ACE-CaP-0: Real-World Code-as-Policies in the Era of Experience

ACE-CaP Team

Please see Contributions and Author List for more author details.

Abstract

Recent advances in frontier models have pushed the boundaries of agentic robot control, enabling agents to perform complex robotic tasks through reasoning and interaction. However, repeated calls to these models during execution can substantially increase task completion time. Code as Policies (CaP) offers a route for coding agents to acquire physical skills through experience and retain them in executable programs, allowing the reasoning invested in development to be reused across runs. Developing reliable code policies while limiting the cost of physical experimentation remains a central challenge. We investigate human guidance as a source of priors for feasible actions and simulation as an auxiliary environment for exploring and testing candidate policies. Our findings favor robot demonstrations available on request and simulation used selectively during real-world policy refinement. These findings inform ACE-CaP-0, an agentic recipe that couples policy revision with experience acquisition through *Demonstrations on Request* and *Simulation on Demand*. Across eight real-world manipulation tasks spanning alignment, multi-object handling, and bimanual coordination, the resulting frozen programs achieve 92.1% success (221/240 trials). Mean execution times across tasks range from 0.97 to 2.27 minutes. The programs execute with runtime perception and control, without a human or coding agent in the loop. These results highlight CaP's potential to turn experience into reliable, reusable physical skills.

Project Page: https://acerobotics2025.github.io/ACE-CaP/



Figure 1 ACE-CaP-0: from targeted experience to standalone robot execution. Left: a coding agent refines policies on the robot, requesting robot demonstrations and using simulation as needed. Right: eight manipulation tasks, with each panel reporting the success rate and mean physical execution time in minutes. Each frozen policy is evaluated in 30 trials and executes with runtime perception and control, without a human or coding agent in the loop.

Contents

Abstract . . . 1

1 Introduction . . . 3

2 Investigation . . . 4

2.1 How Do Human Priors Shape Real-World CaP? . . . 4

2.2 When and How Should Simulation Support Real-World CaP? . . . 6

3 The ACE-CaP-0 Recipe . . . 7

3.1 A Unified Policy Development Loop . . . 7

3.2 Demonstrations on Request . . . 7

3.3 Simulation on Demand . . . 9

3.4 From Policy Development to Standalone Execution . . . 9

4 Real-World Evaluation . . . 10

4.1 Evaluation Setup . . . 10

4.2 Reliable Execution for Real-World Manipulation . . . 10

5 Related Work . . . 12

5.1 Code-as-Policies and Agentic Robot Learning . . . 12

5.2 Human and Simulated Experience for Policy Development . . . 12

6 Conclusion . . . 13

A Contributions and Author List . . . 17

1 Introduction

Foundation models are changing how robots acquire and execute skills. Learned action models, including vision-language-action (VLA) and world action models (WAMs), combine pretrained knowledge with large-scale data and interaction experience to support generalist robot control [1–3]. Advances in frontier models (*e.g.*, GPT-6 Astra [4] and Claude Fable 5.2 [5]) are expanding the role of agents in both robot control and policy development. In agentic robot control, agents interpret observations, choose actions, and revise their decisions in response to physical feedback [6]. When execution relies on repeated model calls, however, inference latency can account for a substantial part of task completion time [7, 8]. Agents can also apply these capabilities during policy development, organizing experiments, diagnosing failures, and revising the policies a robot executes [9, 10].

Code as Policies (CaP) represents robot behavior as executable programs, allowing decisions made during development to be retained and reused across runs [11, 12]. Through demonstrations and experimentation, a coding agent can discover manipulation strategies and encode their action structure, geometric relationships, and motion parameters in code [13]. The resulting program can execute with runtime perception and control, without requiring the coding agent to select each action. Its explicit task logic can be inspected and revised, supporting the accumulation of executable skills in embodied Recursive Self-Improvement (RSI) [10, 14].

Developing reliable code policies requires physical knowledge that a task instruction often leaves unspecified. The agent must resolve choices about grasping, alignment, and coordination, yet a failed attempt may not reveal how to revise the program [10, 13]. Repeated physical trials also consume robot time and can require human scene resets [9, 15]. Policy development therefore couples two decisions: how to revise the program, and what experience to acquire next. We ask: *how should agents acquire experience to develop reliable code policies when physical experimentation is costly?*

Human guidance can supply feasible strategies and motion references, while simulation provides an auxiliary environment for exploring and testing candidate implementations [13, 15]. We examine their use through six development conditions on charger insertion, with one development run per condition (Section 2).

- **What form of human guidance helps an agent develop a physically realizable policy, and when should it be provided?** Our results favor robot demonstrations paired with textual guidance and available on request during development. Robot demonstrations supply motion references in the target embodiment, while on-request access lets the agent seek guidance for its current manipulation problem.
- **When and how should simulated experience support real-world program development?** Our results favor using simulation on demand to investigate current development questions within a real-world feedback loop. Simulated experiments examine candidate revisions, while physical execution checks whether the sensing and action assumptions they rely on hold on the robot.

Building on these findings, we introduce **ACE-CaP-0**, a recipe that couples policy revision with experience acquisition through *Demonstrations on Request* and *Simulation on Demand* (Section 3). During development, the coding agent chooses whether to request a robot demonstration, examine a candidate revision in simulation, or test the program on the robot. The resulting program is frozen for evaluation and executes with runtime perception and control, without a human or coding agent in the loop.

We evaluate ACE-CaP-0 on eight real-world manipulation tasks requiring alignment, object reorientation, multi-object handling, and bimanual coordination (Section 4). Development allows at most ten human-assisted scene resets per task, excluding resets for demonstration collection. Each frozen program is then evaluated in 30 trials without further updates. As summarized in Figure 1, the programs achieve 92.1% overall success (221/240 trials). Mean physical execution times range from 0.97 to 2.27 minutes across tasks. These results support CaP as a practical route from targeted experience to reliable, reusable physical skills.

2 Investigation

We study charger insertion to examine how human guidance and simulation support code-policy development under limited physical interaction. Six experimental conditions address three questions: what form of guidance to provide, when to provide it, and how to incorporate simulation into exploration. Here, exploration involves executing candidate programs, inspecting feedback, and revising code. All three comparisons share a real-only reference with on-request textual guidance and robot demonstrations.

Shared setting and development budget. All conditions use the same foundation model, task instruction, and real-world perception and control interfaces. We conduct one independent development run per condition, allowing at most ten human-assisted scene resets for the agent's physical trials. Resets for demonstration collection are excluded; autonomous executions and demonstrations do not count against the budget.

Evaluation and cost accounting. After development, we freeze each selected program and evaluate it in 30 real-world trials under common success criteria and initial-state conditions, without further updates [16]. Final success rate is the fraction of these trials in which the program completes charger insertion. Development time is the elapsed time in hours, including model calls, simulated experiments, and real-world interaction. Output token usage, reported in millions (M), sums the tokens generated across all model calls during development, including guidance interpretation, simulation construction, execution analysis, and code revision. Both costs exclude frozen-policy evaluation.

2.1 How Do Human Priors Shape Real-World CaP?

Human guidance supplies prior knowledge about manipulation strategies and their implementation [13, 17]. For charger insertion, the task goal leaves choices about grasping, plug orientation, and approach motion unresolved. Demonstrations can supply a workable reference when execution feedback alone does not indicate how to revise the program [13, 18]. We examine guidance form and timing using four conditions developed without simulation. Figures 2 and 3 summarize the protocols and results.

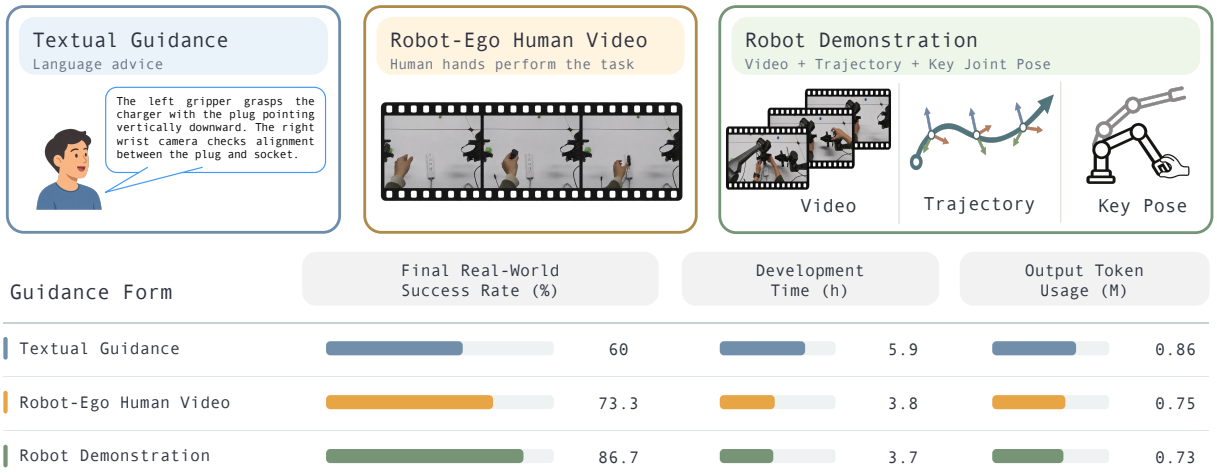


Figure 2 Guidance form for charger insertion. Top: textual guidance, robot-egocentric human video, and robot demonstrations, all available on request during real-only development. Both demonstration conditions also include textual guidance. Bottom: final success rate over 30 real-world trials of each frozen program, development time, and output token usage. Robot Demonstrations is the shared reference in Figures 3 and 4.

Guidance Form. We compare three forms of assistance, each available when the agent requests it:

- **Textual Guidance.** The agent can request textual advice about manipulation strategies, action constraints, and corrections. No demonstration is provided in this condition.

- **Robot-Egocentric Human Video.** In addition to textual guidance, the agent can request videos of a person performing the task with their hands, recorded from the robot's execution viewpoint.
- **Robot Demonstrations.** In addition to textual guidance, the agent can request kinesthetic demonstrations in which a person physically guides the robot arms through the task. Each demonstration provides video, arm trajectories, and key joint poses capturing arm configurations at critical action stages.

Both demonstration conditions use the same camera, fixed centrally between the robot arms on an independent mount. Here, *robot-egocentric* denotes the shared viewpoint during demonstration and robot execution.

The Robot Demonstrations condition achieves the highest final success rate, at 86.7% (26/30), compared with 73.3% (22/30) for Robot-Egocentric Human Video and 60% for Textual Guidance (Figure 2). The two demonstration conditions have similar development costs: 3.7 and 3.8 hours, with 0.73 and 0.75 million output tokens, respectively. Textual Guidance requires 5.9 hours and 0.86 million output tokens. At similar development cost, the robot-demonstration condition achieves higher final success than human video.

Why demonstrations help. Both demonstration conditions combine higher final success with lower development costs than text alone. A visible example conveys action order and object interactions, complementing textual descriptions of intent and constraints [13, 19]. Human video supplies such a strategy, but the agent must still translate human hand movements into robot gripper and arm motions. Robot demonstrations provide the strategy in the target embodiment together with recorded robot motion. The video shows what the actions accomplish; arm trajectories and key joint poses provide quantitative references for approach paths and arm configurations. These sources jointly constrain the action sequence and motion parameters to implement in code, reducing the motion details that must be inferred from images alone [13, 17].

Guidance Timing. Active demonstration learning treats when and what to request as part of the learning process [20]. We examine this choice in code-policy development by comparing two provision protocols for textual guidance and robot demonstrations:

- **Upfront Guidance.** The agent receives textual guidance and robot demonstrations at the outset.
- **On-Request Guidance.** The agent decides whether and when to request assistance, either before its first physical attempt or during subsequent program revision.

On-Request Guidance is the Robot Demonstrations condition above and the Real-Only condition below; all three labels refer to the same development run and evaluation results.

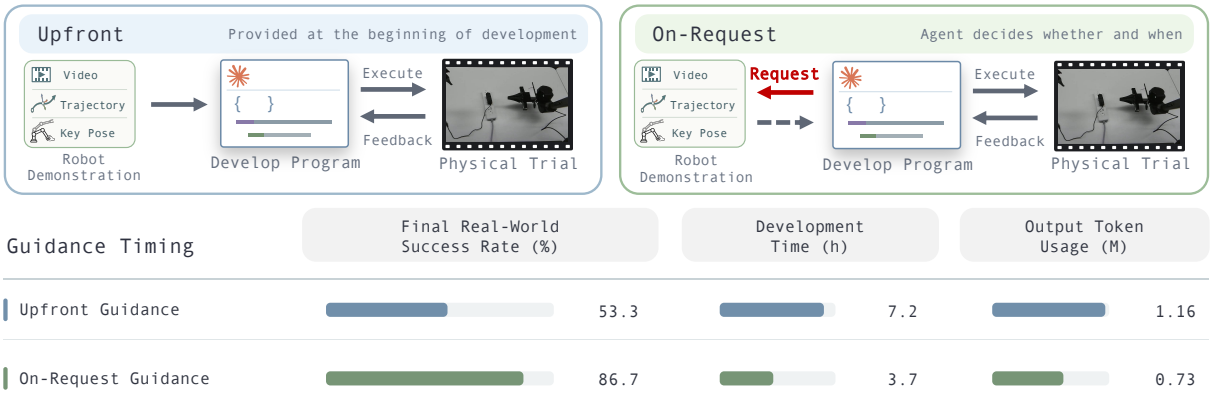


Figure 3 Guidance timing for charger insertion. Textual guidance and robot demonstrations are provided upfront or requested during real-only development. The lower panels report final success rate over 30 real-world trials of each frozen program, development time, and output token usage. On-Request Guidance shares the Robot Demonstrations condition and results from Figure 2.

On-Request Guidance achieves 86.7% (26/30) final success, compared with 53.3% (16/30) for Upfront Guidance,

while development time decreases from 7.2 to 3.7 hours and output token usage from 1.16 to 0.73 million (Figure 3). On-request access ties assistance to the agent's current uncertainty: an initial grasp can be demonstrated before execution, or an unsuccessful motion can motivate a targeted example or correction. This lets guidance inform the next code revision as new difficulties emerge during physical testing.

2.2 When and How Should Simulation Support Real-World CaP?

Simulation can serve as a development environment before transfer or as an on-demand tool for experiments during real-world development [15]. We examine these roles through three exploration protocols:

- **Real-Only.** The agent develops its program through real-world execution and feedback, without simulation. Textual guidance and robot demonstrations are available on request.
- **Sequential Real-Sim-Real.** The agent first receives robot demonstrations in the real environment and constructs a simulation based on the real setup. It then refines its program through ten rounds of simulated exploration in four parallel environments, the maximum supported by the available hardware. Each round ends when all four environments are reset together. The resulting program is then transferred to the robot for evaluation.
- **Simulation on Demand.** The agent develops its program within a real-world feedback loop and can use or construct a simulator to investigate a current development question. It decides whether and when to simulate, which task segment and initial state to examine, and which candidate program revisions to test. Simulated experiments can precede the first physical attempt or occur during later revision. Textual guidance and robot demonstrations remain available on request throughout development.

Comparison structure. Real-Only and Simulation on Demand share on-request access to textual guidance and robot demonstrations and the same physical scene-reset budget. Their comparison examines what simulated experimentation adds to real-world development. Sequential Real-Sim-Real provides a complementary workflow baseline with upfront demonstrations and a separate simulation stage before transfer. Its ten simulation rounds are counted separately from human-assisted scene resets for physical trials.

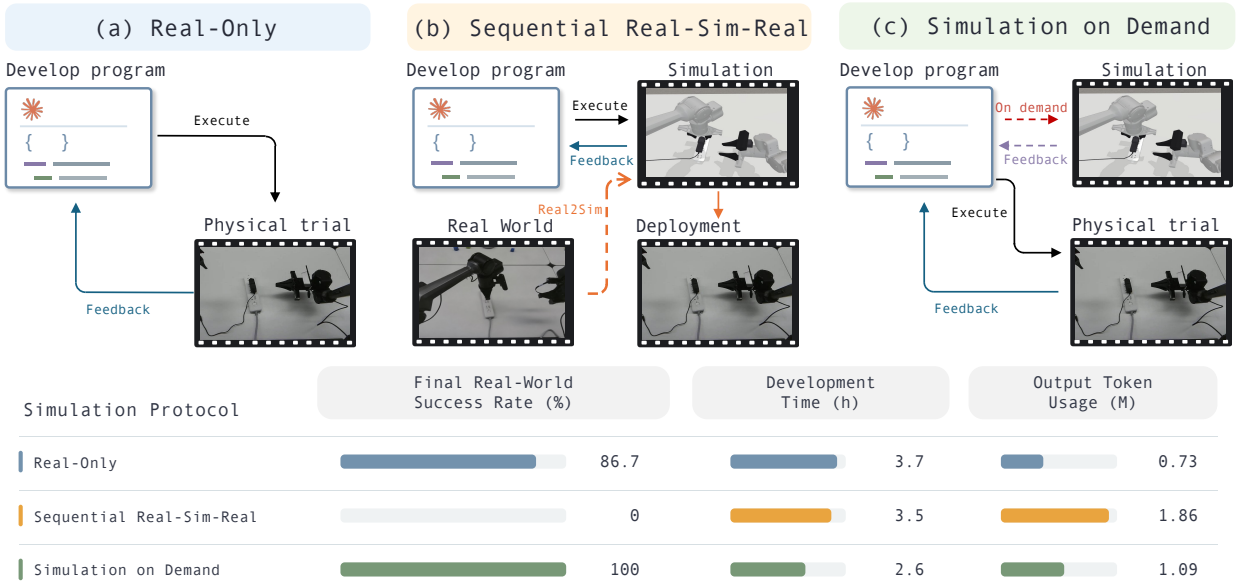


Figure 4 Simulation protocols and charger-insertion results. Top: (a) Real-Only uses physical feedback; (b) Sequential Real-Sim-Real uses ten simulation rounds before transfer, each ending with a joint reset of four parallel environments; (c) Simulation on Demand makes simulated experiments available within real-world development. Bottom: final success rate over 30 real-world trials of each frozen program, development time, and output token usage. Real-Only is the shared reference from Figures 2 and 3.

Results and development costs. Simulation on Demand achieves 100% final real-world success, compared with 86.7% for Real-Only, while development time decreases from 3.7 to 2.6 hours (Figure 4). Output token usage increases from 0.73 to 1.09 million.

Failure analysis. Sequential Real-Sim-Real achieves 0% final real-world success after 3.5 hours of development and 1.86 million output tokens. The transferred program fails at grasping the plug, before insertion. In our setup, simulated images are cleaner, and simulated depth does not reproduce the real sensor's close-range blind regions. The agent refines the image- and depth-dependent code directly against these simulated observations. The failure is consistent with an observation mismatch: grasping decisions can depend on visual cues and depth measurements that are unreliable or unavailable on the robot [21, 22]. The sequential workflow postpones testing these sensing assumptions on the robot until after simulated refinement.

Implications. These results favor keeping robot demonstrations available on request and using simulation within real-world development. Simulated experiments examine candidate revisions, while physical trials test the sensing and action assumptions on which those revisions depend [15]. This emphasis on interaction evidence also aligns with the diagnosis-driven use of offline priors advocated in online reinforcement learning [23]. Together, the guidance and simulation studies motivate adaptive experience acquisition: the agent chooses what evidence to obtain as it revises the program. Section 3 develops this approach in a unified loop.

3 The ACE-CaP-0 Recipe

ACE-CaP-0 makes experience acquisition part of policy development. The coding agent writes and revises an executable program, choosing when to request robot demonstrations, run simulated experiments, or test the program on the robot. Building on the guidance and simulation choices examined in Section 2, this section describes how the resulting evidence informs program revision and how the final policy executes independently of the coding agent. Figure 5 shows these two stages.

3.1 A Unified Policy Development Loop

The program as the working artifact. The agent begins with a task instruction, observations of the workspace, and access to the robot's perception and control interfaces. It constructs an executable program that specifies action sequences, geometric computations, motion parameters, and gripper commands. The coding agent writes and revises this program during development; the program controls the robot through the available interfaces [11]. ACE-CaP-0 builds on the write–execute–inspect–revise loop of embodied coding agents [12], with demonstrations and simulation available as additional sources of evidence.

Choosing the next development step. At each step, the agent considers the current program, the evidence already available, and the question that remains unresolved. It can revise the code directly, request a demonstration, organize a simulated experiment, or execute a candidate program on the robot. A missing manipulation strategy can motivate a demonstration request; uncertainty about a candidate implementation can motivate a simulated comparison; physical execution tests how the program behaves in the real setup. Demonstrations and simulation can be used before the first physical attempt or during later revision, and neither is required in every iteration.

Using feedback to continue development. Physical outcomes inform both the next code revision and the choice of further experience. A failure can expose an incorrect action sequence, geometric assumption, or motion parameter; it can also show that the current evidence is insufficient to identify a useful correction. The sources can inform one another: demonstrations can suggest interactions to simulate, while inconclusive simulations can motivate demonstration requests. Development is subject to the scene-reset budget in Sections 2 and 4.1; the budget counts resets rather than autonomous executions or human demonstrations.

3.2 Demonstrations on Request

Requesting evidence for a manipulation decision. The agent requests a demonstration when it judges that observing how a person performs the task with the robot would help it construct or revise the policy [13]. Before an initial attempt, the unresolved issue may be how to establish an object interaction or coordinate

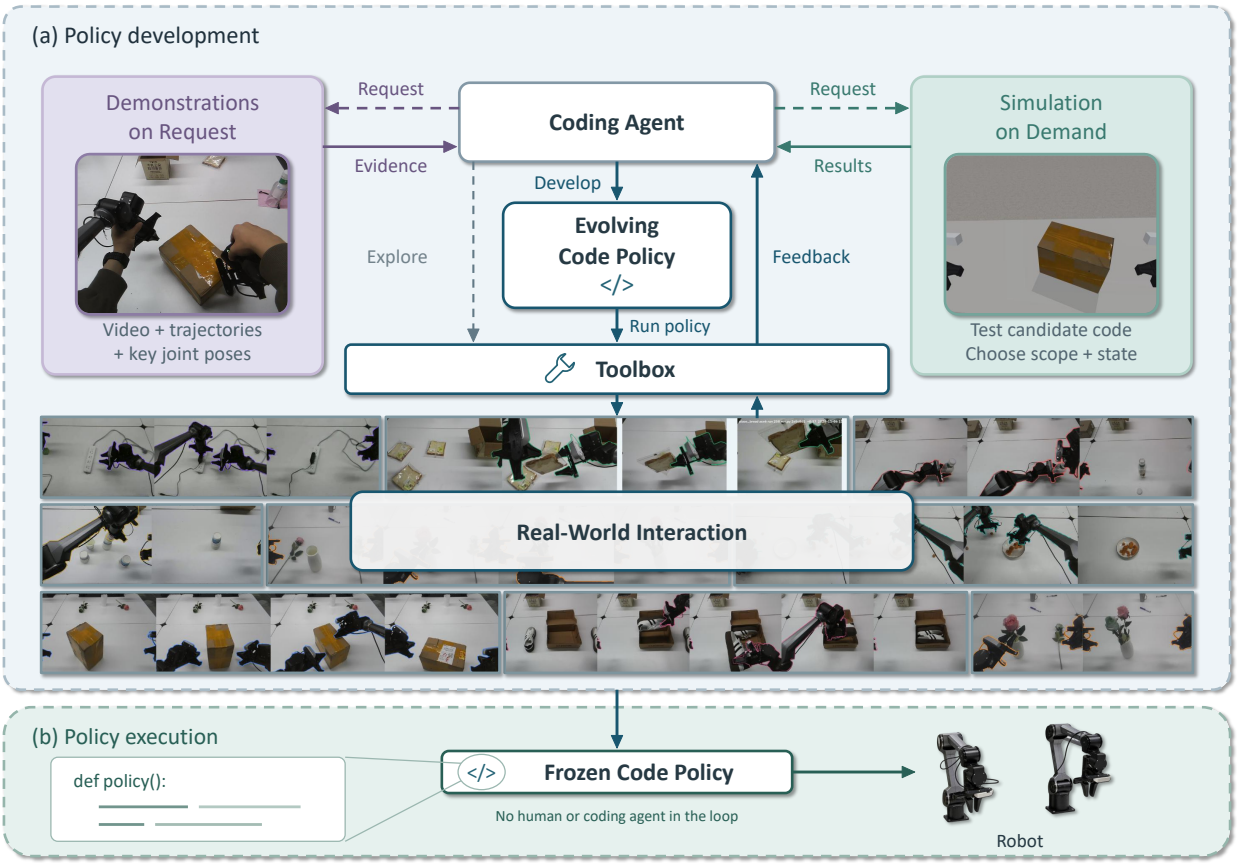


Figure 5 ACE-CaP-0: from policy development to standalone execution. (a) The coding agent writes and revises an executable policy using real-world feedback, with a shared robot toolbox that supports both policy execution and direct exploration by the agent. Throughout development, the agent can request kinesthetic demonstrations that provide video, arm trajectories, and key joint poses, or test candidate code in simulation for a chosen task segment and initial state. (b) The resulting code policy is frozen and executes through the robot's interfaces without a human or coding agent in the loop. Section 4 evaluates these frozen policies without further updates.

the two arms. After a failed attempt, the agent may understand where the program failed but lack a workable alternative for that action. In the guidance protocol of Section 2.1, the agent also has access to textual suggestions about actions, constraints, or corrections.

Reading object interaction and robot motion together. We use kinesthetic demonstrations in which a person physically guides the robot arms through the task. Each demonstration provides video, recorded arm trajectories, and key joint poses that capture arm configurations at critical action stages. These poses show the gripper placement needed to grasp the bread's plastic wrapper upon closure, the arm configurations for a shoe handover, and the plug–socket alignment for downward insertion. The agent interprets these motion references together with the video to identify the configuration from which each action can proceed.

Translating the demonstration into program changes. The demonstrated action order and the roles of the two arms inform the program's structure. Approach directions, motion paths, and key joint poses inform the implementation of individual actions. The agent interprets these references in relation to the current scene and the interfaces available to the policy, then writes or revises the corresponding code. The resulting implementation still contains choices about geometry, timing, and contact whose physical consequences must be assessed. The agent can examine a candidate implementation in simulation or test it on the robot, using the outcome to determine what requires further revision.

3.3 Simulation on Demand

Defining the experiment. Simulation on Demand lets the agent examine candidate program changes through simulated execution [15]. An experiment begins with a development question, such as whether an approach path avoids an obstruction, how a motion parameter affects the interaction, or how coordinated actions should be ordered. The agent chooses the task segment, initial state, and candidate changes relevant to that question. The scope can cover a single interaction or the complete task. The initial state specifies which preceding task conditions the experiment takes as given. The model must represent the geometry and interactions relevant to comparing the candidates.

Constructing a test for the current question. The agent can use an existing simulator or construct and modify one for the experiment. Workspace observations and previous physical outcomes inform the modeled geometry, state, and behavior. Available demonstrations can further guide the choice of actions and interactions to examine. The resulting simulator provides an executable setting in which the agent runs candidate programs and inspects their behavior. The required correspondence with the physical system depends on the decision being made: a comparison of approach paths and a comparison of contact behaviors place different demands on the model [22]. For code that uses images or depth, this includes whether the required observations are available and reliable on the robot.

Taking simulated evidence back to the robot. The agent interprets the outcomes in relation to the question that motivated the experiment, then uses them to choose a code revision or the next physical test. For example, a simulated path comparison can suggest an approach that avoids an obstruction while leaving the reliability of the eventual grasp to be checked on the robot. The conclusion applies to the behavior and conditions examined by that experiment. Physical execution then tests the revision on the robot.

If the real behavior differs in a way that affects the development decision, the agent reconsiders both the program and the assumptions of the simulated experiment. It can revise the simulator, change the task segment or starting state, request a demonstration, or continue directly with real-world development.

3.4 From Policy Development to Standalone Execution

Retaining experience in an executable policy. At the end of development, the agent produces a *condensed code policy* that retains the task logic and execution details developed through interaction. These choices are expressed through the program's action structure, geometric computations, and motion parameters. We freeze the program before evaluation and assess it without further updates.

Executing fixed rules with runtime feedback. During execution, the program calls the robot's perception and control interfaces. Its decision rules remain fixed, while runtime observations can change the values computed by those rules and the resulting actions. The perception and control components remain active, and the policy executes without a human or coding agent in the loop. Section 4 measures the success rates and physical execution times of these programs across eight real-world manipulation tasks.

4 Real-World Evaluation

We evaluate ACE-CaP-0 on eight real-world manipulation tasks, measuring the success rate and physical execution time of the programs produced through the development process in Section 3. Each task program is frozen before evaluation and executes through the robot's perception and control interfaces without a human or coding agent in the loop.

4.1 Evaluation Setup

Platform and tasks. Our platform consists of two Agilex PIPERX robot arms and three Orbbec DaBai DC1 cameras, providing one top-down view and two wrist-mounted views (Figure 6). The eight tasks span alignment, object reorientation, multi-object handling, and bimanual coordination. They include inserting a charger into a socket, removing a bottle cap while stabilizing the bottle, and placing a pair of shoes into a box. Table 1 lists the objective of each task alongside its evaluation results.

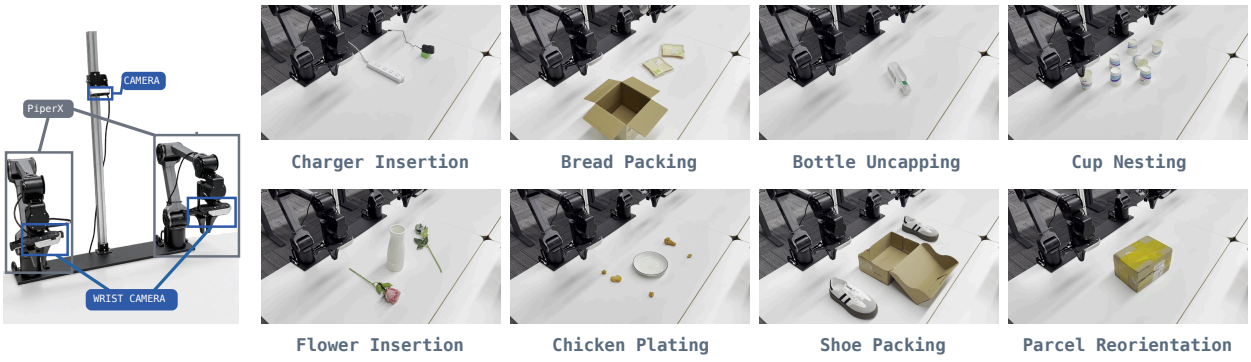


Figure 6 Real-world evaluation platform and tasks. Left: the dual-arm platform with one top-down camera and two wrist-mounted cameras. Right: representative scenes for the eight tasks, covering alignment, object reorientation, multi-object handling, and bimanual coordination.

Policy development. We use Claude Opus 5.5 [24] as the coding agent, with access to perception, reasoning, and robot control interfaces. For each task, the agent develops an executable policy using the recipe in Section 3: it writes, executes, inspects, and revises the program, requesting robot demonstrations and conducting simulated experiments as needed. Development allows at most ten human-assisted scene resets per task. As in Section 2, resets used to collect demonstrations are excluded from this budget. We measure output token usage by summing the tokens generated across all model calls during policy development, including demonstration interpretation, simulated experimentation, and program revision.

Evaluation protocol. After development, we freeze each task program and evaluate it in 30 real-world trials, for 240 trials in total. The same program is used throughout a task's evaluation, with runtime observations processed by its fixed decision rules. Success rate is the fraction of trials in which the robot achieves the task objective. We report mean physical execution time in minutes. The evaluation trials are separate from those used for policy development.

4.2 Reliable Execution for Real-World Manipulation

ACE-CaP-0 achieves an overall success rate of **92.1%**, succeeding in **221 of 240 trials** across the eight tasks (Figure 7 and Table 1). Seven tasks reach success rates of at least 90%. *Charger Insertion* succeeds in all 30 trials, while *Chicken Plating* reaches 96.7% (29/30). *Bread Packing* succeeds in 25 trials (83.3%).

Mean physical execution times range from **0.97 to 2.27 minutes**, with six of the eight tasks averaging less than two minutes. *Parcel Reorientation* averages 0.97 minutes with 93.3% success, while *Charger Insertion* averages 1.65 minutes with 100% success. The two longest tasks, *Bottle Uncapping* and *Shoe Packing*, average 2.23 and 2.27 minutes, respectively, both with 90% success.



Figure 7 Real-world manipulation with the code policies developed by ACE-CaP-0. Each panel shows a task image and the success rate over 30 evaluation trials, with the successful trial count alongside it. Selected panels include end-state insets.

The results demonstrate how ACE-CaP-0 turns experience acquired during policy development into reusable programs for reliable real-world manipulation.

Table 1 Task objectives, execution performance, and output token usage. Each frozen program is evaluated in 30 trials; parentheses give successful and total trial counts. Time denotes mean physical execution time in minutes. Output token usage is the cumulative output across all model calls during development, in millions (M).

Task	Task objective	Success rate	Mean time (min)	Output token usage (M)
Charger Insertion	Insert a charger plug into a socket.	100.0% (30/30)	1.65	1.09
Bread Packing	Pack bread bags into a cardboard box.	83.3% (25/30)	1.93	2.14
Bottle Uncapping	Stabilize a bottle and remove its cap.	90.0% (27/30)	2.23	1.17
Cup Nesting	Insert multiple cups into one another.	93.3% (28/30)	1.58	0.95
Flower Insertion	Insert flower stems into a vase.	90.0% (27/30)	1.37	0.92
Chicken Plating	Plate the scattered chicken pieces.	96.7% (29/30)	1.12	0.75
Parcel Reorientation	Rotate a parcel to the target orientation.	93.3% (28/30)	0.97	1.07
Shoe Packing	Place a pair of shoes into a shoebox.	90.0% (27/30)	2.27	1.87
Overall		**92.1% (221/240)**	—	—

5 Related Work

We first review code-based robot control and learning from execution, including the practical constraints of physical trials. We then examine how human guidance and simulated experience support policy development, focusing on the information they provide and when it is used.

5.1 Code-as-Policies and Agentic Robot Learning

Code-based robot control. Foundation models support robot manipulation through both learned action policies and executable programs. Vision-language-action models learn mappings from observations and instructions to actions [1, 25], while Code-as-Policies expresses behavior through programs that compose perception, geometric computation, and control interfaces [11]. This representation makes task logic and motion parameters available for inspection and revision, but the capabilities of a code policy also depend on the interfaces it can call. Studies across levels of abstraction show that high-level primitives can supply substantial task knowledge, while lower-level interfaces place more of the reasoning burden on the agent [12]. Code-based systems also differ in where decision-making resides: a generated program can govern execution, or an agent can remain in the loop to select tools and write code as the task unfolds [6, 8, 11]. The resulting behavior depends on both the available interfaces and the model's role in the control loop. COAP studies fully code-driven execution through a shared stateful library, with its reported evaluation restricted to simulated RoboDojo tasks and public benchmark assets available during development [14]. Experience-driven program development has also been evaluated on physical robots [15, 26]. We study how human guidance and simulated experiments support policy development under real sensing and contact conditions.

Robot learning from experience. Recent agentic systems use execution feedback to turn program generation into a process of policy improvement. Execution traces support failure diagnosis and iterative repair, while validated solutions and repair knowledge can be retained as reusable skills [8, 10, 12]. Online exploration extends this process across tasks and embodiments [26]. Execution outcomes can also be used to train code-generating models through reinforcement learning [12, 27]. Learned action policies can likewise improve through a combination of demonstrations, autonomous rollouts, and human interventions [3]. Systems that coordinate updates to an orchestrator and the learned policy it invokes further connect these mechanisms [28]. Program revision, skill reuse, and parameter learning offer complementary routes to improvement.

On physical robots, this learning process also depends on how experience is acquired and evaluated. Improvements from retained experience need to be distinguished from those due to additional attempts or larger interaction budgets [29]. Real-world learning also requires execution, outcome verification, and scene reset; automating these operations is itself a substantial part of the learning system [9, 30]. These constraints motivate our study of how additional experience can guide program revision when physical trials are scarce.

5.2 Human and Simulated Experience for Policy Development

Demonstrations for policy development. Beyond providing training supervision, demonstrations can initialize exploration, condition a policy at deployment, or specify the behavior that a program should implement [31–33]. In robot programming, language and visual examples help recover task structure, while richer demonstration records provide evidence for choosing and revising a manipulation strategy [13, 19]. What the agent can infer depends on the information preserved in the demonstration: human videos convey object interactions and temporal structure, while robot demonstrations may pair visual evidence with recorded poses, actions, and gripper states [13, 17]. Studies of in-context adaptation show that action references and finer temporal resolution can improve performance for the models and tasks evaluated [17, 34]. In code-policy development, the form of guidance shapes the strategies and motion details an agent can infer and test.

Selective human assistance. Interactive robot learning also considers when and where human guidance is needed. Confidence-based and interactive imitation methods request demonstrations or control interventions according to uncertainty, novelty, and risk [35, 36]. Active demonstration learning also considers trajectory-level decisions about when to request an example and which example would be useful [20]. For language-model agents, requests can resolve ambiguity in a plan, while planning-based demonstration collection invokes

human assistance for task stages beyond the planner's supported behaviors [37, 38]. Across these settings, the common principle is to seek assistance in response to the learner's current needs, with different query targets and learning objectives. Our study examines this principle in program revision by comparing guidance provided at the outset with guidance requested by the agent as development proceeds.

Simulation for policy development. Simulation complements physical interaction by making candidate behaviors available for repeated experimentation. Dyna integrates modeled experience with learning and planning, while multi-fidelity policy search chooses between simulated and physical experiments according to their information value and cost [39, 40]. Real-world feedback can also refine the simulator. Real rollouts can update its parameters between policy-training rounds, while task-driven adaptation optimizes it for the resulting policy's real-world performance [41, 42]. The usefulness of simulated evidence depends on the decision it supports and the errors that affect that decision; evaluations of sim–real agreement therefore examine policy rankings and behavior as well as simulated success [22].

Agentic methods use simulation to support planning and program revision. Language models can generate tasks and executable environments that provide opportunities for testing [43, 44]. Simulation can support action planning, including replanning from an updated real-world state after failure [45], and the construction and verification of robot programs [46, 47]. Demonstrations and physical trials can further shape what an agent tests. RAPID derives task specifications, manipulation primitives, and verification environments from a human demonstration to guide iterative program synthesis [18]. SimEX first develops a reusable toolbox through open-ended simulated experimentation, then uses limited physical trials to align the simulator, diagnose failures, and screen candidate repairs [15].

Together, these lines of work motivate examining how human guidance and simulated experimentation are organized during real-world policy development. We investigate guidance modality and timing, and compare simulation-first development with allowing the agent to invoke or construct simulation during work on the real task. In the latter setting, the agent chooses whether to simulate, which part of the task to model, which initial state to use, and which candidate changes to test. The resulting recipe combines demonstrations on request and simulation on demand within a common code-policy development process.

6 Conclusion

ACE-CaP-0 makes experience acquisition part of real-world code-policy development. Our charger-insertion investigation favors robot demonstrations available on request and simulated experiments targeted at current development questions. Together, these findings suggest organizing experience around unresolved manipulation decisions, while checking the assumptions behind program revisions through physical execution. The comparison with real-only development also reveals a resource trade-off: simulation on demand combines higher final success and shorter development time with greater output token usage.

Across eight tasks, the resulting frozen programs achieved 221 successes in 240 real-world evaluation trials (92.1%). These results show that skills acquired during development can persist in executable programs that respond to runtime observations, without a human or coding agent in the loop. Their explicit task logic could support future improvements in execution efficiency, with task success revalidated after changes.

The investigation uses one development run per condition on one task; repeated evaluation measures program reliability, while consistency across independent development runs remains to be established. Broader studies should include repeated development runs across tasks and test reuse under new conditions, with cost accounting that includes demonstration effort. A central next question is how agents can estimate the value of additional experience for policy revision and weigh it against the human effort, computation, and robot time required. This includes deciding when simulated evidence warrants a physical trial and when an unresolved manipulation choice calls for a demonstration.

References

[1] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[2] Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

[3] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, et al. $\pi^*_{0.6}$: A VLA that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

[4] OpenAI. GPT-6 Astra: A new generation of intelligence, September 2026. URL https://openai.com/index/gpt-6-astra/.

[5] Anthropic. Claude Fable 5.2 and Mythos 5.2, November 2026. URL https://www.anthropic.com/claude-fable-and-mythos-5-2. Published November 9, 2026.

[6] Mengzhao Jia, Yang Lin, Xixin Zhang, Zhihan Zhang, Xiaobai Liu, and Meng Jiang. Agent as policy for robotic manipulation. *arXiv preprint arXiv:2609.12541*, 2026.

[7] SenseNova, Joint Research Team, Zhongang Cai, Ziang Cao, Tailai Chen, Han Chong Chua, Linjun Dai, Hanming Deng, Hokin Deng, Haiwen Diao, Chenjing Ding, Weichen Fan, Xiao Fu, Zerui Gong, Chenyang Gu, Zexin He, Zhi Hou, Bo Li, Qinyue Li, Quanzhou Li, Xiaowan Li, Zhichao Liao, Dahua Lin, Zilong Lin, Bo Liu, Chunxiao Liu, Pengfei Liu, Tianqi Liu, Ziwei Liu, Chen Change Loy, Lewei Lu, Jiapeng Luo, Yuexiao Ma, Liang Pan, Xingang Pan, Oscar Qian, Ziyi Shang, Yuyao Shao, Dawei Su, Dacheng Tao, Qingyi Tao, Quan Wang, Ruisi Wang, Xiaogang Wang, Ziyi Wang, Beichen Wen, Donghao Wu, Jiaxu Wu, Silei Wu, Xiaoming Wu, Zhonghua Wu, Hongcan Xiao, Weiye Xu, Atsushi Yamashita, Lei Yang, Zhe Yang, Baiqiao Yin, Wanqi Yin, Hanfei Zhao, Tianbo Zhong, Long Zhuo, and Jing Zuo. Has gpt-6-astra achieved embodied intelligence?, 2026. URL https://arxiv.org/abs/2610.22471.

[8] Shijia Ge, Alex Zhou, Jianshu Zeng, Yexing Wan, Di Wu, Zelin Zheng, Yazhe Wang, Zhiqi Jia, Xuan Shangguan, Jay Zhu, et al. Make code as policy great again: Frontier agents write, call, and evolve robot tools. *arXiv preprint arXiv:2609.39018*, 2026.

[9] Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, et al. Enpire: Agentic robot policy self-improvement in the real world. *arXiv preprint arXiv:2606.19980*, 2026.

[10] Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, et al. Aspire: Agentic/skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026.

[11] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[12] Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, et al. Cap-x: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026.

[13] Yifan Kang, Zihan Wang, Zhiwen Fan, and Bangya Liu. Encore: Few-shot agentic discovery of manipulation strategies. *arXiv preprint arXiv:2609.37359*, 2026.

[14] Kairui Hu, Siyuan Hu, Fangzhou Hong, Zhaoxi Chen, and Ziwei Liu. Embodied turing machines: Stateful code for robot recursive self-improvement. *arXiv preprint arXiv:2610.12369*, 2026. URL https://arxiv.org/abs/2610.12369.

[15] Jiaheng Hu, Roberto Martin-Martin, Peter Stone, Rocky Duan, Zhenyu Jiang, and Guanya Shi. Simex: Simulation-integrated robotics autoresearch. *arXiv preprint arXiv:2609.38982*, 2026.

[16] Hadas Kress-Gazit, Kunimatsu Hashimoto, Naveen Kuppuswamy, Paarth Shah, Phoebe Horgan, Gordon Richardson, Siyuan Feng, and Benjamin Burchfiel. Robot learning as an empirical science: Best practices for policy evaluation. *arXiv preprint arXiv:2409.09491*, 2024.

[17] Dongzhou Cheng, Taoran Yi, Ye Fang, Xingwu Zhang, Fan Feng, Yixuan Li, Gengxiong Zhuang, Rongze Wang, Shuai Yang, Wei Song, et al. In-context robot learning with vlm agents. *arXiv preprint arXiv:2609.19138*, 2026.

[18] Yuyao Liu, Jiayuan Mao, David Hsu, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Rapid: Robot agentic programming from demonstrations. *arXiv preprint arXiv:2609.30249*, 2026.

[19] Michael Murray, Abhishek Gupta, and Maya Cakmak. Teaching robots with show and tell: Using foundation models to synthesize robot policies from language and visual demonstration. In *8th Annual Conference on Robot Learning*, 2024.

[20] Muhan Hou, Koen Hindriks, Guszti Eiben, and Kim Baraka. "give me an example like this": Episodic active reinforcement learning from demonstrations. In *Proceedings of the 12th International Conference on Human-Agent Interaction*, pages 287–295, 2024.

[21] Xiaoshuai Zhang, Rui Chen, Ang Li, Fanbo Xiang, Yuzhe Qin, Jiayuan Gu, Zhan Ling, Minghua Liu, Peiyu Zeng, Songfang Han, Zhiao Huang, Tongzhou Mu, Jing Xu, and Hao Su. Close the optical sensing domain gap by physics-grounded active stereo sensor simulation. *IEEE Transactions on Robotics*, 39(3):2429–2447, 2023. doi: 10.1109/TRO.2023.3235591.

[22] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024.

[23] Guozheng Ma, Lu Li, Zilin Wang, Pierre-Luc Bacon, and Dacheng Tao. Beyond one-size-fits-all: Diagnosis-driven online reinforcement learning with offline priors. In *The Fortieth Annual Conference on Neural Information Processing Systems Position Paper Track*, 2026. URL https://openreview.net/forum?id=XZmIeAgd6e.

[24] Anthropic. Claude Opus 5.5, September 2026. URL https://www.anthropic.com/claude-opus-5-5. Published September 22, 2026.

[25] 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. *arXiv preprint arXiv:2307.15818*, 2023.

[26] Runhan Huang, Xiangcheng Zhang, Haoyu Chen, Yitong Li, Yen-Jen Wang, Bohan Lyu, Haotian Ye, Chuye Hong, Yixuan Wang, Chloe Su, et al. Scaling online exploration for robot agents. *arXiv preprint arXiv:2610.22496*, 2026.

[27] Yijia Fan, Xilin He, Yilei Jiang, Dongming Wu, Jiaxu Wang, and Xiangyu Yue. CaPtain: A pipeline for post-training code-as-policy robot agents. *arXiv preprint arXiv:2610.40799*, 2026. URL https://arxiv.org/abs/2610.40799.

[28] Xilun Zhang, Maggie Wang, Erik Bauer, Hong-Xing Yu, Huang Huang, Jiajun Wu, and Marco Pavone. Co-evolving robot orchestrators and policies through deployment. *arXiv preprint arXiv:2610.09228*, 2026.

[29] Xinyi Wang, Bangzheng Li, Hanyu Wang, Lin Zhang, Andrea Yaoyun Cui, Zhenggang Tang, Shyamal Buch, and Dejia Xu. Sear: Self-evolving agentic robotics. *arXiv preprint arXiv:2610.18792*, 2026.

[30] Zhiyuan Zhou, Pranav Atreya, You Liang Tan, Karl Pertsch, and Sergey Levine. Autoeval: Autonomous evaluation of generalist robot manipulation policies in the real world. *arXiv preprint arXiv:2503.24278*, 2025.

[31] Shikhar Bahl, Abhinav Gupta, and Deepak Pathak. Human-to-robot imitation in the wild. In *Proceedings of Robotics: Science and Systems*, 2022. doi: 10.15607/RSS.2022.XVIII.026. URL https://www.roboticsproceedings.org/rss18/p026.html.

[32] Letian Fu, Huang Huang, Gaurav Datta, Lawrence Yunliang Chen, William Chung-Ho Panitch, Fangchen Liu, Hui Li, and Ken Goldberg. In-context imitation learning via next-token prediction. *arXiv preprint arXiv:2408.15980*, 2024. URL https://arxiv.org/abs/2408.15980.

[33] Yuki Wang, Gonzalo Gonzalez-Pumariega, Yash Sharma, and Sanjiban Choudhury. Demo2Code: From summarizing demonstrations to synthesizing code via extended chain-of-thought. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/30699996ff411d48903c9752b782a5c1-Abstract-Conference.html.

[34] Austin Patel, Ben Pekarek, Joel Enrique Castro Hernandez, and Shuran Song. What enables in-context behavior prompting for manipulation? *arXiv preprint arXiv:2606.30457*, 2026. URL https://arxiv.org/abs/2606.30457.

[35] Sonia Chernova and Manuela Veloso. Confidence-based policy learning from demonstration using gaussian mixture models. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, 2007.

[36] Ryan Hoque, Ashwin Balakrishna, Ellen Novoseller, Albert Wilcox, Daniel S Brown, and Ken Goldberg. Thriftydagger: Budget-aware novelty and risk gating for interactive imitation learning. *arXiv preprint arXiv:2109.08273*, 2021.

[37] Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. Robots that ask for help: Uncertainty alignment for large language model planners. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 661–682. PMLR, 2023. URL https://proceedings.mlr.press/v229/ren23a.html.

[38] Samrat Sahoo, Liang Ji, Tom Silver, and Yixuan Huang. Tandem: Task and motion planning with as-needed demonstrations for efficient vision-language-action model fine-tuning. *arXiv preprint arXiv:2609.28314*, 2026.

[39] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2 (4):160–163, 1991.

[40] Alonso Marco, Felix Berkenkamp, Philipp Hennig, Angela P Schoellig, Andreas Krause, Stefan Schaal, and Sebastian Trimpe. Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with bayesian optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1557–1563. IEEE, 2017.

[41] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 international conference on robotics and automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[42] Allen Z Ren, Hongkai Dai, Benjamin Burchfiel, and Anirudha Majumdar. Adaptsim: Task-driven simulation adaptation for sim-to-real transfer. *arXiv preprint arXiv:2302.04903*, 2023.

[43] 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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=14YqPG4cm0.

[44] Pranjal Aggarwal and Sean Welleck. Gym-anything for robotics: Scaling physical simulations with digital agents. *arXiv preprint arXiv:2610.40565*, 2026.

[45] Haowen Liu, Shaoxiong Yao, Haonan Chen, Jiawei Gao, Jiayuan Mao, Jia-Bin Huang, and Yilun Du. Simpact: Simulation-enabled action planning using vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20790–20801, 2026.

[46] Li Kang, Yutao Fan, Rui Li, Heng Zhou, Yiran Qin, Zhemeng Zhang, Songtao Huang, Xiufeng Song, Zaibin Zhang, Bruno N.Y. Chen, Zhenfei Yin, Dongzhan Zhou, Wangmeng Zuo, and Lei Bai. Coenv: Code-driven exploration in real-to-sim digital twins for multi-arm manipulation. *arXiv preprint arXiv:2604.05484*, 2026.

[47] Yihan Li, Yating Feng, Shengjiu Sun, Jianing Chen, Hao Ren, Bowen Yang, Weisheng Xu, Qiwei Wu, Hui Cheng, and Renjing Xu. Agentic rsr: Real-to-sim-to-real through scene reconstruction and execution-grounded robot policies. *arXiv preprint arXiv:2610.10479*, 2026.

A Contributions and Author List

Core Contributors: Changxu Wei[1,2], Yunpeng Qing[1,3], Guozheng Ma[1,4],

Contributors: Bowen Liu[1], Zexin He[1], Yilun Kong[1], Haoming Gu[1], Jingyuan Liu[1], Tianyu Zhu[1], Jianfei Wang [1], Junqiang Kou[1], Huinan Mo [1], Hao Lv [1], Pengbin Li [1], Yuesong Wang [1], Bo Li[1], Huan Zhou[1], Shunyu Liu[4], Chunxiao Liu[1], Bixuan Jin[1], Shaopeng Yin[1], Fei Wang[1],

Advisors:

Zhi Hou[1,†], Jiaxing Huang[2], Changqing Zou[3], Dacheng Tao[4], Xiaogang Wang[1].

[†]Project Leader

Affiliations:
[1]ACE Robotics
[2]The Hong Kong Polytechnic University
[3]Zhejiang University
[4]Nanyang Technological University