STATESKILL-AGENT: AN EFFECTIVE AND EFFICIENT EMBODIED AGENT VIA STATE-CONDITIONED SKILL

Anonymous authors
Paper under double-blind review

ABSTRACT

In this work, we aim to develop an embodied MLLM agent that understands and solves embodied tasks by learning, abstracting, and reusing transferrable and generalizable behavioral patterns. To this end, we propose StateSkill, a new skill representation for embodied agents, which introduces the concept of "finite-state machines" into skill modeling for effective and efficient skill construction and learning. The core idea of StateSkill is to model embodied interaction experiences as a finite-state machine, where a skill is organized around a set of task-progress states, state transitions, and state-conditioned guidance, enabling embodied agents to dynamically adjust their skills as the task progresses while keeping skill contexts compact and stage-relevant. We further develop StateSkill-Evolve, an experience-driven evolution mechanism that iteratively refines StateSkill through retrospective analysis of new successful and failed state-guided trajectories. Upon this, we introduce StateGRPO, which integrates StateSkill evolution and model optimization via state-conditioned GRPO into a unified iterative learning loop, enabling state-conditioned skills and model capabilities to co-evolve and reinforce each other. Extensive experiments demonstrate the effectiveness of our proposed methods and trained model (i.e., StateSkill-Agent) across various embodied tasks.

1 INTRODUCTION

Embodied models like Vision-Language-Action (VLA) models have recently emerged as a promising paradigm for building general-purpose agents for embodied AI by mapping visual observations and language instructions directly to robot actions (Zitkovich et al., 2023; Kim et al., 2024). Despite rapid progress driven by large-scale collections of robot demonstrations (Open X-Embodiment Collaboration, 2024; Ghosh et al., 2024), current VLA methods remain highly dependent on the coverage and diversity of available trajectories, and their performance can degrade substantially when demonstrations are limited (Lin et al., 2025). A key reason lies in that these methods predominantly learn by directly imitating task-specific demonstrations, with few explicit mechanisms for identifying, abstracting, and reusing behavioral regularities shared across demonstrations of different tasks as in Figure 1 (a). Consequently, the resulting models may overfit to training tasks, leading to limited transferability and generalization, particularly in data-scarce scenarios.

In this work, we aim to develop an embodied MLLM agent that understands and solves embodied tasks by learning, abstracting, and reusing transferrable and generalizable behavioral patterns. Recent advances in NLP, such as Anthropic's Claude (Anthropic, 2025), have shown strong potential for enabling large language models (LLMs) to abstract past experiences into compact behavioral knowledge and utilize such knowledge to solve new tasks, substantially improving experience transfer and generalization (Wang et al., 2024a; Zhao et al., 2024; Xia et al., 2026). A core design underlying these advances lies in the concept of "skill": it summarizes generalizable behavioral patterns from successful or failed trajectories, capturing what actions are effective, under what conditions they should be applied, and what mistakes should be avoided, which together can subsequently guide the LLM in solving relevant but unseen tasks (Xia et al., 2026; Ma et al., 2026).

An intuitive idea is to directly apply these LLM skill methods to summarize and reuse behavioral knowledge for embodied MLLM agents, which, however, does not perform well. As illustrated in Figure 1 (b), we attribute this limitation primarily to two key observed challenges. (1) Skill Representation: Most existing LLM skills are typically represented in textual form while embodied agents

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.

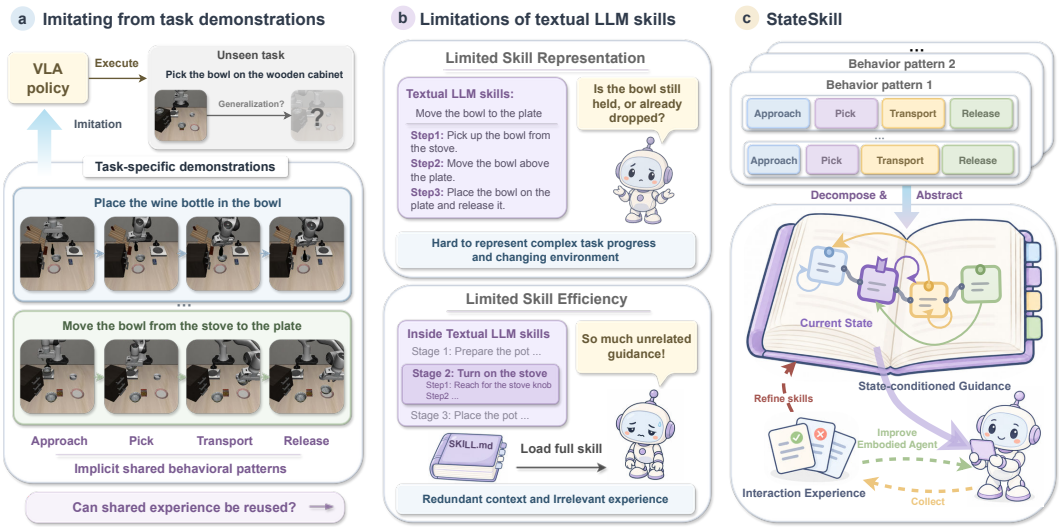


Figure 1: **Motivation. (a)** Imitating from task-specific demonstrations offers limited transferability and generalization. **(b)** Textual LLM skills face two key challenges for embodied agents: ***Limited Skill Representation*** in representing complex behavioral experiences in interactive physical environments; and ***Limited Skill Efficiency*** due to loading entire skill descriptions that introduce redundant and irrelevant context. **(c)** StateSkill organizes shared behavioral patterns into finite-state machines as skills and dynamically provides state-conditioned guidance as tasks progress, while new interaction experiences enable StateSkill and the model to co-evolve and mutually reinforce each other.

are required to perceive, understand and make long-horizon decisions in complex and interactive physical environments, where such text-based skills are often insufficient to represent the complex behavioral experiences and task states arising from agent-environment interactions (e.g., spatial relationships, agent states, and environment states), thereby leading to limited performance. (2) Skill Efficiency: Most existing LLM skill paradigms commonly load an entire skill description at once. For embodied tasks involving multiple execution stages and repeated physical interactions, this can introduce excessively long contexts containing substantial redundant or stage-irrelevant information, which increase computational overhead and may introduce distracting or conflicting guidance, particularly for embodied multimodal agents with limited long-context understanding capabilities.

To tackle these challenges, we propose StateSkill, a new skill representation for embodied agents, which introduces the concept of "finite-state machines" into skill modeling for effective and efficient skill construction and learning as illustrated in Figure 1 (c). The core idea of StateSkill is to model embodied interaction experiences as a finite-state machine, where a skill is organized around a set of task-progress states, state transitions, and state-conditioned guidance. Specifically, StateSkill first decomposes embodied trajectories of similar tasks into multiple progress stages, from which it abstracts a set of progress states and their associated state transitions, and further summarizes state-conditioned skills. Each state is equipped with (a) state-conditioned language guidance describing how to accomplish the current stage or subtasks, (b) a state transition gate specifying when and how the agent should transition to next states, and (c) state-conditioned action priors modeled from embodied trajectories to guide action prediction. By explicitly conditioning skills on progress states, StateSkill enables an embodied agent to track the continuously changing states of the agent, environment, and task, determine state transitions based on the current state and newly observed information, and dynamically adjust its skills in a state-conditioned manner as the task progresses.

In this way, StateSkill enables effective and efficient skill construction for embodied agents. (1) The finite-state representation enables StateSkill to organize embodied interaction experiences into explicit task-progress states and transitions, equipping each state with state-conditioned language guidance, a state transition gate and trajectory-derived action priors. Thus, it enables skills to represent complex behavioral experiences and task states arising from agent-environment interactions in a structured, state-conditioned manner, allowing embodied agents to perceive, understand and

make long-horizon decisions effectively in complex and interactive physical environments. (2) The state-conditioned guidance and state transition gates enable StateSkill to load the language guidance associated with the current progress state and dynamically adjust it as the task progresses, with transitions determined by the current state and newly observed information. This effectively keeps skill contexts compact and stage-relevant, reducing redundant information and the risk of distracting or conflicting guidance and thus lowering computational overhead and easing the long-context processing burden on embodied multimodal agents.

Furthermore, we extend StateSkill with an iterative experience-driven evolution mechanism, termed StateSkill-Evolve, enabling it to continuously evolve by learning from new state-guided experiences beyond its initial construction from pre-generated trajectories. StateSkill-Evolve retrospectively analyzes new successful and failed trajectories, identifies strengths and diagnoses weaknesses in the current StateSkill, and revises its state-conditioned guidance, state transition gates, and action priors.

In addition, we introduce StateGRPO, which integrates StateSkill evolution and model optimization into a unified iterative learning loop. At each iteration, the current StateSkill guides the model to generate embodied trajectories with state-conditioned guidance, transition control, and action priors, while the generated trajectories are then used to optimize the model via state-conditioned GRPO. The updated model then generates new embodied trajectories with enhanced capabilities, which in turn provide fresh experience and feedback for further StateSkill evolution. By alternating between skill evolution and model optimization, StateSkill and the model progressively co-evolve, enabling structured behavioral knowledge and learned model capabilities to mutually reinforce one another. With StateSkill, StateSkill-Evolve and StateGRPO, we train our model, StateSkill-Agent.

The main contributions of this work are fourfold. First, we introduce the concept of finite-state machines into skill modeling and learning for embodied agents and propose StateSkill, which structures reusable behavioral knowledge into state-conditioned skills for effective and efficient embodied execution. To the best of our knowledge, this is the first work that explores finite-state machines for skill modeling and learning in embodied MLLM agent. Second, we develop StateSkill-Evolve, an experience-driven evolution mechanism that iteratively refines StateSkill through retrospective analysis of new state-guided trajectories. Third, we introduce StateGRPO, which unifies StateSkill evolution and model optimization via state-conditioned GRPO in an iterative loop, enabling skills and model capabilities to co-evolve. Fourth, extensive experiments demonstrate the superiority of our proposed methods and trained model (i.e., StateSkill-Agent) on various benchmarks.

2 Related Work

2.1 Embodied Model

Embodied models aim to enable agents to perceive environments and execute physical actions based on observations and instructions (Brohan et al., 2022; Chi et al., 2025; Driess et al., 2023; ACE-Brain Team, 2026). In recent years, Vision-Language-Action (VLA) models have significantly improved the generalization of embodied agents across diverse tasks by jointly modeling vision, language, and actions (Zitkovich et al., 2023; Kim et al., 2024; Black et al., 2025b; Jiang et al., 2023). Beyond standard VLA modeling, recent studies have further explored world models and agentic frameworks for enhancing long-horizon decision making (Liang et al., 2023; Ye et al., 2026). For example, CaP-X (Fu et al., 2026) composes robotic manipulation through code generation, Harness VLA (Zhang et al., 2026c) employs a high-level agent to dynamically orchestrate VLA policies and control primitives, while ASPIRE (Lu et al., 2026) constructs and updates robotic skills through interaction. Unlike these works that mainly focus on model architectures, policy orchestration, or skill invocation, we focus on the representation and learning of embodied skills themselves, and propose StateSkill, a new skill representation that models embodied interaction experiences as a finite-state machine, enabling skills to dynamically adapt to task progress and environment states.

2.2 Skill for Large Language Model

LLM skills abstract historical interaction experiences into reusable behavioral knowledge, allowing agents to transfer previously acquired experience to new tasks (Wu et al., 2026; Wang et al., 2026; 2025b; Jiang et al., 2026). Specifically, Voyager (Wang et al., 2024a) stores successfully executed programs in a retrievable skill library, ExpeL (Zhao et al., 2024) summarizes reusable linguistic

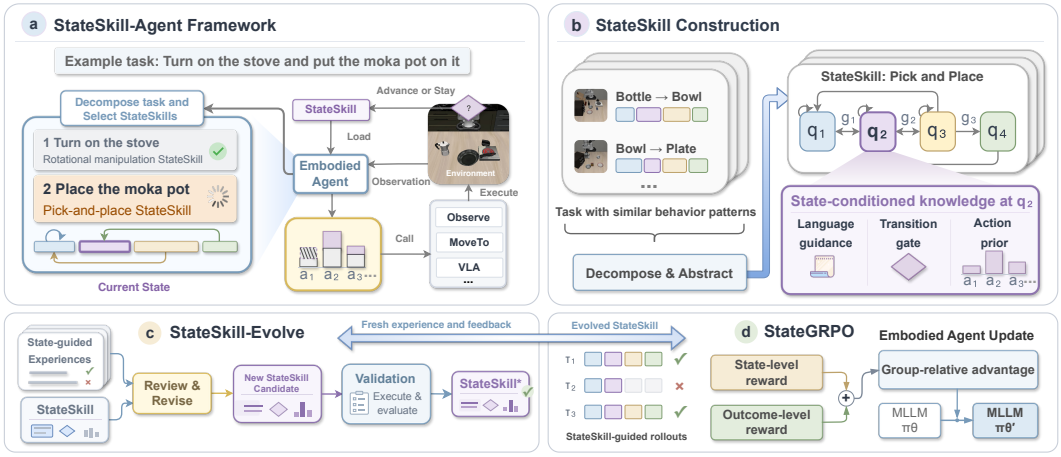


Figure 2: **Overview.** **(a)** StateSkill guides embodied execution through explicit state transitions and progressively loaded state-conditioned guidance, adapting to task progress and environment states. **(b)** StateSkill abstracts shared behavioral patterns from embodied trajectories into a finite-state machine, equipping each task-progress state with language guidance, a transition gate, and trajectory-derived action priors. **(c)** StateSkill-Evolve iteratively refines StateSkill by reviewing new successful and failed state-guided trajectories and validating candidate revisions. **(d)** StateGRPO integrates StateSkill evolution and state-conditioned embodied agent optimization into a unified iterative loop, enabling StateSkill and model capabilities to co-evolve.

experience from historical trajectories, while recent works such as SkillRL (Xia et al., 2026) and SkillGen (Ma et al., 2026) further explore automatically constructing and continuously updating skills from successful and failed trajectories. Existing LLM skills are typically represented as textual or programmatic task-level knowledge, with limited explicit modeling of continuously changing task states during execution. In contrast, StateSkill models a skill as a finite-state machine consisting of task-progress states, state transitions, and state-conditioned guidance, enabling skills to dynamically adapt to changing task and environment states throughout embodied task execution.

2.3 Reinforcement Learning in LLM Agent

Reinforcement learning enables LLM agents to directly optimize their decision-making policies through environmental interaction and reward feedback, and has recently been widely adopted for reasoning, tool use, and multi-turn agent tasks (Guo et al., 2025; Jin et al., 2025; Wang et al., 2025a; Feng et al., 2026). Methods such as GRPO (Shao et al., 2024) optimize model policies based on group-relative rewards. RL has also been applied to interactive agent settings through WebRL (Qi et al., 2025), ToolRL (Qian et al., 2025), and Agent-R1 (Cheng et al., 2025). These methods mainly optimize model parameters using task outcomes or trajectory-level feedback, while the internal state structure of long-horizon tasks remains less explicitly exploited. Different from existing RL methods for LLM agents, we propose StateGRPO, which introduces task states and state-level feedback into policy optimization, while using newly collected interaction trajectories to further evolve StateSkill, enabling the joint optimization of skill knowledge and model capabilities.

3 Methodology

3.1 StateSkill-Agent Framework

As illustrated in Figure 2, StateSkill-Agent uses task instructions and current visual observations to decompose tasks into subtasks and retrieve corresponding reusable StateSkills. It tracks execution progress through the explicit states and transitions of each active skill. Language guidance is introduced progressively as execution advances through these states, without loading the entire skill

description upfront. After each interaction, a skill-specific gate interprets observations, executor feedback, and retained facts to update the active state or return a local skill outcome.

StateSkill-Agent starts with StateSkills constructed by abstracting shared behavioral patterns from embodied trajectories. Building on these initial skills and the interaction trajectories generated during execution, StateSkill-Evolve refines skill structure and guidance using new successful and failed state-guided trajectories, while StateGRPO optimizes StateSkill-Agent's underlying MLLM with state-conditioned feedback. The updated agent then generates further interaction experience for skill refinement, forming an iterative cycle of skill evolution and model optimization.

3.2 StateSkill: Modeling Embodied Skills as Finite-State Machines

StateSkill organizes reusable embodied interaction experience into explicit task-progress states rather than a single task-level textual description. We formulate a StateSkill as

$$\mathcal{S} = (\mathcal{Q}, \mathcal{E}, \{\kappa_q\}_{q\in\mathcal{Q}}), \qquad \kappa_q = (\ell_q, g_q, p_q), \tag{1}$$

where $\mathcal{Q}$ denotes the task-progress states, $\mathcal{E}$ denotes their transitions, and κ_q represents the state-conditioned knowledge associated with state q. ℓ_q provides language guidance for the current task stage, g_q specifies its transition conditions, and p_q denotes a trajectory-derived action prior. Different skills define their own state vocabularies, evidence requirements, and termination conditions.

StateSkill construction. We use GPT-5.5 (OpenAI, 2026) to construct reusable StateSkills from exploration trajectories $\mathcal{D}_{\text{init}}$ through segmentation, cross-task alignment, and knowledge synthesis. Within a given family of local operations, trajectories are partitioned into stages according to local objectives and progress conditions. Compatible stages are aligned into shared states, with task-specific entities parameterized by functional roles while retaining relevant relations. Stage ordering provides candidate transitions; recorded feedback and execution evidence inform their conditions, including progress, retries, and termination. For each state q, we use supporting segments to synthesize language guidance ℓ_q, specify transition conditions g_q, and identify action preferences encoded by p_q, yielding $\kappa_q = (\ell_q, g_q, p_q)$ and the initial StateSkill $\mathcal{S}^{(0)}$. Appendix A provides details of the construction procedure, prompt, output representation, and LIBERO instantiation.

StateSkill inference. Let q_t denote the active state and h_t StateSkill-Agent's context, including the task instruction, interaction history, and progressively introduced guidance. The active guidance ℓ_{q_t} conditions the selection of a high-level action category $c \in \mathcal{A}$. When the MLLM backend exposes action-category scores, the prior p_{q_t} reweights its decision distribution:

$$\widetilde{\pi}_\theta(c \mid h_t, q_t) = \frac{\pi_\theta(c \mid h_t)\, p_{q_t}(c)}{\sum_{c'\in\mathcal{A}} \pi_\theta(c' \mid h_t)\, p_{q_t}(c')}. \tag{2}$$

The agent generates the corresponding action arguments based on the selected action category and its current context. The action vocabulary is fixed within the configured runtime.

After execution, a deterministic skill-specific gate interprets the returned observations and feedback e_{t+1} together with retained execution facts m_t. For a non-terminal interaction, it updates

$$(q_{t+1}, m_{t+1}) = \delta_{\mathcal{S}}\big(q_t, m_t, e_{t+1}; g_{q_t}\big). \tag{3}$$

The gate can retain the state, follow a valid transition, or return a local skill outcome to task-level control. Local skill completion does not imply overall task success. On a state transition, the new guide is appended to the context and supersedes earlier guides as operating guidance; the active action prior changes accordingly. Appendix B describes this shared interface and illustrates it with pick-and-place and contact StateSkills from LIBERO, using Figure 3 as a pick-and-place example.

3.3 StateSkill-Evolve

StateSkill-Evolve uses StateSkill-Agent's experience to refine how skills represent progress and guide behavior. At iteration k, an offline LLM updater analyzes $\mathcal{D}^{(k)} = \mathcal{D}_+^{(k)} \cup \mathcal{D}_-^{(k)}$, the successful and failed trajectories collected under $\mathcal{S}^{(k)}$. For each visited state, it compares the active guidance, transition conditions, and action priors with the recorded actions and execution evidence. Successful

trajectories identify effective behavior and its conditions of applicability; failed trajectories help identify missing guidance, inappropriate action preferences, or state distinctions and recovery paths absent from the current representation. Together, they support revisions grounded in specific states and trajectory segments to preserve effective behavior and address weaknesses.

The updater can revise the state structure $(\mathcal{Q}, \mathcal{E})$ and state-conditioned knowledge $\kappa_q = (\ell_q, g_q, p_q)$ to produce a candidate $\widetilde{\mathcal{S}}^{(k+1)}$. Knowledge updates refine guidance, transition conditions, and action preferences; structural updates adjust the state decomposition and transition graph when the execution evidence warrants them. Each candidate remains a finite-state skill over the available action and feedback interfaces. Revisions are tied to supporting evidence and an expected behavioral effect, with unaffected state knowledge retained to preserve useful experience from earlier iterations.

The candidate and current StateSkill are evaluated on the same validation task instances with the MLLM checkpoint and execution configuration held fixed. After consistency and behavioral checks, the candidate is accepted only if it improves aggregate task success; otherwise, $\mathcal{S}^{(k)}$ is retained. Validation seeds are separate from the unseen seeds reserved for final evaluation. This procedure turns state-level revision hypotheses into execution-tested skill updates. Appendix C specifies diagnosis, candidate generation, paired validation, acceptance criteria, and data use during evolution.

3.4 STATEGRPO

StateSkill-Evolve refines skills, while StateGRPO optimizes StateSkill-Agent's MLLM using task success and state-wise execution efficiency. At iteration k, we collect a group of G trajectories $\{\tau_i\}_{i=1}^{G}$ for a fixed task and environment seed through StateSkill-guided execution with the MLLM policy $\pi_{\theta^{(k)}}$. Training seeds are disjoint from those used for final evaluation.

Let $s_i \in \{0, 1\}$ indicate task success for trajectory τ_i, and let $T_{i,q}$ denote the total number of interaction turns spent in state q, including repeated visits. Let $\mathcal{I}_q^+$ denote the indices of successful trajectories in the group that visit state q. For $|\mathcal{I}_q^+| \geq 2$, we define the state-specific reference:

$$\bar{T}_q = \frac{1}{|\mathcal{I}_q^+|} \sum_{j \in \mathcal{I}_q^+} T_{j,q}. \tag{4}$$

This avoids comparing absolute turn counts across states with different execution requirements.

Let $\mathcal{C}_i$ contain the states visited by τ_i for which $|\mathcal{I}_q^+| \geq 2$. For $\mathcal{C}_i \neq \varnothing$, we define

$$R_{\text{state}}(\tau_i) = \frac{s_i}{|\mathcal{C}_i|} \sum_{q \in \mathcal{C}_i} \frac{\bar{T}_q}{T_{i,q}}, \tag{5}$$

and set $R_{\text{state}}(\tau_i) = 0$ when $\mathcal{C}_i$ is empty. Failed trajectories receive zero state reward, while successful ones are rewarded for using fewer turns relative to the reference in each comparable state. The comparison does not require identical state sequences across trajectories.

We combine task success with state efficiency and compute group-relative advantages:

$$R_i = s_i + R_{\text{state}}(\tau_i), \qquad \widehat{A}_i = \frac{R_i - \mu_R}{\sigma_R + \epsilon}, \tag{6}$$

where μ_R and σ_R are the mean and standard deviation of rewards within the trajectory group, and $\epsilon > 0$ ensures numerical stability. Differences in state efficiency can provide a learning signal even when all trajectories in a group succeed and therefore receive the same task-success reward.

We optimize the MLLM policy using a clipped GRPO objective (Shao et al., 2024) with advantages $\widehat{A}_i$. Truncated importance weighting accounts for differences between the rollout and optimization distributions. The current StateSkills remain fixed throughout each policy update.

We then collect new StateSkill-guided trajectories with the updated policy $\pi_{\theta^{(k+1)}}$, providing fresh successful and failed interaction experience for StateSkill-Evolve. This alternates model optimization with skill refinement, enabling reusable skill knowledge and model capabilities to co-evolve. Appendix D specifies the state accounting, policy objective, and alternating update schedule.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate StateSkill-Agent on LIBERO-PRO (80 tasks) (Liu et al., 2023; Zhou et al., 2025), RoboCasa365 (Target50) (Nasiriany et al., 2026), and five real-world manipulation tasks, using GPT-5.5 (OpenAI, 2026) or Qwen3.5-9B (Qwen Team, 2026) as the high-level MLLM. For LIBERO-PRO, we use 10 environment seeds per task; for RoboCasa365, we use 10 seeds per atomic task and five per composite task. Each real-world task is evaluated over 20 trials. For low-level execution, we use π_{RLinf} for LIBERO-PRO, RLDX-1 for RoboCasa365, and $\pi_{0.5}$ for real-world tasks. VLA executors remain frozen; StateGRPO fine-tuning (FT) updates only Qwen3.5-9B. We report success rates (%), with Overall computed as the unweighted mean of per-task success rates, giving each task equal weight regardless of its number of evaluation episodes. Simulation evaluation seeds are held out from skill construction, skill evolution, candidate validation, and MLLM training (Appendix C.4). Appendix E details benchmark configurations, evaluation protocols, and deployment.

Table 1: **Results on LIBERO-PRO.** Success rates (%) under task (T) and swap (S) perturbations. Gains (pp) are relative to the corresponding baseline; "–" means unavailable.

Method	Spat-T	Spat-S	Obj-T	Obj-S	Goal-T	Goal-S	L10-T	L10-S	Overall
Vision-Language-Action Models									
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., 2025b)	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.3
$\pi_{0.5}$ (Black et al., 2025a)	1.0	20.0	1.0	17.0	2.0	38.0	1.0	8.0	11.0
MolmoAct (Lee et al., 2025)	0.0	0.0	0.0	6.0	0.0	0.0	6.0	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., 2026)	0.0	0.0	8.0	2.0	9.0	1.0	10.0	0.0	3.8
AtomVLA (Sun et al., 2026)	1.0	16.0	0.0	10.0	11.0	2.0	9.0	1.0	6.3
π_{RLinf} (Zang et al., 2026)	42.0	59.0	71.0	78.0	45.0	42.0	49.0	14.0	50.0
Code-as-Policy Agents									
CaP-X (Fu et al., 2026)	14.0	12.0	18.0	22.0	17.0	26.0	–	–	18.2
RATS (Zhang et al., 2026a)	31.0	29.0	63.0	61.0	36.0	43.0	–	–	43.8
RHO (Elmaaroufi et al., 2026)	61.6	53.0	37.2	12.2	55.6	50.6	–	–	45.0
ASPIRE (Lu et al., 2026)	60.0	51.0	95.0	98.0	45.0	81.0	–	–	71.7
GPT-5.5 (Baseline)	40.0	46.0	55.0	29.0	41.0	35.0	11.0	12.0	33.6
Harness VLA Zhang et al. (2026c)	81.0	69.0	94.0	91.0	75.0	66.0	52.0	49.0	72.1
StateSkill-Agent	83.0	72.0	95.0	92.0	77.0	68.0	54.0	51.0	74.0$^{\uparrow 40.4}$
Qwen3.5-9B (Baseline)	24.0	19.0	17.0	22.0	11.0	14.0	7.0	8.0	15.3
Harness VLA Zhang et al. (2026c)	36.0	22.0	27.0	31.0	40.0	26.0	11.0	13.0	25.8
StateSkill-Agent	79.0	52.0	89.0	91.0	69.0	68.0	50.0	49.0	68.4$^{\uparrow 53.1}$
StateSkill-Agent w/ FT	82.0	56.0	90.0	91.0	72.0	70.0	54.0	52.0	70.9$^{\uparrow 55.6}$

Table 2: **Results on RoboCasa365 (Target50).** Success rates (%) on atomic and composite tasks. Gains (pp) are relative to the corresponding baseline.

Method	Atomic-Seen	Composite-Seen	Composite-Unseen	Overall
PRTS (Zhang et al., 2026b)	66.3	30.3	18.8	39.6
RLDX-1 (Kim et al., 2026)	67.6	27.9	8.5	36.0
WorldDreamer (Wang et al., 2024b)	66.3	26.7	9.0	35.3
GR00T N1.5 (Bjorck et al., 2025)	50.7	14.8	2.7	23.9
$\pi_{0.5}$ (Black et al., 2025a)	39.6	7.1	1.2	16.9
π_0 (Black et al., 2025b)	34.6	6.1	1.1	14.8
GPT-5.5 (Baseline)	54.4	33.8	8.8	33.2
Harness VLA (Zhang et al., 2026c)	91.7	56.3	13.8	55.4
StateSkill-Agent	94.4	58.8	16.3	58.0$^{\uparrow 24.8}$
Qwen3.5-9B (Baseline)	22.8	11.3	2.5	12.6
Harness VLA (Zhang et al., 2026c)	52.2	16.3	3.8	25.2
StateSkill-Agent	72.8	37.5	10.0	41.4$^{\uparrow 28.8}$
StateSkill-Agent w/ FT	75.6	40.0	11.3	43.6$^{\uparrow 31.0}$

4.2 Main Results on Simulation Benchmarks

Tables 1 and 2 summarize the simulation results. With Qwen3.5-9B, StateSkill-Agent w/ FT achieves overall success rates of 70.9% on LIBERO-PRO and 43.6% on RoboCasa365, outperforming Harness VLA by 45.1 and 18.4 percentage points, respectively. On LIBERO-PRO, improvements span all four task families under both task and swap perturbations. On RoboCasa365, gains are concentrated in atomic-seen and composite-seen tasks, while composite-unseen success increases from 3.8% to 11.3%, indicating that unseen compositions remain challenging. With GPT-5.5, StateSkill-Agent also improves over Harness VLA by 1.9 and 2.6 percentage points on the two benchmarks, respectively, showing that the benefits extend beyond the Qwen3.5-9B configuration.

4.3 Real-World Experiments

We investigate sim-to-real skill transfer on five related real-world manipulation tasks using simulation-derived StateSkills, preserving their finite-state structure and adapting only environment-dependent components. We evaluate each embodied MLLM agent with and without StateSkill on the same set of tasks, using a frozen $\pi_{0.5}$ policy for low-level execution. As shown in Table 3, StateSkill increases overall success from 60.0% to 76.0% for Qwen3.5-9B and from 75.0% to 86.0% for GPT-5.5, yielding gains of 16.0 and 11.0 percentage points, respectively. The gains span all five tasks for Qwen3.5-9B and four for GPT-5.5, whose success rate on Object Sorting remains unchanged at 100.0%. On Flower Insertion, Qwen3.5-9B improves from 30.0% to 50.0%, while GPT-5.5 improves from 55.0% to 70.0%, yielding gains of 20.0 and 15.0 percentage points, respectively. These results support sim-to-real StateSkill transfer with environment-specific adaptation.

Table 3: **Real-world manipulation results.** Success rates (%) on five real-world tasks. GPT-5.5 and Qwen3.5-9B are used as baselines.

Method	Block Stacking	Flower Insertion	Object Sorting	Pen Placement	Cup Nesting	Overall
GPT-5.5 (Baseline)	80.0	55.0	100.0	55.0	85.0	75.0
GPT-5.5 + StateSkill	90.0	70.0	100.0	75.0	95.0	86.0$^{\uparrow 11.0}$
Qwen3.5-9B (Baseline)	70.0	30.0	70.0	60.0	70.0	60.0
Qwen3.5-9B + StateSkill	80.0	50.0	95.0	75.0	80.0	76.0$^{\uparrow 16.0}$

4.4 Ablation Studies and Discussion

Ablation studies. Table 4 examines the progression from finite-state skill construction to skill evolution and skill–model co-evolution on LIBERO-PRO with Qwen3.5-9B. StateSkill achieves 63.5% success by organizing interaction experience into task-progress states with state-conditioned guidance. Further with StateSkill-Evolve, we build our **StateSkill-Agent**, which increases success to 68.4% (+4.9 percentage points) through retrospective refinement using new successful and failed state-guided trajectories, supporting continued skill improvement beyond initial construction. Further involving StateGRPO, our final model, **StateSkill-Agent w/ FT**, raises success to 70.9% (+2.5 percentage points) by alternating skill evolution with state-conditioned GRPO updates to the embodied MLLM agent. Appendix F separately examines transition verification, reward design with fixed initial skills, and state-localized revision.

Table 4: **Ablation studies on LIBERO-PRO.** Success rates (%) with Qwen3.5-9B. Rows progressively add the indicated components.

StateSkill	StateSkill-Evolve	StateGRPO	Overall
–	–	–	15.3
✓	–	–	63.5
✓	✓	–	68.4
✓	✓	✓	**70.9**

Comparison with LLM skills. Table 5 compares StateSkill-Agent with Harness VLA and representative LLM skill methods using the same Qwen3.5-9B backbone. During task execution, the agent reuses a shared skill library across tasks without retrieving task-matched reference trajectories. Appendix G.1 contrasts how StateSkill-Agent and Harness VLA organize and use exploration experience. StateSkill-Agent w/ FT achieves 70.9% success versus 25.8% for Harness VLA, with reported skill length of 11.0K versus 169.4K and token usage of 0.44M versus 3.62M. Compared with our reproductions of XSkill (Jiang et al., 2026), SkillGen (Ma et al., 2026), and Trace2Skill (Ni et al., 2026), it achieves higher success rates with fewer interaction turns and lower token usage.

Qualitative analysis. Figure 3 contrasts pick-and-place execution with and without StateSkill after a missed grasp. Without StateSkill, the agent assumes grasping succeeded and proceeds with empty transport and release, causing task failure. With StateSkill, it detects the missed grasp, retries grasping, verifies the updated state, and completes the task. This example illustrates how task-state verification supports recovery instead of allowing failed actions to be treated as completed stages. Appendix B.5 provides an additional qualitative comparison between StateSkill and HarnessVLA, including guidance excerpts, executed tool calls, and environment-verified outcomes.

Table 5: **Comparison with LLM skills on LIBERO-PRO.** All methods use Qwen3.5-9B as the baseline. Superscripts in the shaded column indicate improvements over Harness VLA: percentage-point gains in success rate and relative reductions (%) in other metrics.

Metric	Harness VLA	XSkill	SkillGen	Trace2Skill	StateSkill-Agent	StateSkill-Agent w/ FT
Success Rate (%) ↑	25.8	23.6	28.9	31.5	68.4	**70.9**$^{\uparrow 45.1}$
Skill Length (K) ↓	169.4	186.7	104.3	231.5	11.0	**11.0**$^{\downarrow 93.5\%}$
Interaction Turns ↓	74.96	76.80	69.50	66.10	23.10	**21.91**$^{\downarrow 70.8\%}$
Token Usage (M) ↓	3.62	3.74	3.05	3.22	0.47	**0.44**$^{\downarrow 87.8\%}$

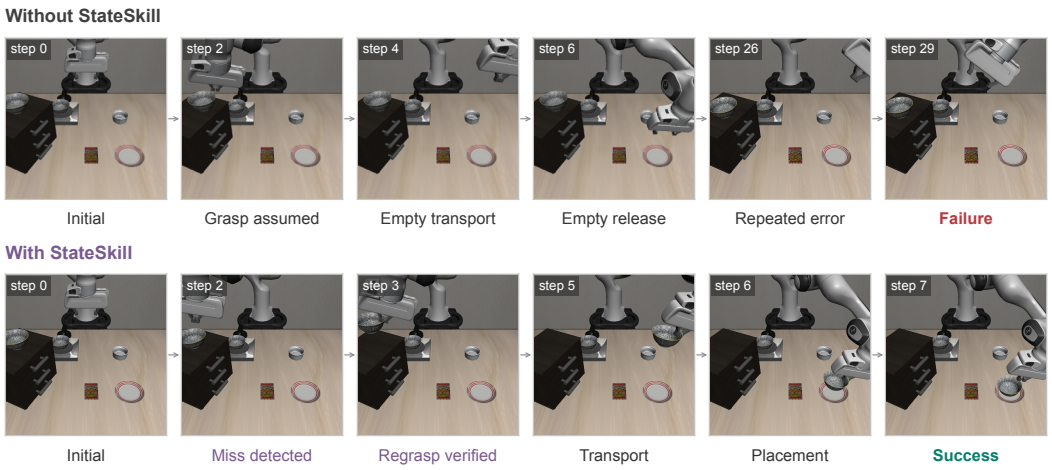


Figure 3: **Qualitative results of StateSkill.** Without StateSkill, the agent incorrectly assumes a successful grasp and continues with empty transport and release, causing repeated errors and eventual failure in a pick-and-place task. With StateSkill, the missed grasp is detected, enabling the agent to regrasp, verify the updated task state, and successfully complete the task.

5 Conclusion

This paper presents StateSkill, a new skill representation for embodied agents, which introduces the concept of "finite-state machines" into skill modeling for effective and efficient skill construction and learning. The core idea of StateSkill is to model embodied interaction experiences as a finite-state machine, where a skill is organized around a set of task-progress states, state transitions, and state-conditioned guidance. By explicitly conditioning skills on progress states, StateSkill enables an embodied agent to track the continuously changing states of the agent, environment, and task and dynamically adjust its skills in a state-conditioned manner as the task progresses, while keeping skill contexts compact and stage-relevant. Furthermore, we extend StateSkill with an iterative experience-driven evolution mechanism, termed StateSkill-Evolve, which retrospectively analyzes new successful and failed trajectories and refines the state-conditioned guidance, state transition gates, and action priors. In addition, we introduce StateGRPO, which integrates StateSkill evolution and model optimization into a unified iterative learning loop. With StateSkill, StateSkill-Evolve, and StateGRPO, we train our model, StateSkill-Agent. Extensive experiments demonstrate the effectiveness of our proposed methods and trained model across various embodied tasks.

AI USE STATEMENT

We used generative AI tools to assist with language editing and improve the clarity and readability of the manuscript. All AI-assisted content was reviewed and revised by the authors, who take full responsibility for the accuracy and integrity of the final manuscript.

REPRODUCIBILITY STATEMENT

We detail StateSkill construction, execution, evolution, and StateGRPO optimization in Appendices A–D. Benchmark configurations, evaluation protocols, model settings, and training and deployment infrastructure are described in Appendix E. Additional ablations and baseline comparisons are provided in Appendices F and G. All simulation evaluation seeds are held out from skill construction, evolution, candidate validation, and model training.

REFERENCES

ACE-Brain Team. ACE-Brain-0.5: A unified embodied foundational model for physical agentic AI. *arXiv preprint arXiv:2607.04426*, 2026.

Anthropic. Introducing agent skills. `https://claude.com/blog/skills`, October 2025. Accessed: 2026-09-14.

Johan Bjorck, Valts Blukis, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, et al. GR00T N1.5: An improved open foundation model for generalist humanoid robots. NVIDIA Research, 2025.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Proc. Conf. Robot Learn. (CoRL)*, 2025a.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. In *Proc. Robot.: Sci. Syst. (RSS)*, 2025b.

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.

Mingyue Cheng, Shuo Yu, Daoyu Wang, Qingchuan Li, Xiaoyu Tao, Jie Ouyang, Yucong Luo, Yitong Zhou, Qi Liu, and Enhong Chen. Agent-R1: A unified and modular framework for agentic reinforcement learning. *arXiv preprint arXiv:2511.14460*, 2025.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *Int. J. Robot. Res.*, 44(10–11):1684–1704, 2025.

Seonglae Cho, Franklin Cardenoso Fernandez, Umar Mohammed, Zekun Wu, Kleyton Da Costa, Ilham Wicaksono, and Adriano Koshiyama. Automata from agent traces: Failure and next-step prediction. *arXiv preprint arXiv:2608.23670*, 2026. URL `https://arxiv.org/abs/2608.23670`.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An embodied multimodal language model. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2023.

Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A. Seshia, and Matei Zaharia. RHO: Your coding agent is secretly a roboticist. *arXiv preprint arXiv:2606.16458*, 2026.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. ReTool: Reinforcement learning for strategic tool use in LLMs. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2026.

Max Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Jim Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2026.

Dibya Ghosh, Homer Rich Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Quan Vuong, Ted Xiao, Pannag R. Sanketi, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proc. Robot.: Sci. Syst. (RSS)*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025.

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.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. XSkill: Continual learning from experience and skills in multimodal agents. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2026.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. VIMA: General robot manipulation with multimodal prompts. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2023.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Dongyoung Kim, Huiwon Jang, Myungkyu Koo, Suhyeok Jang, Taeyoung Kim, Beomjun Kim, Byungjun Yoon, Changsung Jang, Daewon Choi, Dongsu Han, et al. RLDX-1 technical report. *arXiv preprint arXiv:2605.03269*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proc. Conf. Robot Learn. (CoRL)*, 2024.

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.

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 *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2023.

Fanqi Lin, Yingdong Hu, Pingyue Sheng, Chuan Wen, Jiacheng You, and Yang Gao. Data scaling laws in imitation learning for robotic manipulation. In *Proc. Conf. Robot Learn. (CoRL)*, 2025.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2023.

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. *arXiv preprint arXiv:2607.00272*, 2026.

Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. SkillGen: Verified inference-time agent skill synthesis. *arXiv preprint arXiv:2605.10999*, 2026.

Soroush Nasiriany, Sep Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2026.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2Skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Open X-Embodiment Collaboration. Open x-embodiment: Robotic learning datasets and RT-X models. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2024.

OpenAI. GPT-5.5 system card, 2026. URL `https://deploymentsafety.openai.com/gpt-5-5`.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2025.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. ToolRL: Reward is all tool learning needs. In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, 2026. URL `https://huggingface.co/Qwen/Qwen3.5-9B`.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026.

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. *Trans. Mach. Learn. Res.*, 2024a.

Xiaofeng Wang, Zheng Zhu, Guan Huang, Boyuan Wang, Xinze Chen, and Jiwen Lu. WorldDreamer: Towards general world models for video generation via predicting masked tokens. *arXiv preprint arXiv:2401.09985*, 2024b.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Monica Lam, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proc. Int. Conf. Mach. Learn. (ICML)*, 2025b.

Xiyang Wu, Zongxia Li, Guangyao Shi, Alexander Duffy, Tyler Marques, Matthew Lyle Olson, Tianyi Zhou, and Dinesh Manocha. Co-evolving LLM decision and skill bank agents for long-horizon tasks. *arXiv preprint arXiv:2604.20987*, 2026.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

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

Hongzhi Zang, Mingjie Wei, Si Xu, Yongji Wu, Zhen Guo, Yuanqing Wang, Hao Lin, Peihong Wang, Liangzhi Shi, Yuqing Xie, Zhexuan Xu, Zhihao Liu, Kang Chen, Wenhao Tang, Quanlu Zhang, Weinan Zhang, Chao Yu, and Yu Wang. RLinf-VLA: A unified and efficient framework for reinforcement learning of vision-language-action models. In *Proc. Robot.: Sci. Syst. (RSS)*, 2026.

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

Yang Zhang, Jiangyuan Zhao, Chenyou Fan, Fangzheng Yan, Tian Li, Haitong Tang, Sen Fu, Xuan'er Wu, Qizhen Weng, Weinan Zhang, Xiu Li, Chi Zhang, Chenjia Bai, and Xuelong Li. PRTS: A primitive reasoning and tasking system via contrastive representations. *arXiv preprint arXiv:2604.27472*, 2026b.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Zhengru Fang, Yi Nie, 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*, 2026c.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proc. AAAI Conf. Artif. Intell. (AAAI)*, 2024.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Tai Wang, Ya-Qin Zhang, Jingjing Liu, and Xianyuan Zhan. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2026.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proc. Conf. Robot Learn. (CoRL)*, 2023.

A StateSkill Construction

This appendix specifies the construction protocol for an initial StateSkill from trajectories belonging to a given family of local operations. GPT-5.5 infers the progress states, their transitions, and the associated state-conditioned knowledge without a predefined state sequence. The procedure consists of trajectory segmentation, cross-trajectory alignment, and state-wise knowledge synthesis.

A.1 Inputs and State Abstraction

Construction inputs. For one operation family, the input is a trajectory collection $\mathcal{D}_{\text{init}} = \{\tau_i\}_{i=1}^{N}$. Each trajectory contains a local instruction u_i and ordered interaction records:

$$\tau_i = \left(u_i, \{(o_{i,t}, c_{i,t}, a_{i,t}, f_{i,t})\}_{t=0}^{T_i-1}\right), \tag{7}$$

where $o_{i,t}$ is the recorded observation available before the call, $c_{i,t}$ is its high-level action category, $a_{i,t}$ contains its arguments, and $f_{i,t}$ is the returned feedback. Low-level control steps executed within one VLA invocation remain associated with that interaction record. Unavailable observations or feedback are marked as missing. The constructor also receives the action schemas and feedback-field definitions exposed by the configured runtime.

LIBERO instantiation. For LIBERO, we collect construction trajectories through task exploration at seed 0, using Harness VLA's existing exploration records as guidance. Each rollout records observations, executed actions, and execution feedback. GPT-5.5 segments and aligns the resulting trajectories to infer shared progress states and their state-conditioned knowledge. For each state, it specifies the evidence conditions, transition destinations, and recovery branches. These specifications are implemented as deterministic checks over the runtime feedback interface, with geometric tolerances and retry budgets supplied by the execution configuration.

Trajectory segmentation. The constructor partitions each trajectory into contiguous, non-overlapping segments covering all interaction records. Each segment records its local objective, entry conditions, expected completion evidence, observed outcome, relevant entities, and source-record range. A boundary is introduced when the local objective or progress conditions require a distinct stage of reasoning. Different tool calls, camera changes, and repeated attempts do not by themselves require separate stages: complementary actions can serve the same progress objective. Returning to an earlier objective after an intervening stage produces a new segment that can subsequently align with the same shared state.

Expected completion and observed outcome are recorded separately. Issuing an action does not establish that its intended effect occurred. Failed, truncated, and uncertain segments retain their observed status.

Cross-trajectory alignment. The constructor parameterizes task-specific entities by functional roles, such as `target_object`, `destination`, and `selection_reference`. Relations and attributes that affect the operation are preserved, including placement relations and constraints used to distinguish object instances. Segments are aligned when their local objectives and entry and progress conditions are compatible after role substitution. Matching tool names or temporal positions alone is insufficient, while different action counts or observation sequences can still support a shared state.

For a segment collection $\mathcal{Z}$, alignment produces a state vocabulary $\mathcal{Q}$ and an assignment

$$\phi : \mathcal{Z} \to \mathcal{Q}, \qquad \mathcal{D}_q = \{z \in \mathcal{Z} : \phi(z) = q\}, \tag{8}$$

where $\mathcal{D}_q$ contains the segments supporting state q. States can be revisited, and trajectories need not visit identical state sequences.

Candidate transitions. Consecutive segment assignments provide candidate edges, including forward progress and observed recovery. Each edge requires explicit conditions justified by recorded evidence or declared operating rules and compatible with the destination state's entry requirements. Temporal adjacency alone does not establish that these conditions were satisfied. The segment assignments are offline construction annotations; online tracking evaluates returned feedback and retained execution facts rather than replaying annotations or accessing future observations.

A.2 State-Conditioned Knowledge Synthesis

For each state q, the constructor uses $\mathcal{D}_q$ to produce $\kappa_q = (\ell_q, g_q, p_q)$. Together with the inferred states and transitions, these components define $\mathcal{S}^{(0)}$.

Language guidance ℓ_q. The constructor synthesizes the state's objective, useful action strategies, required evidence, and responses to insufficient or contradictory observations. Guidance uses functional roles and runtime quantities so that it can be instantiated across tasks. It captures reusable decision principles rather than copying one action sequence.

Transition conditions g_q. The constructor specifies condition–outcome rules from the associated evidence. A rule identifies required feedback, a predicate over returned observations and retained facts, any fact updates, and an outcome: remain in the current state, transition to another state, or terminate the skill attempt. Its deterministic realization must explicitly handle missing evidence and overlapping conditions. Numerical tolerances and retry budgets are supplied as operating parameters, with their sources distinguished from trajectory-derived rules. Local skill completion remains distinct from authoritative completion of the full task.

Action prior p_q. The constructor identifies a recommended action set $\mathcal{A}_q^{\mathrm{rec}} \subseteq \mathcal{A}$, where $\mathcal{A}$ is the high-level action vocabulary of the configured runtime. Recommendations reflect which actions serve the state objective under the conditions observed in $\mathcal{D}_q$. An action appearing in a failed segment is interpreted together with its context and feedback, rather than automatically being considered unsuitable.

The recommendations are encoded as a soft categorical prior:

$$p_q(c) = \frac{\exp\big(\beta\,\mathbf{1}[c \in \mathcal{A}_q^{\mathrm{rec}}]\big)}{\sum_{c' \in \mathcal{A}} \exp\big(\beta\,\mathbf{1}[c' \in \mathcal{A}_q^{\mathrm{rec}}]\big)}, \qquad \beta > 0. \tag{9}$$

The recommended set is inferred from the supporting segments, while the shared coefficient β controls preference strength. This is a trajectory-informed prior rather than an empirical frequency estimate. Under Equation 2, it adds β to the log probability of each recommended category before renormalization. All categories retain positive prior probability; arguments are generated from the current execution context.

A.3 Illustrative Construction Example

Consider "place the mug on the coaster" and "place the bowl on the plate." Both belong to a pick-and-place family, with mug/bowl filling the target-object role and coaster/plate filling the destination role. This schematic example illustrates construction and is not a reported experimental rollout.

Suppose the first trajectory obtains sufficient target-holding evidence from an external-camera query after a grasp attempt. The second first obtains an ambiguous observation, then uses a wrist-camera query to resolve the outcome. Both segments serve the local objective of verifying acquisition of the bound target and can align to one state, despite using different observation sequences. The resulting knowledge can take the following form:

- ℓ_q: obtain fresh, target-specific holding evidence, use another view when needed, and distinguish controller completion from successful acquisition.
- g_q: enter transport when the holding-evidence condition is satisfied, and retain the verification state while further evidence is required.
- p_q: recommend `segment`, with the target and camera arguments selected from the current context, and encode this recommendation using Equation 9.

A shared state captures a common progress objective and its evidence requirements across different action sequences. Task-specific objects are bound at execution time. Additional recovery rules require their own supporting experience or an explicitly identified operating rule.

Table 6: Output representation for StateSkill construction.

Record	Contents
Skill	Operation family, role definitions, entry state, state identifiers, and skill termination outcomes.
Segment	Trajectory identifier, start and end indices, local objective, entry and progress conditions, observed outcome, role bindings, and assigned state.
State	Canonical objective, supporting segments, language guidance ℓ_q, retained fact definitions, and recommended set $\mathcal{A}_q^{\mathrm{rec}}$.
Gate rule	Source state, required evidence, predicate, fact updates, destination state or skill outcome, precedence, and evidence or configuration references.

A.4 Constructor Prompt and Output Format

The constructor first produces source-indexed stage descriptions, then aligns them and synthesizes shared state knowledge. Supporting interaction records remain available during alignment and synthesis so that abstractions can be checked against their source evidence.

Constructor prompt. The following prompt defines the construction protocol. Inputs include the operation-family description, action schemas, feedback-field definitions, trajectories, and declared operating parameters.

> You construct reusable StateSkills from recorded embodied interactions. All supplied trajectories belong to the given family of local operations. Infer its progress states, transitions, and state-conditioned knowledge without assuming a predefined state sequence.
>
> 1. Partition each trajectory into contiguous stages. Report the source-record range, local objective, entry conditions, expected completion evidence, observed outcome, and relevant object roles. Distinguish intended actions from observed effects.
> 2. Align stages after parameterizing entities by functional roles. Preserve behaviorally relevant relations. Merge only compatible objectives and progress conditions. A state can contain multiple actions and can be visited repeatedly.
> 3. Propose transitions from the aligned sequences. Attach supporting records and required evidence, distinguishing verified progress, recovery, and attempted progress whose conditions failed.
> 4. For each state, synthesize language guidance, transition rules, and recommended action categories. Express predicates using declared feedback fields and retained facts. Specify required evidence, fact updates, and precedence between overlapping rules.
> 5. Retain source references for states and their knowledge. Identify externally supplied rules and parameters. Do not invent observations, feedback fields, thresholds, or recovery experience.
> 6. Return the entry state, state definitions, transitions, termination outcomes, recommended action sets, and source references in the specified output structure.

Output representation. Table 6 summarizes the construction records. Recommended sets are converted into p_q using Equation 9; numerical action probabilities are not generated independently by the constructor.

Consistency requirements. Segments must cover the input without overlap, state and evidence references must resolve, and recommended actions must belong to the declared vocabulary. Gate predicates must use execution-time information and explicitly handle missing evidence and overlapping conditions. These checks establish structural consistency; behavioral validity is assessed through subsequent execution. Initial construction artifacts and source references are retained separately from later StateSkill-Evolve revisions.

B State Tracking and Action Selection

This appendix describes the runtime interface shared by StateSkills. We use two StateSkills from LIBERO—pick-and-place and contact interaction—as concrete examples of how this interface is instantiated. Their state decompositions, verification predicates, and numerical thresholds are specific to these examples, rather than universal requirements of StateSkill. Figure 3 illustrates the pick-and-place example.

B.1 Runtime Interface of StateSkills

A StateSkill exposes an active progress state q, retained facts m, and state-conditioned knowledge (ℓ_q, g_q, p_q). The state identifies the current local objective; the retained facts provide the information needed to interpret subsequent evidence, including role bindings, previous action outcomes, and unresolved conditions. Following an executor call, returned observations and feedback provide evidence e. The runtime evaluates g_q against this evidence and the retained facts, updating q and m as in Equation 3. The resulting state selects the guidance and action prior for the next decision by the agent.

Within a configured runtime, the available action interface remains fixed across states. StateSkills change the context and preferences associated with these actions, while executor checks separately enforce argument validity and applicable execution constraints. Thus, a state describes progress toward an objective and can involve several complementary actions, including observation, manipulation, and verification. Different StateSkills may use different state decompositions.

B.2 Evidence-Based State Transitions

The gate g_q specifies when accumulated evidence permits a transition, requires another observation or attempt, or establishes a terminal outcome. Issuing an action and observing its intended effect are distinct events: an executor acknowledgment can establish that a call was dispatched without establishing that its objective was achieved. Likewise, unavailable or ambiguous feedback is retained as uncertainty rather than converted into positive physical-verification evidence.

The retained facts m allow a transition to depend on multiple calls. For example, evidence about an interaction outcome and evidence about a subsequent recovery action need not arrive together. The gate can retain either fact until the remaining condition is established. Transitions therefore depend on the current state and its evidence history, rather than solely on the latest image or the name of the latest action. The treatment of uncertainty, retry limits, and terminal conditions is specified by the instantiated skill.

A skill can also return a local outcome to the task-level controller, which determines whether another skill is needed. Local completion records one operation; authoritative environment success determines whether the full task has succeeded.

B.3 State-Conditioned Action Selection

Language guidance ℓ_q describes the current objective, relevant evidence, and useful next actions. The prior p_q expresses relative preferences over action categories, following the construction rule in Equation 9. When explicit prior reweighting is enabled, these preferences influence category selection. Task-specific arguments, such as an object description or a motion target, still depend on the current instruction and observations. An action preference does not itself satisfy a transition condition.

When the backend exposes action-category scores, adding $\beta\,\mathbf{1}[c \in \mathcal{A}_q^{\mathrm{rec}}]$ to each category's log score and normalizing over the action vocabulary realizes Equation 2. Action arguments are generated conditioned on the selected category and current context. Every category retains positive prior probability.

Implementation-specific realization. In LIBERO, the Qwen3.5-9B configuration applies the explicit prior with bias strength $\beta = 2$. The GPT-5.5 configuration retains state-conditioned guidance and gate evaluation but does not apply this Qwen-specific bias mechanism. The action inventory

StateSkill Example: Grasp Verification and Recovery

Task and execution context

```
Task: "Put the plate on the stove."
> Initial grasp: controller success = true
```

The controller has completed its motion; target holding still needs verification.

Activated skill: Pick-and-place **Active state:** Verify grasp

Initiation: A grasp attempt has completed, but target acquisition is not yet confirmed.

Language guidance (ℓ)

1. Inspect the bound target in a fresh post-grasp view.
2. Check whether the target was acquired or remains at its source.
3. If the external view is unclear, inspect the wrist view of the same attempt.
4. Do not infer a successful grasp from controller completion alone.

Transition gate (g)

HELD: Holding is confirmed; proceed to transport.
MISS: Retry grasping if attempts remain; otherwise end the failed skill attempt.
UNKNOWN: Inspect another view; if still unresolved, transport with holding marked as unverified.

Action prior (p)

Recommended category: Visual inspection (segment).
Prior: Higher weight for this category; other actions remain selectable.

Recorded execution *LIBERO-PRO · goal task 1 · seed 10*

First grasp: controller success → **MISS → retry grasp**
Second grasp: controller success → **HELD → transport**
Outcome: *The environment confirms task success after release.*

Figure 4: **A StateSkill example: grasp verification and recovery.** The card summarizes the verification state's guidance, transition gate, and action prior. The recorded execution illustrates missed-grasp detection and recovery. Wording and execution traces are condensed for presentation.

and prior realization are properties of the configured runtime, rather than a requirement that all benchmarks expose an identical tool set.

B.4 Illustrative Instantiations

Perception and geometric evidence. In the LIBERO examples below, SAM 3 segmentation masks are combined with simulator-rendered depth and camera calibration. Valid masked depth points are back-projected into world coordinates, and each candidate location is represented by their coordinate-wise median. End-effector positions are obtained from robot-state feedback at the corresponding observation step. The geometric localization backend is shared with the Harness VLA baseline; StateSkill organizes its outputs into state-conditioned verification evidence.

LIBERO pick-and-place. This instantiation uses eight stages: grounding, pre-grasp preparation, VLA grasping, grasp verification, transport, approach, release, and placement verification. Figure 4

illustrates a representative portion of this StateSkill, focusing on grasp verification and recovery. The card presents its state-conditioned knowledge alongside a recorded execution fragment.

Grounding retains the source position for later grasp checks. The geometric predicates below use candidate coordinates from this perception pipeline. For a candidate, d_e is its XY distance to the end effector, d_s its XY distance to the retained source position, and Δz its vertical rise relative to that position. All distances are in meters. Verification requires fresh evidence matching the bound target query, expected camera, and post-grasp observation step, together with valid source and end-effector measurements. Ineligible evidence cannot advance the state.

Table 7: Evidence-based grasp verification for eligible observations in the LIBERO pick-and-place instantiation. Verify, Pick, and Transport refer to its grasp-verification, VLA-grasping, and transport states. These rules illustrate one skill's gate; they do not define the state structure of other StateSkills.

Outcome	Evidence condition	State update
`HELD`	Exactly one candidate satisfies $d_e \leq 0.10$ and $\Delta z > 0.01$; this test takes precedence.	Verify $\rightarrow$ Transport with confirmed holding evidence.
`MISS`	No candidate satisfies the held predicate, and exactly one candidate satisfies $d_s \leq 0.05$ and $\Delta z \leq 0.01$.	Verify $\rightarrow$ Pick while the attempt budget remains; otherwise return failure.
`UNKNOWN`	Neither condition above establishes a unique result.	Remain in Verify for a wrist query; if still inconclusive, use the declared unverified-holding fallback.

A miss permits another grasp attempt within a limit of three dispatched or possibly dispatched VLA attempts. A unique agent-view `MISS` may refresh the source position for the next attempt; a wrist-view `MISS` does not. An `UNKNOWN` result from the agent-view camera triggers wrist-view verification. If the wrist view also remains `UNKNOWN`, this instantiation permits transport while retaining an unverified-grasp risk. Its default held-object geometry uses zero horizontal offset relative to the gripper and a 0.10 m end-effector-to-object-bottom distance. This fallback is a skill-specific execution policy, not positive grasp evidence. Figure 3 illustrates the role of grasp verification in detecting a missed grasp.

After release, the agent inspects a fresh image and reports `CONFIRMED`, `REFUTED`, or `UNVERIFIED` placement. An accepted, non-refuted assessment together with a valid return to the start configuration establishes local skill completion. In particular, `UNVERIFIED` can be accepted by this policy; local completion is recorded separately from the environment's official task-success signal.

LIBERO contact interaction. The contact skill instead uses one progress state with accumulated facts. It freezes the local instruction for one VLA attempt, obtains a fresh post-attempt view and an assessment from the agent, and records a return to the start configuration. Assessment and reset can occur in different orders; memory preserves the facts required for the final decision. A confirmed outcome together with a valid reset establishes local success, whereas a refuted outcome with a valid reset establishes ordinary failure. An unsafe outcome is terminal, and an unverified verdict does not establish success. This single-state example demonstrates that evidence accumulation does not require assigning a separate progress state to every executor call.

B.5 Recorded Execution Case Study

Task: "Put the plate on the stove" | LIBERO-PRO goal task 1, seed 10 | Qwen3.5-9B

With StateSkill (Success)	HarnessVLA (Failure)

Phase 1: Knowledge Guidance (excerpts from the rollout)

State-conditioned guide: ACQUIRE_VERIFY
"Do not treat controller success as visual holding evidence."

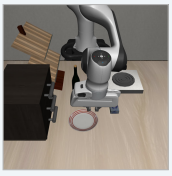
```
MISS -> ACQUIRE_VLA_PICK
HELD -> DEPOSIT_TRANSPORT
```

General guides and task memory
Successfully read the guidance documents:

```
strict_hybrid_guide.md
pro_hybrid_guide.md
env_calibration.md
```

Phase 2: Agent Execution and Verification (condensed traces)

Step 2: Initial grasp and verification
`pi0_pick(...) -> success=true`



`segment(...) -> MISS`
Reload the guide:
`ACQUIRE_VLA_PICK`
Plate remains at source.

Step 2: Initial grasp
`pi0_pick(...) -> success=true`



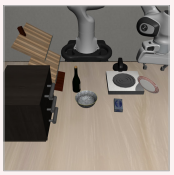
Controller reports success; transport follows.
Peak lift: 0.072 m.

Step 3: Retry and confirm holding
`pi0_pick(...); segment(...)`



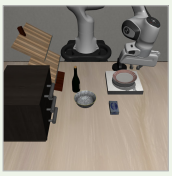
HELD
Load the guide:
`DEPOSIT_TRANSPORT`
Target verified in hand.

Step 6: Release after transport
`release(...) -> terminated=false`



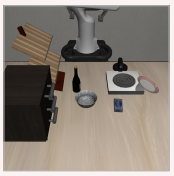
Task remains unsuccessful after release.
Task success: false

Step 7: Place and release
`release() -> terminated=true`



Environment confirms
SUCCESS
Plate on the stove.

Step 9: Final completion claim
`finish(status="success")`



Environment still reports
`terminated=false`
FAILURE

Key Insights

(1) Verification: observed MISS refutes the controller's success report.
(2) Recovery: MISS reloads acquisition guidance; HELD activates transport guidance.
(3) Outcome: task success is determined by the environment, not the agent's claim.

Figure 5: **Recorded StateSkill–Harness VLA comparison.** Selected guidance and execution evidence on LIBERO-PRO goal task 1, seed 10. The case illustrates how verification feedback triggers grasp recovery during execution.

C STATESKILL EVOLUTION

StateSkill-Evolve uses execution experience to refine the skill's progress-state structure $(\mathcal{Q}, \mathcal{E})$ and its state-conditioned knowledge $\kappa_q = (\ell_q, g_q, p_q)$. State-level diagnosis identifies where a revision is needed; the revision may change knowledge within a state or the way states divide and connect local stages. This appendix specifies how evidence supports these candidates and how new rollouts determine whether to retain them.

C.1 STATE-LEVEL DIAGNOSIS

Inputs and evidence. The offline LLM updater receives the current StateSkill, the available action and feedback interfaces, and newly collected successful and failed trajectories. Each trajectory is associated with the skill version used during collection and records the task instruction, observations, agent actions, executor feedback, progress-state trace, and full-task outcome. The updater analyzes a decision against the guidance and evidence available at that point in execution.

For each suspected failure, the analysis locates the earliest unresolved discrepancy between the intended local objective and the observed execution. A later terminal state does not necessarily identify where the problem originated, and a failed action that was subsequently recovered need not motivate a revision. Successful trajectories through the same state provide examples of effective behavior and conditions that a revision should preserve.

Revision eligibility. A state identifies where to inspect the trajectory; it does not by itself establish the cause of failure. The updater distinguishes deficiencies in skill structure or knowledge from failure to follow adequate guidance, low-level execution errors, and missing evidence. A revision hypothesis must identify an actionable gap, such as an ambiguous instruction, an inappropriate action preference, distinct interaction stages conflated in one state, or a missing verification stage or recovery transition. Insufficient evidence leaves the diagnosis unresolved. Candidate execution subsequently tests the proposed behavioral improvement; the diagnosis alone is not treated as proof of its effectiveness.

C.2 EVIDENCE-GUIDED STRUCTURAL AND KNOWLEDGE REVISION

The updater identifies the affected portion of the skill and produces

$$\widetilde{\mathcal{S}}^{(k+1)} = \left(\widetilde{\mathcal{Q}}, \widetilde{\mathcal{E}}, \{\widetilde{\kappa}_q\}_{q \in \widetilde{\mathcal{Q}}}\right). \tag{10}$$

The candidate retains a finite-state representation and the configured action and feedback interfaces, while its state set and transition graph may change. Revisions can combine the following operations:

- **State structure** $(\mathcal{Q}, \mathcal{E})$**:** add a missing stage, split states with distinct interaction requirements, merge overlapping stages, remove redundant states, and add, remove, or redirect transitions. Associated knowledge is defined or revised to match the resulting state meanings and connections.
- **Language guidance** ℓ_q**:** clarify applicable conditions, action arguments, verification requirements, and responses to failure or uncertainty.
- **Transition conditions** g_q**:** revise the evidence required to remain in a state, follow a transition in the candidate graph, or return a local outcome, while maintaining explicit handling of unavailable or conflicting observations.
- **Action prior** p_q**:** revise the recommended action set using the soft-prior construction in Equation 9, while preserving the configured action vocabulary and positive prior support.

Each candidate records its supporting trajectory segments, the diagnosed gap, affected states and transitions, proposed edits, expected behavioral effect, and behavior to preserve. Structural and knowledge edits can be combined to express one coherent hypothesis. Edits use functional roles and observable conditions to support reuse beyond the reviewed episode. Knowledge is retained for surviving states whose meanings and behavior are unaffected; new or redefined states receive corresponding knowledge.

Before rollout validation, consistency checks verify a finite state set, valid entry and transition references, and complete knowledge for every executable state. Revised gates must agree with the candidate graph, local-outcome interface, and executable realization; action and evidence fields must remain valid. Guidance, gates, and priors must describe compatible behavior, including explicit handling of unavailable evidence. These checks establish that a candidate is well formed; new execution is required to evaluate its utility.

C.3 Paired Validation and Candidate Acceptance

The current and candidate skills are evaluated on the same validation set $\mathcal{V}$, containing both instances that exercise the targeted behavior and other instances affected by the revised states, transitions, or knowledge. The MLLM checkpoint, low-level executors, sampling configuration, initial conditions, and interaction budgets are matched between the two versions. Both are evaluated with the same current MLLM checkpoint, even if StateGRPO has updated it since the preceding iteration. Validation uses complete task rollouts and authoritative task-success signals; local skill completion is insufficient.

Let $s_j(\mathcal{S}; \pi_{\mathrm{val}})$ denote success on validation instance j with skill $\mathcal{S}$ and the fixed MLLM policy π_{val}. Define

$$\widehat{P}_{\mathcal{V}}(\mathcal{S}) = \sum_{j \in \mathcal{V}} \omega_j \, s_j(\mathcal{S}; \pi_{\mathrm{val}}), \qquad \omega_j > 0, \quad \sum_{j \in \mathcal{V}} \omega_j = 1. \tag{11}$$

The weights are fixed before comparison and give each validation task equal total weight, consistent with the task-averaged evaluation metric. Budget-exhausted episodes without task success count as failures. The behavioral checks are specified with the candidate before validation and concern its declared structural and knowledge changes. A separate verification pass checks the candidate's rollout evidence against its proposed behavioral effect and the conditions declared for preservation. Let $v^{(k)} = 1$ when the consistency and behavioral checks pass, and $v^{(k)} = 0$ when they fail or the required evidence is insufficient. The update rule is

$$\begin{aligned} \Delta_{\mathcal{V}}^{(k)} &= \widehat{P}_{\mathcal{V}}(\widetilde{\mathcal{S}}^{(k+1)}) - \widehat{P}_{\mathcal{V}}(\mathcal{S}^{(k)}), \\ \mathcal{S}^{(k+1)} &= \begin{cases} \widetilde{\mathcal{S}}^{(k+1)}, & \text{if } v^{(k)} = 1 \text{ and } \Delta_{\mathcal{V}}^{(k)} > 0, \\ \mathcal{S}^{(k)}, & \text{otherwise.} \end{cases} \end{aligned} \tag{12}$$

Ties retain the current skill. Per-instance gains and regressions are recorded alongside the aggregate score. Acceptance is determined by performance across the full validation set; individual regressions may coexist with an aggregate gain. The selection criterion establishes improvement on the validation set; generalization is assessed separately on final test instances.

C.4 Development and Final Evaluation

Within each benchmark with seeded resets, the seeds used for final evaluation are disjoint from all seeds used for initial skill construction, skill evolution, candidate validation, and MLLM training. Writing these sets as $\mathcal{Z}_{\mathrm{test}}$ and $\mathcal{Z}_{\mathrm{dev}}$, respectively,

$$\mathcal{Z}_{\mathrm{test}} \cap \mathcal{Z}_{\mathrm{dev}} = \varnothing. \tag{13}$$

Validation supports candidate selection and remains development data. Its feedback can inform subsequent revision, but final-test trajectories and outcomes are excluded from skill construction, revision, candidate selection, and MLLM policy optimization. Skill contents and MLLM parameters are fixed before final evaluation. Seed separation concerns unseen initializations; task-level generalization additionally depends on the benchmark's task split.

C.5 Illustrative Structural and Knowledge Revision

Consider a skill for opening an articulated container that represents actuation and completion assessment in one state. Suppose an action acknowledgment triggers completion without confirming the opening. The updater can split this state into actuation and verification, with a recovery transition for confirmed incomplete opening. Table 8 illustrates a possible structural edit and its associated knowledge updates. This is a schematic example, not a reported experimental result.

Table 8: Schematic structural and knowledge revision for opening an articulated container. A new verification state distinguishes confirmed completion, incomplete opening, and insufficient evidence.

Component	**Evidence or proposed revision**
Diagnosis	Execution acknowledgment and evidence of opening are conflated, allowing unsupported local completion.
Structure $(\mathcal{Q}, \mathcal{E})$	Split the original state into ACTUATE and VERIFY; connect actuation to verification and add a return transition for recovery.
Guidance ℓ_q	Inspect a fresh view after actuation; obtain additional observations when occlusion prevents judgment.
Gate g_q	Confirmed opening returns local completion. Confirmed incomplete opening permits a retry within the configured budget; uncertainty retains VERIFY for further observation.
Prior p_q	Favor contact actions in ACTUATE and observation actions in VERIFY.
Validation target	Check whether the revised flow prevents unsupported completion while preserving valid executions, and evaluate its effect on full-task success.

D StateGRPO Optimization

This appendix specifies the state accounting, policy objective, and alternating update schedule of StateGRPO. Progress states provide a shared basis for comparing execution efficiency across successful trajectories, while task success remains the outcome criterion.

D.1 State Accounting and Group-Relative Rewards

State identity and interaction counts. Each trajectory group corresponds to one task and one environment seed. All G trajectories start from the same seeded environment initialization and use the same MLLM checkpoint and fixed StateSkills. The shared seed controls the environment reset, while sampling from the MLLM policy remains stochastic. State-wise efficiency comparisons therefore hold the task and initial conditions fixed. All seeds used for these training groups are disjoint from the seeds reserved for final evaluation, following Appendix C.4. A state identifier q denotes a pair consisting of the skill identity and its local progress state. States with the same local name in different skills are therefore distinct. Let H_i be the number of agent interaction turns in τ_i, and let $q_{i,t}$ denote the state active before turn t. The state-specific interaction count is

$$T_{i,q} = \sum_{t=1}^{H_i} \mathbf{1}[q_{i,t} = q]. \tag{14}$$

One turn is one decision cycle of the agent, irrespective of the number of low-level control steps executed by its selected action. Repeated visits to a state, including visits across invocations of the same skill, contribute to the same count. A transition following an action affects the state assigned to the next turn. For turns without an active StateSkill, set $q_{i,t} = \perp$; this symbol is excluded from state-efficiency comparisons.

Comparable successful experience. The indicator s_i records authoritative success of the full task; local skill completion does not suffice. Within each trajectory group, define

$$\begin{aligned} \mathcal{I}_q^+ &= \{i : s_i = 1,\ T_{i,q} > 0\}, \\ \mathcal{C}_i &= \{q : T_{i,q} > 0,\ |\mathcal{I}_q^+| \geq 2\}. \end{aligned} \tag{15}$$

For each eligible state, Equation 4 averages the counts of successful trajectories that visit it. The state reward in Equation 5 compares a trajectory against these references and averages over its comparable states. Failed trajectories and trajectories with $\mathcal{C}_i = \varnothing$ receive zero state reward. References are computed separately for each task group and are not pooled across skill versions. This comparison does not require identical state sequences or assume that different states take equally many turns. It measures agent interaction efficiency within a state, rather than physical execution time.

Advantages and uninformative groups. Using $R_i = s_i + R_{\text{state}}(\tau_i)$, the group statistics in Equation 6 are

$$\mu_R = \frac{1}{G}\sum_{i=1}^{G} R_i, \qquad \sigma_R = \sqrt{\frac{1}{G}\sum_{i=1}^{G}(R_i - \mu_R)^2}. \tag{16}$$

If all total rewards coincide, every advantage is zero and the group contributes no policy gradient. A group of successful trajectories can still provide a learning signal through differences in state efficiency; it is therefore not discarded solely because its success indicators are identical. State rewards enter policy optimization through the trajectory-level advantages.

D.2 Policy Optimization under StateSkill-Guided Sampling

Behavior and optimization policies. At iteration k, let $\pi_{\text{old}} = \pi_{\theta^{(k)}}$ denote the MLLM policy before optimization, and let $\mu^{(k)}$ denote the behavior distribution used to collect trajectories. The latter includes StateSkill-guided action selection, with the prior in Equation 2 when enabled, and the configured sampling procedure. It can therefore differ from π_{old} even before any parameter update.

Let $y_{i,j}$ be the jth MLLM-generated token in τ_i, and $h_{i,j}$ its recorded context, including observations, interaction history, active skill guidance, and the generated response prefix. All policy

likelihoods below condition on this same context. Write L_i for the number of generated tokens. Generated reasoning, action names, and action arguments participate in the objective; prompts, injected guidance, and executor feedback are conditioning information and are excluded from the optimized tokens.

Truncated importance weighting. For each sampled token, define

$$w_{i,j} = \mathrm{stopgrad}\left[\min\left\{\bar{w}, \frac{\pi_{\mathrm{old}}(y_{i,j} \mid h_{i,j})}{\mu^{(k)}(y_{i,j} \mid h_{i,j})}\right\}\right], \tag{17}$$

where $\bar{w} \geq 1$ bounds the correction and $\mathrm{stopgrad}$ treats its argument as constant during optimization. The denominator is the normalized probability under the distribution actually used to sample the token. These bounded token-level weights account for conditional sampling differences at recorded histories, without providing an unbiased correction of the full trajectory distribution.

Clipped policy objective. Define the policy-update ratio

$$r_{i,j}(\theta) = \frac{\pi_\theta(y_{i,j} \mid h_{i,j})}{\pi_{\mathrm{old}}(y_{i,j} \mid h_{i,j})}. \tag{18}$$

We maximize the token-averaged clipped surrogate

$$\begin{aligned}\mathcal{J}_{\mathrm{StateGRPO}}(\theta) = \frac{1}{\sum_{i=1}^{G} L_i} \sum_{i=1}^{G} \sum_{j=1}^{L_i} w_{i,j} \\ \times \min\left\{r_{i,j}(\theta)\widehat{A}_i,\ \mathrm{clip}(r_{i,j}(\theta), 1-\epsilon_{\mathrm{clip}}, 1+\epsilon_{\mathrm{clip}})\,\widehat{A}_i\right\},\end{aligned} \tag{19}$$

where $\epsilon_{\mathrm{clip}} \in (0, 1)$ controls policy-ratio clipping and is distinct from the reward-normalization constant ϵ. Each trajectory's advantage is shared by its generated tokens. The reward references, advantages, importance weights, and old-policy likelihoods remain fixed throughout optimization of the collected batch. Gradients update only the MLLM parameters, with no differentiation through state transitions, reward computation, or skill contents.

D.3 Alternating Skill and MLLM Updates

One iteration proceeds as follows:

1. Refine the current StateSkills using successful and failed state-guided trajectories through StateSkill-Evolve (Section 3.3). Keep the current MLLM checkpoint fixed during candidate validation and apply only accepted revisions.
2. With the accepted StateSkills and current MLLM checkpoint fixed, collect each trajectory group for a fixed task and environment seed, retaining state traces and behavior-policy token probabilities.
3. Compute state counts, successful-state references, total rewards, and group-relative advantages from the collected groups.
4. Hold the collected contexts and StateSkills fixed, and update the MLLM using Equation 19.
5. Collect fresh interaction experience with the updated MLLM and current StateSkills for StateSkill-Evolve in the next iteration.

Skill revisions affect subsequent rollouts; they do not retroactively change state annotations or rewards in a batch already collected for optimization. Thus, MLLM optimization uses a fixed progress representation within each update, while skill evolution refines reusable knowledge between updates.

E Experimental Protocols and Deployment Details

E.1 Benchmarks and Evaluation Protocols

LIBERO-PRO contains 80 tasks across four task families under task and swap perturbations. We evaluate each task using 10 environment seeds. For RoboCasa365, we use the 50-task target set (Target50), comprising 18 atomic-seen, 16 composite-seen, and 16 composite-unseen tasks. Following Harness VLA's Target50 task–seed configuration, we use seeds 1–10 for atomic tasks and 1–5 for composite tasks, yielding 340 evaluation episodes. Each of the five real-world manipulation tasks is evaluated over 20 trials.

We compute each task's success rate over its evaluation episodes and report Overall as the unweighted mean of these per-task rates. Thus, tasks receive equal weight even when their evaluation episode counts differ. Simulation evaluation seeds are disjoint from those used for skill construction, skill evolution, candidate validation, and MLLM training, following Appendix C.4.

E.2 Agent Configuration and Co-Evolution Schedule

GPT-5.5 or Qwen3.5-9B serves as the high-level MLLM in the main comparisons. Our agent uses SAM 3 for segmentation across all evaluation settings. The low-level VLA executors are π_{RLinf} for LIBERO-PRO, RLDX-1 for RoboCasa365, and $\pi_{0.5}$ for real-world tasks. These executors remain frozen, while StateGRPO updates only the Qwen3.5-9B MLLM.

For the full framework, we first refine StateSkills through StateSkill-Evolve, then optimize the MLLM with StateGRPO using trajectories collected under the accepted skills. Skill contents and MLLM parameters are fixed before final evaluation, as specified in Appendix C.4.

E.3 Training Infrastructure

StateGRPO training uses 16 NVIDIA A800 GPUs with 80 GB of memory per GPU, distributed across four nodes with four GPUs per node. The Qwen3.5-9B MLLM is optimized using FSDP2 in BF16 precision, with vLLM serving rollout generation. The MLLM, VLA executor, SAM 3, and simulation services share the allocated GPU resources. Only the MLLM is updated; the VLA executor and SAM 3 remain frozen throughout training.

E.4 Inference Deployment

The local Qwen3.5-9B deployment uses two NVIDIA A100 GPUs with 80 GB of memory each. One GPU serves the MLLM through vLLM in BF16 precision. The other GPU hosts the VLA executor, SAM 3, and simulation rendering. The VLA and SAM 3 run as separate services accessed through HTTP RPC interfaces. This separates MLLM inference, perception, and execution at the process level while allowing the latter two to share a GPU. Experiments with GPT-5.5 access the MLLM through a hosted service rather than deploying its model weights locally.

E.5 Real-World Robotic Platform

Table 9 summarizes the robotic platform. The two arms are arranged as a left–right pair, with an electric gripper and a wrist-mounted camera on each arm. A fixed overhead camera provides a shared view of the workspace. The firmware-reported gripper opening ranges are 0–70 mm for the left arm and 0–100 mm for the right arm.

Table 9: Hardware configuration of the real-world robotic platform.

Component	**Specification**	**Quantity**
Robotic arm	AgileX Piper series; six joint degrees of freedom per arm	2
Electric gripper	One gripper per arm	2
Overhead camera	Intel RealSense D455; fixed overhead mounting	1
Wrist camera	Intel RealSense D435i; one on each wrist	2

F CONTROLLED ABLATION STUDIES

We examine three aspects of the framework: how skill knowledge guides execution, how progress states inform the training reward, and how execution experience is used to revise skills. These comparisons complement the incremental system evaluation in Table 4 by examining specific alternatives within each experimental block.

F.1 COMPARISONS AND REPORTING

We report task success rates, token usage per episode in millions, and, where available, agent interaction turns. The execution comparison additionally reports erroneous advancement, while the revision comparison records changes in success relative to its fixed-skill reference. For the B1–B2 reward comparison, we additionally report variation in the success-rate difference across ten training repetitions. Other comparisons are reported as point estimates and interpreted descriptively.

F.2 KNOWLEDGE REPRESENTATION AND EXECUTION

Table 10 compares four execution conditions using the same underlying skill knowledge. A0 presents the knowledge as a complete text. A1 uses explicit StateSkill progress states but relies on the MLLM to assess progression, without an explicit action prior. A2 replaces MLLM-assessed progression with evidence-based transition gates and also omits the prior. A3 adds the explicit action prior, yielding the complete StateSkill. A2 versus A1 examines transition evaluation; A3 versus A2 examines action-prior reweighting.

Table 10: Ablations of knowledge presentation and StateSkill execution on LIBERO-PRO with Qwen3.5-9B. Err. advance. denotes the reported erroneous-advancement rate. Token usage is measured in millions per episode.

Condition	**Success (%) ↑**	**Tokens/ep. (M) ↓**	**Turns ↓**	**Err. advance. (%) ↓**
A0: Full-knowledge text	47.0	1.10	38.5	18.0
A1: MLLM-assessed progression, no prior	54.5	0.67	31.0	13.0
A2: Evidence gates, no prior	60.0	0.65	29.5	6.5
A3: Full StateSkill	**63.5**	**0.60**	**27.5**	**6.0**

Textual knowledge and structured execution. With the same underlying skill knowledge, A1 achieves 54.5% success at 0.67M tokens per episode, compared with 47.0% and 1.10M for A0. Interaction turns decrease from 38.5 to 31.0. This comparison evaluates the combined effect of organizing execution by explicit progress states and presenting state-conditioned guidance, relative to full-text prompting.

Transition verification and action preferences. Replacing MLLM-assessed progression with evidence gates (A2 versus A1) increases success from 54.5% to 60.0% and reduces the reported erroneous-advancement rate from 13.0% to 6.5%, alongside reductions in interaction turns and token usage. This pattern is consistent with verification helping prevent premature progression. Adding the explicit action prior (A3 versus A2) raises success by a further 3.5 percentage points, while reducing token usage from 0.65M to 0.60M and interaction turns from 29.5 to 27.5. These comparisons separate the benefit of evidence-based transition evaluation from the additional benefit of action-prior reweighting.

F.3 STATE-AWARE REWARD DESIGN

To examine the reward design used in StateGRPO, we hold the initial StateSkills $\mathcal{S}^{(0)}$ fixed and disable skill evolution. All three GRPO variants start from the same MLLM checkpoint. Table 11 compares a task-success reward (B0), task success combined with global efficiency (B1), and task success combined with the state-aware efficiency reward in Equation 5 (B2).

Table 11: Comparison of GRPO reward designs with fixed initial StateSkills $\mathcal{S}^{(0)}$ and a shared MLLM initialization. Skill evolution is disabled. The reference row reports pre-training performance. B1 and B2 success rates are means over ten training repetitions.

Condition	Success (%) ↑	Tokens/ep. (M) ↓	Turns ↓
Before training: Initial StateSkills	63.5	0.60	27.5
B0: Task-success reward	67.0	0.56	26.0
B1: Success + global efficiency	67.5	**0.48**	**22.9**
B2: Success + state-aware efficiency	**69.3**	0.49	23.2

Task-success-only GRPO increases success from 63.5% to 67.0%. Adding global efficiency shaping raises success to 67.5% while reducing token usage and interaction turns. With the state-aware efficiency reward, B2 reaches 69.3%, exceeding B0 and B1 by 2.3 and 1.8 percentage points, respectively. Relative to B1, B2 uses slightly more tokens (0.49M versus 0.48M) and interaction turns (23.2 versus 22.9). Under fixed initial skills, state-aware reward shaping thus yields the highest observed success rate, with execution cost close to that of global efficiency shaping.

Variation across training repetitions. Across ten training repetitions in simulation, the mean success-rate difference between B2 and B1 is 1.80 percentage points, with a sample standard deviation of 1.56 percentage points and a 95% confidence interval of $[0.69, 2.91]$ percentage points. B2 improves over B1 in eight repetitions, ties in one, and performs worse in one. Mean gains are positive across all four LIBERO-PRO task families: 0.8 percentage points on Spatial, 1.2 on Object, 2.2 on Goal, and 3.0 on LIBERO-10. The confidence interval excludes zero, supporting a positive average benefit of state-aware reward shaping over global efficiency shaping across training repetitions.

F.4 State-Localized Skill Revision

Table 12 compares a fixed skill (C0), whole-skill reflection (C1), and revision following state-localized diagnosis (C2). C2 corresponds to StateSkill-Agent without StateGRPO fine-tuning. In addition to success and deployment token usage, we report paired outcome changes relative to C0 on the same task instances and environment seeds. A repair is an instance on which C0 fails and the revised skill succeeds; a regression is an instance on which C0 succeeds and the revised skill fails. These counts capture both beneficial and detrimental changes.

Table 12: Skill-revision comparison. Repairs and regressions are paired counts relative to the fixed-skill reference C0. Token usage is measured in millions per episode during deployment.

Condition	Success (%) ↑	Repairs ↑	Regressions ↓	Tokens/ep. (M) ↓
C0: Fixed skill	63.5	–	–	0.60
C1: Whole-skill reflection	65.0	32	20	0.58
C2: State-localized revision	**68.4**	**51**	**12**	**0.47**

Whole-skill reflection repairs 32 failures but introduces 20 regressions, yielding 12 additional successes relative to C0. State-localized revision repairs 51 failures and introduces 12 regressions, yielding 39 additional successes. C2 achieves 68.4% success at 0.47M deployment tokens per episode, compared with 65.0% at 0.58M for C1. Relative to C0, C2 improves success by 4.9 percentage points and reduces token usage from 0.60M to 0.47M per episode. In this comparison, state-localized revision repairs more failures and introduces fewer regressions than whole-skill reflection.

G COMPARISON WITH LLM SKILLS

G.1 MEMORY PREPARATION AND REUSE

Reusing interaction experience across tasks requires deciding both what to retain and when that knowledge applies. We contrast the memory-based Harness VLA protocol (Zhang et al., 2026c) with StateSkill to clarify the unit of reuse and the reference experience needed for deployment. Table 13 summarizes these differences.

Task references and general memory. Harness VLA combines task-specific execution references with general memory. In its reference-based protocol, exploration on a reference seed produces an audit of the task execution and a primitive-command trace. These records supply task-associated strategies and procedural context, while general memory supplies reusable operating principles and failure lessons. The agent consults relevant memories and available task references, then re-grounds objects and spatial arguments from current observations. This is closed-loop execution with verification and recovery, rather than literal trajectory replay. The comparison here concerns this reference-based protocol; Harness VLA also permits execution when task references are unavailable.

Shared skills for local operations. For LIBERO, StateSkill's construction trajectories are collected through seed-0 exploration guided by Harness VLA's existing records (Appendix A.1). StateSkill organizes experience around local operations and their progress conditions. Trajectory segments with compatible objectives and interaction requirements support shared states, whose knowledge comprises guidance ℓ_q, transition conditions g_q, and action priors p_q. A shared library is reused across evaluation tasks: the agent selects skills for the required local operations, and the active state determines the relevant guidance and action prior. Task instructions and current observations specify the objects and relations for each invocation. At deployment, the library supplies reusable ways to carry out and verify local operations without retrieving task-matched reference trajectories.

Table 13: Experience preparation and reuse in the reference-based Harness VLA protocol and StateSkill. Both use current observations to ground execution.

Dimension	Harness VLA	StateSkill
Unit of reuse	Task-associated execution references and general operating lessons	Shared local-operation skills with explicit progress states
Knowledge access	Relevant memory entries and available task-matched references	Skills selected for local operations; guidance and priors indexed by active state
Progress evaluation	MLLM interprets feedback using instructions and memory	Explicit transition gates interpret evidence to update skill progress
Preparation	Reference-seed exploration yields task references and general memory	Reference-guided exploration yields trajectories for skill construction

Implication for reuse. The same local operation can recur within tasks with different goals and subtask orderings. StateSkill makes its applicability explicit through progress states and interaction conditions, allowing the agent to reuse the skill knowledge wherever that operation is needed. StateSkill performs this reuse through shared state-conditioned knowledge, while reference-guided exploration remains part of offline construction. The execution ablations in Appendix F.2 complement this protocol comparison by examining alternative ways to present and use the same skill knowledge.

G.2 Task Performance and Execution Time

Table 14 extends the comparison in Table 5 with execution time. All methods use Qwen3.5-9B under comparable hardware and serving conditions. Execution time is the mean end-to-end duration over all evaluated episodes, including both successes and failures. It includes perception, agent inference, action execution, and retries.

Table 14: **Extended comparison with LLM skills on LIBERO-PRO.** All methods use Qwen3.5-9B as the high-level embodied agent; w/ FT denotes StateGRPO fine-tuning. Superscripts in the shaded column indicate improvements over Harness VLA: percentage-point gains in success rate and relative reductions (%) in other metrics.

Metric	Harness VLA	XSkill	SkillGen	Trace2Skill	StateSkill-Agent	StateSkill-Agent w/ FT
Success Rate (%) ↑	25.8	23.6	28.9	31.5	68.4	**70.9** $^{\uparrow 45.1}$
Skill Length (K) ↓	169.4	186.7	104.3	231.5	11.0	**11.0** $^{\downarrow 93.5\%}$
Interaction Turns ↓	74.96	76.80	69.50	66.10	23.10	**21.91** $^{\downarrow 70.8\%}$
Token Usage (M) ↓	3.62	3.74	3.05	3.22	0.47	**0.44** $^{\downarrow 87.8\%}$
Execution Time (min) ↓	8.3	8.5	7.6	7.2	1.8	**1.7** $^{\downarrow 79.5\%}$

Execution time. StateSkill-Agent with StateGRPO fine-tuning reduces mean execution time from 8.3 to 1.7 minutes relative to Harness VLA, a 79.5% reduction, while increasing success from 25.8% to 70.9%. The variant without fine-tuning takes 1.8 minutes, so a substantial runtime reduction is already observed before StateGRPO optimization. The reduction accompanies fewer interaction turns (74.96 to 21.91) and lower token usage (3.62M to 0.44M).

Deployment optimization. These measurements use our research implementation, with BF16 vLLM serving and separate perception and execution services (Appendix E), without model quantization or additional task-specific latency optimization. Hardware-supported quantization and further optimization of inference serving and inter-service communication may reduce runtime. The end-to-end benefit depends on the time spent in MLLM inference, perception, and low-level execution. These deployment optimizations have not been evaluated here; their runtime benefits and effects on task success remain to be measured.

G.3 Discussion of Concurrent Work

Concurrent work by Cho et al. (2026) extracts finite-state automata from LLM-agent traces for behavioral analysis, next-step prediction, and failure monitoring. Our work focuses on embodied AI, developing reusable skills for agents that interact with physical environments.

The motivations differ accordingly: their work makes shared behavioral structure in agent logs explicit for prediction and monitoring, whereas ours addresses how embodied experience can provide efficient, stage-appropriate guidance as task progress and physical conditions change.

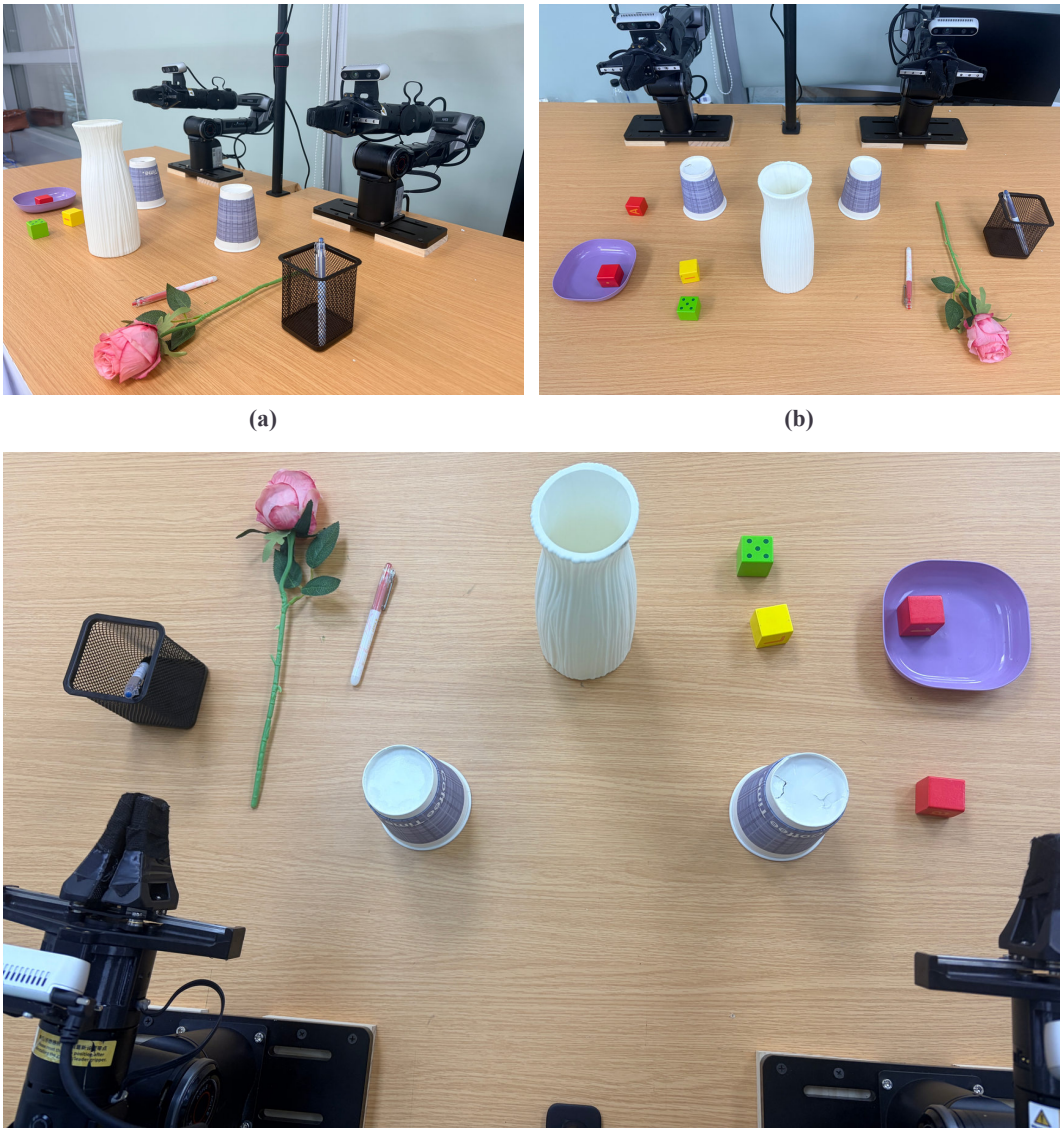
Their standard construction groups trace prefixes by the latest activity symbol. StateSkill abstracts semantic task-progress states from embodied trajectories and equips each state with language guidance, transition gates, and action priors. Transitions depend on observations, execution feedback, and retained facts. For example, issuing a grasp command does not establish success; verification evidence determines whether the skill advances or initiates recovery.

These designs yield different properties and advantages. Their compact automata support efficient trace analysis and state-conditioned workflow context. StateSkill supports transferable skill reuse, stage-relevant guidance, and evidence-based verification and recovery during embodied execution. StateSkill-Evolve further refines skills using successful and failed interactions, while StateGRPO improves the MLLM using task success and state-wise efficiency, enabling skills and agent capabilities to co-evolve.

H SIM-TO-REAL TRANSFER AND LONG-HORIZON EXECUTION

Using simulation-derived StateSkills, the agent completes a long-horizon real-world sequence of five successive subtasks: flower insertion, pen placement, object sorting, cup nesting, and block stacking (Figures 7–11). We preserve the skills' finite-state structure and adapt only environment-dependent components. GPT-5.5 serves as the high-level MLLM, and a trained, frozen $\pi_{0.5}$ policy provides grasping throughout the sequence.

Across subtasks, the agent tracks completed outcomes and coordinates subsequent skill invocations. Within each subtask, StateSkill organizes progress through state-conditioned guidance and verification, from object acquisition to placement assessment. The shared structure is reused with task-specific objects, destinations, and completion criteria, allowing transferred local skills to support a longer execution sequence. Figure 6 shows the dual-arm workspace, equipped with wrist-mounted cameras and a fixed overhead camera.



(a) **(b)** **(c)**

Figure 6: **Real-world task scene.** (a) Oblique view of the dual-arm platform. (b) Frontal view of the workspace. (c) Top view of the task objects.

Each subtask figure shows seven selected moments, with the fixed-camera view above and the active wrist-camera view below. Reading from left to right follows acquisition, holding assessment, transport and alignment, release, and placement assessment, highlighting both execution and verification. The fixed-camera view shows the target object and destination within the workspace, while the wrist-camera view provides local interaction details. Together, these views help relate the illustrated progress stages to changes in the physical scene.

The annotations below each sequence connect the shared progress structure to task-specific guidance and completion criteria. For example, flower insertion and pen placement reuse the same acquisition-and-placement structure, while their guidance concerns different openings and their completion criteria require placement in the corresponding receptacles. Across successive figures, these local operations form the longer five-subtask sequence, illustrating how transferred StateSkills support task composition.

Nearest-time camera frames are paired at unequal intervals. State labels and annotations are semantic summaries rather than frame-aligned runtime traces; intervening observations, corrections, and retries are omitted. The figures complement the quantitative per-task results in Table 3.

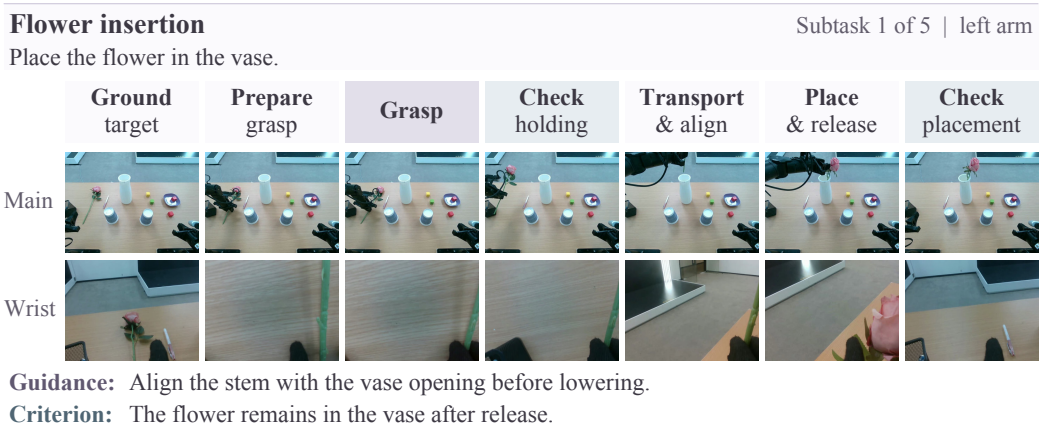


Figure 7: **Subtask 1: Flower insertion.** State-conditioned acquisition and placement of the flower in the vase.

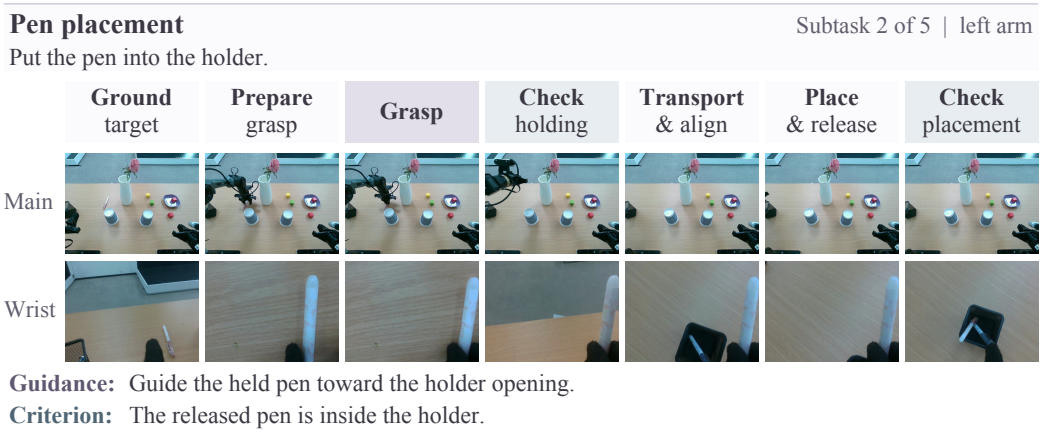


Figure 8: **Subtask 2: Pen placement.** Reuse of the acquisition-and-placement structure to place the pen in its holder.

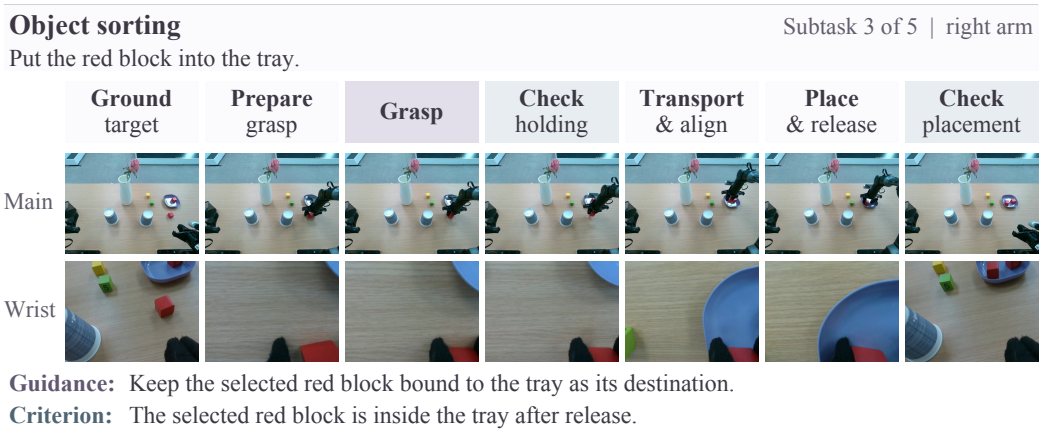


Figure 9: **Subtask 3: Object sorting.** The shared skill structure guides transfer of the selected red block to the tray.

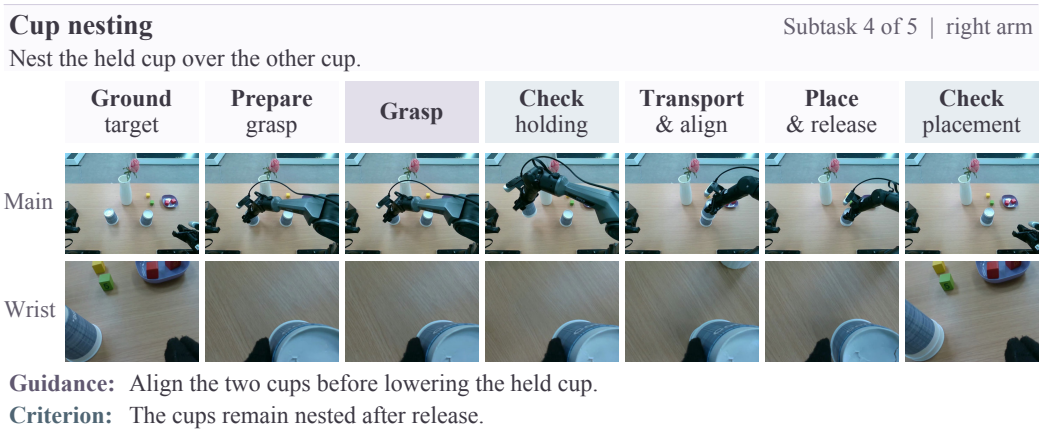


Figure 10: **Subtask 4: Cup nesting.** Task-specific guidance supports cup alignment, with the post-release nesting relation as the completion criterion.

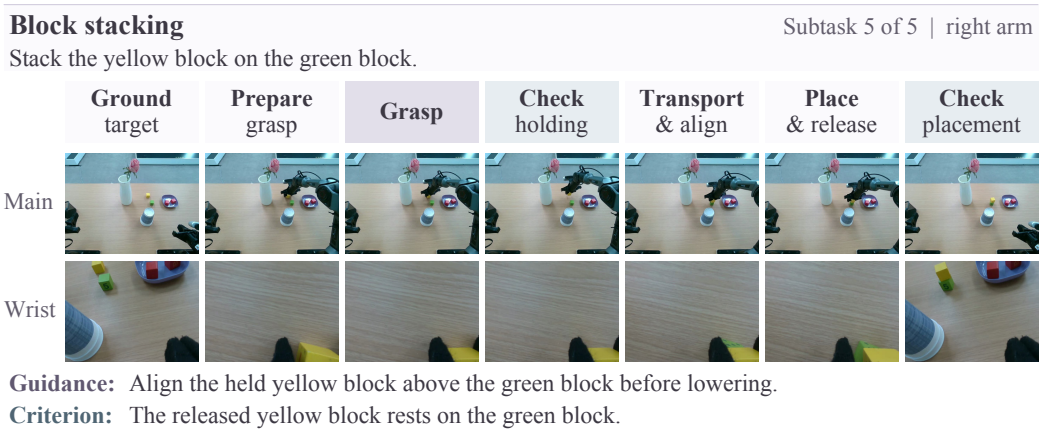


Figure 11: **Subtask 5: Block stacking.** Placement of the yellow block on the green block, with the post-release support relation as the completion criterion.