SIMEX: SIMULATION-INTEGRATED ROBOTICS AUTORESEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents powered by large language models (LLMs) have shown remarkable abilities to autonomously reason about and achieve goals in the digital world. However, bringing this success to the physical world remains challenging. On the one hand, direct generation methods (e.g., Code as Policies) often suffer from the LLMs' insufficient understanding of robots and physical environments. On the other hand, iterative trial-and-error tuning in the physical world (e.g., physical autoresearch) induces significant experimental cost and safety concerns. We introduce SimEX: Simulation-Integrated Robotics AutoResearch, an autoresearch framework that tightly integrates simulated experimentation, enabling coding agents to efficiently acquire physical capabilities for controlling real robots. SimEX operates in two stages. First, the agent conducts open-ended *probe-and-optimize* iterations in simulation, developing a robot toolbox with robust and generalizable capabilities. Second, the agent adapts the toolbox and the simulator together through only a few physical trials: each trial corrects the simulator, and the corrected simulator is used to diagnose failures and screen candidate repairs. We evaluate SimEX extensively in sim-to-sim settings and on physical robots. On challenging real-world manipulation tasks including towel folding, barcode scanning, and plate manipulation, SimEX enables coding agents to efficiently acquire robot skills without any demonstration and with only 10 minutes of real-robot interaction. These results suggest that simulation can be a critical component in achieving physical intelligence, not only as a source of training data for policies, but also as a laboratory where a coding agent iteratively develops capabilities through experimentation before acting on the robot. More details and robot videos at `https://robo-simex.github.io/`.

1 INTRODUCTION

Coding agents powered by large language models (LLMs) have emerged as capable problem solvers in digital environments. Given access to a workspace, they can write and debug software, resolve repository-level issues (Jimenez et al., 2024), conduct scientific experiments (Lu et al., 2024), and discover new mathematical constructions and algorithms (Romera-Paredes et al., 2024; Novikov et al., 2025). These capabilities already enable coding agents to automate substantial amounts of digital work. The next step toward realizing their full potential is to extend them beyond computers and enable them to act through robots in the physical world.

However, bringing coding agents into the physical world introduces challenges that do not arise in purely digital environments. One popular approach directly generates robot control code from a natural-language instruction (aka "Code as Policies") (Liang et al., 2023; Singh et al., 2023). While this approach produces a control program in a single pass, without any trial and error on the robot, its performance often depends on carefully designed, user-provided physical primitives, such as perception (Kamath et al., 2021; Gu et al., 2021), motion-planning (Huang et al., 2023a; Zucker et al., 2013), and manipulation functions (Sundermeyer et al., 2021). Without these primitives, which are often embodiment- and environment-specific and costly to hand-engineer, the LLM must guess quantities that appear neither in its training data nor in the prompt, such as camera calibration errors or reachable grasps. These guesses rarely meet the tight tolerances of physical tasks, and often result in catastrophic failure of these direct generation methods.

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

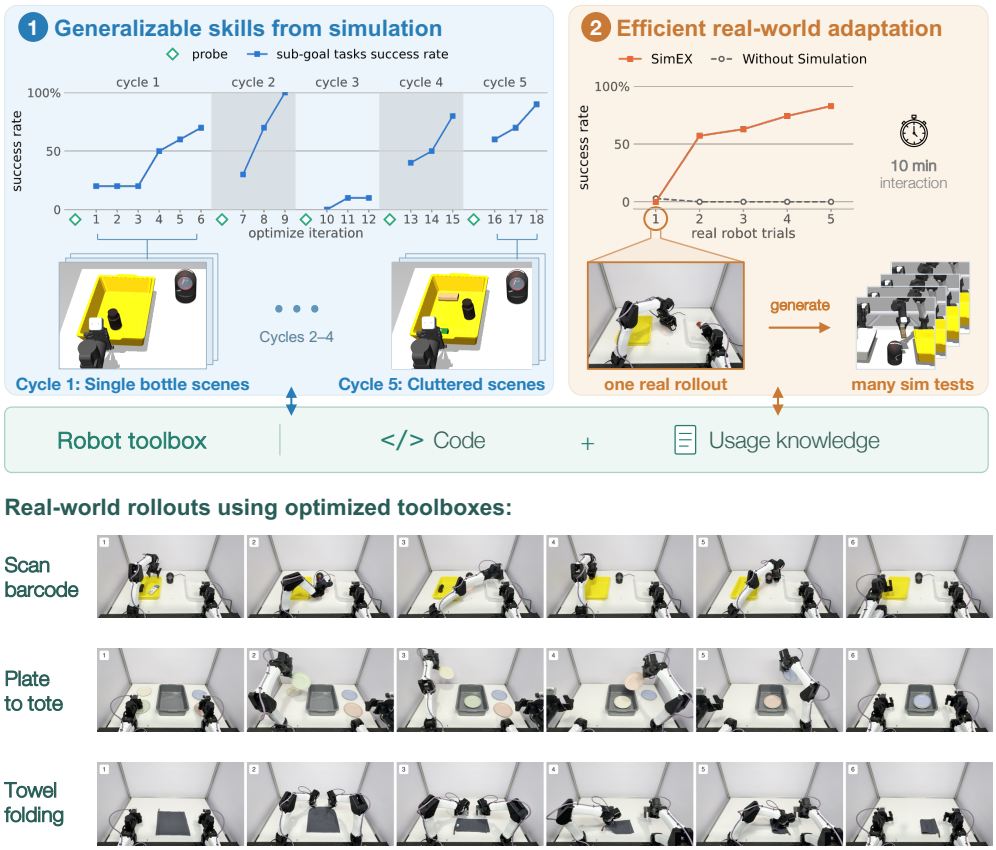


Figure 1: SimEX brings coding agents to physical robots through simulation. (1) The agent develops diverse robot capabilities through open-ended autoresearch in simulation. (2) The agent then efficiently adapts to the real robot within a few trials, turning each real rollout into many simulated tests of candidate repairs. Bottom: physical-robot rollouts using the optimized toolboxes.

Building on iterative autoresearch in software (Karpathy, 2026), a second line of work lets coding agents improve robot policies from physical feedback (Xiao et al., 2026). Unlike direct generation, this process allows the agent to learn from the consequences of its actions and correct mistakes that could not be anticipated from the initial prompt. However, experimentation on real robots is costly and potentially unsafe: each trial consumes robot time, untested code can cause collisions, and noisy rollouts can provide misleading signals for subsequent revisions. As a result, real-world autoresearch systems such as ENPIRE (Xiao et al., 2026) depend on carefully engineered physical harnesses, including automatic scene resets, success verification, and safety monitoring, to sustain long sequences of robot trials.

How can we bring coding agents onto robots without relying extensively on physical trials? Our key idea is to use simulation as a bridge between the coding agent and the physical world. Simulation is particularly well suited to this role: on the one hand, it is defined and manipulated through code, making it naturally accessible to a coding agent; on the other hand, it encodes physical laws, robot dynamics, and workspace geometry needed to solve real-world tasks. The agent can therefore repeat experiments and compare hypotheses safely in simulation before acting on the robot. However, no simulator perfectly reproduces the physical world, and capabilities optimized purely in simulation can often exploit its inaccuracies. An effective approach must therefore tolerate an imperfect simulator and ground what it learns in a small number of real-world trials.

We introduce SimEX, a simulation-integrated autoresearch framework that enables a coding agent to efficiently acquire robot capabilities. Given an autonomously constructed simulation of the workspace, SimEX follows a two-stage process (Figure 2). In Stage 1, the agent conducts open-ended autoresearch in simulation without a fixed, human-specified evaluation metric. It probes the current toolbox on self-generated tasks and workspace variations that expose its weaknesses, then improves the toolbox through repeated simulated experiments. In Stage 2, the agent adapts

the resulting toolbox through a few physical trials, using each robot rollout to generate improvement hypotheses and returning to simulation to screen candidate repairs. Importantly, the two stages make SimEX robust to an imperfect simulator (e.g., the coarse deformable simulator for our towel folding experiments): open-ended exploration in Stage 1 prevents the toolbox from overfitting to any single simulated scenario, and real-world grounding in Stage 2 corrects what the simulator gets wrong. This distinguishes SimEX from real-to-sim-to-real methods (Torne et al., 2024), which use high-fidelity simulation to train one policy for one reconstructed scene; SimEX instead uses simulation to develop reusable capabilities that transfer across tasks.

We evaluate SimEX in sim-to-sim settings and on physical robots, across four types of coding agents (Fable 5.1, Opus 5, GPT-5.5 and GPT-6 Astra) and three manipulation families spanning long-horizon object handling, contact-rich manipulation, and deformable-object control. SimEX solves these challenging tasks without demonstrations and substantially outperforms approaches that directly generate robot code or adapt purely through physical trials. On physical hardware, with only 10 minutes of on-robot interaction per task, SimEX succeeds in 26 of 30 evaluation trials across the three task families, while the strongest baseline succeeds in only 3.

To summarize, our contributions are:

- We introduce a simulation-integrated autoresearch framework that enables coding agents to efficiently control physical robots.
- We develop a novel open-ended simulated autoresearch procedure (Stage 1), in which the agent generates tasks and environment variations without a ground-truth distribution or human-specified evaluation metric, and develops generalizable capabilities.
- We develop a sample-efficient physical adaptation procedure (Stage 2) that interleaves limited physical experience with simulated experimentation.
- We demonstrate that SimEX solves challenging long-horizon, contact-rich, and deformable-object manipulation tasks on physical robots without any demonstrations.

2 METHOD

SimEX aims to enable a coding agent to efficiently acquire reusable capabilities for controlling physical robots. It begins by autonomously constructing a simulation of the physical workspace (Section 2.2). SimEX then uses this simulation to improve a portable robot toolbox through two stages. In Stage 1, SimEX develops generalizable capabilities through open-ended simulated experimentation, producing a toolbox that supports diverse tasks under workspace variations (Section 2.3). In Stage 2, SimEX adapts this toolbox with a small physical trial budget, returning to simulation to screen repairs between trials (Section 2.4). Importantly, this two-stage design confines on-robot execution to a compact final phase after the agent has already acquired an initial toolbox. This design has two benefits. First, the initial capabilities make physical rollouts safer and more efficient. Second, concentrating these trials into one phase provides the practical benefit of allowing a human operator to initialize and monitor the robot in one session rather than repeatedly throughout the longer optimization process. Figure 2 visualizes the full pipeline.

2.1 PROBLEM SETUP

SimEX optimizes a portable robot capability library, which we call the *toolbox*. It comprises two files. `toolbox.py` implements perception and control primitives, while `skill.md` documents how to use them, including measured limits and known failure modes. When the agent encounters a task, SimEX makes a fresh query to an LLM with the current toolbox and a natural-language task description. Acting as a code-as-policy model (Liang et al., 2023), the LLM writes an executable program that invokes the primitives in `toolbox.py` using the guidance in `skill.md` (prompt in Appendix A.2). Together, the LLM and toolbox can be viewed as a multi-task policy that maps a natural-language task description to a visuomotor program.

Toolbox boundary. The toolbox is an editable layer built above a fixed low-level robot API. For scene perception, the toolbox has access only to raw RGBD images, and receives no privileged scene state. The robot API also provides joint and end-effector proprioception, inverse-kinematics and reachability queries, Cartesian end-effector or absolute joint-position targets, and gripper com-

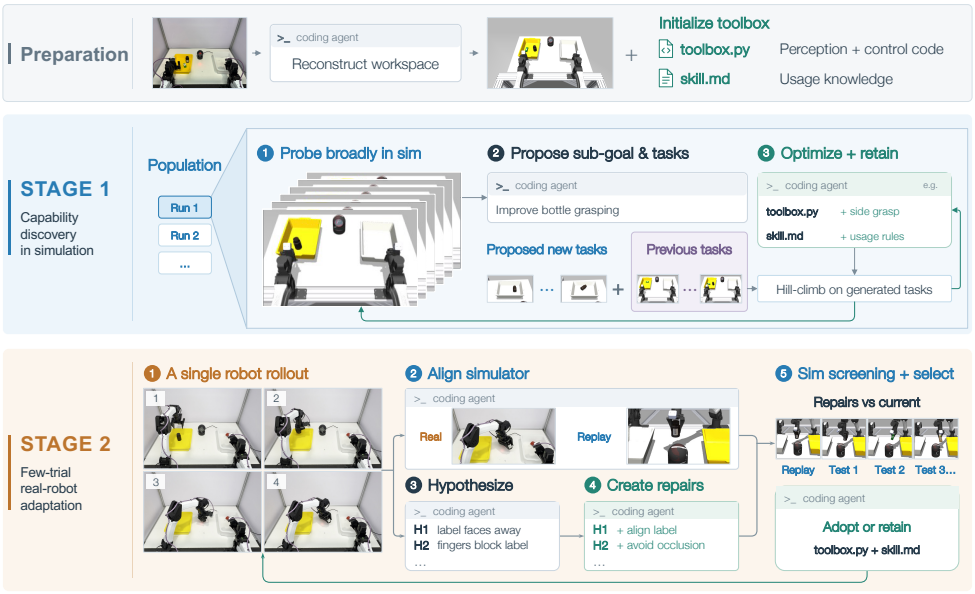


Figure 2: Overview of SimEX. Preparation reconstructs the physical workspace in simulation and initializes the toolbox. Stage 1 runs independent probe–focus–optimize cycles, retaining improvements on focused tasks and capability-bank replay. Stage 2 executes one robot rollout, aligns the simulator through fixed-policy replay, creates candidate repairs, and returns simulated screening evidence to the coding agent as supportive evidence for repair selection.

mands, but no predefined task-level manipulation skills. During optimization, SimEX improves this toolbox while the underlying robot API remains fixed.

Agent-driven autoresearch. Both optimization stages run as autoresearch loops (Karpathy, 2026) driven entirely by a coding agent. Each stage starts a coding-agent session with an `agent_program.md` that states the goal, the rules the agent must follow, the steps of one iteration, and the harness commands available to it. The agent then drives the loop autonomously. In each iteration, it chooses what to test and edits the toolbox accordingly. It then launches simulated rollouts or robot trials through the harness and records what the results teach it before the next iteration begins. The harness only executes and scores rollouts, so every optimization decision belongs to the agent. The loop ends when the stage's iteration budget is spent.

Within this loop, the agent revises `toolbox.py` and `skill.md` as a pair. The task-specific program generated via CaP is discarded after execution; only the shared toolbox persists. Consequently, knowledge transfers across iterations only through updates to the toolbox. The toolbox therefore stores task-agnostic capabilities rather than a solution to one instruction, allowing it to support many tasks and workspace variations.

2.2 Preparation: Constructing the Simulation Sandbox

SimEX starts by constructing a simulation sandbox from an image of the physical workspace. The goal is to produce an executable approximation that subsequent autoresearch can build on, rather than a precisely calibrated or photorealistic digital twin. In principle, this initialization can be produced by an off-the-shelf real-to-sim system, such as SimFoundry or VIGA (Ranawaka et al., 2026; Yin et al., 2026). In our implementation, we simply give a frontier coding agent the high-level objective of reconstructing the observed physical workspace in a target simulation platform (e.g., mjlab). The agent has access to an empty YAM station scene template and a predefined library of 3D meshes, and receives a workspace image and a natural-language description of the objects present. It autonomously selects and configures the assets.

The reconstruction initializes three components: the simulator with the robot and workspace geometry, an initial `toolbox.py` containing an initial perception module based on the input image, and an initial `skill.md` describing the fixed robot API. Together, these components provide an executable starting point for the agent's optimization.

2.3 STAGE 1: OPEN-ENDED CAPABILITY DISCOVERY IN SIMULATION

With this simulation initialization in place, the most direct approach would be to optimize only for the requested task in the reconstructed simulator. Doing so, however, risks overfitting to simulator-specific geometry and dynamics that may not transfer to the physical system. Stage 1 instead asks the coding agent to acquire broad capabilities before any robot trial by repeatedly discovering and repairing weaknesses in the toolbox. The agent generates both the tasks used to expose these weaknesses and the success conditions used to evaluate them, rather than optimizing against a fixed task distribution or human-specified metric.

Specifically, the coding agent first probes the current toolbox with a broad set of self-generated tasks and environment variations. It then selects one systematic weakness, constructs a focused training set around it, and improves the toolbox on these tasks together with replay tasks drawn from previously acquired capabilities. The updated toolbox begins the next cycle, which probes broadly again to discover a new weakness. SimEX runs several independent copies of this complete cycle in parallel to diversify the resulting capabilities, where the copies can learn from each other. We describe these steps in detail below. Figure 2 (blue panel) illustrates this stage.

Step 1: open-ended capability probing. The probing phase searches for capabilities that the current toolbox is missing. The coding agent generates open-ended task and environment variations, such as changes to the task goal, object categories, object physical properties (e.g., mass and friction), object placements, or the robot's initial configuration. For each configuration, it also writes an executable success predicate over simulator state. These predicates make the self-generated tasks automatically scorable without imposing one global evaluation metric. The agent runs these configurations through the harness and compares their successes and failures to identify a systematic limitation. Probing therefore prioritizes discovering informative weaknesses over maximizing performance on a predetermined task set. The outcome of probing is a set of candidate weaknesses for the next step to choose from.

Step 2: sub-goal selection. A broad probe set may reveal several unrelated failures. To produce a coherent toolbox update, the coding agent selects one systematic weakness as the sub-goal for the next optimization cycle and constructs a fixed set of training tasks that isolates it. Fixing the sub-goal and training tasks within one cycle provides a stable objective for comparing successive toolbox revisions in the next step.

Step 3: sub-goal-driven autoresearch. With the training set fixed, the agent proposes revisions to `toolbox.py`, `skill.md`, or both and hill-climbs their performance on this task set through repeated simulated experiments. The improved toolbox is then probed again, which starts the next cycle. Repeating the full probe–focus–optimize cycle over newly discovered sub-goals gradually expands the toolbox's capability coverage. Two further mechanisms keep this loop from narrowing: capability-bank replay guards against forgetting across cycles, and population-based optimization diversifies the self-generated curriculum.

Capability-bank replay. Focusing on one weakness creates the risk that an edit removes capabilities learned in earlier cycles. To guard against such regressions, the coding agent maintains a capability bank of previously solved tasks and samples replay tasks from it during hill-climbing. Autoresearch therefore optimizes the new sub-goal and previously acquired capabilities jointly.

Population-based optimization. Even repeated cycles can follow a narrow self-generated curriculum and/or get stuck in local optima. To diversify the search, SimEX maintains a population of independent optimization processes. Each member executes the full probe–focus–optimize cycle with its own worktree, toolbox, generated task sets, and capability bank, and terminates after a fixed exploration budget of 20 optimization iterations in our experiments. At probe boundaries, one member may inspect a short description of another member's experiments. It may reimplement an idea but cannot copy the other implementation. This exchange allows useful hypotheses to spread while preserving diverse optimization trajectories and independent implementations.

2.4 STAGE 2: SIMULATION-ASSISTED FEW-TRIAL ADAPTATION ON THE ROBOT

Although SimEX develops broad capabilities in Stage 1, directly porting the resulting toolbox to a physical robot can still produce task failures due to the sim-to-real gap. Stage 2 bridges this gap by using a small number of real-robot rollouts to improve the toolbox. The central idea of Stage 2 is to

turn each scarce robot rollout into a batch of simulated experiments. Rather than spending the next physical trial on the first plausible edit, SimEX uses simulation to screen several candidate repairs, each built from a distinct improvement hypothesis.

Each iteration begins with one rollout on the robot. The harness records the raw execution and replays the executed policy in simulation. When necessary, the coding agent first updates the simulator until the replay reproduces the relevant deployment behavior. Next, the agent formulates distinct, testable hypotheses about how the toolbox can be improved, and implements several candidate repairs based on these hypotheses. Finally, the agent screens these repairs by running every candidate and the current toolbox through simulated replays and test tasks. The resulting videos, traces, and scores are returned to the coding agent as context. Based on this context, the agent judges how much to trust each simulated test, and decides which repair, if any, to use in the next robot rollout. Thus, one physical rollout supports many offline experiments instead of one untested physical edit. We describe the five steps of one iteration below. Figure 2 (orange panel) illustrates this stage.

Step 1: robot rollout. At the start of an iteration, the harness gives the current toolbox and task description to the policy-writing model, which produces one script through the same prompt used during Stage 1 (Appendix A.2). The script is executed once. The harness stores the camera videos, every image frame read by the policy, and the full policy transcript, including the perception and motion information printed during execution. These raw artifacts provide the evidence for subsequent simulated experiments.

Step 2: replay-based simulator alignment. Immediately after the robot rollout is scored, the harness replays the exact executed policy in simulation under the same task configuration. The coding agent qualitatively compares the recorded real and simulated traces and uses its own judgment to determine whether the simulator reproduces the behavior relevant to the observed outcome. If it does not, the agent keeps updating the simulator and reruns the same fixed-policy replay. These corrections persist and are applied to subsequent screening rollouts.

Step 3 & 4: hypothesis and candidate repair generation. In parallel, the agent diagnoses the rollout. A robot rollout reveals what happened but may support several explanations for what should change. The coding agent therefore formulates concrete hypotheses from the recorded evidence, each identifying a toolbox limitation and predicting an observable improvement. Based on these hypotheses, the coding agent creates several copies of the current toolbox and implements a distinct candidate repair in each one. The next step tests these predictions in simulation.

Step 5: repair screening. To test these predictions before the next robot trial, the agent launches a screen in which the harness runs each candidate and the current toolbox in two settings. It first replays the executed robot script from the reconstructed trial state, holding the script fixed. It then evaluates them on a small set of simulated test tasks designed to expose their differences and check whether a local fix generalizes beyond the observed failure. Importantly, screening does not rank the candidates or apply an automatic selection rule. Instead, the coding agent receives the resulting videos, traces, and scores as context, judges how much to trust each test given any remaining mismatch between simulation and the robot, and selects a repair or retains the current toolbox. The selected toolbox is used for the next robot rollout, which begins the next iteration.

3 EXPERIMENTS

We evaluate SimEX on three families of challenging robot manipulation tasks spanning long-horizon object handling, contact-rich manipulation, and deformable-object control. The evaluation combines controlled sim-to-sim comparisons with baselines, tests on a physical robot, ablations of each stage, and comparisons across four coding agents.

Through these experiments, we seek to answer five questions: (1) How effectively does SimEX enable coding agents to solve challenging manipulation tasks compared with alternative approaches (Section 3.2)? (2) Does this advantage carry over to a physical robot (Section 3.3)? (3) Which design choices within each stage account for its performance (Section 3.4)? (4) Do these conclusions hold across different coding agents (Section 3.4)? (5) How does synthesizing a program compare with using the coding agent itself as an online policy (Section 3.4)?

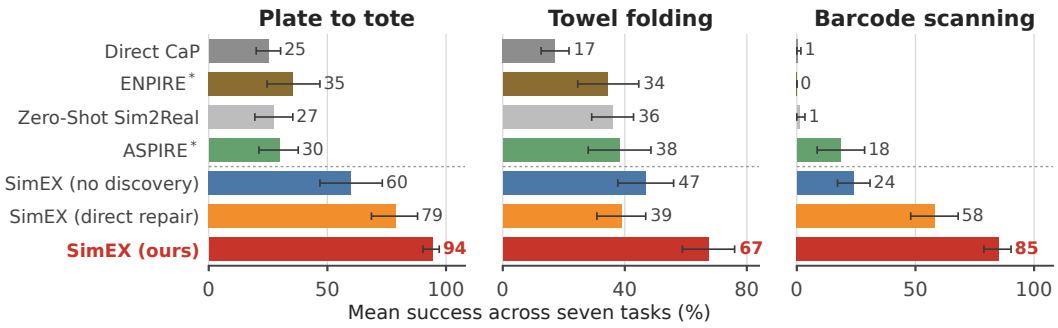


Figure 3: Aggregate performance across the three evaluation task families.

3.1 Experimental setup

Tasks. We consider three manipulation task families that require different capabilities (Figure 7). In *plate to tote*, the robot moves plates from a table into a tote; a plate lying flat cannot be picked up with a straightforward top-down grasp, so the agent must discover a geometry-aware strategy. In *towel folding*, the robot brings a deformable towel into a target state, such as shifted, compacted, or folded, and the outcome is highly sensitive to grasp location and drag trajectory. *Barcode scanning* is a long-horizon task in which the robot presents the label of each object to a fixed barcode reader before stowing it in a second tote. All tasks use a dual-arm YAM robot with one overhead camera and two wrist cameras. For each family, we define seven evaluation tasks that are held out from the coding agent. Appendix B describes each family and task in detail.

Evaluation protocol. For each task, every adaptive method receives on-robot five trials. After optimization, we freeze the resulting toolbox and evaluate it in five i.i.d. rollouts, generating a fresh task program from the frozen toolbox for each rollout. We repeat this process with five random seeds, yielding 25 evaluation trials per method–task pair. Non-adaptive baselines follow the same five-seed protocol without making deployment-time updates. We report successful trials over the 25 scheduled evaluation trials. Unless noted otherwise, scene-level results are unweighted means of the seven task-level success rates, and error bars show 95% bootstrap confidence intervals that resample runs within each task. All coding-agent and policy-writing calls use Fable 5.1, with maximum reasoning effort in Stage 1 and medium reasoning effort in Stage 2, except in the cross-agent comparison (Section 3.4). Table 2 summarizes the autoresearch and evaluation budgets used throughout the experiments. We report the agent thinking time and token usage of both stages in Appendix A.3.

Baselines. We compare against Direct CaP (Liang et al., 2023), ENPIRE* (Xiao et al., 2026), Zero-Shot Sim2Real, which follows DrEureka (Ma et al., 2024b), and ASPIRE* (Lu et al., 2026). We also include two SimEX variants, SimEX (no discovery) and SimEX (direct repair), which remove open-ended discovery in Stage 1 and sandbox-guided repair in Stage 2, respectively. An asterisk marks a baseline that we reimplement under our common interface and budget. Appendix C describes each method and the controlled-comparison setup.

3.2 Controlled sim-to-sim evaluation

Comparing trial efficiency requires many repeated adaptation runs, which is impractical to perform entirely on hardware. We therefore use an independently implemented simulator as a stand-in for the physical robot. Barcode scanning and plate to tote use MuJoCo (Todorov et al., 2012) as the sandbox and Isaac Sim (NVIDIA, 2025) as the evaluation environment. Towel folding uses FLASH (Luo et al., 2026) as the sandbox and MJWarp Flex (Google DeepMind & NVIDIA, 2025) as the evaluation environment. In each pair, the two simulators have distinct physics implementations, contact solvers, assets, rendering pipelines, and system parameters. The coding agent receives the same observations and task outcomes available in the real robot setup, but cannot inspect the evaluation simulator's state or source code. This setting preserves a substantial transfer gap while supporting controlled repetition.

We first summarize aggregate performance across the three task families in Figure 3, then report per-task success counts in Table 4 (Appendix D.1). Figure 3 shows that SimEX achieves the highest average success in all three task families. It also outperforms both stage-level ablations in every

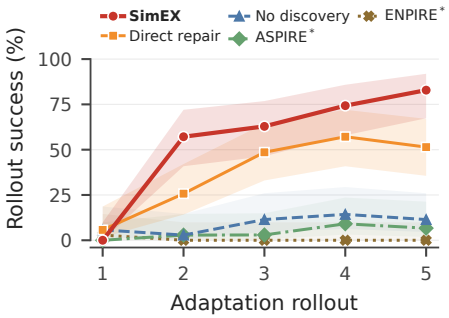


Figure 4: Online adaptation (Stage 2). Each point is the average success rate of the deployment rollouts at that iteration.

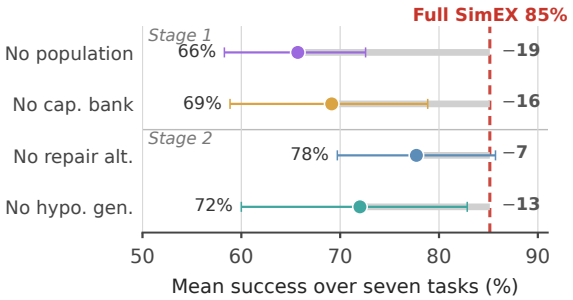


Figure 5: Fine-grained ablations. Points show mean task success, with the full SimEX result marked by the red line. Every ablation reduces performance.

family, supporting the contributions of open-ended discovery in Stage 1 and sandbox-guided repair in Stage 2. The per-task results show that this advantage is broad rather than driven by a few tasks: SimEX matches or exceeds every external baseline on all 21 tasks. The separation is largest on barcode scanning, the longest-horizon family, where SimEX reaches 85% average success and no external baseline exceeds 18%.

Figure 4 reports online adaptation (Stage 2) using the actual deployment rollout from each iteration on the barcode task. The two methods that share the same Stage 1 toolbox both begin near zero, so the toolbox does not transfer directly. However, a single deployment rollout and one repair improve both sharply, suggesting that Stage 1 still gives Stage 2 a strong starting point for repair. Sandbox guidance more than doubles this gain, and the gap persists through the fifth rollout, showing that the sandbox improves how effectively limited deployment feedback is converted into repairs.

3.3 Real-robot evaluation

We next evaluate whether the same adaptation loop can operate on physical hardware. We deploy each method on the physical YAM robot under common task criteria; adaptive methods receive the same five-trial adaptation budget. These five trials amount to only about 10 minutes of wall-clock robot interaction per task. Between each trial, a human operator resets the scene. We select one representative task from each family: plate to tote, towel folding, and barcode scanning. The selected tasks respectively require stowing four plates, folding a towel in half, and scanning and stowing two objects. We then evaluate each method in 10 physical trials per task.

Table 1: Physical-robot successful evaluation trials out of 10.

Method	Plate	Towel	Barcode
Direct CaP	0/10	0/10	0/10
ENPIRE*	2/10	1/10	0/10
ZS Sim2Real	0/10	2/10	0/10
ASPIRE*	2/10	0/10	0/10
SimEX (ours)	**10/10**	**8/10**	**8/10**

As shown in Table 1, SimEX succeeds in 26 of 30 physical evaluation trials, while no baseline succeeds in more than 3. The advantage is consistent across all three task families, including the two-object barcode scanning task, on which every baseline fails all trials. These results show that the capabilities developed in simulation and repaired through limited physical feedback transfer across contact-rich, deformable, and long-horizon real-world manipulation tasks.

Qualitatively, the adapted barcode-scanning policy discovers an efficient single-arm strategy that grasps, presents, and stows both objects without unnecessary bimanual coordination; the plate policy learned to change grasp side based on the distance between the plate and the tote; while the towel policy learned to flatten the towel after folding. Videos showcasing these real-robot trajectories can be found on our website.

3.4 Ablations and further analysis

Ablation study. The main comparison in Section 3.2 already ablates each stage as a whole. Here, we ablate the mechanisms within each stage: removing the population or capability-bank replay in Stage 1, and removing repair alternatives or hypothesis generation in Stage 2 (Figure 5). Appendix D.2 defines each ablation and reports per-task counts. Results show that every ablation

reduces performance. The largest drops arise from the two Stage 1 ablations, removing population diversity and capability-bank replay.

Robustness across coding agents. The main experiments use Fable 5.1 for both the optimization loop and all policy-writing calls. To test whether our conclusions depend on this agent, we repeat the complete seven-task barcode scanning comparison with three alternatives: Opus 5, GPT-5.5, and GPT-6 Astra each replaces Fable 5.1 as both the optimization agent and policy-writing model. All agents use their highest reasoning effort in Stage 1 and medium reasoning effort in Stage 2.

Our conclusion is stable across coding agents. With every agent, SimEX is the strongest method and leads the best external baseline by more than 50 percentage points (Figure 6; Appendix D.3, Figure 11). Interestingly, absolute performance does depend on the strength of the coding agent. Under otherwise identical end-to-end settings, replacing GPT-5.5 with GPT-6 Astra raises SimEX from 67% to 82%. This result suggests that improvements in coding-agent reasoning can translate into stronger robot performance, allowing our framework to benefit from future model progress. Per-task results for all four configurations appear in Appendix D.3.

	Fable 5.1	Opus 5	GPT-5.5	GPT-6 Astra
Direct CaP	1	0	0	0
ENPIRE*	0	9	0	0
Zero-Shot Sim2Real	1	2	4	0
ASPIRE*	18	0	11	6
SimEX (no discovery)	24	18	13	11
SimEX (direct repair)	58	62	55	55
SimEX (ours)	**85**	**77**	**67**	**82**

Figure 6: Cross-agent comparison. Our main conclusion holds with different coding agents.

Comparison with online agent control. An alternative to synthesizing a robot program is to use the coding agent itself as an online policy (Zhang et al., 2026b): after each observation, the agent emits the next low-level action. We compare the two deployment modes on three barcode scanning tasks (Appendix D.4). SimEX makes a single model call of about 30 seconds per task before execution, and all three generated programs complete their validation rollout. The online policy instead waits over 50 times longer for model responses than it spends executing actions, and none of its rollouts completes the task within the allotted wall-clock budget. Moving coding-agent inference out of the execution loop therefore makes SimEX substantially more practical for robotics.

4 Related Work

SimEX is most closely related to four lines of work: LLM-based robot control (Liang et al., 2023; Huang et al., 2023b; Vemprala et al., 2024), agentic optimization (Karpathy, 2026; Ma et al., 2024a; Lu et al., 2026; Xiao et al., 2026), generative simulation and environment design (Dennis et al., 2020; Wang et al., 2023b; 2024), and sim-to-real transfer (Tobin et al., 2017; Ma et al., 2024b; Kumar et al., 2021). Unlike these works, SimEX uses simulation as a laboratory in which a coding agent builds a reusable code toolbox, and returns to simulation between physical trials to diagnose failures and screen repairs. We discuss each line of work in detail in Appendix E.

5 Conclusion

We introduced SimEX, a framework that uses simulation as a bridge between coding agents and physical robots. SimEX acquires broad capabilities through open-ended simulated autoresearch in Stage 1, then uses limited physical experience in Stage 2 to identify and repair transfer failures. Across controlled sim-to-sim benchmarks and physical-robot tasks, this design outperforms direct generation, deployment-only iteration, simulation-only transfer, and simpler simulated search, while remaining robust across four coding agents. More broadly, our results show that simulation is useful not only for direct policy transfer, but also as a laboratory in which a coding agent iteratively develops and repairs robot capabilities before returning to the physical world. One limitation is that our evaluation uses a single robot platform; an important direction for future work is to evaluate SimEX on additional robot platforms. In addition, a human operator still resets the scene between physical trials in Stage 2; future work that automates these resets would make SimEX fully autonomous.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology and experiments, implement methods, and support qualitative and thematic data analysis. We did not use generative AI to develop theoretical models or conceptual frameworks, propose or refine hypotheses, assist with translation, interpret results, generate synthetic datasets, formulate mathematical claims, provide ingredients for proofs, write proofs, or clean and reformat datasets. We reviewed all AI-assisted work, and verified and tested LLM-generated code. We take responsibility for all text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Sections 2.3 and 2.4 specify the optimization loops, while Section 3.1 describes the controlled evaluation protocol. Table 2 reports the hyperparameters, Appendix A.2 gives the policy-writing prompt, and Appendix B provides every evaluation task and success condition. We will release the implementation, prompts, initial toolbox, task specifications, and per-iteration records on GitHub.

REFERENCES

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)*, pp. 8973–8979. IEEE, 2019.

Yanzhe Chen, Zechen Bai, Zhijun Cao, Wenzheng Zeng, Kevin Qinghong Lin, Yiqi Lin, Guoqiang Liang, Kevin Yuchen Ma, Qiming Huang, and Mike Zheng Shou. Show-harness: Just a vlm agent can play robots. *arXiv preprint arXiv:2609.10522*, 2026.

Tianyuan Dai, Josiah Wong, Yunfan Jiang, Chen Wang, Cem Gokmen, Ruohan Zhang, Jiajun Wu, and Li Fei-Fei. Automated creation of digital cousins for robust policy learning. *arXiv preprint arXiv:2410.07408*, 2024.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

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.

Google DeepMind and NVIDIA. Mujoco warp (mjwarp): A gpu-optimized implementation of mujoco in nvidia warp, 2025. URL `https://github.com/google-deepmind/mujoco_warp`.

Xiuye Gu, Tsung-Yi Lin, Weicheng Kuo, and Yin Cui. Open-vocabulary object detection via vision and language knowledge distillation. *arXiv preprint arXiv:2104.13921*, 2021.

Huy Ha, Pete Florence, and Shuran Song. Scaling up and distilling down: Language-guided robot skill acquisition. In *Conference on robot learning*, pp. 3766–3777. PMLR, 2023.

Jiaheng Hu, Peter Stone, and Roberto Martín-Martín. SLAC: Safe and efficient real-robot reinforcement learning via unsupervised simulation pre-training, 2025. URL `https://arxiv.org/abs/2506.04147`.

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

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*, 2023a.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In Karen Liu, Dana Kulic, and Jeff Ichnowski (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 14–18 Dec 2023b. URL https://proceedings.mlr.press/v205/huang23c.html.

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.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pp. 4940–4950. PMLR, 2021.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world Github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Aishwarya Kamath, Mannat Singh, Yann LeCun, Gabriel Synnaeve, Ishan Misra, and Nicolas Carion. Mdetr-modulated detection for end-to-end multi-modal understanding. In *2021 IEEE/CVF international conference on computer vision (ICCV)*, pp. 1760–1770. IEEE, 2021.

Andrej Karpathy. autoresearch. GitHub repository, 2026. URL https://github.com/karpathy/autoresearch.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, May 2023. doi: 10.1109/ICRA48891.2023.10160591. URL https://doi.org/10.1109/ICRA48891.2023.10160591.

Fangchen Liu, Kuan Fang, Pieter Abbeel, and Sergey Levine. Moka: Open-vocabulary robotic manipulation through mark-based visual prompting. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery, 2024. URL https://arxiv.org/abs/2408.06292.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Yuke Zhu, Linxi "Jim" Fan, and Guanzhi Wang. ASPIRE: Agentic /skills discovery for robotics, 2026. URL https://arxiv.org/abs/2607.00272.

Siyuan Luo, Bingyang Zhou, Chong Zhang, Xin Liu, Zhenhao Huang, Gang Yang, Zhengtao Han, Xiaotian Hu, Eric Yang, Rymon Yu, Ziqiu Zeng, and Fan Shi. FLASH: Fast learning via GPU-accelerated simulation for high-fidelity deformable manipulation in minutes, 2026. URL https://arxiv.org/abs/2604.17513.

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 *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=IEduRUO55F.

Yecheng Jason Ma, William Liang, Hung-Ju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. Dreureka: Language model guided sim-to-real transfer, 2024b. URL https://arxiv.org/abs/2406.01967.

Yao Mu, Junting Chen, Qinglong Zhang, Shoufa Chen, Qiaojun Yu, Chongjian Ge, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, et al. Robocodex: Multimodal code generation for robotic behavior synthesis. *arXiv preprint arXiv:2402.16117*, 2024.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

NVIDIA. NVIDIA Isaac Sim, 2025. URL https://docs.isaacsim.omniverse.nvidia.com/5.1.0/index.html. Version 5.1.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pp. 17473–17498. PMLR, 2022.

Nadun Ranawaka, Josiah Wong, Wei-Lin Pai, Wei-Teng Chu, Tianyuan Dai, Masoud Moghani, Hang Yin, Yunfan Jiang, Wesley Durbano, Brandon Huynh, Yu Fang, Danfei Xu, Ruohan Zhang, Li Fei-Fei, Linxi Fan, Bowen Wen, Ajay Mandlekar, and Yuke Zhu. Simfoundry: Modular and automated scene generation for policy learning and evaluation, 2026. URL https://arxiv.org/abs/2606.28276.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, Jan 2024. ISSN 1476-4687. doi: 10.1038/s41586-023-06924-6. URL https://doi.org/10.1038/s41586-023-06924-6.

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 *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, May 2023. doi: 10.1109/ICRA48891.2023.10161317. URL https://doi.org/10.1109/ICRA48891.2023.10161317.

Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE international conference on robotics and automation (ICRA)*, pp. 13438–13444. IEEE, 2021.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30. IEEE, Sept 2017. doi: 10.1109/IROS.2017.8202133. URL https://doi.org/10.1109/IROS.2017.8202133.

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, Oct 2012. doi: 10.1109/IROS.2012.6386109. URL https://doi.org/10.1109/IROS.2012.6386109.

Marcel Torne, Anthony Simeonov, Zechu Li, April Chan, Tao Chen, Abhishek Gupta, and Pulkit Agrawal. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. *arXiv preprint arXiv:2403.03949*, 2024.

Sai H Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. *Ieee Access*, 12:55682–55696, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023a. URL https://arxiv.org/abs/2305.16291.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. In *International Conference on Learning Representations*, volume 2024, pp. 4890–4924, 2024.

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*, 2023b.

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.

Mengdi Xu, Peide Huang, Wenhao Yu, Shiqi Liu, Xilun Zhang, Yaru Niu, Tingnan Zhang, Fei Xia, Jie Tan, and Ding Zhao. Creative robot tool use with large language models. *arXiv preprint arXiv:2310.13065*, 2023.

Shaofeng Yin, Jiaxin Ge, Zora Zhiruo Wang, Chenyang Wang, Xiuyu Li, Michael J. Black, Trevor Darrell, Angjoo Kanazawa, and Haiwen Feng. Vision-as-inverse-graphics agent via interleaved multimodal reasoning, 2026. URL https://arxiv.org/abs/2601.11109.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montserrat Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 374–404. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/yu23a.html.

Jesse Zhang, Jiahui Zhang, Karl Pertsch, Ziyi Liu, Xiang Ren, Minsuk Chang, Shao-Hua Sun, and Joseph J Lim. Bootstrap your own skills: Learning to solve new tasks with large language model guidance. *arXiv preprint arXiv:2310.10021*, 2023.

Junyi Zhang, Jiaxin Ge, Hanjun Yoo, Letian Fu, Zihan Yang, Yaowei Liu, Raj Saravanan, Shaofeng Yin, Justin Yu, Dantong Niu, et al. Playful agentic robot learning. *arXiv preprint arXiv:2606.19419*, 2026a.

Wenbo Zhang, Kaixuan Wang, Yutao Ouyang, Xiaoyu Huang, Liyang Li, Kailun Su, Weiyang Jin, Wenhao Chai, Haotian Liang, Zhiyang Dou, Yue Chen, and Tianxing Chen. An unexpected robot policy: Early evaluations of gpt-6 astra on robodojo and beyond, 2026b. URL https://arxiv.org/abs/2609.24170.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, et al. Harness vla: Steering frozen vlas into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026c.

Enshen Zhou, Qi Su, Cheng Chi, Zhizheng Zhang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, and He Wang. Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6919–6929. IEEE, 2025.

Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International journal of robotics research*, 32(9-10):1164–1193, 2013.

A IMPLEMENTATION DETAILS

A.1 HYPERPARAMETERS

Table 2 lists the default settings used in the main experiments.

Table 2: Default hyperparameters used in the main experiments. Population size, per-process pre-training budget, deployment-trial budget, and evaluation protocol are matched across methods whenever the corresponding phase applies.

Phase	Hyperparameter	Value
Stage 1 & 2	Coding-agent orchestrator	Fable 5.1
Stage 1 & 2	Policy-writing model	Fable 5.1
Stage 1	Reasoning effort	max
Stage 2	Reasoning effort	medium
Stage 1	Independent population members	4
Stage 1	Optimization iterations per member	20
Stage 1	Broad probe configurations per iteration	24–32
Stage 1	Focused configurations per selected sub-goal	∼10
Stage 1	Capability-bank replay configurations per candidate	4–6
Stage 2	Robot rollouts per iteration	1
Stage 2	Repair candidates per iteration	4
Stage 2	Adaptation trials per run	5
Stage 2	Agent-written test scenes per iteration	5
Evaluation	Independent runs per method–task pair	5
Evaluation	I.i.d. evaluation trials per run	5

A.2 POLICY-WRITING PROMPT

Every script executed in this paper, in simulation or on the robot, comes from the same one-shot, code-as-policy call. The prompt below is shared by SimEX, every baseline, and every evaluation trial, across all three task families. It concatenates the current `skill.md`, the current `toolbox.py` source, the task, and a fixed output format. Slots shown in <...> and *italics* are filled in verbatim; everything else is literal. The call is text-only: if the model uses a tool or takes more than one turn, the trial is voided rather than retried.

Policy-writing (code-as-policy) prompt

```
<skill.md>

----- TOOLBOX SOURCE (reference) -----
The current Toolbox implementation is below. Use it however you like:
call any helper, read what's available, or write your own code that
bypasses tb entirely. The only HARD rule is the no-cheating constraint
in the skill text above (no reading or writing of privileged object
state).

<toolbox.py>

----- TASK -----
Name: $TASK_NAME
Description: $TASK_DESC

This is a pure text-generation call. Do not inspect files, do not run
shell commands, do not patch the filesystem, and do not use tools. The
skill text, toolbox source above, and task description contain all
required context.

OUTPUT FORMAT - overrides any conflicting instruction in skill.md:

===PLAN_START===
```

```
concise plan, <=200 words
===PLAN_END===

===SCRIPT_START===
complete Python script, no markdown fences
===SCRIPT_END===

Output ONLY the marker blocks. Do NOT call any tools.
```

A.3 AGENT THINKING TIME AND TOKEN USAGE

Table 3 reports the cost of the orchestrating coding agent on average for the barcode scanning tasks. Agent thinking time sums, over all orchestrator calls, the time from the tool result or prompt to the model's reply. Output tokens include reasoning tokens. Note that these costs are for toolbox optimization: once the toolbox is optimized, running inference for a given task takes only around 30 seconds.

Table 3: Coding agent orchestration cost.

	Stage 1 (per member)	Stage 2 (per run)
Orchestrator calls	840	124
Agent thinking time	5.4 h	31 min
Output tokens (reasoning)	1.98M (1.16M)	161k (64k)
Cache-read input tokens	155M	19.5M

B EVALUATION TASK DETAILS

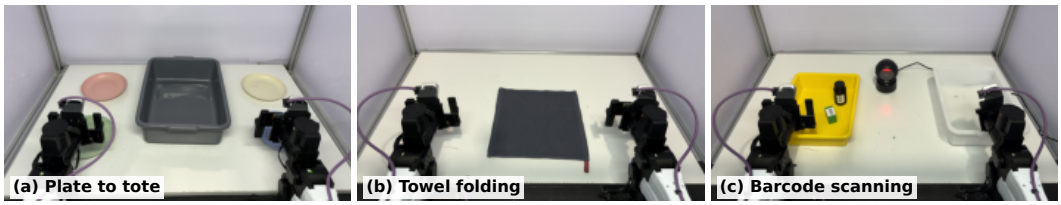


Figure 7: Physical setups for the three evaluation task families: plate to tote, towel folding, and barcode scanning. All use the same dual-arm YAM platform and workspace enclosure, while task-specific objects and fixtures produce distinct manipulation challenges.

B.1 PLATE TO TOTE

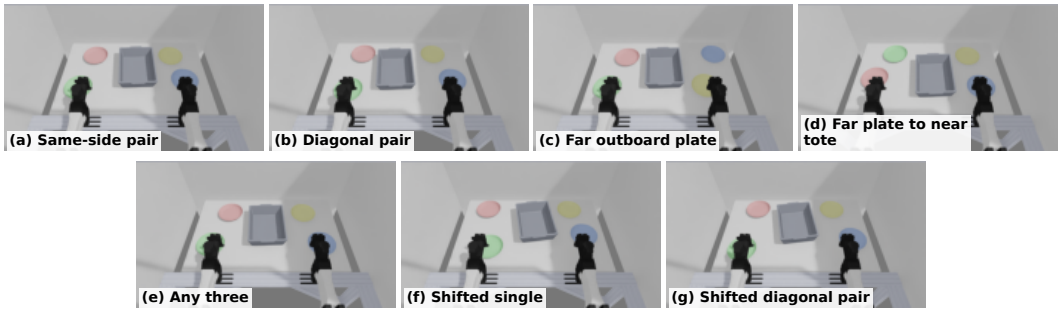


Figure 8: Initial states of the seven plate to tote tasks. (Isaac Sim)

The robot must move one or more plates from a table into a tote. A plate lying flat cannot be picked up with a straightforward top-down grasp. The task therefore tests whether the coding agent can discover a geometry-aware strategy, such as a lateral rim pinch, and turn it into collision-free grasp, lift, and transport motions.

Every delivered plate must first be lifted at least 5 cm and end inside the tote; plates not named by the task must remain outside.

- **Same-side pair.** Move the near green and far red plates from the left side into the tote, one after the other.
- **Diagonal pair.** Move the near-left green and far-right yellow plates, requiring both arms and two distinct reaches.
- **Far outboard plate.** Move the blue plate from the far, outboard edge of the right arm's workspace.
- **Far plate to near tote.** Move a far-left green plate into a tote shifted toward the robot, producing a long return carry.
- **Any three.** Choose and move any three of the four plates; exactly one may remain on the table.
- **Shifted single.** Move the near-left green plate after the tote is translated 3.5 cm, shifted laterally by 3 cm, and rotated 6°; all plate positions are also perturbed.
- **Shifted diagonal pair.** Repeat the green–yellow diagonal delivery under the same shifted fixture layout.

B.2 TOWEL FOLDING

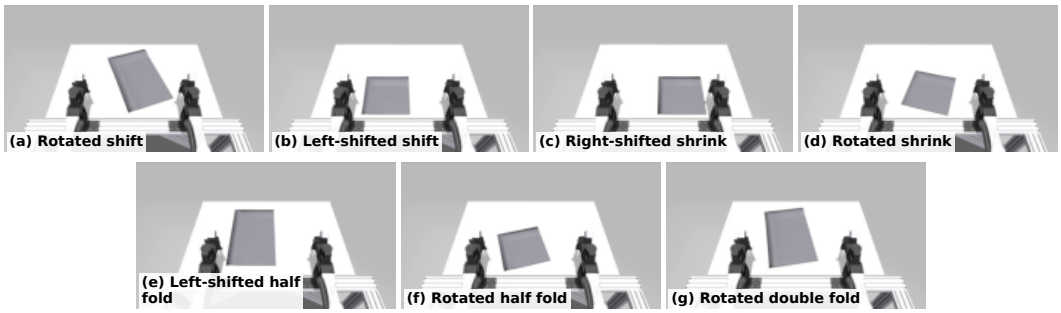


Figure 9: Initial states of the seven towel folding tasks. (mjlab)

The robot must bring a deformable towel from varied initial poses into a target state, such as shifted to a new location, compacted into a smaller footprint, or folded once or twice. Deformable-object dynamics make the outcome highly sensitive to grasp location, drag trajectory, and accumulated error across folds, creating a large gap between the sandbox and deployment environment.

- **Rotated shift.** Move the center of a flat 56×36 cm towel, initially rotated by 20°, by at least 5 cm.
- **Left-shifted shift.** A 34 cm square towel starts 10 cm left of its nominal pose and its center must move by at least 5 cm.
- **Right-shifted shrink.** The 34 cm towel starts 10 cm right of nominal; reduce its covered area to at most 80% of the flat footprint.
- **Rotated shrink.** Perform the same shrink on the 34 cm towel, rotated by 15° and placed 3 cm farther from the robot.
- **Left-shifted half fold.** Fold a towel shifted 12 cm left so its footprint is at most 62% of the flat footprint along one axis.
- **Rotated half fold.** Fold the 34 cm towel, rotated by 15° and placed 8 cm left and 3 cm farther, in half. Success requires a footprint of at most 62% of flat and a half-fold shape—one side at most 62% and the other at least 78% of its flat length—with the towel lying flat; piled configurations fail.
- **Rotated double fold.** Starting 8 cm left and rotated by 10°, fold twice until the footprint is at most 38% of flat.

B.3 BARCODE SCANNING

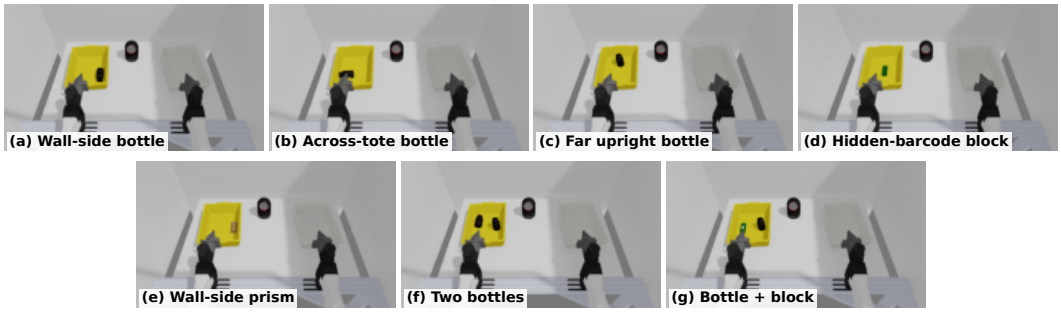


Figure 10: Initial states of the seven barcode scanning tasks. (Isaac Sim)

The robot moves labeled objects from one tote to another after presenting each label to a fixed barcode reader. This is a long-horizon task with a strict logical sequence: the robot must discover the objects, acquire a feasible grasp, expose the correct label face, align it with the reader, and only then stow the object. The reader has a narrow operating window, tote walls constrain the approach to objects near an edge, and multi-object variants require reasoning about clutter and manipulation order.

- **Wall-side bottle.** A bottle lies beside the near wall with its barcode facing into the wall. The policy must extract or roll it, present the barcode to the fixed scanner, and stow it.
- **Across-tote bottle.** The bottle spans the tote, with its cap close to the far wall and its barcode facing away. Success requires a clearance-aware grasp, reorientation, scanning, and stowing.
- **Far upright bottle.** An upright bottle starts near the far-reach boundary with its barcode already visible. The challenge is the long reach and subsequent transfer to the scanner and drop-off tote.
- **Hidden-barcode block.** A green block lies in open space with its printed face against the floor. It must be flipped or regrasped before it can be scanned and stowed.
- **Wall-side prism.** A flattened prism lies beside the near wall with its barcode against the floor. The policy must combine a wall-tight grasp with flipping, scanning, and stowing.
- **Two bottles.** One bottle is upright and one lies along a wall. Both barcodes must be read and both bottles must end in the drop-off tote; order is unconstrained.
- **Bottle + block.** The scene contains an upright bottle and a flat, barcode-up block. Both objects must be identified, scanned, and stowed, requiring shape-specific grasps and multi-object sequencing.

C BASELINE DETAILS

Controlled comparison. All methods start from the same low-level robot API (Section 2.1) and initial toolbox (Section 2.2), including the same raw camera observations, calibration, kinematic queries, and generic motion wrappers; none receives a predefined task-level manipulation skill. We hold the coding agent, reasoning effort, task seeds, and evaluation protocol fixed across different methods. Every method with simulated pretraining uses a four-member population with 20 optimization iterations per member, while every method with real-world adaptation receives five deployment trials. All simulation-based methods also start from the same initial sandbox constructed in Section 2.2 for each task family. The comparison therefore isolates how each method utilizes simulation and robot experience rather than differences in initialization or compute.

Setting-matched baselines. Our evaluation setting combines code-policy generation, a persistent toolbox, open-ended simulation pretraining, and few-trial deployment adaptation. Since some prior methods were introduced under slightly different setups, we mark a baseline with an asterisk when we preserve its central mechanism but reimplement it under our common interface and budget. The asterisk denotes a setting-matched implementation, not additional information or a relaxed evaluation.

Methods.

- **Direct CaP** independently generates a task program from the shared initial toolbox and low-level robot API for each trial, following Code as Policies (Liang et al., 2023).
- **ENPIRE*** starts from the same initial toolbox and iteratively edits the policy using only deployment rollouts. It instantiates ENPIRE's reset–execute–verify–refine loop (Xiao et al., 2026); the asterisk denotes our code-policy implementation under the shared five-trial budget.
- **Zero-Shot Sim2Real** uses a coding agent to optimize the toolbox entirely through simulated experiments using our Stage 1 pipeline, then transfers it directly to the target environment without deployment-time adaptation. This baseline follows the simulation-only transfer principle of DrEureka (Ma et al., 2024b), instantiated through the shared code-policy interface.
- **ASPIRE*** builds a textual skill library in simulation and uses it for deployment-time repair, following ASPIRE (Lu et al., 2026). The asterisk marks our interface- and budget-matched implementation.
- **SimEX (no discovery)** replaces Stage 1 with a fixed-target autoresearch run (Karpathy, 2026), then applies the same Stage 2 as SimEX.
- **SimEX (direct repair)** starts from the same Stage 1 toolbox as SimEX, but performs Stage 2 repair directly from each deployment rollout, without simulator replay, hypothesis generation, or candidate screening.

D Additional results

D.1 Per-task sim-to-sim results

Table 4 reports the per-task counts behind Figure 3. Each table cell reports successful evaluation trials over 25 scheduled trials; bold marks the highest success proportion in each row.

Table 4: Per-task controlled sim-to-sim results, reported as successful evaluation trials out of 25.

Scene	Task	Direct CaP	ENPIRE*	Zero-Shot Sim2Real	ASPIRE*	SimEX (no discovery)	SimEX (direct repair)	**SimEX (ours)**
Plate to tote	Same-side pair	1/25	4/25	1/25	0/25	16/25	**21/25**	**21/25**
	Diagonal pair	3/25	8/25	0/25	0/25	12/25	14/25	**24/25**
	Far outboard plate	17/25	14/25	12/25	17/25	15/25	19/25	**22/25**
	Far plate to near tote	9/25	18/25	10/25	19/25	20/25	**25/25**	**25/25**
	Any three	0/25	6/25	0/25	0/25	**23/25**	16/25	**23/25**
	Shifted single	11/25	5/25	**25/25**	16/25	6/25	23/25	**25/25**
	Shifted diagonal pair	3/25	7/25	0/25	0/25	13/25	20/25	**25/25**
Towel folding	Rotated shift	0/25	8/25	5/25	0/25	16/25	5/25	**21/25**
	Left-shifted shift	4/25	4/25	5/25	11/25	14/25	10/25	**19/25**
	Right-shifted shrink	5/25	14/25	17/25	20/25	19/25	15/25	**24/25**
	Rotated shrink	10/25	16/25	12/25	18/25	20/25	19/25	**21/25**
	Left-shifted half fold	7/25	7/25	8/25	**11/25**	9/25	10/25	**11/25**
	Rotated half fold	1/25	3/25	5/25	3/25	2/25	5/25	**8/25**
	Rotated double fold	3/25	8/25	11/25	4/25	2/25	4/25	**14/25**
Barcode scanning	Wall-side bottle	0/25	0/25	0/25	0/25	2/25	17/25	**25/25**
	Across-tote bottle	0/25	0/25	0/25	1/25	7/25	17/25	**24/25**
	Far upright bottle	0/25	0/25	0/25	15/25	17/25	16/25	**24/25**
	Hidden-barcode block	1/25	0/25	0/25	10/25	8/25	**25/25**	24/25
	Wall-side prism	0/25	0/25	0/25	6/25	6/25	3/25	**19/25**
	Two bottles	0/25	0/25	2/25	0/25	2/25	**20/25**	19/25
	Bottle + block	0/25	0/25	0/25	0/25	0/25	4/25	**14/25**

D.2 Per-task ablation results

For Stage 1, *No population* uses one optimization process instead of four, while *No capability bank* removes capability-bank replay. For Stage 2, *No repair alternatives* limits each iteration to one proposed repair, removing comparison among alternatives, while *No hypothesis generation* formulates repairs directly from the robot evidence without first stating hypotheses. Each setting uses five runs per task and five evaluation trials per run. Table 5 reports the per-task counts behind Figure 5.

Table 5: Per-task barcode scanning ablations, reported as successful evaluation trials out of 25.

Task	SimEX (ours)	No population (Stage 1)	No capability bank (Stage 1)	No repair alternatives (Stage 2)	No hypothesis generation (Stage 2)
Wall-side bottle	25/25	17/25	15/25	24/25	23/25
Across-tote bottle	24/25	20/25	14/25	25/25	20/25
Far upright bottle	24/25	25/25	25/25	25/25	20/25
Hidden-barcode block	24/25	25/25	24/25	25/25	25/25
Wall-side prism	19/25	15/25	21/25	5/25	13/25
Two bottles	19/25	12/25	15/25	18/25	16/25
Bottle + block	14/25	1/25	7/25	14/25	9/25

Table 6: Per-task barcode scanning results for the all-Fable 5.1 configuration used in the main experiments, reported as successful evaluation trials out of 25.

Task	Direct CaP	ENPIRE*	Zero-Shot Sim2Real	ASPIRE*	SimEX (no discovery)	SimEX (direct repair)	**SimEX (ours)**
Wall-side bottle	0/25	0/25	0/25	0/25	2/25	17/25	**25/25**
Across-tote bottle	0/25	0/25	0/25	1/25	7/25	17/25	**24/25**
Far upright bottle	0/25	0/25	0/25	15/25	17/25	16/25	**24/25**
Hidden-barcode block	1/25	0/25	0/25	10/25	8/25	**25/25**	24/25
Wall-side prism	0/25	0/25	0/25	6/25	6/25	3/25	**19/25**
Two bottles	0/25	0/25	2/25	0/25	2/25	**20/25**	19/25
Bottle + block	0/25	0/25	0/25	0/25	0/25	4/25	**14/25**

Table 7: Per-task barcode scanning results for the end-to-end Opus 5 configuration, reported as successful evaluation trials out of 25.

Task	Direct CaP	ENPIRE*	Zero-Shot Sim2Real	ASPIRE*	SimEX (no discovery)	SimEX (direct repair)	**SimEX (ours)**
Wall-side bottle	0/25	4/25	0/25	0/25	0/25	16/25	**25/25**
Across-tote bottle	0/25	1/25	0/25	0/25	3/25	20/25	**25/25**
Far upright bottle	0/25	8/25	0/25	0/25	7/25	20/25	**22/25**
Hidden-barcode block	0/25	2/25	0/25	0/25	17/25	**24/25**	23/25
Wall-side prism	0/25	0/25	0/25	0/25	4/25	2/25	**10/25**
Two bottles	0/25	0/25	3/25	0/25	1/25	21/25	**23/25**
Bottle + block	0/25	0/25	0/25	0/25	0/25	5/25	**6/25**

D.3 Per-task results across coding agents

Tables 6–9 report the complete per-task results underlying Figures 6–11. Each configuration uses the same seven tasks, five independent runs, and five evaluation trials per run. Bold marks the highest success proportion in each row.

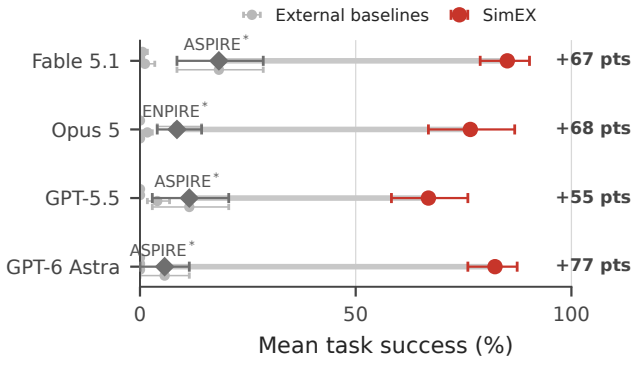


Figure 11: Across all coding agents, SimEX consistently outperforms the strongest baseline methods (marked with darker diamond) by a large margin.

D.4 Comparison with online agent control

An alternative to synthesizing a robot program is to use the coding agent itself as an online policy (Zhang et al., 2026b): after each observation, the agent emits the next low-level action. This approach avoids committing to a program before execution, but places serial model inference directly on the robot's control path. We compare these two deployment modes on three representative

Table 8: Per-task barcode scanning results for the end-to-end GPT-5.5 configuration, reported as successful evaluation trials out of 25.

Task	Direct CaP	ENPIRE*	Zero-Shot Sim2Real	ASPIRE*	SimEX (no discovery)	SimEX (direct repair)	**SimEX (ours)**
Wall-side bottle	0/25	0/25	0/25	0/25	2/25	19/25	**21/25**
Across-tote bottle	0/25	0/25	0/25	9/25	1/25	13/25	**21/25**
Far upright bottle	0/25	0/25	0/25	1/25	0/25	10/25	**17/25**
Hidden-barcode block	0/25	0/25	0/25	5/25	15/25	**25/25**	24/25
Wall-side prism	0/25	0/25	0/25	**5/25**	4/25	0/25	**5/25**
Two bottles	0/25	0/25	7/25	0/25	0/25	**25/25**	24/25
Bottle + block	0/25	0/25	0/25	0/25	0/25	**5/25**	**5/25**

Table 9: Per-task barcode scanning results for the end-to-end GPT-6 Astra configuration, reported as successful evaluation trials out of 25.

Task	Direct CaP	ENPIRE*	Zero-Shot Sim2Real	ASPIRE*	SimEX (no discovery)	SimEX (direct repair)	**SimEX (ours)**
Wall-side bottle	0/25	0/25	0/25	0/25	4/25	17/25	**25/25**
Across-tote bottle	0/25	0/25	0/25	0/25	0/25	17/25	**24/25**
Far upright bottle	0/25	0/25	0/25	0/25	6/25	20/25	**25/25**
Hidden-barcode block	0/25	0/25	0/25	10/25	9/25	**25/25**	**25/25**
Wall-side prism	0/25	0/25	0/25	0/25	0/25	0/25	**8/25**
Two bottles	0/25	0/25	0/25	0/25	0/25	17/25	**20/25**
Bottle + block	0/25	0/25	0/25	0/25	0/25	0/25	**17/25**

barcode scanning tasks. For SimEX, the coding agent writes one task program before execution. For the online policy, the same coding-agent model observes camera-derived object locations and proprioception, then repeatedly emits end-effector actions. We evaluate the online policy over five initializations per task.

SimEX makes a single model call per task, before the task begins. This call takes about 30 seconds, and all three generated programs complete their validation rollout. Execution itself contains no model calls. Moreover, when the task interface is unchanged, the same program can be reused across initial states, allowing the synthesis cost to be amortized.

In contrast, the online policy makes about 30 model calls per rollout and waits 40 minutes for their responses, over 50 times longer than it spends executing actions. None of its 15 rollouts completes the task within the allotted wall-clock budget. The bottleneck is therefore not robot motion, but repeated serial inference while the robot waits for its next command. Although future models may improve both latency and control quality, moving coding-agent inference out of the execution loop makes the SimEX system substantially more practical for robotics.

E Extended related work

In this section, we review four areas most closely related to our method: LLM-based robot control, agentic optimization, generative simulation and environment design, and sim-to-real transfer.

LLM-based Robot Control. Prior work uses LLMs to map language instructions to robot behavior by generating executable plans or code, incorporating language feedback, defining optimization objectives, or orchestrating learned policies (Liang et al., 2023; Singh et al., 2023; Huang et al., 2023b; Yu et al., 2023; Hu et al., 2026; Chen et al., 2026; Zhang et al., 2026c; Fu et al., 2026; Jia et al., 2026; Vemprala et al., 2024; Liu et al., 2024; Zhou et al., 2025; Mu et al., 2024; Xu et al., 2023). These methods demonstrate that an LLM can compose robot behavior from a provided interface, but generally assume that the underlying capabilities, feedback channels, or optimization interface have already been specified. SimEX instead studies how a coding agent can construct and calibrate these underlying robot capabilities through experimentation.

Agentic Optimization. Autoresearch casts improvement as an iterative loop in which a coding agent proposes, implements, and evaluates changes to an executable artifact (Karpathy, 2026). In digital domains, this paradigm has been used to discover algorithms and automate computational research (Romera-Paredes et al., 2024; Novikov et al., 2025; Lu et al., 2024). It has also been extended to embodied settings through persistent skill libraries, automatically optimized reward

programs, and robot policies revised from simulated or physical feedback (Wang et al., 2023a; Ma et al., 2024a; Lu et al., 2026; Xiao et al., 2026; Zhang et al., 2026a; 2023). SimEX develops a robot capability library through open-ended simulated autoresearch and then calibrates it with feedback from deployment. Its simulated stage assumes neither a fixed task distribution, a prescribed set of environment variations, nor a ground-truth evaluation metric.

Generative Simulation and Environment Design. Unsupervised environment design automatically constructs curricula or selects environments that adapt to a learner (Dennis et al., 2020; Jiang et al., 2021; Parker-Holder et al., 2022), while recent generative simulation frameworks use foundation models to produce simulator scenes, task specifications, rewards, and training supervision (Wang et al., 2024; 2023b; Ha et al., 2023; Dai et al., 2024; Torne et al., 2024). Both directions reduce the manual effort required to create diverse training environments and curricula. They mainly use generated content to train policies. SimEX instead generates task variations and success predicates in a simulation of the target workspace to stress-test, diagnose, and expand a persistent symbolic code toolbox.

Sim-to-Real Transfer. Because a simulator cannot exactly reproduce real-world perception, geometry, and contact dynamics, policies developed in simulation may fail after physical deployment. Sim-to-real transfer studies how to bridge this gap. Prior methods improve transfer by exposing policies to varied simulation parameters, or automating reward and randomization design (Tobin et al., 2017; Ma et al., 2024b). A particularly relevant design pattern is simulation pretraining followed by limited real-world adaptation (Hu et al., 2025; Kumar et al., 2021; Chebotar et al., 2019). SimEX is inspired by this two-stage structure, but differs in both the learned object and the role of simulation: it constructs a reusable code toolbox through open-ended autoresearch, then returns to simulation between physical trials to diagnose failures and screen code repairs.