels	Control sequence
Grasp, transport, and place	lift, place-on, place-near, put-inside, sort, cover, stack, arrange-in-line, insert	Select a grasp; approach, close, and lift; specialize the destination and release motion.
Closed-gripper pushing	push, topple	Approach a side contact and move the closed gripper through pushing waypoints.
Surface wiping	wipe	Maintain the gripper or tool configuration and sweep surface waypoints.
Joint actuation	open, close, slide, turn	Follow a linear or circular trajectory determined by the joint geometry.

D Details of Atomic Actions and Composition

Action labels describe changes to objects or relations, while low-level functions implement motion and contact. Several labels share the same control sequence and differ mainly in the grasp, destination, contact direction, or joint parameters. Table 7 groups the documented labels into four execution patterns. These are construction and oracle patterns; their availability does not imply that each has a corresponding transferable skill in the frozen library.

D.1 Shared Control Functions

The generated-task interface includes the following core functions. Arguments and available wrappers depend on the runtime version.

```
sample_grasp_pose(name, quaternion_wxyz=None)
get_object_pose(name, return_bbox_extent=False)
goto_pose(position, quaternion_wxyz, z_approach=0.0)
open_gripper()
close_gripper()
```

Grasp sampling provides a candidate position and orientation, and pose queries provide the geometry needed to compute clearances and destinations. The motion controller uses inverse kinematics to move the end effector toward a requested pose. A return from `goto_pose` does not certify that the requested pose was reached; controller and forward-kinematics checks are separate from the function return. Later public interfaces and frozen skills may wrap these operations in grasping or placement functions with additional verification. Such wrappers are not interchangeable with individual oracle primitives.

D.2 Four Execution Patterns

Grasp, transport, and place. The shared sequence opens the gripper, samples a grasp, approaches the object, closes the gripper, and lifts to a clearance height. Placement then queries the target geometry, computes an approach and release pose, moves to these poses, opens the gripper, and retreats. The destination specializes the action: a support surface for `place-on` or `stack`, an

offset for `place-near`, a container interior for `put-inside`, a class-dependent slot for `sort`, a covering pose for `cover`, an indexed slot for `arrange-in-line`, and an insertion axis for `insert`. The `lift` action retains the grasp instead of proceeding to release.

Closed-gripper pushing. The controller queries the object pose, closes the gripper, and follows approach, contact, push, and retreat waypoints. A push sets the displacement direction and distance. A topple uses a higher contact point to produce a turning moment. Neither requires the object to be grasped before contact.

Surface wiping. The controller maintains a closed gripper or installed tool and follows a serpentine sequence of surface waypoints. Success depends on accumulated cleaning coverage rather than completion of an individual motion command. The RoboSuite Wipe evaluation uses its own tool configuration and is reported separately from this atomic oracle.

Joint actuation. The privileged oracle exposes the following operation:

```
actuate_joint(obj, joint, target_fraction, grasp=...)
```

Internally, it uses the joint type, range, axis, anchor, and grasp point to generate linear or circular motion. These simulator-state inputs are privileged. The five source skill bundles do not contain validated joint-specialist skills, and the S2 cross-domain condition does not supply simulator joint axes to the model. Oracle success with this function therefore does not demonstrate visual-policy competence in joint manipulation.

D.3 Success Predicates and Capability Boundaries

Action execution and success evaluation remain separate. For example, a placement-on-target predicate requires both release of the subject and the specified support relation. Containment additionally depends on container geometry and a stable final relation; lifting requires both holding and sufficient height change; wiping uses accumulated cleaning state. Normal program termination or a motion-function return cannot replace these native predicates.

The action inventory also does not establish the availability of specialized source skills. In particular, the source bundles do not provide a precision-insertion skill for NutAssembly. Some repair runtimes do not export an insertion operation at all, so a handwritten insertion sequence is not evidence of reusing an existing insertion skill. These distinctions separate an action label, an executable privileged oracle, and a skill available to the evaluated policy.

E Details of Policy Execution and Evaluation

This section specifies the policy interface, distinguishes execution failures by stage, and describes the scope of skill reuse across evaluation protocols.

E.1 Policy Interface and Skill Usage

The policy model receives a task description and the API documentation and returns a Python code block for execution in a sandbox. Depending on the experimental condition, the program may call, rewrite, or bypass library skills. Native task success and recorded skill calls are therefore distinct outcomes: success alone does not establish that a supplied or newly admitted skill was used.

E.2 Execution Failures and Error Attribution

Failure stages. The experimental audit distinguishes request and transport errors, code-extraction errors, guard rejections, initialization failures, control failures, manipulation failures, and unmet native predicates. Observed interface faults include loss of a complete Python code block during parsing, rejection of otherwise valid `map` or `type` usage, treating a return value as proof of action completion, loss of candidate identity between observation frames, and interpreting an entire instruction as an object name. Control and manipulation faults include inverse-kinematics or end-effector errors, failed grasps, and invalid placement geometry. A trial can exhibit errors at more than one stage; a later parser error does not establish that preceding physical execution was correct.

E.3 Frozen Programs and Cross-Benchmark Protocols

Development and evaluation. The LIBERO external condition uses task-level programs or skills frozen after human-assisted development. It is not an evaluation of autonomous skill discovery from scratch. The updated RoboSuite experiments include geometry cards, target-domain development guidance, and policy generation, and are not strictly zero-shot transfer. When a single frozen program is replayed across 50 initial states, those trials measure physical execution across initial states, not 50 independent language-model generations.

Protocol separation. Five-room, RoboSuite, LIBERO/PRO, and RATs experiments differ in their models, interfaces, initializations, budgets, and native predicates. Their denominators and success rates are reported within each protocol rather than pooled into one benchmark score. Asset-screening counts, oracle-validation rates, generated-program counts, and policy success rates likewise measure different stages of the system.

F Reproducibility and Limitations

This section lists the records needed to identify tasks, assets, programs, and episodes, and states the limits of the claims supported by those records.

F.1 Reproducibility Records

The reproducibility schema separates the following records. Missing historical fields remain unknown rather than being imputed as failures.

1. **Task:** task name, instruction, object references, relation or constraints, and specification hash.
2. **Asset:** source, converter version, scale, bounding box, reference sites, collision settings, and object-XML hash.
3. **Scene:** arena XML, layout seed, support surfaces, robot base pose, and workspace reference.
4. **Validation:** predicate and oracle versions, oracle successes by seed, and initial-state nontriviality checks.
5. **Policy:** model and API/runtime versions, program hash, request and parsing status, and recorded skill calls.
6. **Episode:** reset hash, control steps, exceptions, physical completion, and native task success.
7. **Skill admission:** whether the candidate was executed, independently validated, and shown to recover a baseline failure.

F.2 Evidence Scope and Capability Limitations

Evidence scope. These records support different claims. An asset manifest establishes the identity of an object artifact; oracle success establishes execution of a particular template program; a native-success record establishes the result of an evaluated policy under its specified conditions. None substitutes for the others. Similarly, a scene load-check rendering demonstrates successful loading, a layout image shows spatial organization, and an episode's final frame illustrates its terminal state. Images alone do not certify task success.

Capability limitations. The evidence is strongest for execution reuse of skills frozen after human-assisted, target-domain development. Establishing that autonomous accumulation at L1–L4 reliably improves L5 requires a matched comparison with the same runtime, tasks and initial states, and budget, together with evidence that newly acquired skills were actually called. The audited results do not yet provide that complete comparison. High scores on fixed layouts, lower scores on randomized layouts, and failures on difficult insertion tasks delimit the demonstrated capability; unfinished randomized batches should not be treated as a completed robustness study.