ROBOBRIDGE: A SELF-EVOLVING EMBODIED AGENT FRAMEWORK FOR SIM-TO-REAL TRANSFER

Anonymous authors
Paper under double-blind review

ABSTRACT

A key challenge in bringing embodied intelligence into the real world is transferring capabilities from simulation to reality and enabling agents to continually adapt after deployment. End-to-end vision-language-action policies provide strong manipulation capabilities, but their transfer to physical environments typically relies on calibrating simulated visual and dynamical conditions, collecting additional target-domain demonstrations, and optimizing the policy through further training. Tool-using embodied agents offer flexible task orchestration, yet existing systems primarily emphasize task execution and experience reuse within a given environment, with limited support for transferring procedural knowledge and continuously adapting it across simulation and reality. We propose RoboBridge, a framework that treats sim-to-real transfer as the continued adaptation of executable task skills. The agent represents task knowledge as procedures connecting task intent, observations, tool operations, and outcome verification. Interaction feedback is used to generate candidate skill revisions, which are evaluated before being persisted or rejected. A pretrained vision-language-action policy is exposed as a reusable action tool and enhanced with inference-time guidance, enabling fine-grained execution without retraining the underlying policy. RoboBridge grounds transferable skills in task semantics and interaction interfaces shared across simulation and reality. This representation preserves reusable task structure while allowing environment-dependent operations to be selectively revised through real-world execution feedback. We evaluate the framework on LIBERO-PRO and corresponding physical tasks, studying both skill evolution and post-transfer adaptation. Our framework provides a route from one-shot policy deployment to continual procedural learning across environments.

1 INTRODUCTION

Reliable sim-to-real transfer requires robotic systems to preserve task performance across differences in visual observations and physical dynamics (Zhao et al., 2020). End-to-end vision-language-action models provide fine-grained manipulation capabilities by mapping visual observations and language instructions to action sequences (Kim et al., 2024; Black et al., 2024). The policy-centered pathway illustrated in Figure 1 includes adapting simulation conditions using real-world experience (Chebotar et al., 2018). For pretrained VLA models, adaptation to new robotic settings commonly involves target-domain demonstrations and policy fine-tuning (Kim et al., 2025). In this formulation, task knowledge is largely embedded in model parameters and learned action distributions. This makes it difficult to isolate a reusable task procedure and selectively revise it when execution conditions change, tying procedural adaptation closely to policy optimization.

Tool-using embodied agents offer a way to make task procedures explicit (Singh et al., 2022; Rana et al., 2023). An agent harness provides the observation and tool interfaces through which the agent organizes subtasks, invokes operations, and verifies their outcomes (Liu et al., 2026a). Successful procedures can then be retained as skills for subsequent execution (Wang et al., 2023; Lu et al., 2026). Feedback-based memory updates and skill refinement extend this reuse across interactions (Shinn et al., 2023; Wang et al., 2026), and recent robotic systems further incorporate repair validation and validation-gated skill updates (Lu et al., 2026; Ding et al., 2026). Sim-to-real deployment requires these mechanisms to govern which parts of a transferred procedure remain applicable and how the others should be refined using target-environment feedback. Meanwhile, task decompo-

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.

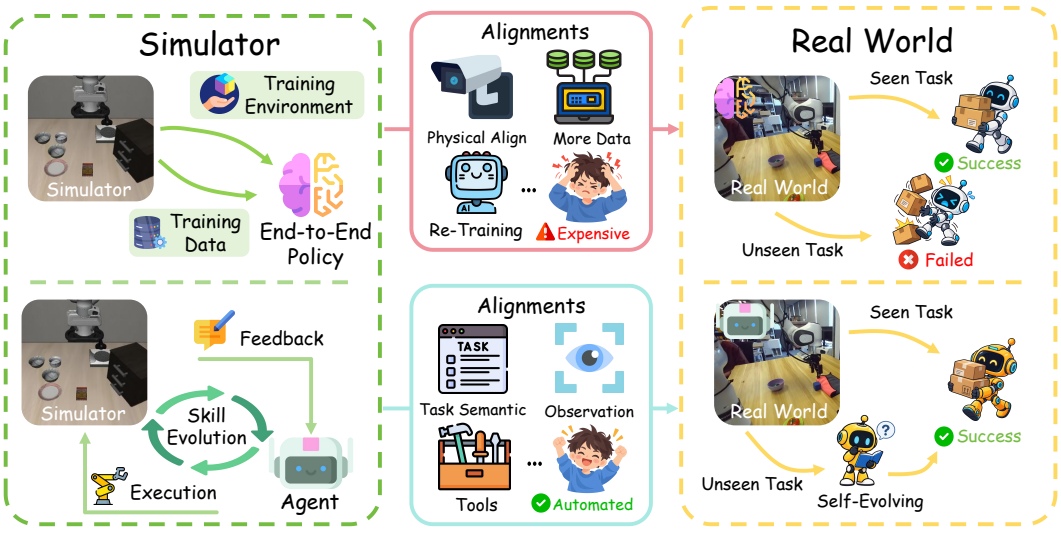

Figure 1: **Sim-to-real transfer with end-to-end models and RoboBridge.** The policy-centered pathway adapts an end-to-end model for real-world execution through physical alignment, additional target-domain data, and retraining. **RoboBridge** evolves procedural skills through a feedback-driven execution loop in simulation and transfers them through aligned task semantics and compatible observation and tool interfaces. After deployment, real-world execution feedback drives further skill evolution, refining the transferred procedures for the target environment.

sition and tool selection still require an underlying capability to execute continuous, fine-grained manipulation (Yu et al., 2023). Recent harnesses address this execution requirement by exposing frozen VLA policies as reusable manipulation primitives (Zhang et al., 2026b). This perspective motivates a framework that combines revisable task knowledge with the manipulation capabilities of pretrained policies.

We propose RoboBridge, a self-evolving embodied agent framework for sim-to-real transfer of procedural skills. RoboBridge incorporates a pretrained VLA policy as a reusable action tool within an agent harness. The policy supplies short manipulation segments, while the agent determines how these operations are organized, when they should be invoked, and whether their outcomes satisfy the task. Task knowledge is stored as procedural skills connecting intent, observations, tool operations, and outcome verification. This organization preserves pretrained execution capabilities while exposing task procedures for revision through interaction. As illustrated in Figure 1, skills evolved in simulation are reused in reality through aligned task semantics and interaction interfaces, with action tools instantiated for each environment. Real-world execution feedback then supports continued skill adaptation. Effective transfer requires both reliable revision of the transferred procedures and accurate execution of their local operations.

Reliable execution presents a complementary challenge: even a suitable task procedure can fail when a frozen policy produces actions that do not meet the current subtask requirements (Liu et al., 2026b). Prior work has explored adapting pretrained policy behavior during inference through external guidance or verification (Wang et al., 2024; Wu et al., 2025; Liu et al., 2026b). RoboBridge therefore augments the policy tool with inference-time guidance. The agent determines whether guidance is needed and configures a reward specification from the subtask objective, execution constraints, and observed feedback. This reward steers action generation toward the intended operation while keeping the policy parameters fixed. Skill evolution thus adjusts the procedure across episodes, while inference-time guidance adapts action generation within an invocation.

Our contributions are summarized as follows:

1. **A self-evolving framework for procedural sim-to-real transfer.** We integrate pretrained action capabilities with explicit task skills, enabling procedures evolved in simulation to be reused through aligned interaction interfaces and further adapted from real-world feedback.

2. **Controlled skill evolution.** We develop a feedback-driven process for adding, revising, and deleting procedural knowledge, with validation governing which updates are persisted and which are rejected or rolled back.
3. **Adaptive policy-tool orchestration.** We enable the agent to determine when to invoke the pretrained policy, when to activate inference-time guidance, and how to configure subtask rewards, supporting context-dependent execution with frozen policy parameters.

2 Related Work

2.1 End-to-End Embodied Models

End-to-end embodied models learn to map visual observations and language instructions directly to robot actions. Early systems such as RT-1 (Brohan et al., 2022) and RT-2 (Brohan et al., 2023) established this paradigm, while later models expanded its scope through embodied multimodal reasoning, multi-robot datasets, and generalist policies, including PaLM-E (Driess et al., 2023), Open X-Embodiment (O'Neill et al., 2024), Octo (Team et al., 2024), OpenVLA (Kim et al., 2024), and π_0 (Black et al., 2024). These models improve generalization by increasing the diversity of training tasks, observations, robot embodiments, and demonstrations, but their behavior remains primarily encoded in a fixed policy. To address the gap between simulation and physical deployment, existing approaches increase the diversity of simulated observations (Tobin et al., 2017), perturb simulated dynamics (Peng et al., 2018), or adapt the simulator distribution using informative variations and posterior estimation (Mehta et al., 2020; Muratore et al., 2022). Recent data-centric methods further scale simulated experiences and adaptation data to improve transfer across tasks and robot embodiments (Liu et al., 2025), while inference-time steering adjusts a frozen policy during execution (Liu et al., 2026b). These approaches use a single end-to-end policy to map observations and instructions to actions, leaving adaptation coupled to the policy itself. Our framework instead uses an agent to organize reusable skills and invoke action tools, so the transferred procedure can be revised in the target environment without retraining the entire policy.

2.2 Embodied Agents

Embodied-agent research introduces explicit reasoning, tool use, and feedback into robotic execution. Early systems grounded language in executable affordances and used environment feedback for closed-loop planning (Ahn et al., 2022; Huang et al., 2022), while program-based approaches synthesized control logic or composed spatial value functions for manipulation (Liang et al., 2023; Huang et al., 2023). Subsequent work explored skill bootstrapping, lifelong skill libraries, and open-vocabulary adaptation through interaction and memory (Zhang et al., 2023; Tziafas & Kasaei, 2024; Zhao et al., 2025). Recent systems have further studied workflow reasoning, self-directed skill acquisition, feedback-based repair, and continual adaptation in simulated or physical environments (Lu et al., 2026; Ding et al., 2026; Wang et al., 2026; Nie et al., 2026; Liu et al., 2026a; Lei et al., 2026; Zhang et al., 2026a). These studies show that explicit procedures, persistent memory, and execution feedback can extend embodied reasoning beyond direct policy execution. Their primary focus, however, varies between workflow planning, skill discovery, memory expansion, and runtime self-improvement. Our work focuses on the migration process itself: a skill is treated as a revisable task hypothesis, and feedback from the target environment determines which parts of the transferred procedure should be retained, modified, or removed.

3 Method

RoboBridge integrates skill-guided execution, inference-time action guidance, and feedback-driven skill evolution into the closed loop illustrated in Figure 2. During task execution, the agent consults the active skill and current observations to select tool operations, then uses task checks and program outputs to guide subsequent decisions. When invoking the pretrained end-to-end action model, the agent can construct a reward function to steer action refinement during denoising while keeping model parameters fixed. Execution logs from the rollout provide evidence for adding, deleting, or replacing skill content. Candidate skills are evaluated through task execution, and validated improvements are retained to guide subsequent rollouts while unsuccessful revisions are rejected.

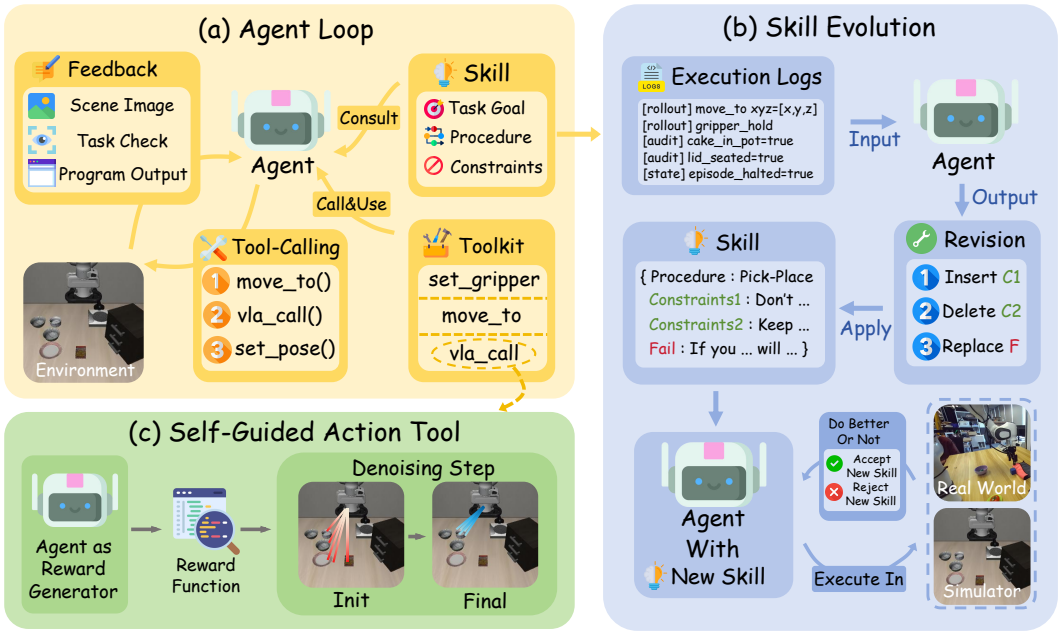

Figure 2: **RoboBridge framework.** **(a) Agent loop:** The agent consults the active skill, invokes manipulation tools, and uses observations and execution feedback to guide subsequent actions. **(b) Skill evolution:** Execution logs inform candidate insertions, deletions, and replacements in the skill. Candidates are evaluated through task execution in the current environment, with validated improvements retained and unsuccessful updates rejected. **(c) Self-guided action tool:** When needed, the agent configures a reward function to guide action refinement during denoising, adapting execution to the current subtask while keeping the pretrained model parameters fixed.

This loop allows execution experience to improve reusable procedures and updated procedures to guide future actions. With aligned task semantics and compatible observation and tool interfaces, skills evolved in the simulator can initialize real-world execution and continue to adapt through physical feedback.

3.1 Problem Formulation and Agentic Execution Loop

Task Setup and Skill Evolution. We consider language-conditioned manipulation in a source simulator, LIBERO-PRO (Zhou et al., 2026), and corresponding real-robot tasks in a target environment. Each task is specified by an instruction ℓ and a completion condition G. The agent maintains an editable skill S that connects observations to tool operations and verification conditions, providing procedural knowledge for task execution.

Skill evolution uses interaction feedback to revise this knowledge. In either environment, a task rollout τ_k informs the validated update process $\mathcal{U}$. Once evolution in simulation is complete, the resulting skill initializes adaptation in the physical environment through an interface mapping Φ:

$$S_{k+1} = \mathcal{U}(S_k, \tau_k), \qquad S_0^r = \Phi(S_K^s), \tag{1}$$

where k indexes evolution steps, s and r denote simulation and reality, and K is the number of source-side evolution steps. The update process retains the current skill when a candidate fails validation. The mapping Φ captures the alignment of task semantics, observation formats, and tool interfaces across environments. The transferred skill continues to evolve through $\mathcal{U}$ using real-world execution feedback. Our objective is to improve task success with limited target-environment interaction while preserving valid procedures and revising those affected by environmental differences.

Agentic Execution Loop. Each rollout consists of successive interactions between a high-level planner Π and a unified tool interface $\mathcal{P}$. At turn t, the planner receives an observation o_t containing the structured scene description, camera observations, robot state, and observation freshness and

uncertainty. Together with instruction ℓ, previous execution feedback f_{t-1}, and the active skill S_k, this information determines the next tool invocation:

$$c_t = \Pi(\ell, o_t, f_{t-1}, S_k), \qquad (o_{t+1}, f_t) = \mathcal{E}(o_t, c_t), \tag{2}$$

where c_t is a structured call selected through $\mathcal{P}$, and $\mathcal{E}$ denotes tool execution and observation refresh. The interface provides deterministic manipulation primitives, including gripper and pose controls, policy-based action tools, and task termination.

Each turn contains at most one physical operation. Once the operation completes or reports a failure, the updated observation and feedback return to the planner, allowing subsequent decisions to reflect the actual execution outcome. The loop ends when G is satisfied, the agent explicitly terminates, or the episode budget is exhausted. The resulting trajectory records observations, tool calls, and feedback as evidence for subsequent skill updates, linking feedback-guided execution to skill evolution across rollouts.

3.2 Embodied Feedback-Driven Skill Evolution

As the experience-bearing unit of embodied-agent self-evolution, each skill is organized as an executable hypothesis about how a task should be performed. It captures the relation between changes in the observed scene and the agent's subsequent behavior: given the current observation and feedback from the previous operation, it specifies how the next operation should be selected and executed. At the same time, a skill is not tied to a particular object identity. It is organized around task-relevant states and manipulation affordances. For example, a grasping skill is not intended for a single object such as a mug, but for different targets that share similar manipulation requirements. This organization separates task intent from its physical realization, allowing the skill to preserve reusable behavioral structure as the environment changes.

Effective skill evolution requires more than adding new experience; it requires continual revision of existing knowledge. To prevent erroneous experience from permanently contaminating the skill memory and to avoid uncontrolled append-only accumulation, a skill must be tested through embodied feedback during task execution. The system collects scene observations, tool outcomes, intermediate verification signals, and the final task result throughout each rollout. Together, these signals provide multiple forms of evidence about the current skill. Successful trajectories can indicate that an operation and its surrounding conditions are reusable, whereas failed trajectories may reveal an unsuitable operation, an incorrect execution order, a missing transition, or a verification condition that no longer holds in the target environment. After a rollout, the agent integrates this evidence to propose a local revision of the current skill. The revision may add a missing operation, replace an unreliable transition, or remove behavior that is no longer supported, while preserving unaffected parts of the procedure.

To distinguish evidence-supported improvements from unverified behavioral hypotheses, each candidate skill is evaluated against the current skill under comparable task conditions. A candidate is consolidated as a new skill version only when it remains valid and improves task execution. If it fails to improve performance or introduces invalid behavior, the current skill remains unchanged and the candidate is excluded from persistent knowledge. Skill evolution is therefore not a simple accumulation of experience, but a feedback-driven process in which behavioral evidence is tested, selectively incorporated, and gradually consolidated into reusable task knowledge.

3.3 From a Frozen Policy to a Self-Guided Action Tool

RoboBridge integrates the fine-grained manipulation capabilities of a pretrained end-to-end action model with the agent's explicit task procedures. The agent uses the active skill and multimodal context to specify a local operation, which the model translates into a short action segment. Operation selection and outcome verification remain explicit in the agent's execution loop. This organization allows procedural knowledge to be revised independently of the model parameters while retaining pretrained manipulation capabilities.

The central execution challenge is to align the model's generated actions with the requirements of the selected operation. A valid task procedure specifies the intended outcome, but a frozen model can still produce unsuitable behavior under different observations and physical conditions. We address this discrepancy through inference-time guidance configured by the agent. When guidance

is needed, the agent uses task context and execution feedback to construct a reward that expresses subtask objectives and execution constraints. Let $\mathbf{A}_t^{(j)}$ denote the candidate action sequence at denoising step j during the tool invocation at turn t. The reward-guided correction is expressed as

$$\mathbf{A}_{t,\text{guided}}^{(j)} = \mathbf{A}_t^{(j)} + \eta_{t,j} \nabla_{\mathbf{A}_t^{(j)}} r_t\big(\mathbf{A}_t^{(j)}; o_t, u_t\big), \tag{3}$$

where o_t is the current observation, u_t specifies the local subtask, and r_t is the agent-configured reward function. The coefficient $\eta_{t,j} \geq 0$ controls the guidance strength, with $\eta_{t,j} = 0$ corresponding to no reward-guided correction. The gradient is taken with respect to the candidate actions, steering their refinement toward higher reward during denoising while keeping the pretrained model parameters fixed. This mechanism connects the agent's task reasoning to action generation, enabling behavioral adaptation within an individual tool invocation.

This mechanism complements skill evolution at the level of individual operations. Skill evolution revises the procedural knowledge governing operation selection, ordering, and verification across episodes, while inference-time guidance adjusts the execution of the selected operation under the current conditions. Execution feedback informs both subsequent planning and further skill revision, linking immediate action adjustment to persistent changes in task knowledge.

4 EXPERIMENTS

We organize our experiments around two deployment stages. In the simulation stage, the agent repeatedly interacts with LIBERO-PRO to evolve task skills and evaluates the resulting procedures. In the transfer stage, the skills obtained in simulation are initialized on corresponding real-robot tasks and further adapted from feedback collected in the target environment.

For each task in both the simulated and real-world settings, we follow the Embodied Feedback-Driven Skill Evolution process described in Section 3.2 to obtain the skill used for subsequent evaluation. The agent gathers evidence from multiple training episodes and uses it to generate candidate skills. After evaluation, an accepted candidate is persisted as a new skill version for subsequent deployment.

Section 4.1 describes the common experimental configuration and skill evolution protocol. Section 4.2 reports the results on the LIBERO-PRO benchmark, and Section 4.3 evaluates skill transfer and continued adaptation in the real-robot environment.

4.1 EXPERIMENTAL SETUP

We evolve skills and initially evaluate the framework in LIBERO-PRO, then transfer the resulting skills to a real-robot environment for further evolution and evaluation. We use Codex with GPT-5.5 as the coding agent. Across all experiments, the agent operates through a unified observation and tool interface. The planner can combine deterministic operation tools with policy-based action tools and receives an updated observation and execution feedback after each operation.

The end-to-end action primitive is instantiated with different checkpoints for simulation and real-robot execution while preserving the same action interface. Specifically, we use the RLinf-released `pi05_libero130_fullshot` $\pi_{0.5}$-SFT checkpoint for LIBERO-PRO and the OpenPI-released $\pi_{0.5}$-DROID checkpoint for the real-robot tasks.

For each task unit in LIBERO-PRO, we perform two rounds of skill evolution, with five training seeds used in each round, to obtain the skill version used for evaluation. The detailed allocation of training, validation, and test seeds is provided in Appendix. In the real-robot environment, we perform one round of skill evolution at the end of each episode.

4.2 SIMULATION EVALUATION

We deploy the skills produced by skill evolution and evaluate our framework on previously unseen seeds. This evaluation tests whether the framework can reuse strategies for organizing actions learned on the training seeds under new task conditions. Specifically, we examine whether the evolved skills help the agent maintain robustness to instruction-redirection (T) and position-swap (S) perturbations in LIBERO-PRO.

Table 1: Success rates (%) on LIBERO-PRO (Zhou et al., 2026) across SPATIAL, OBJECT, GOAL, and LIBERO-10 under instruction-redirection (T) and position-swap (S) perturbations. Our evaluation uses 100 trials per suite–perturbation pair (10 tasks $\times$ 10 seeds). Spat and Long denote SPATIAL and LIBERO-10, respectively. Dashes indicate unreported results. Overall scores for prior methods are reproduced as reported, with differing suite coverage where results are unavailable. Bold indicates the highest reported score in each column.

Method	Instruction redirection (T)				Position swap (S)				Overall
	Spat	Object	Goal	Long	Spat	Object	Goal	Long	
OpenVLA (Kim et al., 2024)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
π_0 (Black et al., 2024)	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.3
$\pi_{0.5}$ (Intelligence et al., 2025)	1.0	1.0	2.0	1.0	20.0	17.0	38.0	8.0	11.0
MolmoAct (Lee et al., 2025)	0.0	0.0	0.0	6.0	0.0	6.0	0.0	0.0	1.5
NORA (Hung et al., 2025)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
X-VLA (Zheng et al., 2025)	0.0	8.0	9.0	10.0	0.0	2.0	1.0	0.0	3.8
AtomVLA (Sun et al., 2026)	1.0	0.0	11.0	9.0	16.0	10.0	2.0	1.0	6.3
π_{RLinf} (Yu et al., 2025)	42.0	71.0	45.0	49.0	59.0	78.0	42.0	14.0	50.0
CaP-X (Fu et al., 2026)	14.0	18.0	17.0	—	12.0	22.0	26.0	—	18.2
RATs (Zhang et al., 2026a)	31.0	63.0	36.0	—	29.0	61.0	43.0	—	43.8
ASPIRE (Lu et al., 2026)	60.0	**95.0**	45.0	—	51.0	**98.0**	**81.0**	—	71.6
Harness VLA (Codex) (Zhang et al., 2026b)	81.0	94.0	75.0	52.0	69.0	91.0	66.0	49.0	72.1
Ours (Codex)	**83.0**	94.0	**80.0**	**55.0**	**73.0**	95.0	65.0	**50.0**	**74.3**

Table 1 summarizes the results on LIBERO-PRO across four task suites, SPATIAL, OBJECT, GOAL, and LIBERO-10, under both T and S perturbations. Existing end-to-end VLA models exhibit marked performance degradation under these distribution shifts. Among prior methods evaluated under identical or similar experimental settings, Harness VLA is the strongest reported baseline, with an overall success rate of 72.1%. Our framework achieves an overall success rate of 74.3% when using Codex, exceeding Harness VLA by 2.2 percentage points in the main comparison and by 5 percentage points on GOAL-T. Under our experimental setup, directly using π_{RLinf} yields an overall success rate of 50.0%, indicating that the improvement does not arise solely from the underlying end-to-end policy. The gains across both generalization settings suggest that our framework evolves reusable experience through interaction feedback and a fixed library of action primitives, supporting rapid transfer to similar scenes or tasks with the same semantics.

We further investigate the contributions of four components: procedural skill guidance, deletion and revision during skill updates, validation before skill persistence, and inference-time guidance for the pretrained end-to-end action model. Figure 3 summarizes the results on SPATIAL-T and OBJECT-T, where the full framework achieves success rates of 83% and 94%, respectively. Without skill guidance, success rates drop to 21% and 16%, corresponding to decreases of 62 and 78 percentage points. Restricting skill evolution to append-only updates reduces success rates to 44% and 57%, which are 39 and 37 percentage points below the full framework. These results highlight the importance of revising and removing existing procedural knowledge during skill evolution. Removing the validation gate lowers success rates to 69% and 63%, corresponding to decreases of 14 and 31 percentage points. This result supports evaluating candidate skill updates through task execution before incorporating them into persistent skill memory. With the full skill evolution process retained, disabling inference-time guidance reduces success rates to 75% and 82%. Activating guidance therefore provides additional gains of 8 and 12 percentage points, respectively. Together, these ablations show that validated skill evolution and inference-time action guidance make complementary contributions to task performance.

4.3 Real-Robot Evaluation

We evaluate whether task skills evolved in simulation can be transferred directly to real-robot settings and further improved through feedback from physical execution. This evaluation examines whether simulation experience can facilitate adaptation to real-robot tasks and reduce the effort

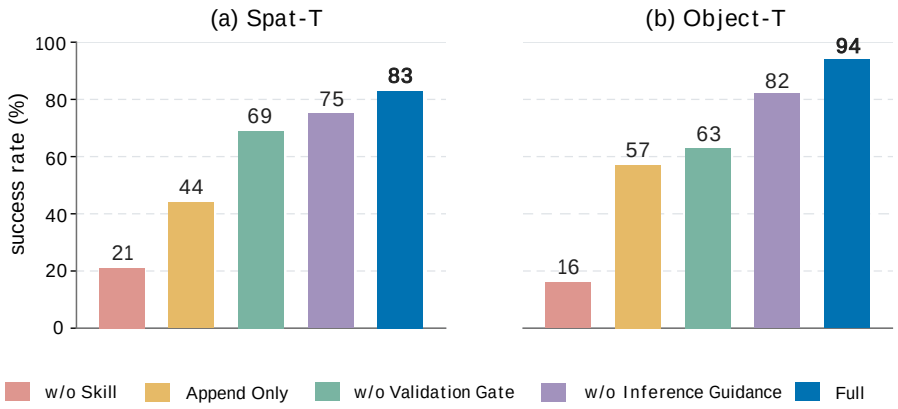

Figure 3: Ablation study on SPATIAL-T and OBJECT-T in LIBERO-PRO. We compare the full framework with four variants: without skill guidance, with append-only updates that exclude revision and deletion, without validation before skill persistence, and without inference-time policy guidance. Bars report success rates (%), with the full framework achieving 83% and 94% on the two suites, respectively.

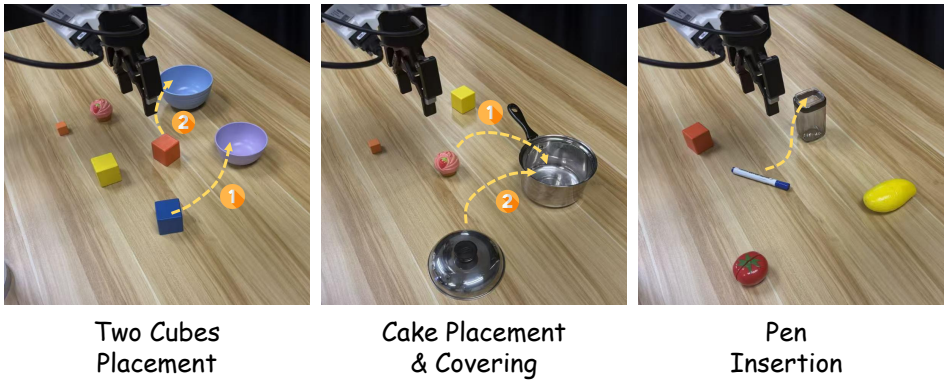

Figure 4: **Real-robot evaluation tasks.** Left: Placing two cubes into their respective target bowls. Middle: Placing a cake into a pot and covering it with the lid. Right: Inserting a pen into a holder.

required for debugging. To support reuse across environments, we encapsulate the real robot's perception, calibration, and control pipelines behind a unified interface, exposing observation specifications and compatible tool interfaces aligned with those used in simulation. We investigate three questions: (1) whether skills evolved in simulation can be applied directly to real-robot tasks with the same or similar task semantics; (2) whether transferred skills can continue to evolve through real-world interaction and improve task performance; and (3) whether simulation skills reduce the exploration and debugging effort required for learning and evolution in the physical environment.

As shown in Figure 4, we design three real-robot tasks whose task semantics and high-level operation procedures correspond to existing simulation tasks, while their specific objects and manipulation modes differ. We first apply the skills evolved in simulation directly to the physical tasks, before performing any real-robot evolution, to measure their transfer value. Figure 5 (a) shows that, compared with execution without skill guidance, the transferred skills improve the success rates of the three tasks from $4/15$, $2/15$ and $3/15$ to $9/15$, $7/15$ and $7/15$. These results indicate that simulation experience provides effective initial behavioral guidance for real-robot execution.

We then initialize real-robot adaptation with the transferred skills. For the performance evaluation in Figure 5 (a), we run 10 real-robot episodes per task and perform one round of skill evolution after every episode. Execution feedback is used to revise the skills for the target environment and its manipulation conditions. The adapted skills achieve $15/15$, $14/15$, and $15/15$ successful trials on

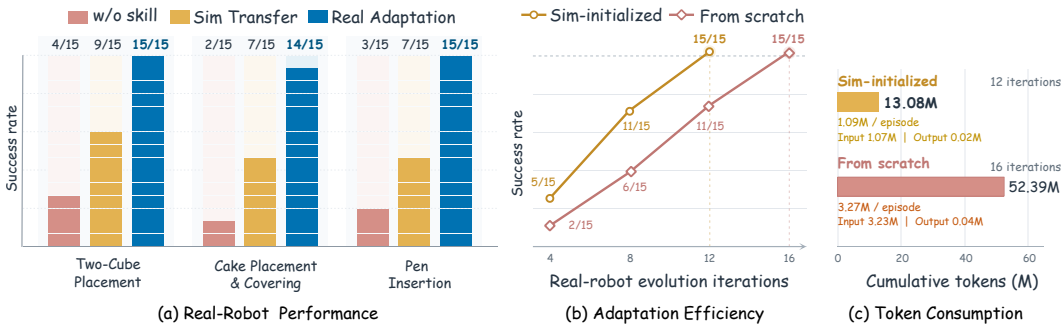

Figure 5: **Real-robot performance and adaptation efficiency. (a) Real-robot performance:** Success counts out of 15 trials on three tasks without skill guidance, with direct transfer of simulation-evolved skills, and after 10 episodes of real-robot skill evolution. **(b) Adaptation efficiency:** Under a separate experimental setting on the cake placement and covering task, simulation-initialized skill evolution reaches $15/15$ after 12 iterations, compared with 16 iterations from an empty skill state. Each iteration consists of one physical execution episode followed by one skill update. **(c) Token consumption:** Cumulative token usage for the same task and setting as (b), measured at the respective 12- and 16-iteration endpoints. Annotations report mean total, input, and output tokens per evolution episode. M denotes million tokens.

the three tasks, respectively. These improvements over direct transfer show that physical interaction can further refine the transferred skills after deployment.

To further quantify the effect of simulation experience on adaptation efficiency, we compare skill evolution initialized from simulation skills with evolution from an empty skill state. Figure 5(b) shows that simulation-initialized evolution reaches $15/15$ after 12 iterations, compared with 16 iterations from scratch, corresponding to a 25% reduction in iterations. At these respective endpoints, cumulative token consumption is 13.08M and 52.39M, respectively, as shown in Figure 5(c). Simulation initialization thus reduces cumulative token consumption by approximately 75%. Together, these results show that transferred skills reduce both the interaction budget and token consumption needed to reach the same performance level on this task.

Together, these results show that, when task semantics and interaction interfaces are aligned, procedural skills formed in simulation provide reusable initial knowledge for real-robot tasks and can continue to adapt from physical execution feedback. The gains from direct transfer demonstrate the reuse value of simulation experience, while the additional improvements after real-robot evolution show that target-environment feedback can refine the transferred procedures. Compared with evolution from scratch, reusing simulation skills also reduces the number of iterations and the token budget required to reach the same performance level, indicating benefits in both task effectiveness and adaptation efficiency. Combining simulation-time experience accumulation with post-deployment skill evolution therefore provides a more efficient starting point for real-robot adaptation by reducing the interaction and reasoning overhead of physical exploration.

5 Conclusion And Limitations

We presented RoboBridge, a self-evolving embodied agent framework for transferring procedural skills from simulation to reality and adapting them through execution feedback. Reusable, revisable skills guide tool selection and outcome verification, while inference-time guidance adapts action generation without updating the underlying policy. Experiments on LIBERO-PRO and three real-robot tasks show that simulation-evolved skills can provide useful initialization for physical execution and continue to improve through real-world feedback, reducing adaptation effort compared with evolution from scratch. Our current evaluation adopts Codex with GPT-5.5 as the coding-agent configuration. A direction for future work is to explore RoboBridge with a broader range of foundation models and agent harnesses, further examining its applicability across different model capabilities and agent execution frameworks.

AI USE STATEMENT

We used OpenAI Codex to assist with translation, manuscript drafting and language editing, and the writing, debugging, modification, and execution of research code. Codex also assisted in organizing experimental results and qualitatively analyzing multimodal execution records to support the interpretation of experimental processes and observed phenomena. Additional uses included literature searches, citation and bibliography preparation, and the creation and editing of scientific figures and plotting code. Generative AI was not used to formulate the core research framework or hypotheses, design the experiments, generate synthetic datasets, clean research datasets, or formulate mathematical claims or proofs.

The authors reviewed the AI-assisted text, translations, and interpretations against the implemented method and experimental evidence. We verified references against the original publications, checked numerical results, token statistics, and figures against the underlying experimental records and logs, and inspected and tested the AI-assisted code. The authors take responsibility for the final content of this work, including its text, claims, references, experimental reporting, and AI-assisted artifacts.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

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.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

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.

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. *arXiv preprint arXiv:1810.05687*, 2018. URL https://arxiv.org/abs/1810.05687.

Xin Ding, Liang Mi, Mingzhe Huang, Zixuan Wang, Chao Zhang, Zixu Hao, Fu Chen, Xiangyu Li, Yikai Zheng, Yaoyu Guo, et al. Zetta ζ: An efficient closed-loop embodied harness for self-evolving physical intelligence. *arXiv preprint arXiv:2608.16590*, 2026.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi "Jim" Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026. URL https://arxiv.org/abs/2603.22435.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

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

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. *arXiv preprint arXiv:2502.19645*, 2025. URL https://arxiv.org/abs/2502.19645.

Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru Wang, Sangho Lee, Winson Han, Wilbert Pumacay, Angelica Wu, Rose Hendrix, Karen Farley, Eli VanderBilt, Ali Farhadi, Dieter Fox, and Ranjay Krishna. MolmoAct: Action reasoning models that can reason in space. *arXiv preprint arXiv:2508.07917*, 2025. URL https://arxiv.org/abs/2508.07917.

Iok Tong Lei, QianZhi Li, Ying Jie Yap, Yujie Zhang, Rui Zhong, Haichao Gui, Xiaolong Liu, and Zhidong Deng. Ace: Agentic control for embodied manipulation via zero-shot workflow reasoning. *arXiv preprint arXiv:2607.04162*, 2026.

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

Guiliang Liu, Yueci Deng, Runyi Zhao, Huayi Zhou, Jian Chen, Jietao Chen, Ruiyan Xu, Yunxin Tai, and Kui Jia. Dexscale: Automating data scaling for sim2real generalizable robot control. In *Forty-second international conference on machine learning*, 2025.

Haowen Liu, Xirui Li, Shaoxiong Yao, Peng Shi, Tianyi Zhou, Jia-Bin Huang, Furong Huang, and Jiayuan Mao. Guava: An effective and universal harness for embodied manipulation. *arXiv preprint arXiv:2606.18363*, 2026a.

Shuo Liu, Ishneet Sukhvinder Singh, Yiqing Xu, Jiafei Duan, and Ranjay Krishna. Vls: Steering pretrained robot policies via vision-language models. *arXiv preprint arXiv:2602.03973*, 2026b.

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.

Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. In *Conference on robot learning*, pp. 1162–1176. PMLR, 2020.

Fabio Muratore, Theo Gruner, Florian Wiese, Boris Belousov, Michael Gienger, and Jan Peters. Neural posterior domain randomization. In *Conference on robot learning*, pp. 1532–1542. PMLR, 2022.

Chang Nie, Zhe Liu, and Hesheng Wang. Teach and grow: An agent-centered architecture for general robot learning. *arXiv preprint arXiv:2608.17209*, 2026.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903. IEEE, 2024.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810. IEEE, 2018.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. 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. 23–72. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/rana23a.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

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. *arXiv preprint arXiv:2209.11302*, 2022. URL https://arxiv.org/abs/2209.11302.

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

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

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, 2017.

Georgios Tziafas and Hamidreza Kasaei. Lifelong robot library learning: Bootstrapping composable and generalizable skills for embodied control with language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 515–522. IEEE, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Peidong Wang, Zhiming Ma, Ying Chang, Xufang Luo, Xiaocui Yang, Shi Feng, Yuqing Yang, and Dongsheng Li. Self-evolving embodied agents via skill-harness evolution. *arXiv preprint arXiv:2608.11350*, 2026.

Yanwei Wang, Lirui Wang, Yilun Du, Balakumar Sundaralingam, Xuning Yang, Yu-Wei Chao, Claudia Perez-D'Arpino, Dieter Fox, and Julie Shah. Inference-Time Policy Steering through Human Interactions. *arXiv preprint arXiv:2411.16627*, 2024. URL https://arxiv.org/abs/2411.16627.

Yilin Wu, Ran Tian, Gokul Swamy, and Andrea Bajcsy. From Foresight to Forethought: VLM-In-the-Loop Policy Steering via Latent Alignment. *arXiv preprint arXiv:2502.01828*, 2025. URL https://arxiv.org/abs/2502.01828.

Chao Yu, Yuanqing Wang, Zhen Guo, Hao Lin, Si Xu, Hongzhi Zang, Quanlu Zhang, Yongji Wu, Chunyang Zhu, Junhao Hu, Zixiao Huang, Mingjie Wei, Yuqing Xie, Ke Yang, Bo Dai, Zhexuan Xu, Jiakun Du, Xiangyuan Wang, Xu Fu, Letong Shi, Zhihao Liu, Kang Chen, Weilin Liu, Gang Liu, Boxun Li, Jianlei Yang, Zhi Yang, Guohao Dai, and Yu Wang. RLinf: Flexible and efficient large-scale reinforcement learning via macro-to-micro flow transformation. *arXiv preprint arXiv:2509.15965*, 2025. URL https://arxiv.org/abs/2509.15965.

Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to Rewards for Robotic Skill Synthesis. *arXiv preprint arXiv:2306.08647*, 2023. URL https://arxiv.org/abs/2306.08647.

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.

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

Han Zhao, Jiaxuan Zhang, Wenxuan Song, Pengxiang Ding, and Donglin Wang. Vlaˆ 2: Empowering vision-language-action models with an agentic framework for unseen concept manipulation. *arXiv preprint arXiv:2510.14902*, 2025.

Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey. *arXiv preprint arXiv:2009.13303*, 2020. URL https://arxiv.org/abs/2009.13303.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Ya-Qin Zhang, Jiangmiao Pang, Jingjing Liu, Tai Wang, and Xianyuan Zhan. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025. URL https://arxiv.org/abs/2510.10274.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2026. URL https://arxiv.org/abs/2510.03827.

A Planner Design with Decoupled Observation and Execution

RoboBridge separates observation construction from action selection through an environment adapter. In centralized-perception mode, the adapter supplies an initial observation after reset and refreshes it after each action-tool invocation. The planner receives the task instruction, procedural skills, current observation, and execution feedback, then selects the next tool and its arguments. Segmentation, geometric reconstruction, and observation packaging are handled by the adapter. The planner is responsible for task decomposition, object selection, motion offsets, and outcome verification. This division allows sensing and control implementations to vary across environments while preserving the observation–action–verification loop used by the skills.

In simulation, observation construction begins with a scene inventory initialized from the camera images and task instruction. The inventory provides entity identities and descriptions for subsequent segmentation. After each action, the adapter associates the resulting masks with current metric geometry and extracts measured reference points. Each point retains its entity identifier, source camera, pixel location, and capture step. Missing or unreliable measurements are represented through availability and uncertainty fields. The resulting snapshot describes the scene and robot state; target–destination assignments, grasp offsets, and end-effector goals are determined by the planner. For example, a measured point on an object's upper surface supplies a geometric reference, while the planner determines the approach and grasp positions using the task, skill, and current observations.

Table 2 summarizes the simulation observation schema. It specifies the coordinate frame and units alongside entity geometry and robot proprioception. Four fixed image slots contain external and wrist RGB views and their annotated counterparts. The serialized snapshot stores image metadata, while the runtime attaches image content to the planner request. Task language and execution feedback are supplied alongside the snapshot. This organization keeps observed scene properties distinct from the task-dependent decisions made from them.

Table 2: Structured observation provided to the planner in simulation. The snapshot combines observation identity, coordinate conventions, external and wrist image metadata, scene entities and their measured geometry, robot proprioception, and validity information. Positions are expressed in the simulation world frame in metres. Optional estimates are included when available. Task instructions and execution feedback accompany the snapshot as separate inputs.

Field	Contents
`observation_id` `episode_step`	Unique observation identifier and the corresponding action-tool step.
`coordinate_frame` `units`	Reference frame and measurement units: simulation world frame and metres.
`images`	External and wrist RGB views and their annotated counterparts, with availability, camera, capture step, resolution, and content references.
`entities`	Entity identifiers, visual labels, visibility, available geometric estimates, and measured reference points with provenance.
`relations`	Supported relations between entity identifiers.
`robot`	End-effector position and orientation, gripper state, and gripper opening.
`freshness` `uncertainties`	Observation validity and explicit caveats about missing or unreliable measurements.

Action execution is synchronized with observation refresh. The planner loop permits one tool invocation per response, and the dispatcher serializes each physical action with publication of its resulting observation. An action attempt invalidates the previous frame. After the primitive returns, the adapter records its result and refreshes perception before the next action decision. In simulation, this refresh is attempted even when the primitive raises an error; unavailable observations are reported

explicitly. A tool invocation can contain multiple low-level control steps, so this synchronization occurs at tool boundaries. The API planner retains text and tool history while keeping only the latest observation images in subsequent requests.

B Action Tools and Agent-Guided Execution

The action library provides a common interface between the planner's subtask decisions and robot execution. It covers Cartesian motion, orientation adjustment, gripper control, and policy-based manipulation, as summarized in Table 3. The planner uses the current observation and procedural skill to select an operation and express its intent through a position, a pose, a gripper command, or a language instruction. The environment adapter translates the request into executable robot actions. This organization lets skills describe how operations should be composed while delegating their low-level execution to the corresponding tools.

Geometric tools support operations with explicit spatial objectives, including approaching an object, aligning the end effector, transporting a held object, and positioning it for release. Orientation and gripper tools provide complementary control over alignment and object handling. Policy-based tools supply learned manipulation behavior for grasping and other contact interactions. A task procedure can combine geometric positioning with policy-based manipulation, with the planner selecting the appropriate tool according to the current subtask and observed state.

For policy-based manipulation, the agent specifies the subtask instruction and determines whether additional inference-time guidance is needed. When guidance is activated, the agent configures a reward specification from the subtask objective, execution constraints, and observed feedback. This specification steers action generation toward the intended operation while keeping the policy parameters fixed. The policy produces short action sequences from current sensor observations, and the agent evaluates the resulting progress at tool boundaries to inform subsequent invocations. The policy thus supplies continuous manipulation capability, while the agent controls its invocation and the guidance appropriate to the current operation.

Table 3: Functional organization of the action library. Tool names describe their roles in task execution. Geometric tools provide explicit motion and gripper control, while policy-based tools support learned manipulation with inference-time guidance configured by the agent when needed.

Function	**Tool names**	**Role**
Cartesian motion	Cartesian Move Pose Alignment	Approach, transfer, and positioning with optional orientation adjustment.
Orientation adjustment	Wrist Rotation Wrist Tilt	Adjust end-effector orientation while maintaining its position.
Gripper control	Open Gripper Gripper Control	Open or actuate the gripper for object handling.
Policy-based manipulation	Policy Grasp Policy Contact	Execute learned grasping and contact behaviors, with agent-configured inference-time guidance when needed.

Tool execution is embedded in the observation–action–verification loop. An invocation returns execution feedback and triggers an observation refresh, allowing the planner to check whether the intended intermediate outcome has been achieved. The planner then continues the procedure, adjusts the next operation, or stops in response to the observed result. Each tool operates within its stopping conditions and execution budget. Tool completion and task success are assessed separately: a returned operation does not by itself establish that the task is complete. Simulation success is determined by the benchmark, while physical evaluation records an operator-provided final label.

Across environments, adapters preserve compatible operation semantics while accommodating differences in sensing, control, and tool availability. The physical adapter checks workspace and mo-

tion constraints before execution. Gripper operations associated with Cartesian motion follow verified arrival at the target, and policy-tool invocation requires successful positioning and a refreshed observation. Execution errors halt the physical episode. These conditions govern how tools can be composed on the robot while allowing the planner to reuse task procedures through the adapted interfaces.

C LIBERO-PRO BENCHMARK AND EVALUATION PROTOCOL

LIBERO-PRO augments LIBERO with controlled variations in task requirements and scene configurations. Our evaluation covers the SPATIAL, OBJECT, GOAL, and LIBERO-10 families, which provide complementary settings involving spatial relationships, object identities, goal conditions, and longer manipulation sequences. Together, these families allow us to examine procedural skills across different manipulation requirements under a common benchmark framework.

Each family is evaluated under two perturbation conditions, T and S. In the task or instruction-redirection condition T, the instruction specifies another valid target object or goal condition. The agent must therefore interpret the revised objective and bind its procedure to the appropriate objects. In the swap or position-swap condition S, object starting locations are exchanged or rearranged while the instruction is kept unchanged. This condition requires the agent to ground its actions in the current layout. The two conditions examine adaptation to changes in task meaning and spatial configuration, respectively.

For every task, seeds 11–20 are assigned to training for skill evolution, seeds 21–25 to validation, and seeds 1–10 to testing. Training episodes supply the execution feedback used to construct candidate skill revisions. The validation gate compares a candidate with the current skill under matched initial conditions and determines whether the revision is retained. The selected skill is subsequently fixed throughout testing. The three seed sets are disjoint, so test initial states are excluded from both skill-update evidence and validation decisions. Reported success rates are calculated from test episodes alone.

A test rollout starts from the initial state specified by its task and seed. RoboBridge receives the task instruction and initial observation, then uses the selected skill to organize tool execution and verify intermediate outcomes. Observations and execution feedback are refreshed as the rollout proceeds, while the skill itself remains unchanged. The rollout continues until the task succeeds, execution is stopped, or the allotted budget is exhausted. This distinction allows the planner to respond to the current scene during testing without performing further skill evolution.

The four families and two perturbation conditions form $4 \times 2 = 8$ evaluation groups. Each group contains 10 tasks, with one test rollout for each of the 10 test seeds per task. Its reported test count is therefore

$$N_{\text{group}} = N_{\text{tasks}} \times N_{\text{test seeds}} = 10 \times 10 = 100. \tag{4}$$

Across all groups, each evaluated configuration covers 80 task–perturbation combinations and

$$N_{\text{test}} = 4 \times 2 \times N_{\text{group}} = 800 \tag{5}$$

test rollouts. Training episodes and validation rollouts are accounted for separately and are excluded from these test counts.

LIBERO-PRO's task-completion predicate supplies the success label for each test rollout. Let $y_{g,i,s}$ denote the binary outcome for task i and test seed s in evaluation group g. It equals 1 if the official predicate is satisfied within the execution budget and 0 otherwise. With $\mathcal{S}_{\text{test}} = \{1, \ldots, 10\}$, the success rate of group g is

$$\text{SR}_g = \frac{1}{N_{\text{group}}} \sum_{i=1}^{10} \sum_{s \in \mathcal{S}_{\text{test}}} y_{g,i,s}. \tag{6}$$

The same task-level criterion applies regardless of the sequence of tools used to execute the task. A primitive returning control to the planner, or the agent declaring completion, does not by itself establish success.

D VLA MODEL INSTANTIATIONS

RoboBridge uses a frozen $\pi_{0.5}$ policy as its contact-rich action primitive in both simulation and physical deployment. The two deployments use different checkpoints because the simulated and real environments provide different robot embodiments, sensor layouts, state representations, and control interfaces. In simulation, we use the RLinf LIBERO checkpoint, denoted π_{RLinf}, while the physical experiments use the OpenPI DROID checkpoint, denoted π_{DROID} (Intelligence et al., 2025; Yu et al., 2025). During each policy invocation, the agent may construct an inference-time reward specification from the current subtask, execution constraints, and observed feedback. This reward guidance steers the generated action segment without updating the VLA parameters.

D.1 π_{RLinf}: RLINF LIBERO CHECKPOINT

Architecture. π_{RLinf} is the LIBERO-specific $\pi_{0.5}$ checkpoint released with RLinf, corresponding to the full-shot `pi05-libero130` model. The policy combines visual observations, language instructions, and robot state before producing continuous robot actions. Within RoboBridge, the planner specifies the current operation through a task or subtask instruction, while the agent can supplement this context with an inference-time reward specification. The policy then converts the multimodal context and the guidance signal into a short manipulation segment.

Input, Guidance, and Action Interface. For LIBERO execution, the policy receives an external RGB view, a wrist-camera RGB view, proprioceptive state, and a language prompt. The environment adapter maps these fields to the input structure expected by the $\pi_{0.5}$ implementation and applies the normalization statistics associated with the checkpoint. Before a guided invocation, the agent interprets the active skill and execution feedback to define the desired local outcome, relevant constraints, and undesirable behaviors. This information is expressed as an inference-time reward specification and used to steer action generation during the current policy call. The resulting action chunk is then executed before the next observation is returned to the planner.

Action Modeling. The inspected LIBERO service produces a short chunk of continuous actions rather than a single control command. Each chunk contains seven-dimensional environment actions and is executed before the planner receives the refreshed observation and execution status. The reward guidance operates during this action-generation process: it evaluates whether candidate behavior is consistent with the local subtask and biases the generated segment toward the desired outcome. The guidance is therefore local to the current invocation and does not become a permanent modification of the policy.

Training and Deployment. The checkpoint is adopted without additional policy training or target-domain fine-tuning. Its parameters are loaded once, kept in evaluation mode, and used without gradient computation. The reward specification supplied by the agent is an inference-time control signal rather than a supervised training label or a parameter-update objective. Skill evolution consequently operates at the procedural level, modifying operation ordering, target binding, tool selection, and verification conditions, while reward-guided inference adjusts the execution of the selected operation.

Performance and Role in RoboBridge. When used directly as a policy-only baseline on LIBERO-PRO, π_{RLinf} obtains an overall success rate of 50.0% under the reported evaluation protocol. This result provides a reference for separating the contribution of the frozen action policy from the contribution of procedural planning, skill evolution, and inference-time reward guidance. In RoboBridge, π_{RLinf} supplies fine-grained manipulation capability, while the agent determines the operation to execute, constructs guidance when necessary, and verifies whether the resulting state satisfies the intended subtask.

D.2 π_{DROID}: OPENPI DROID CHECKPOINT

Architecture and Checkpoint. π_{DROID} is the OpenPI $\pi_{0.5}$ checkpoint used for physical-robot execution. It preserves the same general vision-language-action formulation as the LIBERO policy, but is supplied through a DROID-oriented deployment stack. The physical agent does not invoke

the checkpoint with a fixed open-loop procedure alone. Instead, it provides the current manipulation instruction together with an inference-time reward specification when additional behavioral steering is required. The checkpoint remains unchanged while the guidance signal adapts action generation to the current physical context.

Observation, Guidance, and Action Processing. The physical policy service consumes an external view, a wrist view, robot state, and a language instruction describing the current manipulation operation. The agent first obtains a fresh observation and positions the end effector for the intended operation. It then determines whether the policy should be guided and, if so, specifies the desired interaction outcome and physical constraints through the reward signal. The DROID policy generates a short action chunk under this instruction and guidance, which is executed through the physical control interface. After execution, refreshed images and robot state are returned for verification before another policy call is permitted.

Frozen Use and Agent Coordination. The DROID checkpoint remains frozen during real-world skill evolution. The agent controls the conditions under which the policy is invoked: it selects the target object, determines the local operation, checks that the preceding positioning action succeeded, and constructs the reward specification when the action requires additional steering. The reward signal is used only during inference and does not create a new training update for the policy. Subsequent skill revisions operate on the surrounding procedure, including pre-positioning, action ordering, recovery behavior, and outcome verification.

Role in Physical Transfer. In the real-robot stage, π_{DROID} provides the low-level manipulation capability required to execute transferred procedures. RoboBridge transfers the task-level organization of the skill from simulation, while the physical adapter and the agent-generated reward guidance determine how each local operation is realized under the target robot's observations and dynamics. Execution feedback is retained for subsequent skill revision, allowing the transferred procedure to be refined without retraining the entire VLA model. The contribution of π_{DROID} is therefore evaluated as part of the complete real-robot transfer and adaptation process rather than as an isolated policy-only score.

E Prompt, Skill, and Structured Artifact Organization

RoboBridge separates agent behavior into three persistent layers: environment-specific system prompts, executable procedural skills, and structured execution artifacts. The system prompts define how the agent interprets observations, selects tools, and verifies outcomes. Skills describe reusable task procedures and their recovery conditions. Structured artifacts record the observations, tool calls, feedback, and evaluation results generated during an episode. The following examples present the organization and semantics of these components while omitting implementation-specific prompt text.

E.1 Simulation Planner Prompt

The simulation system prompt defines the common planner contract used during LIBERO-PRO execution. It instructs the agent to reason from the newest observation, issue at most one physical operation per response, and wait for the resulting observation before selecting the next operation. It also describes the available geometric and policy-based tools, the distinction between primitive completion and task success, and the requirement to use the benchmark completion predicate for final evaluation. The prompt does not encode a complete task procedure; task-specific behavior is supplied through the active skill.

Prompt 1: Simulation Planner System Prompt

Purpose: Organize task execution in the simulator.

Core semantics: Use the newest observation; select one operation; wait for execution feedback; verify the resulting state.

Tool semantics: Combine geometric motion tools with policy-based manipulation tools. Tool completion is not equivalent to task success.

Evaluation rule: Use the benchmark task-completion predicate to determine the final outcome.

E.2 Physical-Robot System Prompt

The physical-robot system prompt extends the simulation contract with requirements imposed by hardware execution. It requires the agent to inspect a fresh external and wrist observation before acting, respect workspace and motion constraints, and stop when execution or sensing becomes unreliable. A policy-based action call is allowed only after the robot has been positioned for the intended operation and a new observation has been obtained. The prompt also makes clear that closing the gripper, completing an action chunk, or returning from a tool does not prove that the intended object interaction succeeded.

Prompt 2: Physical-Robot Execution Prompt

Purpose: Add hardware-aware constraints to the planner loop.

Observation rule: Inspect the newest external and wrist views before making a physical decision.

Policy-call gate: Position the end effector successfully and refresh the observation before invoking the policy action.

Failure rule: Stop the episode when execution, localization, or observation reliability is lost.

E.3 Skill-Evolution System Prompt

The skill-evolution system prompt is used after one or more training rollouts. It asks the agent to compare the current skill with the observed execution evidence, identify the local cause of failure or improvement, and propose a bounded revision. Candidate revisions may modify an operation, transition, recovery condition, or verification rule while preserving unaffected parts of the skill. The prompt also requires the candidate to be evaluated against the incumbent skill before persistence. A candidate that does not improve execution or introduces unsupported behavior is rejected.

Prompt 3: Skill-Evolution System Prompt

Purpose: Convert rollout evidence into a controlled skill revision.

Evidence: Read observations, tool outcomes, intermediate feedback, and the final task result.

Revision scope: Add, replace, or remove a local operation, transition, recovery condition, or verification rule.

Persistence rule: Retain a candidate only after comparison with the current skill under matched evaluation conditions.

E.4 Stored Skill Structure

A stored skill is represented as a compact procedural document rather than as a raw transcript. It contains the task intent, applicable conditions, expected observations, ordered operations, verification criteria, and recovery behavior. The skill is written around task-relevant states and affordances, so that object names or scene-specific coordinates do not become the only source of generalization. The following schematic shows the organization of a saved skill without reproducing a complete task-specific procedure.

Artifact 1: Stored Skill Schema

Skill ID: `object_transfer`

Intent: Transfer a selected object to its designated destination.

Preconditions:

- The target and destination are observable.
- The robot is in a reachable starting configuration.

Procedure:

1. Bind the instruction to the current target and destination.
2. Select a suitable positioning or manipulation operation.
3. Execute the local action and refresh the observation.
4. Verify the intermediate state before continuing.

Verification: Check target identity, relative placement, and task progress.

Recovery: Re-observe, revise the local operation, or terminate when the required evidence is unavailable.

E.5 Skill Versioning and Validation

Each accepted skill is associated with a version identifier and the evidence used to produce it. During evolution, the incumbent skill remains available as a fallback, while a candidate skill is evaluated separately. The candidate is promoted only when it satisfies the validity checks and improves the selected execution criterion. Rejected candidates are retained as audit records rather than merged into the active skill. This versioned organization prevents a single failed rollout from permanently altering the procedural memory.

Artifact 2: Skill-Update Record

Current skill: `skill_v3`
Candidate skill: `skill_v4-candidate`
Evidence summary: Repeated failure at the local transfer step; preceding positioning operation remains valid.
Proposed change: Replace the local transfer transition and add an intermediate verification condition.
Validation result: Candidate compared with the current skill under the designated validation episodes.
Decision: `accepted` or `rejected`

E.6 Structured Observation and Episode Files

The execution record is divided into files with different responsibilities. The skill file stores procedural knowledge, the observation snapshot stores the agent-visible scene state, the audit file stores perception and inventory information, and the rollout record stores tool calls, feedback, and the final evaluation result. This separation keeps internal perception details out of the planner request while preserving them for later diagnosis. A schematic observation record is shown below.

Artifact 3: Observation Snapshot

```
{
      "observation_id": "obs_k",
      "episode_step": "k",
      "coordinate_frame": "robot_base",
      "images": ["external_rgb", "wrist_rgb"],
      "entities": [
```

```
        {"id": "entity_1", "measurement": "available"}
      ],
      "robot": {"state": "current"},
      "freshness": "latest",
      "uncertainties": ["target_depth"]

}
```

Note: The example shows the schema only; implementation-specific coordinates, detector outputs, and internal metadata are omitted.

E.7 Episode Artifact Organization

A completed episode contains the files needed to reconstruct the interaction without treating the internal prompt transcript as the skill itself. A typical record includes the active skill version, step-indexed observations, image references, tool calls, execution feedback, perception audit information, and the final evaluation label. The stored files support both failure analysis and skill evolution: the agent can identify whether a failure originated from object binding, positioning, action generation, tool ordering, or outcome verification.

Artifact 4: Schematic Episode Record

`episode/`	
`skill.md`	active procedural skill
`snapshots/`	step-indexed observations
`images/`	external and wrist views
`transcript.json`	planner requests and tool calls
`audit.json`	perception and inventory details
`rollout.json`	execution feedback and outcome
`operator_label.json`	physical evaluation label

E.8 Simulation and Real-Robot Prompt Alignment

The simulation and physical prompts share the same high-level behavioral contract but differ in their environment-specific constraints. The simulation prompt emphasizes benchmark interaction and predicate-based completion, whereas the physical prompt additionally enforces fresh sensing, pre-positioning before policy invocation, hardware-aware stopping, and operator-based outcome labeling. The skill format and update protocol remain shared across the two domains. This organization allows a skill evolved in simulation to be transferred as a procedural object while adapting its environment-specific execution conditions through the physical prompt and feedback records.