ROBOHARN-EVO: EVOLVING HIERARCHICAL PHYSICAL KNOWLEDGE FOR SELF-IMPROVING ROBOTIC MANIPULATION

Anonymous authors
Paper under double-blind review

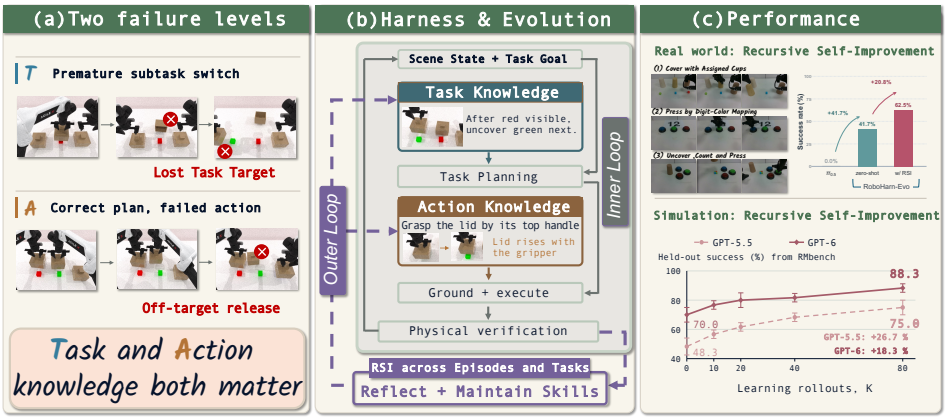


Figure 1: **Self-improving robotic manipulation through hierarchical knowledge evolution.** **(a)** In ordered block manipulation, the robot may switch to the next block before completing the current subtask (top), or select the correct target but release the cover off target (bottom). **(b)** Physical feedback updates Task and Action Knowledge across episodes, with the VLM and executor held fixed. **(c)** Knowledge evolution improves held-out simulation success by 18.3–26.7 percentage points after 80 rollouts. On real-world tasks, simulation-derived knowledge achieves 41.7% mean zero-shot success, which rises to 62.5% after further knowledge updates from real-world interaction.

ABSTRACT

Vision-language models can coordinate long-horizon robot manipulation, yet successful task reasoning still depends on whether local physical interactions produce the intended effects. We study how repeated interaction can improve this capability without updating the base model. We introduce **RoboHarn-Evo**, a dual-loop harness that evolves **Hierarchical Physical Knowledge (HPK)** from physical experience. HPK couples two levels of reusable knowledge: Task Knowledge captures which subtask should be executed and when it is complete, while Action Knowledge captures object-relative geometric strategies and their physical effects. During execution, the agent retrieves knowledge at the corresponding decision level and grounds it in the current scene under the task goal. Across episodes, physical feedback is used to revise historical knowledge, update its applicability, and organize reusable entries for subsequent retrieval. Experiments on RMBench show that HPK improves average success by up to 24.2 percentage points across different agent models. With 80 interaction rollouts, held-out success rises from 48.3% to 75.0% for GPT-5.5 and from 70.0% to 88.3% for GPT-6. RoboHarn-Evo also resolves over 83% of historical knowledge errors while retaining 95.8% of valid knowledge, and transfers zero-shot from RMBench to RoboDojo with gains of 35.0 and 25.0 percentage points. These results demonstrate that physical interaction can be accumulated into reusable knowledge for improving subsequent manipulation.

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.

1 INTRODUCTION

Recent embodied agents increasingly use execution feedback to adapt their behavior during physical interaction (Li et al., 2026c; Chen et al., 2026c). Beyond correcting the current execution, recent systems retain interaction experience as persistent memories (Zhang et al., 2026; Jiang et al., 2026), physical knowledge (Li et al., 2026b), reusable skills (Ju et al., 2026), or evolving runtime components (Ding et al., 2026; Wang et al., 2026b), allowing past interactions to inform future behavior. Recent language agents also refine reusable skills from execution feedback (Yang et al., 2026; Wang et al., 2026a). These advances point toward a broader goal: enabling robots to improve through physical experience without repeatedly retraining their underlying models. A central challenge, however, is how to turn accumulated experience into ***reliable improvement in subsequent behavior***.

Hierarchical skill representations support compositional generalization in robotic policies (Xie et al., 2026a), while language agents evolve hierarchical skill libraries from execution experience (Xia et al., 2026; Wang et al., 2026a). For manipulation, we study how physical feedback should revise knowledge at different decision levels, where a local action effect need not establish task completion. At the *task level*, the agent must decide which subtask to execute and when its objective has been satisfied (Shi et al., 2025); at the *action level*, it must determine the geometry and physical interaction needed to realize that subtask (Huang et al., 2023; 2024). As illustrated in Fig. 1(a), the robot may switch to the next block before confirming completion of the current subtask (top), or select the correct target but release the cover off target (bottom). A successful grasp in the latter example can support the grasping strategy, but does not establish that the cover has been placed at the intended location. This motivates evolving task and action knowledge using their respective completion and effect criteria, while preserving the task goal during reuse.

To address this challenge, we introduce **RoboHarn-Evo**, a harness that evolves **Hierarchical Physical Knowledge (HPK)** while keeping the base vision-language model (VLM) and low-level executor fixed. HPK organizes manipulation experience into *Task Knowledge*, which captures what subtask to execute and its completion condition, and *Action Knowledge*, which captures object-relative geometric strategies and their intended and observed physical effects. As shown in Fig. 1(b), RoboHarn-Evo couples knowledge use and knowledge evolution through an *inner execution loop* and an *outer knowledge-update loop*. The inner execution loop retrieves Task Knowledge for subtask planning and Action Knowledge for action grounding in the current scene. The outer knowledge-update loop uses recorded observations, tool calls, and physical outcomes to extract and consolidate individual pieces of task or action guidance (*knowledge entries*). It checks Task Knowledge against subtask completion and Action Knowledge against intended physical effects. These checks guide content revision and determine eligibility for subsequent retrieval. By linking subtasks to their constituent operations, HPK preserves the relationship between task context and local physical experience for future reuse.

We assess ***self-improvement*** through two complementary criteria with the VLM and executor held fixed. First, *knowledge correction*: accumulated physical evidence should improve persistent knowledge by correcting or deactivating erroneous entries while preserving valid ones. Second, *held-out behavioral improvement*: the resulting knowledge should improve performance on held-out configurations when the knowledge store is frozen and evaluation feedback is excluded from updates. We additionally evaluate *cross-domain knowledge transfer* to examine whether learned knowledge generalizes beyond the environment in which it was acquired.

We evaluate RoboHarn-Evo on RMBench (Chen et al., 2026b) and RoboDojo (Chen et al., 2026a) using different VLM backbones and knowledge configurations, with policy-based methods and Harness VLA as reference systems. On RMBench, full HPK improves average success by up to 24.2 percentage points over the same harness without HPK, and ablations support the complementary benefits of Task Knowledge and Action Knowledge. As shown in Fig. 1(c), after 80 interaction rollouts, mean held-out success increases from 48.3% to 75.0% with GPT-5.5 and from 70.0% to 88.3% with GPT-6. An external GPT-6 audit finds that RoboHarn-Evo repairs or deactivates 10 of the 12 initially incorrect knowledge entries (83.3%) with GPT-5.5 and 11 of 12 (91.7%) with GPT-6. Among the separate set of 24 initially correct entries, both models retain 23 (95.8%). In the additional generalization evaluation, transferring frozen HPK zero-shot from RMBench to RoboDojo improves average success by 35.0 and 25.0 percentage points with GPT-5.5 and GPT-6,

respectively. These results support the use of evolving hierarchical physical knowledge to improve subsequent manipulation under a fixed base model and executor.

Our contributions are threefold. **(1) Experience-driven self-improvement for robotic manipulation.** We formulate robotic self-improvement as the continual refinement and reuse of physical experience under a fixed base VLM and executor, and introduce RoboHarn-Evo to couple online execution with persistent knowledge evolution. **(2) Evidence-driven evolution of HPK.** We link Task Knowledge for subtask selection and completion with Action Knowledge for object-relative geometry and physical effects. Level-specific verification guides knowledge revision and retrieval eligibility, while task-goal consistency constrains the reuse of geometric strategies. **(3) Systematic evaluation of knowledge evolution.** Experiments on RMBench and RoboDojo show complementary benefits from the two knowledge levels, correction of erroneous historical knowledge while retaining valid knowledge, continued improvement on held-out configurations, and zero-shot cross-domain transfer.

2 Related Work

Harnesses for agentic manipulation. Agentic manipulation wraps robot policies in a reasoning-and-execution loop that selects capabilities, monitors outcomes, and replans from feedback. SayCan and Inner Monologue established affordance-grounded skill selection and feedback-driven replanning (Ichter et al., 2022; Huang et al., 2022). Recent systems extend this paradigm to learned robot policies: VLAs-as-Tools and Goal2Skill couple high-level VLM reasoning with VLA execution for long-horizon manipulation (Lei et al., 2026; Liu et al., 2026), while RoboHarness and Harness VLA augment frozen VLAs with memory-guided orchestration and external interventions or analytic primitives (Huang et al., 2026; Zhang et al., 2026). A systematic study further identifies planner–executor orchestration as a key determinant of hierarchical VLA performance (Hu et al., 2026). SkillNet models hierarchical skills within a learned VLA (Xie et al., 2026a). RoboHarn-Evo evolves physical knowledge reused by a fixed planner–executor system.

Self-evolution in robotics. SkillOpt refines external skills from execution feedback (Yang et al., 2026); XSkill consolidates task-level skills and action-level experiences without parameter updates (Jiang et al., 2026). Hierarchical skill evolution is also studied by SkillRL, SkillPyramid, and SPyCE (Xia et al., 2026; Xiong et al., 2026; Zhang & Qiu, 2026). In robotics, REFLECT and AIC support failure correction and geometric adjustment, while PhysMem verifies interaction-derived physical knowledge (Liu et al., 2023; Xiong et al., 2024; Li et al., 2026a). Uni-Skill extends robot skills through a hierarchical demonstration repository (Xie et al., 2026b). EmbodiSkill, SHAPER, Zetta, and ASPIRE evolve procedural skills, runtime components, or executable code (Ju et al., 2026; Wang et al., 2026b; Ding et al., 2026; Lu et al., 2026). Other systems refine policies through autonomous data collection (Li et al., 2026c; Xiao et al., 2026). RoboHarn-Evo focuses on physical knowledge evolution under a fixed VLM and executor: it checks subtask completion separately from local action effects to guide knowledge revision and retrieval eligibility, while preserving task goals during reuse.

3 Problem Formulation

We study experience-driven self-improvement in robotic manipulation, where physical experience continually refines persistent knowledge under a fixed VLM and executor. Let $\mathcal{K}_i$ denote the knowledge available before episode i. For instruction l, episode i is recorded at operation boundaries as $\tau_i = (l;\, o_0, a_0, o_1, a_1, \ldots, o_{T_i})$, where o_t is the observation and a_t the robot-action sequence for operation t. With runtime memory s_t, retrieved knowledge conditions the VLM's tool-use decisions:

$$c_t \sim p_\theta(\cdot \mid s_t, l, \mathcal{K}_i)\,, \qquad a_t = \mathrm{Exec}(s_t, c_t), \tag{1}$$

where c_t is a parameterized tool call and the executor grounds it in the current scene with execution feedback.

Across episodes, a knowledge-update rule $\mathcal{U}$ revises the store using observed task progress and physical action effects:

$$\mathcal{K}_{i+1} = \mathcal{U}(\mathcal{K}_i, \mathrm{View}(\tau_i))\,, \tag{2}$$

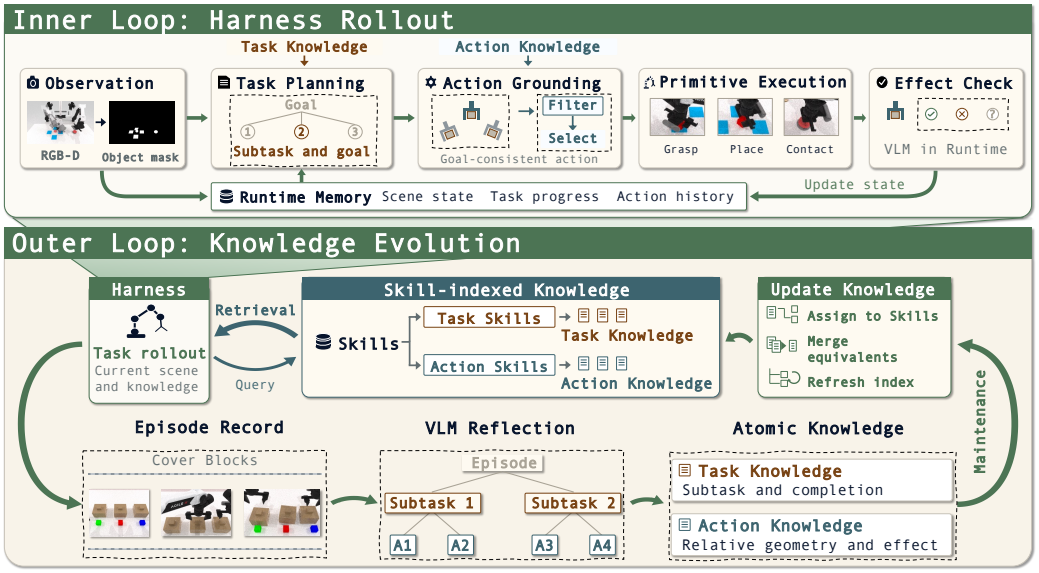


Figure 2: **Overview of RoboHarn-Evo. Top:** The inner loop executes manipulation operations and records their outcomes. **Bottom:** The outer loop organizes this experience into Task and Action Knowledge for subsequent decisions. Concrete motions are computed from the current scene.

where $\mathrm{View}(\tau_i)$ contains observations, tool calls, and execution feedback. Each episode is executed with $\mathcal{K}_i$ and updates the knowledge used in subsequent episodes. We seek an update rule that yields higher task success as interaction accumulates, evaluated using frozen knowledge snapshots on fixed held-out configurations without feeding evaluation outcomes back into the store.

4 Method

RoboHarn-Evo connects a fixed VLM with robot execution through two loops. The inner loop uses *Hierarchical Physical Knowledge* (HPK) to guide manipulation (Sec. 4.1); the outer loop extracts and maintains this knowledge from observed physical outcomes (Sec. 4.2 and 4.3).

4.1 Agentic Manipulation Harness

As shown in the top of Fig. 2, the inner loop maintains runtime memory s_t from RGB-D observations, object segmentation, task progress, and action history. Let $\mathcal{K}_i = (\mathcal{K}_i^{\mathrm{T}}, \mathcal{K}_i^{\mathrm{A}})$ denote the Task and Action Knowledge available before episode i. The VLM first chooses a task strategy u_t and then an action strategy z_t, which together condition its tool call:

$$u_t \sim p_\theta\big(\cdot \mid s_t, l, \mathcal{K}_i^{\mathrm{T}}\big)\,, z_t \sim p_\theta\big(\cdot \mid s_t, u_t, \mathcal{K}_i^{\mathrm{A}}\big)\,, c_t \sim p_\theta(\cdot \mid s_t, u_t, z_t)\,. \tag{3}$$

Here, u_t specifies the current subtask, target relation, and completion condition, while z_t specifies object-relative geometry for an operation $\alpha_t \in \{\texttt{grasp}, \texttt{place}, \texttt{contact}\}$. The harness grounds c_t in the current scene, and the robot executor produces a_t through Eq. (1).

After each operation, the effect check records the observed physical outcome and updates s_t. Local action effects and subtask completion are evaluated separately: an operation may succeed physically without completing the enclosing subtask. Verified completion advances task planning; otherwise the updated runtime state conditions the next decision. The resulting observations, tool calls, and effect checks form the episode record used by the outer loop.

4.2 Hierarchical Physical Knowledge

From trajectories to atomic knowledge. The bottom of Fig. 2 shows how an episode is converted into HPK. A VLM reflector R_ϕ maps the observable trajectory into an ordered hierarchy of Task and

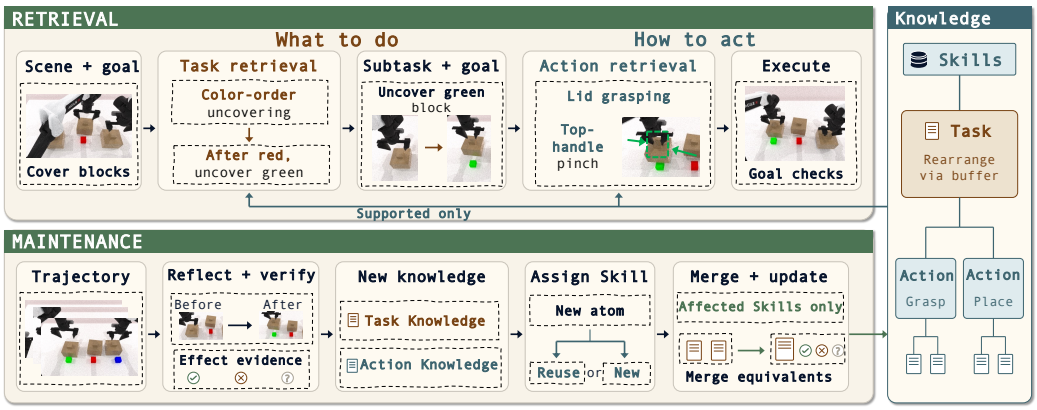


Figure 3: **Skill-routed retrieval and evidence-driven maintenance. Top:** Task retrieval determines *what to do*; conditioned on that subtask, Action retrieval determines *how to act* using supported atomic knowledge. **Bottom:** Before–after execution evidence is reflected into new Task and Action Knowledge, assigned to Skills, and used to consolidate and update only the affected knowledge groups.

Action Knowledge:

$$\mathcal{P}_i = R_\phi(\mathrm{View}(\tau_i)) = \left(k^{\mathrm{T}}_{i,j}, \left(k^{\mathrm{A}}_{i,j,r}\right)_{r=1}^{m_{i,j}}\right)_{j=1}^{m_i}. \tag{4}$$

Each Task entry $k^{\mathrm{T}}_{i,j}$ captures when a subtask applies, what it should achieve, and when it is complete. Its associated Action entries $k^{\mathrm{A}}_{i,j,r}$ capture the operation type, object-relative geometric strategy, and observed physical effect. Linking each subtask to the operations that realize it preserves task context while keeping local action success distinct from task completion.

From atomic knowledge to skill-indexed knowledge. The extracted atomic entries are integrated into the persistent stores $\mathcal{K}^{\mathrm{T}}_{i+1}$ and $\mathcal{K}^{\mathrm{A}}_{i+1}$ and organized by a Skill index $\mathcal{F}_{i+1} = (\mathcal{F}^{\mathrm{T}}_{i+1}, \mathcal{F}^{\mathrm{A}}_{i+1})$, as illustrated in the bottom of Fig. 2. Each Skill is a retrieval index that groups contextually related entries and summarizes when they are useful; execution is conditioned on the selected atomic Task or Action Knowledge. The task–action schema and execution primitives remain fixed; physical experience evolves knowledge content, applicability, and retrieval eligibility. Evidence representation and update details are provided in Appendix C.3.

4.3 Knowledge Retrieval and Maintenance

Retrieval. The top of Fig. 8 mirrors the two decisions in Eq. 3. Given the scene, goal, and runtime memory, Task retrieval selects applicable knowledge for u_t, answering *what to do* next. Conditioned on that subtask and primitive α_t, Action retrieval selects knowledge for z_t, answering *how to act*. Skill summaries route each query to relevant groups, after which the VLM selects supported atomic entries whose conditions match the current context. The retrieved task strategy constrains action grounding so that the selected geometry remains consistent with the intended object relation and subtask goal.

Maintenance. The bottom of Fig. 8 shows how each rollout updates the persistent knowledge used by later episodes:

$$(\mathcal{K}_{i+1}, \mathcal{F}_{i+1}) = \mathrm{Maintain}(\mathcal{K}_i, \mathcal{F}_i, \mathcal{P}_i). \tag{5}$$

Verification follows the HPK hierarchy. Task Knowledge is assessed against subtask completion, whereas Action Knowledge is assessed against its intended physical effect. Thus, a successful grasp can support an Action entry even when the enclosing placement subtask remains incomplete.

For knowledge that is applicable and faithfully executed, the observed outcome provides `support`, `oppose`, or `unverified` evidence. Maintenance uses this evidence to merge equivalent entries and revise conditions or strategies challenged by subsequent interaction. Revised content becomes

Table 1: **Task success on RMBench.** We report task success rates (%) on six RMBench tasks, with Overall denoting the six-task mean. X-VLA and Mem-0 results are reported from the Mem-0 benchmark evaluation; all remaining results are evaluated on 20 held-out episodes per task. Within each model, w/o HPK and Full HPK share the same executor. HarnessVLA is tested with GPT-5.5.

Task	$\pi_{0.5}$	X-VLA	Mem-0	HarnessVLA	Qwen3.8-27B		GPT-5.5		GPT-6	
					w/o HPK	**Full HPK**	w/o HPK	**Full HPK**	w/o HPK	**Full HPK**
Rearrange Blocks	13%	13%	89%	30%	15%	35%	50%	80%	75%	**90%**
Swap Blocks	24%	16%	67%	35%	10%	25%	45%	70%	70%	**85%**
Press Button	0%	0%	0%	75%	35%	60%	95%	**100%**	**100%**	**100%**
Swap T	15%	3%	14%	35%	20%	40%	55%	90%	45%	**95%**
Put Back Block	11%	18%	**90%**	25%	20%	50%	50%	75%	75%	**90%**
Cover Blocks	0%	2%	68%	50%	15%	40%	55%	80%	70%	**90%**
Overall	10.5%	8.7%	54.7%	41.7%	19.2%	41.7%	58.3%	82.5%	72.5%	**91.7%**

retrievable only when supported without applicable opposition, and only the affected Skill summaries are refreshed. Detailed evidence attribution, versioning, and consolidation procedures are given in Appendix C.3.

5 EXPERIMENTS

In this section, we aim to address the following four research questions: **Q1:** Does *Hierarchical Physical Knowledge* improve manipulation, and how do *Task* and *Action Knowledge* contribute?**Q2:** Can *RoboHarn-Evo* resolve historical errors while preserving valid knowledge? **Q3:** Does continued interaction improve performance on held-out configurations? **Q4:** Can source knowledge support execution across benchmarks and sim2real environments?

Experimental setup. We evaluate *RoboHarn-Evo* on six RMBench (Chen et al., 2026b) tasks and assess cross-benchmark knowledge transfer on RoboDojo (Chen et al., 2026a). The main comparison uses Qwen3.8-27B, GPT-5.5, and GPT-6, with $\pi_{0.5}$ (Intelligence et al., 2025) and Harness VLA (Zhang et al., 2026) as reference methods. Detailed tasks, comparisons, and protocols appear in Appendix A.1.

5.1 EFFECTIVENESS OF HIERARCHICAL PHYSICAL KNOWLEDGE

Comparative Performance. As shown in Tab. 1, averaged across the three VLM backbones, RoboHarn-Evo exceeds $\pi_{0.5}$ and Harness VLA by 61.1 and 30.3 percentage points in overall task success, respectively. **Gains across VLM Backbones.** Tab. 1 compares w/o HPK and w/ Full HPK across six tasks. Full HPK raises mean success from 19.2% to 41.7% with Qwen3.8-27B, from 58.3% to 82.5% with GPT-5.5, and from 72.5% to 91.7% with GPT-6. Gains span task planning and action grounding: with GPT-5.5, success increases from 50.0% to 80.0% on Rearrange Blocks and from 55.0% to 90.0% on Swap T, respectively.

Complementarity of Knowledge Levels. Tab. 2 evaluates the individual and combined contributions of Task and Action Knowledge using the same source experience. With Qwen3.8-27B, Task-only and Action-only achieve 27.5% and 28.3% overall success, respectively, compared with 19.2% without HPK. With GPT-5.5, they achieve 70.8% and 66.7%, compared with 58.3% without HPK. Combining both levels raises success to 41.7% and 82.5%, respectively, outperforming either level alone and supporting their complementary contributions. Full HPK also exceeds Flat Reflection, which achieves 24.2% and 64.2%, indicating the benefit of organizing and applying the same experience at distinct decision levels.

Table 2: **Component ablation on RMBench.** Δ is the overall gain over w/o HPK in percentage points.

Method	Qwen3.8-27B		GPT-5.5	
	Overall (%)	Δ	Overall (%)	Δ
w/o HPK	19.2	0.0	58.3	0.0
Flat Reflection	24.2	+5.0	64.2	+5.8
Task-only	27.5	+8.3	70.8	+12.5
Action-only	28.3	+9.2	66.7	+8.3
RoboHarn-Evo	**41.7**	**+22.5**	**82.5**	**+24.2**

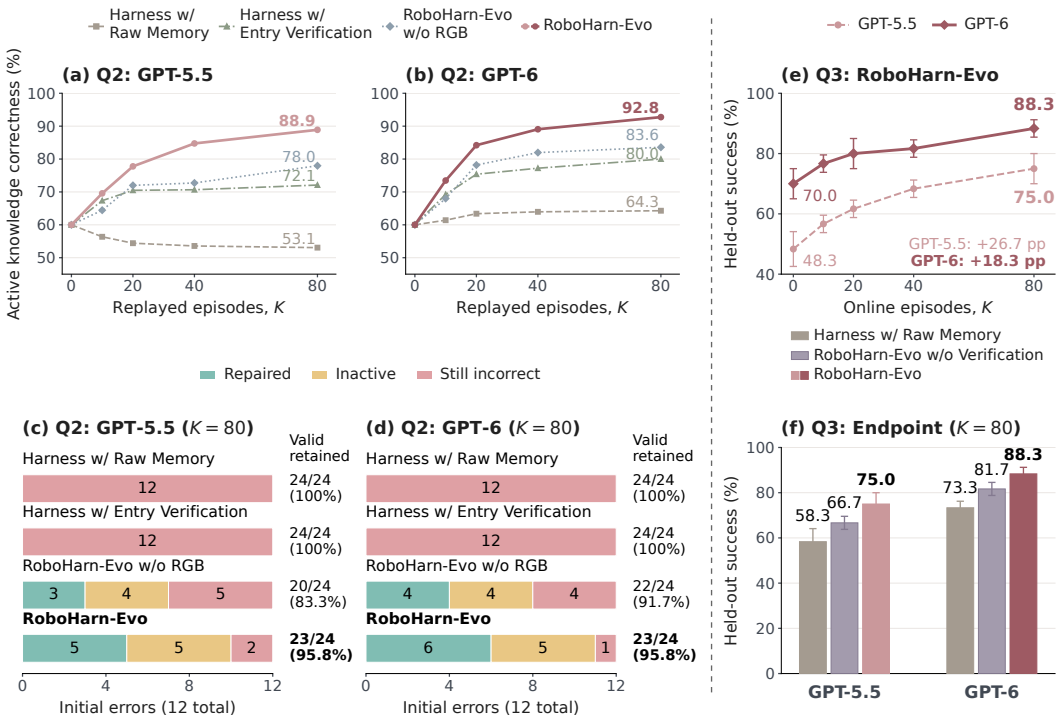


Figure 4: **Knowledge maintenance and held-out manipulation performance.** **(a,b)** Active-knowledge correctness under shared-evidence replay. **(c,d)** Outcomes of the 12 initial errors and retention of the 24 initially correct entries at $K = 80$. **(e)** Held-out success across learning checkpoints. **(f)** Endpoint comparisons using the same source experience at $K = 80$. K counts interaction rollouts. Error bars in (e,f) indicate standard deviations over three learning histories. Q2 replays previously collected interaction records to evaluate knowledge maintenance; Q3 collects experience through online interaction to evaluate subsequent task performance.

5.2 Knowledge Maintenance

We evaluate knowledge maintenance on *Swap Blocks* and *Put Back Block* from RMBench. For each model, four variants start from the same historical pool of 40 entries (24 correct, 12 incorrect, and 4 unverified) and process identical evidence from one predesignated interaction history. *Harness w/ Raw Memory* only appends entries; *Harness w/ Entry Verification* checks only additions; and *RoboHarn-Evo w/o RGB* reviews historical knowledge using textual evidence. Full *RoboHarn-Evo* additionally incorporates visual evidence. Following the LLM-as-a-judge paradigm Li et al. (2025), knowledge correctness is assessed by an external GPT-6 judge, separate from RoboHarn-Evo's execution and knowledge maintenance components, using recorded observations and execution evidence.

Fig. 4(a,b) tracks the overall quality of the active knowledge pool, measured as the proportion of active entries judged correct. At $K = 80$, RoboHarn-Evo reaches 88.9% with GPT-5.5 and 92.8% with GPT-6. To distinguish historical error correction from changes in pool composition, Fig. 4(c,d) track the fixed cohorts of initially incorrect and correct entries. At the same checkpoint, RoboHarn-Evo repairs 41.7% of the initial errors with GPT-5.5 and 50.0% with GPT-6, while deactivating another 41.7% with either model. Both retain 95.8% of initially correct knowledge. Raw Memory and Entry Verification leave all initial errors unresolved, while removing RGB reduces both the repair rate and valid-knowledge retention.

5.3 Self-Improvement through Continued Interactions

We evaluate whether accumulated experience improves manipulation on held-out configurations of *Swap Blocks* and *Put Back Block*. For each model, RoboHarn-Evo completes three independent

Table 3: **Zero-shot knowledge transfer from RMBench to RoboDojo.** Score and success rate (SR) are reported as percentages, with 10 target episodes per task. Δ is the difference in Average between w/ Full HPK and w/o HPK, in percentage points.

Task	GPT-5.5				GPT-6			
	w/o HPK		**w/ Full HPK**		w/o HPK		**w/ Full HPK**	
	Score	SR	Score	SR	Score	SR	Score	SR
Cover Blocks	37.0	30.0	**75.0**	**70.0**	57.0	50.0	**93.0**	**90.0**
Press by Number	50.0	50.0	**80.0**	**80.0**	80.0	80.0	**90.0**	**90.0**
Average	43.5	40.0	**77.5**	**75.0**	68.5	65.0	**91.5**	**90.0**
Δ (pp)		–	**+34.0**	**+35.0**		–	**+23.0**	**+25.0**

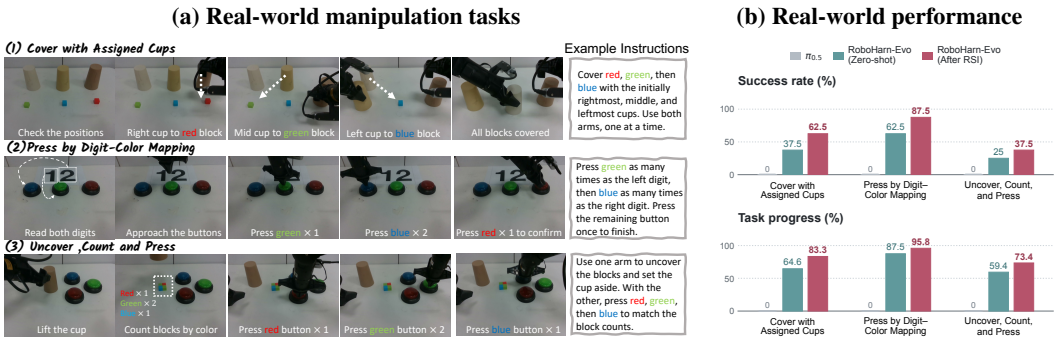


Figure 5: **Real-world transfer and self-improvement.** **(a)** Task sequences and example instructions. **(b)** Success rate **(top)** and normalized task progress **(bottom)** for $\pi_{0.5}$ and RoboHarn-Evo before and after RSI. Zero-shot uses simulation-derived HPK; After RSI incorporates additional real-world interaction. RoboHarn-Evo results use eight scenes per task, with full task completion counted as success.

learning histories of 80 interaction rollouts, each starting from an empty knowledge store. Model parameters and execution tools remain fixed throughout. At predefined checkpoints, knowledge snapshots are frozen and evaluated on the same 20 held-out configurations. Evaluation feedback is excluded from knowledge updates.

For each learning history, we replay its 80 source trajectories to construct two endpoint control stores. *Harness w/ Raw Memory* retains raw experience, while *RoboHarn-Evo w/o Verification* constructs knowledge without physical-evidence verification. Both controls are evaluated only at $K = 80$, using the same held-out configurations and evaluation protocol as RoboHarn-Evo. Runtime execution checks remain enabled in all variants.

Fig. 4 (e) shows that mean held-out success increases from 48.3% to 75.0% with GPT-5.5 and from 70.0% to 88.3% with GPT-6 as interaction accumulates. Under shared source experience at $K = 80$, Fig. 4 (f) shows that RoboHarn-Evo exceeds Raw Memory by 16.7 and 15.0 percentage points for GPT-5.5 and GPT-6, respectively. It also exceeds w/o Verification by 8.3 and 6.7 percentage points. These endpoint comparisons support the value of constructing and verifying reusable knowledge from the same interaction evidence.

5.4 Knowledge Transfer across Environments

Cross-benchmark transfer. We transfer HPK acquired on RMBench to *Cover Blocks* and *Press by Number* in RoboDojo. The source knowledge remains frozen throughout evaluation, while tool arguments and motion targets are determined from current observations. The *w/o HPK* and *w/ Full HPK* conditions share the target agent, executor, initial configurations, and action budget.

Tab. 3 shows that transferred HPK raises mean success from 40.0% to 75.0% with GPT-5.5 and from 65.0% to 90.0% with GPT-6. Mean task-progress scores also increase from 43.5 to 77.5 and from

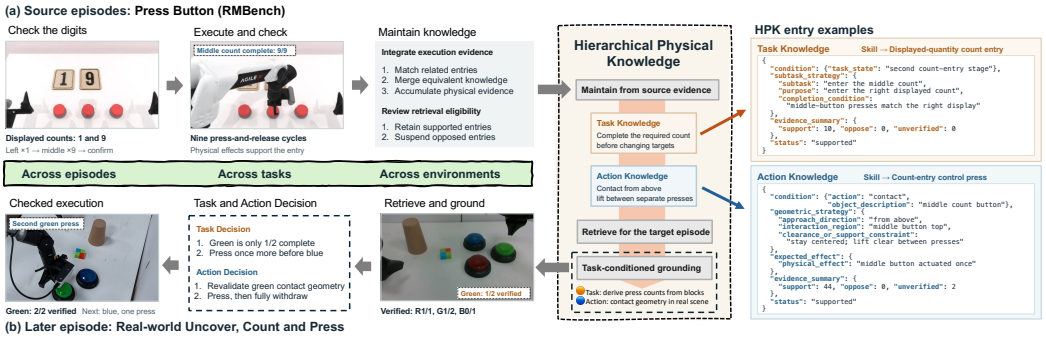


Figure 6: **Task-conditioned knowledge reuse from simulation to the real world.** **(a)** Simulated pressing experience provides physical evidence for maintaining HPK. **(b)** A later real-world episode uses block-derived counts and scene-specific contact geometry to complete the second green press before switching to blue. The right panels show condensed Task and Action Knowledge entries with accumulated source evidence. Arrows illustrate the maintenance and reuse workflow.

68.5 to 91.5, respectively. Both models improve on both target tasks without updating knowledge in the target environment.

Self-improvement across real-world scenes. We deploy RoboHarn-Evo on an AgileX Cobot Magic platform for the three tasks in Fig. 5(a). We compare $\pi_{0.5}$ with RoboHarn-Evo using simulation-derived HPK *(Zero-shot)* and after further knowledge updates from real-world interaction *(After RSI)*. The VLM and executor remain fixed during these updates. Across eight scenes per task, Fig. 5(b) shows that RSI raises mean success from 41.7% to 62.5% and mean normalized task progress from 70.5% to 84.2%, averaged across the three tasks. Both metrics improve on every task. These results support the use of simulation-derived knowledge in real-world manipulation and its further improvement through physical interaction. Hardware, task definitions, and scoring rules are detailed in Appendix A.2.

Case study: task-conditioned knowledge reuse. Fig. 6 illustrates knowledge reuse from simulated *Press Button* to real-world *Uncover, Count and Press*. Source HPK captures guidance for completing the required count before switching targets and performing distinct press-and-release cycles. In the target task, the instruction and observed block counts determine the button identities and required counts, while contact geometry is recomputed from the current scene. With only one of two required green presses verified, the robot selects another green press, revalidates its contact geometry, and verifies the second press before selecting blue. HPK remains fixed during this target episode.

6 Conclusion

We presented RoboHarn-Evo, a harness for experience-driven self-improvement in robotic manipulation with a fixed VLM and low-level executor. Its Hierarchical Physical Knowledge links subtask decisions to object-relative action strategies, allowing past experience to guide planning and execution under a shared task goal. Physical feedback then supports the revision and reuse of this knowledge across episodes. Experiments show that Task and Action Knowledge contribute complementary benefits, while evidence-driven maintenance repairs or deactivates historical errors and preserves valid knowledge. Evaluation with frozen knowledge snapshots demonstrates improved held-out performance as interaction accumulates. Cross-benchmark and real-robot results further show that simulation-derived knowledge supports transfer and can be refined through subsequent physical interaction. These findings support hierarchical knowledge evolution as a mechanism for turning accumulated experience into improved manipulation without updating model parameters.

AI USE STATEMENT

In this work, we used generative AI tools to assist with checking parts of the system implementation and evaluation pipelines, refining experimental methodology and evaluation protocols, and polishing the manuscript and figures. All AI-assisted outputs were reviewed and verified by the authors against the underlying code, experiments, results, and source literature. The authors made all final decisions regarding the methodology, experimental design, interpretation of results, and scientific claims, and take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the RoboHarn-Evo framework, including its knowledge representation, retrieval, maintenance, and execution procedures, in the main text and Appendix C. Appendix A.1 specifies the simulation and real-world experimental setups, model configurations, training settings, and evaluation protocols. Appendix D provides detailed metric definitions and experimental protocols, while Appendix E documents the runtime skill library and its interfaces. These details are provided to facilitate reproduction of the reported experiments and implementation of the proposed system.

REFERENCES

Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Haoran Lu, Weijie Wan, Baijun Chen, Songling Liu, Haowen Yan, Honghao Su, Zhiyang Dou, Kaixuan Wang, Dandan Zhang, Yunze Liu, Yan Qin, Qiwei Liang, Qiwei Wu, Zijian Lin, Wenwei Lin, Yuran Wang, Minghua He, Tianshu Wu, Ruihai Wu, Jingquan Zhou, Kai-Chong Lei, Haibao Yu, Yuanfeng Ji, Weiyang Jin, Guanyu Lin, Xiaofan Li, Qi Xiong, Renjing Xu, Zhongyu Li, Wenhao Chai, Enze Xie, Ziwei Wang, Yao Mu, Hao Dong, Wojciech Matusik, Mingyu Ding, Wenbo Ding, Ping Luo, and Masayoshi Tomizuka. Robodojo: A unified sim-and-real benchmark for comprehensive evaluation of generalist robot manipulation policies. *CoRR*, abs/2607.04434, 2026a. doi: 10.48550/ARXIV.2607.04434. URL https://doi.org/10.48550/arXiv.2607.04434.

Tianxing Chen, Yuran Wang, Mingleyang Li, Yan Qin, Hao Shi, Zixuan Li, Yifan Hu, Yingsheng Zhang, Kaixuan Wang, Yue Chen, Hongcheng Wang, Renjing Xu, Ruihai Wu, Yao Mu, Yaodong Yang, Hao Dong, and Ping Luo. Rmbench: Memory-dependent robotic manipulation benchmark with insights into policy design. *CoRR*, abs/2603.01229, 2026b. doi: 10.48550/ARXIV.2603.01229. URL https://doi.org/10.48550/arXiv.2603.01229.

Yanzhe Chen, Zechen Bai, Zhijun Cao, Wenzheng Zeng, Kevin Qinghong Lin, Yiqi Lin, Guoqiang Liang, Kevin Yuchen Ma, Qiming Huang, and Mike Zheng Shou. Show-harness: Just a vlm agent can play robots, 2026c. URL https://arxiv.org/abs/2609.10522.

Xin Ding, Liang Mi, Mingzhe Huang, Zixuan Wang, Chao Zhang, Zixu Hao, Fu Chen, Xiangyu Li, Yikai Zheng, Yaoyu Guo, Weijun Wang, Kun Li, Hao Wu, Yunxin Liu, and Ting Cao. Zetta ζ: An efficient closed-loop embodied harness for self-evolving physical intelligence. *CoRR*, abs/2608.16590, 2026. doi: 10.48550/ARXIV.2608.16590. URL https://doi.org/10.48550/arXiv.2608.16590.

Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile ALOHA: learning bimanual mobile manipulation using low-cost whole-body teleoperation. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Conference on Robot Learning, 6-9 November 2024, Munich, Germany*, volume 270 of *Proceedings of Machine Learning Research*, pp. 4066–4083. PMLR, 2024. URL https://proceedings.mlr.press/v270/fu25b.html.

Jiaheng Hu, Mohit Shridhar, Caden Lu, Dhruv Shah, Hao-Tien Lewis Chiang, Jie Tan, and Annie Xie. What matters in orchestrating robot policies: A systematic study of hierarchical VLA agents. *CoRR*, abs/2606.10267, 2026. doi: 10.48550/ARXIV.2606.10267. URL https://doi.org/10.48550/arXiv.2606.10267.

Jinbang Huang, Yuanzhao Hu, Zhiyuan Li, Ran Qi, Yixin Xiao, Zhanguang Zhang, Mark Coates, Tongtong Cao, and Yingxue Zhang. Roboharness: Memory-driven orchestration of heterogeneous

robot policies for long-horizon planning. *CoRR*, abs/2607.18060, 2026. doi: 10.48550/ARXIV.2607.18060. URL https://doi.org/10.48550/arXiv.2607.18060.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In Karen Liu, Dana Kulic, and Jeffrey Ichnowski (eds.), *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 2022. URL https://proceedings.mlr.press/v205/huang23c.html.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, volume 229 of *Proceedings of Machine Learning Research*, pp. 540–562. PMLR, 2023. URL https://proceedings.mlr.press/v229/huang23b.html.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Conference on Robot Learning, 6-9 November 2024, Munich, Germany*, volume 270 of *Proceedings of Machine Learning Research*, pp. 4573–4602. PMLR, 2024. URL https://proceedings.mlr.press/v270/huang25g.html.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as I can, not as I say: Grounding language in robotic affordances. In Karen Liu, Dana Kulic, and Jeffrey Ichnowski (eds.), *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, volume 205 of *Proceedings of Machine Learning Research*, pp. 287–318. PMLR, 2022. URL https://proceedings.mlr.press/v205/ichter23a.html.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael 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. *CoRR*, abs/2504.16054, 2025. doi: 10.48550/ARXIV.2504.16054. URL https://doi.org/10.48550/arXiv.2504.16054.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. Xskill: Continual learning from experience and skills in multimodal agents. *CoRR*, abs/2603.12056, 2026. doi: 10.48550/ARXIV.2603.12056. URL https://doi.org/10.48550/arXiv.2603.12056.

Ruofei Ju, Xinrui Wang, Xin Ding, Yifan Yang, Hao Wu, Shiqi Jiang, Qianxi Zhang, Hao Wen, Xiangyu Li, Weijun Wang, Kun Li, Yunxin Liu, Haipeng Dai, Wei Wang, and Ting Cao. Embodiskill: Skill-aware reflection for self-evolving embodied agents. *CoRR*, abs/2605.10332, 2026. doi: 10.48550/ARXIV.2605.10332. URL https://doi.org/10.48550/arXiv.2605.10332.

Zixing Lei, Changxing Liu, Yichen Xiong, Minhao Xiong, Yuanzhuo Ding, Zhipeng Zhang, Weixin Li, and Siheng Chen. Towards long-horizon embodied agents with tool-aligned vision-language-action models. *CoRR*, abs/2605.13119, 2026. doi: 10.48550/ARXIV.2605.13119. URL https://doi.org/10.48550/arXiv.2605.13119.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. From generation to judgment: Opportunities and challenges of llm-as-a-judge. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 2757–2791. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.138. URL https://doi.org/10.18653/v1/2025.emnlp-main.138.

Haoyang Li, Yang You, Hao Su, and Leonidas Guibas. PhysMem: Scaling test-time memory for embodied physical reasoning. *arXiv preprint arXiv:2602.20323*, 2026a. URL https://arxiv.org/abs/2602.20323v6.

Haoyang Li, Yang You, Hao Su, and Leonidas J. Guibas. Learning physical principles from interaction: Self-evolving planning via test-time memory. *CoRR*, abs/2602.20323, 2026b. doi: 10.48550/ARXIV.2602.20323. URL https://doi.org/10.48550/arXiv.2602.20323.

Ruiying Li, Yunlang Zhou, YuYao Zhu, Kylin Chen, Jingyuan Wang, Sukai Wang, Kongtao Hu, Minhui Yu, Bowen Jiang, Zhan Su, Jiayao Ma, Xin He, Yongjian Shen, Yang Yang, Guanghui Ren, Maoqing Yao, Wenhao Wang, and Yao Mu. Roboclaw: An agentic framework for scalable long-horizon robotic tasks. *CoRR*, abs/2603.11558, 2026c. doi: 10.48550/ARXIV.2603.11558. URL https://doi.org/10.48550/arXiv.2603.11558.

Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: summarizing robot experiences for failure explanation and correction. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, volume 229 of *Proceedings of Machine Learning Research*, pp. 3468–3484. PMLR, 2023. URL https://proceedings.mlr.press/v229/liu23g.html.

Zhen Liu, Xinyu Ning, Zhe Hu, XinXin Xie, Weize Li, Zhipeng Tang, Chongyu Wang, Zejun Yang, Hanlin Wang, Yitong Liu, and Zhongzhu Pu. Goal2skill: Long-horizon manipulation with adaptive planning and reflection. *CoRR*, abs/2604.13942, 2026. doi: 10.48550/ARXIV.2604.13942. URL https://doi.org/10.48550/arXiv.2604.13942.

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. *CoRR*, abs/2607.00272, 2026. doi: 10.48550/ARXIV.2607.00272. URL https://doi.org/10.48550/arXiv.2607.00272.

Lucy Xiaoyang Shi, Brian Ichter, Michael Robert Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/shi25d.html.

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. *CoRR*, abs/2604.04804, 2026a. doi: 10.48550/ARXIV.2604.04804. URL https://doi.org/10.48550/arXiv.2604.04804.

Peidong Wang, Zhiming Ma, Ying Chang, Xufang Luo, Xiaocui Yang, Shi Feng, Yuqing Yang, and Dongsheng Li. Self-evolving embodied agents via skill-harness evolution. *CoRR*, abs/2608.11350, 2026b. doi: 10.48550/ARXIV.2608.11350. URL https://doi.org/10.48550/arXiv.2608.11350.

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. *CoRR*, abs/2602.08234, 2026. doi: 10.48550/ARXIV.2602.08234. URL https://doi.org/10.48550/arXiv.2602.08234.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian "Max" Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi "Jim" Fan, Yuke Zhu, and Guanya Shi. ENPIRE: agentic robot policy self-improvement in the real world. *CoRR*, abs/2606.19980, 2026. doi: 10.48550/ARXIV.2606.19980. URL https://doi.org/10.48550/arXiv.2606.19980.

Senwei Xie, Yuntian Zhang, Zhenzhou Tan, Ruiping Wang, Pengwei Wang, Shanghang Zhang, and Xilin Chen. Skillnet: Hierarchical skill modeling for compositional generalization in vision-language action models. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026a.

Senwei Xie, Yuntian Zhang, Ruiping Wang, and Xilin Chen. Uni-skill: Building self-evolving skill repository for generalizable robotic manipulation. *CoRR*, abs/2603.02623, 2026b. doi: 10.48550/ARXIV.2603.02623. URL https://doi.org/10.48550/arXiv.2603.02623.

Chuyan Xiong, Chengyu Shen, Xiaoqi Li, Kaichen Zhou, Jiaming Liu, Ruiping Wang, and Hao Dong. Autonomous interactive correction MLLM for robust robotic manipulation. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Conference on Robot Learning, 6-9 November 2024, Munich, Germany*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3139–3156. PMLR, 2024. URL https://proceedings.mlr.press/v270/xiong25a.html.

Yuan Xiong, Ziqi Miao, Qian Chen, Lijun Li, Yequan Wang, Shizhu He, Jun Zhao, and Kang Liu. Skillpyramid: A hierarchical skill consolidation framework for self-evolving agents. *CoRR*, abs/2606.03692, 2026. doi: 10.48550/ARXIV.2606.03692. URL https://doi.org/10.48550/arXiv.2606.03692.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills. *CoRR*, abs/2605.23904, 2026. doi: 10.48550/ARXIV.2605.23904. URL https://doi.org/10.48550/arXiv.2605.23904.

Ru Zhang and Weijie Qiu. Spyce: Skill-policy co-evolution for multimodal agents. *CoRR*, abs/2607.13854, 2026. doi: 10.48550/ARXIV.2607.13854. URL https://doi.org/10.48550/arXiv.2607.13854.

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. *CoRR*, abs/2607.08448, 2026. doi: 10.48550/ARXIV.2607.08448. URL https://doi.org/10.48550/arXiv.2607.08448.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu (eds.), *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023. doi: 10.15607/RSS.2023.XIX.016. URL https://doi.org/10.15607/RSS.2023.XIX.016.

A Experimental Setup

A.1 Simulation Experimental setup

Benchmarks and tasks. RMBench (Chen et al., 2026b) is our main platform. We evaluate six tasks covering task planning, action grounding, and their interaction, with 20 held-out episodes per task. Knowledge correction and self-evolution use *Swap Blocks* and *Put Back Block*. For transfer, we evaluate RMBench-derived knowledge on RoboDojo (Chen et al., 2026a) using *Cover Blocks* and *Press by Number*, with ten episodes per task.

Agent models and comparisons. The main comparison uses Qwen3.8-27B, GPT-5.5, and GPT-6. Within each model, *w/o HPK* uses the harness without persistent knowledge, while *w/ Full HPK* uses both levels of HPK. We include $\pi_{0.5}$ (Intelligence et al., 2025) and Harness VLA (Zhang et al., 2026) as reference methods. The hierarchy ablations share source trajectories, the acting model, perception, executor, and action budget. *Flat Reflection* summarizes the same source experience without separating task-level and action-level guidance. *Task-only* and *Action-only* use subsets of the same knowledge store. GPT-5.5 and GPT-6 are used for the correction, self-evolution, and transfer comparisons.

Evaluation protocol. We report task success and within-model gains in percentage points (pp). RoboDojo additionally reports a normalized task-progress Score. Evaluation uses fixed held-out configurations and read-only knowledge stores. Across-episode updates use the designated interaction trajectories, with VLM parameters and execution modules fixed. Appendix D provides construction budgets, model configurations, metrics, and replication details.

A.2 Real-World Experimental Setup

Hardware Setup. We conduct real-world experiments on an AgileX Cobot Magic platform configured as an ALOHA-style bimanual system (Fu et al., 2024; Zhao et al., 2023). The platform comprises four 6-DoF Piper arms: two leader arms for human teleoperation and two follower arms for data collection and autonomous execution. Each follower arm is equipped with a parallel-jaw gripper. Three RealSense D435 RGB-D cameras provide a front view of the workspace and a wrist-mounted view from each follower arm. Fig. 7 shows the two follower arms and the camera locations; the leader arms are not shown.

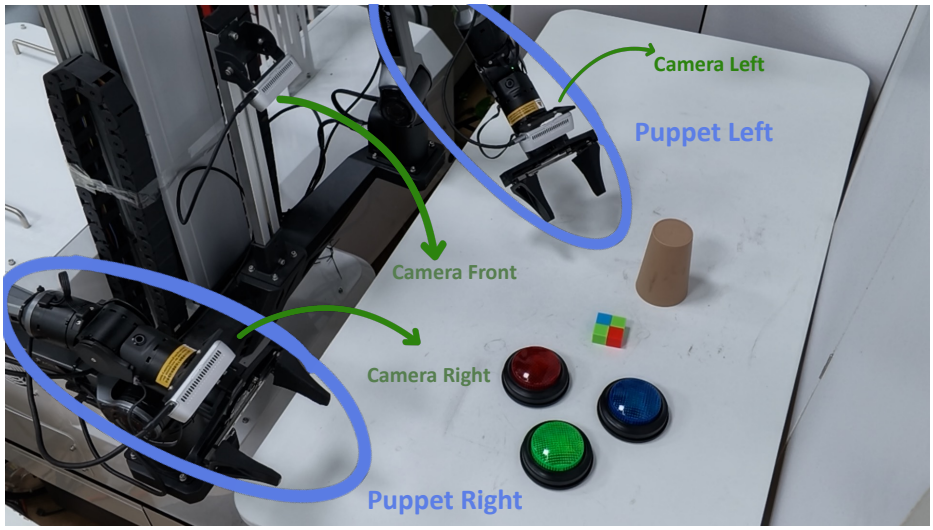


Figure 7: **Real-world hardware setup.** The AgileX Cobot Magic platform uses two follower arms for tabletop manipulation, with one front-view and two wrist-mounted RGB-D cameras. The follower arms are labeled *Puppet Left* and *Puppet Right* in the image.

$\pi_{0.5}$ baseline training and deployment. For the real-robot baseline, we initialize $\pi_{0.5}$ from the pretrained `pi05_base` weights and fine-tune all parameters on 450 RMBench simulation demonstrations spanning nine tasks (277,350 frames). The policy takes a task instruction, three RGB views (workspace and two wrists), and a 14-dimensional bimanual state comprising joint and gripper positions. It predicts 50-step action chunks. During training, joint targets are represented relative to the current state, while gripper targets remain absolute; output transforms recover absolute action targets.

Training uses eight GPUs with seed 42. We run 20,000 updates at global batch size 64, then resume for 5,834 updates at batch size 256, totaling 25,834 updates. AdamW uses $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 10^{-10}, and gradient clipping at norm 1.0, with an exponential moving average decay of 0.99. The learning-rate schedule uses 1,000 warmup steps to 2.5×10^{-5}, followed by cosine decay toward 2.5×10^{-6} over a configured total horizon of 30,000 steps. We directly deploy the final checkpoint (`25833`, indexed from zero) on the real robot without any real-world fine-tuning. Policy parameters remain fixed during evaluation.

Evaluation Rule. We evaluate three real-world tasks: *Cover with Assigned Cups*, *Press by Digit–Color Mapping*, and *Uncover, Count, and Press*. The comparison includes $\pi_{0.5}$, RoboHarn-Evo *(Zero-shot)*, and RoboHarn-Evo *(After RSI)*. Zero-shot uses HPK transferred from simulation without prior real-world knowledge updates. After RSI uses HPK further updated through real-world interaction, with the VLM parameters and low-level executor fixed. Each RoboHarn-Evo setting is evaluated on eight scenes per task. We report both full-task success and normalized task progress to distinguish complete execution from partial completion of these sequential tasks.

Task progress and success. Each task is divided into M ordered subtasks, where $M = 3$ for *Cover with Assigned Cups* and *Press by Digit–Color Mapping*, and $M = 4$ for *Uncover, Count, and Press*. Each subtask receives a score of 0, 0.5, or 1:

- 0: No valid operation toward the required subtask has begun.
- 0.5: The correct operation is underway, but the required outcome has not yet been completed and confirmed.
- 1: The required outcome has been completed and confirmed.

These scores measure achieved task progress; the number of attempts does not by itself reduce a completed subtask to 0.5. Scoring follows the required task order. A subtask receives no credit unless all preceding subtasks have received 1. Using the wrong object or button, or failing to match the required press count, does not qualify as completion. Scores are reassessed if a previously completed state is disrupted; the reported score reflects the state at the end of the rollout, not the highest progress reached earlier. Full task credit additionally requires compliance with all task-specific ordering and arm-use constraints described below.

For rollout n, let $q_{n,j}$ denote the final score of subtask j. The normalized rollout progress P_n and the reported mean task progress are

$$P_n = \frac{1}{M} \sum_{j=1}^{M} q_{n,j}, \qquad \text{TaskProgress} = \frac{100}{N} \sum_{n=1}^{N} P_n, \tag{A.1}$$

where N is the number of evaluated rollouts for the task. Success requires the full task score, with all instruction constraints satisfied. The success rate is the percentage of successful rollouts. When aggregating across tasks, we average the three task-level percentages with equal weight.

Task requirements and scoring criteria. The following criteria apply together with the shared ordering rule.

1. **Cover with Assigned Cups.** The robot must cover the red, green, and blue blocks in that order. Each rollout instruction specifies which cup to use for each block. Cups are identified by their initial left, middle, or right position; these identities remain fixed after the cups move. The cup assignment is instruction-dependent, rather than a fixed left-to-right sequence. Both arms must be used during the task, and only one arm may move at a time. The maximum score is 3.

- *Step 1: Covering the red block with the first designated cup.*
  - 0: The robot has not begun valid manipulation of the designated cup.
  - 0.5: The designated cup has been grasped and is being transported or placed, but coverage of the red block is not confirmed.
  - 1: The red block is covered by the designated cup, and the cup has been released.
- *Step 2: Covering the green block with the second designated cup.*
  - 0: The robot has not begun valid manipulation of the designated cup.
  - 0.5: The designated cup has been grasped and is being transported or placed, but coverage of the green block is not confirmed.
  - 1: The green block is covered by the designated cup, and the cup has been released.
- *Step 3: Covering the blue block with the third designated cup.*
  - 0: The robot has not begun valid manipulation of the designated cup.
  - 0.5: The designated cup has been grasped and is being transported or placed, but coverage of the blue block is not confirmed.
  - 1: The blue block is covered by the designated cup, and the cup has been released.

Full completion requires all three assigned coverings to hold, with both arms used sequentially as instructed.

2. **Press by Digit–Color Mapping.** The robot reads the left and right displayed digits. The instruction assigns each digit to a different button color. The robot first presses the button assigned to the left digit that many times, then presses the button assigned to the right digit that many times. It finally presses the remaining colored button exactly once to finish. The digit-to-color mapping is specified by the current instruction; the finishing button has no fixed color. The maximum score is 3.
   - *Step 1: Completing the presses specified by the left digit.*
     - 0: No valid pressing of the assigned button has begun.
     - 0.5: The robot is pressing the correct button, but the required count has not yet been completed and confirmed.
     - 1: The assigned button has been pressed exactly as many times as specified by the left digit.
   - *Step 2: Completing the presses specified by the right digit.*
     - 0: No valid pressing of the assigned button has begun.
     - 0.5: The robot is pressing the correct button, but the required count has not yet been completed and confirmed.
     - 1: The assigned button has been pressed exactly as many times as specified by the right digit.
   - *Step 3: Pressing the remaining button to finish.*
     - 0: No valid pressing of the remaining button has begun.
     - 0.5: The robot is pressing the remaining button, but completion of the press is not yet confirmed.
     - 1: The remaining button has been pressed exactly once after both digit-specified sequences are complete.

3. **Uncover, Count, and Press.** The robot first uses one arm to remove the cup covering the blocks and release it in a safe location. It then observes the numbers of red, green, and blue blocks and uses the other arm to press the corresponding buttons in red–green–blue order. Each button must be pressed exactly as many times as there are blocks of that color. Cup removal and button pressing must be performed sequentially by different arms. No additional finishing-button press is required. Counting determines the required press counts and is not scored as a separate subtask. The maximum score is 4.
   - *Step 1: Removing and safely placing the cup.*
     - 0: The cup still covers the blocks, and valid removal has not begun.
     - 0.5: The cup has been grasped and is being lifted, transported, or placed, but safe placement is incomplete.
     - 1: The cup has been released in a safe location, leaving all blocks visible.

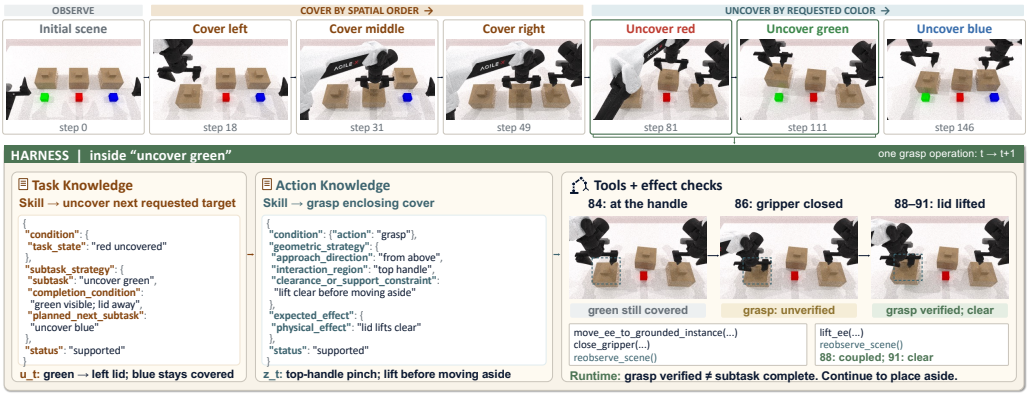


Figure 8: Example of Cover Blocks.

- *Step 2: Pressing the red button to match the red-block count.*
  - 0: No valid pressing of the red button has begun.
  - 0.5: The robot is pressing the red button, but the required count has not yet been completed and confirmed.
  - 1: The red button has been pressed exactly as many times as there are red blocks.
- *Step 3: Pressing the green button to match the green-block count.*
  - 0: No valid pressing of the green button has begun.
  - 0.5: The robot is pressing the green button, but the required count has not yet been completed and confirmed.
  - 1: The green button has been pressed exactly as many times as there are green blocks.
- *Step 4: Pressing the blue button to match the blue-block count.*
  - 0: No valid pressing of the blue button has begun.
  - 0.5: The robot is pressing the blue button, but the required count has not yet been completed and confirmed.
  - 1: The blue button has been pressed exactly as many times as there are blue blocks.

Full completion requires safe cup placement and all three press sequences, with the instructed separation of arm roles.

B Examples of Persistent Task and Action Knowledge

We provide representative entries from the persistent knowledge store for the *Cover Blocks* task. The source package was extracted from ten fixed demonstration trajectories and contains six Task Knowledge entries grouped into two Task Skills, and four Action Knowledge entries grouped into four Action Skills. The examples below preserve the semantic fields stored by the method; identifiers and runtime bindings are intentionally omitted because they are not part of the VLM-facing knowledge representation. Table 4 summarizes the complete persistent store for this task. Tables 5 and 6 then show one Task Knowledge entry and its complementary Action Knowledge entry, respectively, while Figure 9 grounds both entries in the corresponding execution sequence.

B.1 Task Knowledge Example

Table 5 shows how Task Knowledge determines *what to do next* from task progress and the requested semantic order. It deliberately selects the target by identity rather than by spatial position. In Figure 9(a,c), all blocks are initially covered, and the red block is visible after the selected lid has been moved away.

Table 4: Complete catalog of the persistent knowledge entries extracted for *Cover Blocks*. The table summarizes semantic content and skill grouping, rather than treating the independently collected HPK examples as repeated validation trials.

Type	Skill family	Persistent entry
Task	Cover by spatial order	Cover the leftmost visible target
Task	Cover by spatial order	Cover the next (middle) visible target
Task	Cover by spatial order	Cover the remaining rightmost target
Task	Uncover by requested identity	Uncover the red block first
Task	Uncover by requested identity	Uncover the green block next
Task	Uncover by requested identity	Uncover the blue block last
Action	Grasp free cover	Grasp a table-supported lid by its exposed handle
Action	Grasp enclosing cover	Grasp a covering lid and lift it clear of the block
Action	Place cover to enclose	Center a held lid over a visible block and release
Action	Place removed cover aside	Set a removed lid down away from the revealed block

Table 5: Representative Task Knowledge entry (semantic content preserved, reformatted for readability).

overall_goal	Cover all targets from left to right, then uncover them in red, green, blue order.
task_state	Start of the uncovering phase after all targets have been covered.
relevant_relations	All target blocks are covered; the red block is under one lid; the lid over the red block is reachable.
subtask	Uncover the red block.
purpose	Begin the requested uncovering sequence with red.
selection_basis	Select the lid covering the red block, independent of spatial position, because red is first in the requested uncover order.
completion_condition	The red block is visible and its lid has been moved away while later-color blocks remain covered.
planned_next_subtask	Uncover the green block.

B.2 Action Knowledge example

Table 6 specifies *how to realize* the selected subtask. Its applicability condition distinguishes removing an enclosing lid from grasping a free-standing lid, while its expected effect provides an observable verification criterion. The intermediate observation in Figure 9(b) directly verifies this expected effect: the red block becomes visible while the lid moves with the gripper.

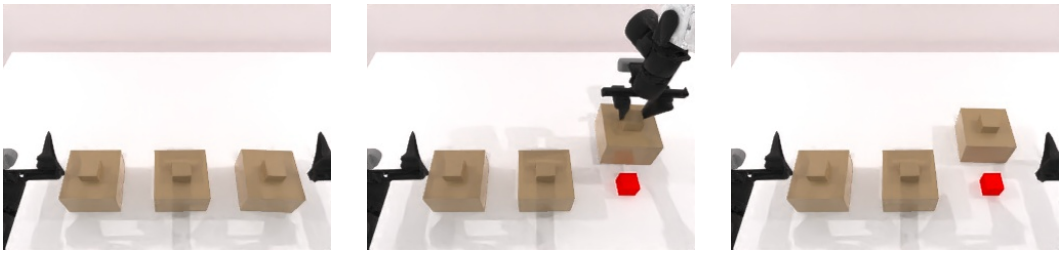
Example provenance and scope. The frames in Figure 9 provide traceability for this independently collected HPK example: they show the state transition from the applicable condition to the expected effect and subtask completion. They are illustrative observations from the same execution sequence, not an aggregate reliability evaluation or repeated validation trial. This example demonstrates the extracted knowledge structure and its grounding in a fixed demonstration; it does not by itself establish real-robot effectiveness.

C Additional Method Details

RoboHarn-Evo evolves the knowledge that guides a fixed VLM and robot executor. Its central mechanism is a shared subtask context: Task Knowledge specifies the objective, Action Knowledge specifies how to act toward it, and physical evidence determines how both are revised before reuse. This appendix details the representation, retrieval, and maintenance procedures that implement this interaction–update cycle.

Table 6: Representative Action Knowledge entry (semantic content preserved, reformatted for readability).

action	grasp.
object_description	A lid currently covering a block.
held_state	No lid held.
support_relation	The lid is supported on the table around or over the block.
target_relation	The top handle is exposed and reachable.
approach	From above, referenced to the covering lid's top handle.
interaction	Close the fingers around opposite sides of the handle; align the gripper span across the handle and keep the lid level.
clearance	Lift vertically high enough to clear the hidden block before moving the lid laterally away.
avoid	The block beneath the lid; neighboring covered or set-aside lids; nearby exposed blocks; workspace edges when near an edge.
physical_effect	The gripper gains control of the covering lid and lifts it clear of the block.
verification	The covered block becomes visible while the lid rises with the gripper.



(a) Before: all blocks are covered.

(b) Action effect: the red block becomes visible while the lid is held.

(c) Subtask complete: the lid is set aside and the red block remains visible.

Figure 9: Evidence associated with the two representative entries from demonstration `episode_9`. The Task Knowledge evidence spans frames 512–687. The Action Knowledge evidence for grasping the enclosing lid spans frames 512–600. The final frame also verifies completion of the subsequent set-aside action.

C.1 Hierarchical knowledge representation

Runtime context and persistent knowledge. Within episode i, the harness maintains s_t from observations and interaction history. It records the active subtask, relevant objects and relations, executed operations, and their observed outcomes. Across episodes, $\mathcal{K}_i = (\mathcal{K}_i^{\mathrm{T}}, \mathcal{K}_i^{\mathrm{A}})$ stores reusable Task and Action Knowledge, and $\mathcal{F}_i = (\mathcal{F}_i^{\mathrm{T}}, \mathcal{F}_i^{\mathrm{A}})$ indexes these entries by Skill. Following Sec. 3, t indexes operation boundaries, c_t denotes parameterized tool calls, and a_t denotes the resulting robot-action sequence. A subtask can remain active across several operations; its completion is assessed separately from the success of each operation.

Task and Action entries. A Task entry describes the conditions under which a subtask is appropriate, its purpose, its selection rationale, and its completion condition. An Action entry describes an operation type, applicable object and support relations, an object-relative geometric strategy, and its intended physical effect. Geometric strategies specify approach direction, contact region, gripper orientation, and clearance or support requirements. They are instantiated using current observations rather than stored robot poses. Both entry types retain links to their source executions, an evidence summary $\mathbf{b}_k = (n_k^+, n_k^-, n_k^?)$, and a retrieval status $\sigma(k)$. Table 7 illustrates the two levels using *Cover Blocks*.

From an episode to a knowledge package. The reflector receives $\mathrm{View}(\tau_i)$: observations, tool calls, and execution feedback associated with the episode. It constructs $\mathcal{P}_i$ as in Eq. equation 4, preserving the order of subtasks $k_{i,j}^{\mathrm{T}}$ and the operations $\{k_{i,j,r}^{\mathrm{A}}\}_{r=1}^{m_{i,j}}$ within each subtask. Before-and-after observations remain associated with the operations they assess. Observed facts and inferred

Table 7: Two linked knowledge entries for *Cover Blocks*, with the requested uncovering order red, green, then blue. The grasp effect is a step toward, rather than the completion of, the uncovering subtask.

Component	Task Knowledge	Action Knowledge
Condition	Red is uncovered; green and blue remain covered.	A lid covers the target block; its top handle is reachable.
Strategy	Uncover green next, according to the requested color order.	Pinch opposite sides of the top handle; lift vertically to clear the block before moving laterally.
Outcome	Green is visible and its lid is moved away; blue remains covered.	The lid rises with the gripper and clears the block.
Evidence	Observations at the subtask boundaries establish the required change in task state.	Observations of the grasp and lift establish control of the lid.

rationales are recorded separately, so a proposed explanation does not replace the physical outcome of an attempt. Atomic entries are the units of retrieval and revision; the package retains their shared task context and temporal associations.

Skill organization. A Skill groups entries that address a related decision context and summarizes their applicability. For example, color-order uncovering groups Task entries, while lid grasping and lid placement group Action entries. A group can contain distinct strategies or conditions. Group membership therefore determines where retrieval and maintenance look; semantic consolidation determines which entries express the same knowledge.

C.2 Hierarchical retrieval and goal-consistent execution

Selecting what to accomplish. Task retrieval uses the instruction, current runtime memory, and the planner's baseline subtask. Skill summaries identify relevant groups, from which the VLM selects supported entries whose conditions match the current state. These entries refine u_t, including the designated object, target, and completion condition. If no entry applies, the planner retains its baseline decision. In the example of Table 7, the requested color sequence determines the next target after red is uncovered, regardless of the targets' left-to-right arrangement.

Selecting how to act. Action retrieval follows the selection of u_t and operation type $\alpha_t \in \{\texttt{grasp}, \texttt{place}, \texttt{contact}\}$. The query includes the target object, held state, support relations, and descriptions of the current candidate geometry. The VLM selects applicable Action Knowledge together with a geometric realization z_t. Its conditions must match the present interaction: grasping a lid that covers a block, for example, requires clearance above the hidden block before lateral motion. The retrieved strategy supplies geometric guidance within the selected subtask rather than changing its target.

Skill routing. The routing defaults use exhaustive recall for stores of at most 24 entries. For larger stores, local inverse-document-frequency overlap ranks Skill summaries and member cards; up to two Skills are selected, and their members are interleaved to form a shortlist of up to eight entries. When lexical overlap is uninformative, the VLM selects Skills from semantic routing cards. On this model-routed path, up to 24 members can be passed directly to the final selector; larger candidate sets are first shortlisted to eight. Both levels filter for supported entries, and Action retrieval additionally matches the operation type. Final selection uses compact semantic cards and the current scene context.

Grounding under the task goal. The harness generates candidate poses from current observations and checks feasibility together with the object, target, and target relation specified by u_t. Candidates matching z_t are prioritized within this admissible set, preserving their relative order. The VLM issues tool calls c_t, and scene grounding, motion planning, and control produce a_t. Without applicable Action Knowledge, the baseline candidate ranking is retained under the same task constraints.

If no candidate satisfies these constraints, the agent observes again or replans instead of substituting a different target.

Closing the execution loop. Execution checks and VLM assessment compare the observed outcome with the intended effect. Their result updates s_t and informs whether to continue, re-observe, or change the attempted realization. The planner advances to the next subtask when its completion condition is verified. Thus control of a grasped object can enable a subsequent transport operation while leaving the placement or uncovering subtask active. Tool-call completion, operation effect, and subtask completion remain distinct throughout the trajectory.

C.3 Physical evidence and knowledge revision

Evidence is tied to the claim being tested. Each judgment concerns a particular knowledge entry, its content version, and a source execution. Assessment first establishes that the entry's condition held and that the executed operation followed its strategy under the designated task goal. The outcome is then labeled `support` when the intended effect is observed, `oppose` when the observations establish failure, and `unverified` when execution deviated or the available observations are inconclusive. An out-of-scope execution does not contribute evidence to that entry. This attribution keeps an off-target local success from reinforcing a strategy for the intended task.

Verification follows the knowledge hierarchy. Action evidence concerns the physical effect of an operation, such as an object moving with the gripper after a grasp, resting on its target support after release, or changing state after contact. Task evidence concerns the completion condition of the enclosing subtask and the required task relations. It is assessed at subtask boundaries rather than inferred by summing successful constituent operations. A successful grasp can therefore support Action Knowledge while the associated Task entry awaits evidence that its completion condition has been reached.

Delayed observations refine the same attempt. Consider removing the lid over the green block in Table 7. Gripper closure alone can leave attachment uncertain. A subsequent lift showing the lid moving with the gripper supports the grasp strategy. The uncovering subtask is completed when green is visible and the lid has been moved away, with blue still covered. These observations address different levels of the hierarchy. At the operation level, the lift refines the pending grasp judgment instead of creating a second grasp trial. Source associations preserve one verdict per attempt for each assessed content version. If a pending verdict has already been recorded, resolving it replaces that verdict rather than appending a duplicate event.

Evidence summaries and retrieval eligibility. For an entry k, n_k^+, n_k^-, and $n_k^?$ count distinct applicable attempts labeled support, opposition, and unverified, respectively. The retrieval status is determined by

$$\sigma(k) = \begin{cases} \texttt{contested}, & n_k^- > 0, \\ \texttt{supported}, & n_k^+ > 0 \text{ and } n_k^- = 0, \\ \texttt{candidate}, & n_k^+ = n_k^- = 0. \end{cases} \tag{C.1}$$

Retrieval considers supported entries whose conditions apply to the current decision. Unverified outcomes remain in the evidence record without increasing support or opposition. A newly proposed strategy has candidate status until an applicable execution supplies support. Counter-evidence withdraws the challenged version from retrieval and directs maintenance toward its content and applicability conditions.

Consolidating equivalent knowledge. New atoms are assigned to existing Skills, or to a new Skill when their decision context is not represented. Maintenance then operates on the corresponding Task or Action partitions, including historical entries in the affected Skills. Entries are consolidated when their conditions, strategies, and intended effects are equivalent. Entries that share an object or operation but prescribe different interactions remain distinct. Evidence summaries are combined from the associated attempts, with repeated references to the same attempt counted once. Runtime and reflected records of an operation therefore contribute the same evidence event. Consolidation preserves supporting, opposing, and unresolved observations together.

Revising challenged content. Maintenance uses before-and-after observations and execution records to review historical entries alongside new knowledge. A revision changes the applicable condition or recommended strategy in response to the observed outcomes. Narrowing an entry to a demonstrated interaction context and changing its grasp geometry are different revisions: each produces a content version whose claims are checked against relevant executions. Evidence is associated with the version it assesses. For a narrower condition, an earlier execution contributes only when that condition held; for a changed strategy, it contributes only when the changed strategy was actually executed. A rewritten description therefore receives support through an execution that tests it, not through the act of rewriting.

The revised version becomes retrievable when it satisfies Eq. equation C.1. Earlier versions retain their evidence, including opposition. This separates withdrawal from repair: withdrawal prevents reuse of a challenged claim, whereas repair establishes an evidence-supported replacement with revised content. A promising but untested revision remains a candidate. An observed failure that still applies to the revised claim remains opposing evidence for that claim.

Refreshing the persistent store. After consolidation and revision, the maintainer refreshes the summaries and membership of the affected Skills. Unaffected entries remain unchanged. The updated $(\mathcal{K}_{i+1}, \mathcal{F}_{i+1})$ supplies the next episode's retrieval, so later decisions reflect both newly supported strategies and changes to historical knowledge. Physical feedback can thus change what the harness recommends and the conditions under which it recommends it, while the base VLM and execution tools remain fixed.

C.4 The interaction–update cycle

Algorithm 1 combines the procedures above. During an episode, observations and effect checks update runtime memory and provide the evidence for the next knowledge update. After the episode, reflection and maintenance update the persistent store. The next trajectory is therefore generated using knowledge shaped by earlier trajectories, closing the recursive loop between execution and knowledge revision.

Algorithm 1 RoboHarn-Evo: execution and evidence-driven knowledge evolution

Require: Fixed VLM parameters θ, executor Exec, initial store $\mathcal{K}_0$ and Skill index $\mathcal{F}_0$

for each interaction episode i with instruction l **do**
  Initialize s_0 from the initial observations; set $t = 0$.
  while the task is active and the execution budget remains **do**
    Select or retain u_t using Task retrieval and verified task progress.
    Choose α_t; retrieve Action Knowledge to select z_t for u_t.
    Ground z_t using feasible, goal-consistent scene candidates.
    if no admissible realization is available **then**
      Re-observe or replan; record the unresolved attempt.
    else
      Issue tool calls c_t and execute robot actions a_t.
      Assess the operation effect and subtask completion separately.
      Retain observations, tool calls, and effect judgments.
    end if
    Update runtime memory s_{t+1} from feedback; increment t.
  end while
  Reflect on $\mathrm{View}(\tau_i)$ to construct $\mathcal{P}_i$.
  Assess new Task and Action entries against level-specific physical evidence.
  Assign new atoms to Skills and collect affected historical entries.
  Review historical entries; consolidate equivalents and revise challenged content.
  Check revisions against relevant executions; update version-specific evidence.
  Apply Eq. equation C.1 and refresh affected Skill summaries.
  Retain $(\mathcal{K}_{i+1}, \mathcal{F}_{i+1})$ for subsequent episodes.
end for

D Detailed experimental protocols and result tables

D.1 Q2: experience-pool self-correction

D.1.1 Metric definitions

Let $\mathcal{V}_K$ be the active pool and $\mathcal{R}_K \subseteq \mathcal{V}_K$ the reviewed entries at checkpoint K. Let C_K, I_K, and U_K count correct, incorrect, and unverified labels in $\mathcal{R}_K$, with $N_K = C_K + I_K + U_K$. Then

$$\mathrm{CER}_K = 100\frac{C_K}{N_K}, \qquad \mathrm{UR}_K = 100\frac{U_K}{N_K}. \tag{D.1}$$

For a census, $\mathcal{R}_K = \mathcal{V}_K$; for a sampled review, the reported ratio is over reviewed active entries, accompanied by the sampling design and coverage. Unequal-probability sampling requires the corresponding weighted pool estimate. Unverified entries remain in the denominator; unaudited entries are neither silently labeled unverified nor counted as correct.

Let $\mathcal{H}_0^-$ and $\mathcal{H}_0^+$ be the fixed initially reviewed incorrect and correct cohorts. Each initially incorrect claim has exactly one current outcome: *repaired*, *inactive*, *still incorrect*, or *unverified*. Repair requires the relevant active descendants to be independently reviewed as correct; inactivation requires that no active descendant retains the erroneous claim. A remaining erroneous branch prevents a split or merge from counting as resolved. If R_K^{fix} and R_K^{off} count the first two outcomes,

$$\mathrm{HERR}_K = 100\frac{R_K^{\mathrm{fix}} + R_K^{\mathrm{off}}}{|\mathcal{H}_0^-|}. \tag{D.2}$$

The two terms are reported separately. For initially valid knowledge, let T_K count claims with an active, correct semantic successor retaining their original valid scope. Correct retention is

$$\mathrm{CERet}_K = 100\frac{T_K}{|\mathcal{H}_0^+|}. \tag{D.3}$$

D.2 Q4: frozen-source transfer and target adaptation

RoboDojo evaluates both partial task progress and terminal success. For episode e, let r_e denote the task score. We report

$$\mathrm{Score} = \frac{100}{N}\sum_{e=1}^{N} r_e, \qquad \mathrm{SR} = \frac{100}{N}\sum_{e=1}^{N} \mathbb{1}[r_e = 1], \tag{D.4}$$

with $N = 10$ episodes for each transfer condition.

For Cover Blocks,

$$r_e \in \{0,\ 0.05,\ 0.15,\ 0.30,\ 1.0\}.$$

The intermediate values correspond to the furthest verified stage reached: all three blocks covered (0.05), red uncovered (0.15), and red and green uncovered while blue remains covered (0.30). A score of 1.0 requires completion of the full sequence and return of both arms.

For Press by Number,

$$r_e \in \{0, 1\},$$

because the task exposes terminal success only. Its mean Score therefore equals its Success Rate.

E Runtime Skill Library

E.1 Overview and Organization

The agent is equipped with a library of *runtime skills* that define structured behavioral contracts for different stages of the manipulation runtime. Each skill specifies the information available to the agent, the responsibility of the corresponding runtime step, the expected output format, and the constraints that must be respected. These skills are prompt-level runtime interfaces rather than additional learned policies or model parameters.

Category	# Skills	Primary Responsibility
Perception	1	Normalize perception queries before segmentation and scene-memory binding.
Memory	1	Convert observation and execution evidence into compact runtime memory.
Planning	5	Maintain task planning, runtime reasoning, and high-level execution decisions.
Monitoring	1	Detect execution health and semantic OOD states.
Tool Calling	19	Route tool use and construct grounded tool-call procedures from reusable primitives and workflows.
Effect Verification	1	Verify whether executed actions produced their intended physical effects.
Total	**28**	

Table 8: Organization of the runtime skill library.

The library contains 28 skills organized into six functional categories: Perception, Memory, Planning, Monitoring, Tool Calling, and Effect Verification. Perception, Memory, Monitoring, and Effect Verification each contain a single dedicated skill. Planning contains five skills for different planning and runtime-reasoning responsibilities. Tool Calling contains 19 skills, organized into one routing skill, ten reusable primitive skills, and eight workflow skills. Experience records, including raw traces, case summaries, learned lessons, and retrieval artifacts, are maintained separately from the runtime skill prompts and are not counted as runtime skills.

The categories define complementary runtime responsibilities rather than independent policies. Perception normalizes task-conditioned queries used by downstream grounding. Memory summarizes task-relevant observation and execution evidence into existing runtime memory surfaces. Planning maintains task and subtask decisions from the available runtime state. Monitoring detects execution anomalies and produces explicit semantic OOD information. Tool Calling maps the current execution context to structured tool-use procedures through routing, workflows, and reusable primitives. Effect Verification evaluates the observed physical consequence of an executed action or tool sequence.

We distinguish these runtime skills from the *semantic Skills* used by Action-Feedback Knowledge (AFK) in the main method. AFK Skills organize atomic Task or Action Knowledge for retrieval, whereas the runtime skills described here define the structured interfaces used by the agent during perception, memory construction, planning, monitoring, tool calling, and effect verification.

E.2 Perception Skills

The Perception category contains a single skill for normalizing perception requests before segmentation and scene-memory binding.

Perception Query Normalization.

Purpose and Invocation. This skill converts raw perception-query objects into a stable semantic interface for downstream perception. It is applied before segmentation and scene-memory binding, and is responsible for normalizing object categories, task roles, relational query types, and optional instance bindings. When a previous query set is rejected by the runtime, the same skill can regenerate the queries using the supplied failure reason and binding requirements.
Inputs. The skill receives the current task context, committed memory, observation summary, raw perception queries, and the candidate identities currently advertised by the runtime. It also receives the current instance-binding phase, whether exact binding is required, an optional binding postcondition describing a rejected previous result, and the maximum number of queries that may be returned. In simulator environments, an optional oracle-object catalog may also be provided. Each raw query may specify an object or part phrase, a segmentation

prompt, a task role, an entity scope, an optional spatial relation, instance-level disambiguation, and an optional advertised runtime or oracle identity.

Core Procedure.

1. Normalize each query into a stable object or part category while preserving descriptors that are useful for segmentation.
2. Determine whether the query serves as a target, tool, or contextual reference, and separate category-level semantics from instance-level disambiguation.
3. Represent ordinary objects as single-instance queries. When multiple visual references jointly define a placement relation, preserve them as a reference set rather than converting the relation into a metric coordinate.
4. Resolve instance identity according to the current binding phase. Candidate discovery permits a task-supported target or tool to remain unbound when no matching candidate has yet been detected, whereas execution-facing selection requires an exact advertised identity when binding is enabled.
5. If a previous query set was rejected, use the reported failure reason and previous output to correct the missing role or identity rather than repeating the rejected result unchanged.

Output Contract. The skill returns JSON only. The output contains at most the runtime-specified maximum number of normalized queries and follows the schema below.

```
{
  "queries": [
    {
      "object_id": "<category>",
      "text_prompt": "<segmentation prompt>",
      "role": "target | tool | context",
      "entity_scope": "single_instance | reference_set",
      "placement_relation": "center_of | omitted",
      "expected_count": "<int | omitted>",
      "instance_hint": "<descriptor | empty>",
      "instance_ref": "<track_id / instance_id | empty>",
      "oracle_id": "<oracle_id | empty>",
      "reason": "<short rationale>"
    }
  ]
}
```

Key Constraints.

- Object categories should remain stable across frames, while instance-specific descriptors are represented separately.
- Objects and identities that are not supported by the task, raw queries, or advertised candidates must not be invented.
- Exact runtime or oracle identifiers must be preserved without shortening or rewriting, and candidates must not be selected by list order.
- A visual reference set remains a relational query. The skill does not choose an arbitrary member of the set or generate the corresponding metric placement coordinate.
- Candidates retained only for recovery-time identity binding cannot provide current geometric evidence for grounded task progress.
- If a required target or tool cannot be identified confidently, the skill omits the unsupported query rather than guessing. If the required role itself cannot be supported, an empty query list is allowed.

Interaction with Other Components. This skill receives raw queries from the perception-query planner and produces normalized queries for downstream segmentation and scene-memory binding. The runtime validates the returned schema, required roles, identifier membership, grounding validity, visibility or retained-track status, supported relations, and query

count before the queries are used downstream. Relational reference sets are converted by the runtime into separately verified placement targets rather than metric poses produced by the language model.

E.3 Memory Skills

The Memory category contains a single skill for converting observation and execution evidence into compact memory for the next control turn.

Observation Memory Summarization.

Purpose and Invocation. This skill converts the current observation, segmentation and grounding results, scene memory, monitor signals, tool-execution history, and previously committed memory into compact task state for the next control turn. It uses the existing runtime memory surfaces rather than introducing an additional persistent memory representation. Its main role is to retain task-relevant facts, recent execution consequences, and unresolved uncertainty without requiring later control turns to repeatedly inspect the full raw observation and execution history.

Inputs. The skill operates over the existing runtime state, including committed task memory, the recent observation summary, scene memory, observation preprocessing results, prior tool calls and outcomes, robot state, and semantic tags when they are already available. Scene memory additionally provides stable object instances, task focus, temporal tracks, grounding information, and the runtime-maintained position state of each tracked instance.

Core Procedure.

1. Collect only task-relevant facts supported by the current observation, scene memory, monitor result, tool-execution history, or previously committed memory.
2. Preserve stable object-instance bindings across control turns when the available geometry and interaction history remain consistent, while retaining the runtime-provided position validity instead of re-deriving it from visibility alone.
3. Summarize the consequences of recent execution, including the current target or tool binding, grounding availability, robot and gripper state, the latest tool-call outcome, repeated failures, and constraints that affect the next decision.
4. Carry forward uncertainty explicitly when perception is weak, missing, contradictory, or retained only from history, and update committed task progress only when it is supported by observation, monitoring, or verified physical evidence.

Output Contract. The skill writes into two existing text-based memory surfaces. The first, `memory_text`, contains one concise sentence describing committed facts that remain relevant across control turns, such as completed subtasks, stable object relations, active target or tool bindings, and verified failures. The second, `recent_observation_summary`, provides a compact execution digest containing only evidence that may affect the next decision. The recommended structure is:

```
focus=target:... tool:...;
grounding=...;
robot=...;
recovery=...;
uncertainty=...
```

Metric grounding is included only when it is useful for execution or disambiguation. The skill does not create additional durable state fields such as a separate manipulation phase or execution-state schema.

Key Constraints.

- Observation preprocessing is treated as evidence rather than as the final task-state decision.
- Task progress must not be inferred from an attempted action alone, and intended future actions must not be stored as completed state.

- A closed gripper does not establish a successful grasp without supporting physical or monitoring evidence.
- A visible object with missing executable grounding remains ungrounded.
- For tracked or temporarily missing objects, geometric usability follows the runtime-provided position state rather than visibility alone. A retained coordinate may remain usable when it is marked as valid, whereas a coordinate made uncertain by contact, failed grasp, collision, or unverified release cannot be used for grounded motion until reacquisition.
- Missing, weak, or contradictory evidence must remain explicit uncertainty rather than being replaced by an unsupported clean state.
- Stable instance identifiers or VLM-provided instance hints are used when multiple instances exist; raw segmentation rank is not used as object identity.

Interaction with Other Components. This skill consumes observation, grounding, monitoring, robot-state, and tool-execution evidence already maintained by the runtime and summarizes it into the memory surfaces used by subsequent control turns. In particular, it preserves the execution consequences and uncertainties that are relevant to later planning or tool-control decisions, so those decisions do not need to reconstruct the current state directly from raw observation JSON.

E.4 Planning Skills

The Planning category contains five skills that cover different levels of task planning, runtime-state interpretation, monitored execution, and data-collection workflows. Table 9 summarizes their responsibilities. We describe representative planning skills in greater detail in the following subsections.

E.4.1 Planning Skill Inventory

Skill	Responsibility
Control Turn Planner	Decides the next top-level action forone agent control turn from the task instruction, committed memory, observations, state information, and available runtime state. It returns the next committed memory, an executor-facing subtask, an action mode, an arm preference, and optionally a selected skill.
Runtime State Reasoning	Defines how the planner interprets structured task, working, perception, scene-memory, monitor, manipulation, and recovery state. It treats structured runtime fields as the authoritative state source and specifies how that state should be used when producing the next control decision.
Long Horizon Execution	Handles global tasks that must be decomposed into ordered subtasks. It refines the subtask plan, selects the next subtask, checks the previous subtask, decides whether to retry, recover, continue, or finish, and delegates concrete execution to a monitored execution boundary.
Monitored Subtask Execution	Executes exactly one narrow executor-facing instruction as a monitored action unit. It monitors success, failure, stall, and timeout conditions, performs deterministic stopping or resetting when needed, and returns a structured execution result to the calling workflow.
EAP Data Collection	Implements a forward/reverse EAP-style data-collection workflow. It initializes a collection run, executes forward behavior, executes reverse or reset behavior, keeps the environment reusable, and records structured run and dataset information.

Table 9: Planning skills and their responsibilities.

Table 9 enumerates the complete Planning skill set. Because several skills address specialized execution or data-collection workflows, we do not describe every skill individually. Instead, the following subsection presents two representative examples that define the core planning interface.

E.4.2 Representative Planning Skills

The following examples illustrate two complementary aspects of the planning stack: Control Turn Planner defines the decision contract for one control turn, while Runtime State Reasoning specifies how structured runtime state is interpreted when producing that decision.

Control Turn Planner

Purpose and Invocation. The Control Turn Planner defines the planner contract for one control turn. It is used when deciding the next top-level action of a robot rollout. Given the current task state, committed memory, visual observations, state summary, and available structured runtime state, it produces the next committed memory and an executor-facing subtask.
Inputs. The skill receives the global task instruction, previous committed memory, the segment-start and current images, and a numeric state-summary vector. When available, it additionally receives structured runtime state containing task memory, working memory, perception results, scene memory, monitor state, recovery state, and available skills. The planner may also receive the current grasp-transport policy, per-arm manipulation state, release-guard setting, action-geometry repair policy, and runtime evaluation containing the environment's authoritative global-success signal, reward, and step counters. Under the compact planner context, scene memory is used as the single structured scene view.
Core Procedure.

1. Determine whether any task-relevant state change has already been supported by observation, scene memory, monitor information, or environment-success evidence, and update committed memory accordingly.
2. Read completed and failed skill histories as transitions already committed by the after-action verifier rather than independently inferring completion from planner intent.
3. Select an actionable executor-facing subtask and one action mode from start, continue, retry, reset, replan, recover, switch, or finish. Global completion is determined by the runtime's authoritative global-success signal.
4. Select the preferred arm from grounded geometry, end-effector poses, gripper state, and recent physical outcomes, while respecting the current manipulation state and runtime policies governing grasp transport, release, and pending action-geometry repair.
5. For count-sensitive repeated transient actions, create one atomic event per subtask and wait for after-action verification before advancing to the next repetition.

Output Contract. The skill returns JSON only using the following schema:

```
{
  "commit_label": "no_update | subtask_complete |
  ↪  state_change",
  "memory_text": "<committed task state>",
  "subtask_text": "<next executor-facing subtask>",
  "selected_skill": "<optional skill name>",
  "action_mode": "start | continue | retry | reset | replan |
  ↪  recover | switch | finish",
  "preferred_arm": "left | right | either",
  "semantic_tags": {}
}
```

The committed memory describes state that is already true, while the subtask field specifies the next action objective for the executor.
Key Constraints.

- Task progress cannot be committed from planner intent alone; it must be supported by runtime evidence.

- A subtask-complete label only summarizes a completion already recorded by runtime and does not create a new completion transition.
- The global-success signal in runtime evaluation is authoritative. The planner may output finish only when this signal is true.
- Retry, continue, and reset require an active skill; when no active skill exists, the planner must instead start or recover with an actionable replacement subtask.
- Stable scene-memory instance identifiers are preferred over raw single-frame detection ranks.
- Arm choice is represented explicitly by the preferred-arm field and must not be encoded indirectly through task wording or object names.
- Manipulation state and its associated runtime policies must be preserved across replanning; confirmed or provisionally authorized attachments must not be silently rewritten into a different physical state.
- When action geometry is marked as relocation-pending, the selected behavior must follow the configured repair policy rather than using quarantined historical approach, grasp, or contact geometry.

Interaction with Other Components. The Control Turn Planner consumes the structured runtime state interpreted within the planning stack and produces the committed memory, executor-facing subtask, action mode, arm preference, and optional selected skill for the next control turn. Completion and failure transitions referenced by the planner are supplied by runtime execution history and the after-action verifier rather than created by the planner itself.

Runtime State Reasoning

Purpose and Invocation. Runtime State Reasoning defines how structured `agent_state` should be interpreted inside a control turn. Its role is to make the structured runtime fields the authoritative source for task, working, perception, scene, monitor, manipulation, and recovery state, while using raw images and summaries as supporting evidence rather than reconstructing long execution histories from them.

Inputs. The reasoning payload exposes three main groups of state. Task memory contains the global task, committed memory, committed facts, execution plan, completed and failed skills, and the runtime task-finished flag. Working memory contains the active skill and instruction, recent observation information, scene memory, recent tool calls, local progress information, recovery history, manipulation state, runtime policy flags, and semantic tags. Monitor and recovery state provide the current rollout status, progress and failure information, together with retry, reset, and replan budgets or pending recovery actions.

When compact planner context is enabled, scene memory is the single planner-visible scene source. Public grasp/contact availability and manipulation state remain visible there, while complete masks, candidate poses, and attachment transforms remain private to runtime.

Core Procedure.

1. Inspect structured runtime fields first and use raw images or observation summaries as supporting evidence rather than reconstructing long histories independently.
2. Use scene memory for stable object identity and spatial grounding, and interpret each instance's position state independently of visibility. Current-verified geometry supports the normal grounded contract, memory-valid geometry permits only a safe approach before fresh verification, and motion-uncertain geometry forbids use of historical coordinates.
3. Use monitor and recovery state when deciding whether execution should continue, retry, recover, replan, or finish.
4. Interpret manipulation state together with the configured grasp-transport and release policies. Confirmed holding, provisional holding, pending release verification, and

release-recovery-required are kept as distinct runtime states with different allowed next actions.

5. Use public operation targets for placement decisions. Reference-region targets are selected from runtime-computed relations rather than recomputing their metric coordinates in the planner.
6. Keep committed memory concise and restricted to facts that are already true; when perception conflicts with previous memory, preserve the uncertainty instead of silently overwriting it.

Output Contract. This skill defines the structured payload that the planner should inspect rather than a separate planner-output schema. The control-turn payload follows the contract:

```
{
  "global_task": "...",
  "trigger": "...",
  "agent_state": {},
  "available_skills": [],
  "memory_harness": {},
  "runtime_state_reasoning": {}
}
```

The planner is instructed to inspect `agent_state` first and then use the memory harness and available skills when producing the next control decision.

Key Constraints.

- Structured runtime fields are the authoritative state source.
- Scene memory is used for stable instance identity and spatial grounding; private candidate identifiers are not exposed as planner choices.
- Verified physical roles must not be silently relabeled by a later single semantic observation.
- Historical coordinates marked motion-uncertain cannot be used for grounded motion.
- Under safe-motion repair, only runtime-validated open-gripper retreat, runtime-derived safe-height motion, and fresh camera observation are permitted while action geometry is pending.
- Transport authorization does not by itself imply confirmed attachment; provisional and confirmed holding states remain distinct.
- A released object awaiting verification must not immediately enter the next manipulation, while a verified released-target miss requires reacquisition rather than continued observation-only verification.
- Committed memory must remain short and contain only state that is already true; working-memory details should not be copied wholesale into persistent memory.

Interaction with Other Components. Runtime State Reasoning defines how the structured state supplied to the planning stack should be read before the next control decision is produced. The Control Turn Planner then uses this interpreted task, scene, monitor, manipulation, and recovery state together with the available skills and memory harness to determine the next control-turn output.

E.5 Monitoring Skills

The Monitoring category contains a single skill for semantic OOD and failure-state detection during VLA rollout execution.

OOD Detection

Purpose and Invocation. This skill judges whether the currently executing subtask has entered a semantic out-of-domain, blocked, failed, or visually invalid state. It belongs to the monitoring path rather than the execution path: the skill classifies execution health and returns an explicit structured OOD result, but does not execute recovery actions, replace progress or success detection, or directly determine low-level robot control. The classification is made relative to the active subtask rather than only to the global task.

Inputs. The skill receives the active subtask, the latest observation summary, current monitor status, recovery state, structured execution context, and the current and maximum environment step counts. The execution context may provide progress, task-success, and action-chunk information. The complete payload is evaluated jointly by the VLM; the runtime does not infer specific OOD classes afterward from a free-text summary.

Core Procedure.

1. Read the active subtask and establish what the VLA is currently expected to accomplish.
2. Inspect the latest observation, monitor status, recovery state, step budget, and execution context together. Explicit evidence such as target disappearance, grasp loss, motion blockage, camera failure, or a contradictory scene state is preferred over generic wording.
3. Select exactly one primary OOD scenario from the supported vocabulary, choosing the narrowest class justified by the available evidence.
4. Return the classification directly together with a short evidence-based reason and, when useful, consistent structured monitor signals.

Output Contract. The skill returns one JSON object. The primary contract is the explicit `OOD_scenario` field; supporting monitor signals are optional.

```
{
  "status": "ok",
  "selected_skill": "ood-detection",
  "OOD_scenario":
    "none | object_not_visible | motion_blocked |
     grasp_lost | scene_drift_detected | requires_replan",
  "reason": "<short evidence-based explanation>",
  "confidence": "<0.0--1.0, optional>",
  "signals": [
    {
      "name": "<runtime-aligned signal>",
      "level": "info | warning | error",
      "reason": "<short explanation>",
      "score": "<0.0--1.0>",
      "details": {}
    }
  ],
  "analysis_note": "<optional concise explanation>"
}
```

The supported primary scenarios are `none`, `object_not_visible`, `motion_blocked`, `grasp_lost`, `scene_drift_detected`, and `requires_replan`. If supporting signals are returned, they must remain consistent with the primary classification.

Key Constraints.

- Classification is conservative: when the available payload does not support a confident OOD diagnosis, the skill returns `none`.
- The skill must not fabricate scene facts, object identities, grasp states, or failure causes.
- It must output the OOD scenario explicitly rather than returning only a natural-language summary and relying on runtime post-processing to classify it.

- Conditions already handled by progress monitoring or task-success logic should not be forced into an OOD class.
- The narrowest justified failure class is preferred over a generic escalation label.
- Recovery tool plans and tool-call arguments are outside the scope of this skill.
- When evidence conflicts, the classification follows the strongest explicit evidence; `requires_replan` is used only when the payload supports such escalation.

Interaction with Other Components. The monitoring runtime constructs the evaluation payload and invokes this skill during VLA rollout. Its structured OOD result is consumed by the monitoring evaluator, and the returned signals are then available to the agent runtime when deciding whether VLA control should continue or be handed off to the tool-calling path. The skill itself performs only semantic execution-health classification and does not generate the subsequent tool calls.

E.6 Tool Calling Skills

The Tool Calling category contains 19 skills that structure the use of runtime manipulation tools. They are organized into one routing skill, ten reusable primitive skills, and eight workflow skills. The routing skill selects an appropriate workflow and post-execution intent; primitive skills specify the invocation contract of individual runtime tools; and workflow skills construct short, situation-dependent sequences from the available primitives.

E.6.1 Tool-Calling Skill Inventory

Table 10 and Table 11 enumerate the complete Tool Calling skill set. To avoid repeating similar invocation and output contracts across closely related skills, we do not describe every skill individually. Instead, the following subsection presents three representative examples spanning the routing, primitive, and workflow levels.

E.6.2 Representative Tool-Calling Skills

Tool-Calling Router

Purpose and Invocation. The Tool-Calling Router is invoked when the runtime needs to select an appropriate tool-use workflow from the current execution state. When invoked from the monitoring path, it may consume an explicit OOD scenario or monitor signal. Its role is to select one available workflow and determine the post-execution intent. The router itself does not execute tools or generate low-level tool calls.
Inputs. The router receives the monitor signal or OOD scenario, its supporting reason, the current subtask, the current observation summary, recovery state, previous tool-calling history, and the set of workflows currently available to the runtime.
Core Procedure.

1. Read the monitor or OOD signal without re-classifying the raw observation.
2. Select exactly one available workflow that best matches the current signal, subtask, and prior tool-calling history. The least disruptive workflow that can safely restore control is preferred.
3. Avoid repeating a workflow that has already failed unless new evidence justifies another attempt.
4. Select a post-execution intent: retry the current subtask, return control to planning, or abort when the state is unsafe or repeated recovery has failed.

Output Contract. The router returns JSON only. The current routing contract is:

```
{
  "selected_skill": "tool-calling-router",
  "OOD_scenario": "grasp_lost",
```

Skill	Responsibility
Tool-Calling Router	Selects one workflow and a post-execution intent from an explicit OOD scenario or monitor signal. It returns a structured routing decision and does not itself execute tools.
Close Gripper	Closes a selected gripper to re-establish or stabilize grasp state after an uncertain, failed, or slipping grasp.
Contact Displace	Applies a very small bounded end-effector displacement when local contact should be released or probed without initiating a new task-level action.
Lift End Effector	Creates bounded vertical end-effector clearance when the arm or gripper is locally blocked or obstructed.
Move EE to Grounded Instance	Moves an end effector toward manipulation geometry associated with a grounded scene-memory instance or a public placement target. Executable pose selection remains runtime-side.
Move EE to Pose	Moves an end effector toward an already validated absolute world-frame target pose or position. It does not perform semantic grounding.
Move to Home	Moves one or both end effectors toward the configured original or home pose before retry, replanning, or termination of an unsafe local rollout.
Open Gripper	Opens a selected gripper to release an unstable or failed grasp state or to prepare for safe retreat.
Reobserve Scene	Refreshes visual and contextual observation after an abnormal rollout state or after another tool changes robot posture or scene visibility.
Retreat Arm	Creates local end-effector clearance after blocked motion, failed grasp, uncertain contact, or a stalled posture.
Safe Reset Posture	Moves the end effector by a bounded step toward a conservative original or home posture when local recovery motion is insufficient.

Table 10: Routing and primitive skills in the Tool Calling category.

```
  "tool_workflow": "recover-grasp-lost",
  "post_execution_intent": "retry",
  "reason": "<short routing rationale>"
}
```

Key Constraints.

- The router assumes that monitoring has already produced the failure signal and does not re-classify raw images.
- It selects exactly one workflow and does not output low-level tool calls.
- The selected workflow must appear in the runtime-provided workflow list.
- If the signal is ambiguous, the router prefers a workflow that refreshes observation or returns control to planning.
- If no workflow fits, the fallback is to return control to planning through the replanning workflow.
- Repeated failed physical recovery should lead to replanning or abort rather than unconditional repetition.

Interaction with Other Components. The router consumes the explicit execution-health signal produced by monitoring and selects the workflow that should handle the current situation. The selected workflow subsequently generates concrete tool calls. Runtime validates the selected workflow before invoking the workflow planner.

Skill	Responsibility
Go Home and Retry	Handles exhausted or unproductive rollouts by constructing a situation-specific tool sequence that returns the robot toward a reusable posture before control is returned to planning.
Recover Grasp Lost	Constructs a bounded tool sequence when the manipulated object is no longer attached to or controlled by the gripper.
Recover Motion Blocked	Constructs a situation-specific tool sequence when local contact or blockage prevents the current motion from continuing safely.
Recover Object Not Visible	Constructs a tool-use plan that improves observability or returns control to planning when a task-relevant object cannot be reliably observed.
Recover Requires Replan	Returns control to planning when the current subtask is no longer valid. It may produce no physical tool calls when physical intervention is unnecessary.
Recover Scene Drift	Refreshes observation and determines whether execution can be retried or should be replanned when the scene has changed enough to invalidate the current rollout context.
Retreat and Reobserve	Handles a locally stalled but potentially retryable subtask by constructing a short tool sequence for clearance and refreshed observation.
Task-Level Tool Control	Uses grounded runtime tools as the temporary low-level controller to make observable progress on the global instruction under the dedicated task-level control mode.

Table 11: Workflow skills in the Tool Calling category. Each workflow constructs a situation-dependent sequence from the runtime tools currently available.

Move End Effector to Grounded Instance

Purpose and Invocation. This primitive is used when a tool-calling procedure needs to move an end effector toward manipulation geometry that is already represented in scene memory. For grasp and contact operations, the planner selects a grounded scene instance. For placement, it selects a public operation target, while runtime binds the transport-authorized held object and resolves the executable arm-specific pose. The planner therefore selects semantic grounding, whereas private pose-candidate selection remains inside the runtime.

Inputs. The primitive uses scene memory, public placement targets, the latest perception evidence, robot state, the current subtask, previous tool-call outcomes, the runtime tool allowlist, and the configured grasp-transport policy. Scene instances may contain grounded operation-pose candidates, but private candidate identifiers are not exposed as planner tool arguments. Public placement targets instead expose stable target identifiers, target type, support and occupancy state, target coordinates, and executable arm choices.

Core Procedure.

1. Determine the manipulation mode. Portable objects intended for later carry or placement use grasp mode, whereas press, push, brace, or articulated interactions use contact mode. Placement uses the dedicated place mode.
2. Select a grounded public reference. Grasp and contact may use an exact scene-instance identity; place uses an exact public operation-target identifier. When identity binding is required, role-based or focus-based selection cannot replace the exact public identity.
3. Select the appropriate grounded point and motion contract. A new portable-object grasp uses the grounded approach pose as the mandatory staging pose before the final grasp pose and gripper closure. Contact uses separately grounded contact geometry. Placement uses the runtime-provided placement target and may first move to its approach pose before issuing the final place motion.

4. Pass the public grounding request to runtime. Runtime resolves the matching private candidate, validates the grounded position and orientation, applies bounded motion limits, and executes the resulting end-effector command.
5. Fail closed when the required identity, finite 3D grounding, position validity, placement support, occupancy state, or transport authority is unavailable.

Output Contract. This skill produces a structured invocation of the runtime tool `move_ee_to_grounded_instance`. The principal arguments are shown below; the runtime interface additionally supports bounded offsets, clearance parameters, focus-based selection when permitted, and gripper preconditions.

```
{
  "tool_name": "move_ee_to_grounded_instance",
  "args": {
    "arm": "left | right",
    "instance_id": "<public scene instance, optional>",
    "action_mode": "grasp | contact | place",
    "target_id": "<public operation target; required for
    ↪  place>",
    "point_key": "<grounded point type>",
    "held_instance_id": "<held instance, optional>",
    "preserve_height": "<boolean, optional>",
    "gripper_precondition": "<requested state, optional>",
    "target_quat_wxyz": "grounded | preserve | <quaternion>",
    "max_translation": "<bounded translation>",
    "steps": "<bounded step count>"
  },
  "reason": "<short tool-call rationale>"
}
```

The runtime result follows the common `RecoveryToolResult` interface. For audit, the runtime may report the resolved operation candidate, operation target, action mode, and geometry source, while the planner never selects private candidate identifiers.

Key Constraints.

- The primitive must use grounded scene-memory geometry and must not invent coordinates.
- Private operation-candidate identifiers never enter planner tool arguments.
- When exact identity binding is required, the public scene-instance identifier must be supplied explicitly.
- A scene instance with motion-uncertain position state cannot be used through its retained historical coordinate.
- A new portable-object grasp must first reach its grounded approach pose before proceeding to its grasp pose and gripper closure, unless recovery history already establishes that the same arm reached that approach for the same grasp attempt.
- Contact interactions with anchored or articulated mechanisms remain contact operations even when the gripper may subsequently close around a handle.
- Place mode uses a public operation-target identifier rather than a destination instance identifier. Runtime recomputes support and occupancy before execution and rejects stale or occupied targets.
- Missing or ambiguous visual evidence does not cause this primitive to insert diagnostic motion, re-observation, gripper opening, retreat, or return-to-grasp actions automatically; such actions must be requested separately when needed.

Interaction with Other Components. This primitive is called by a tool-calling workflow when grounded end-effector motion is required. It consumes public grounding already stored in scene memory and leaves private pose resolution, bounded execution, and validation to the runtime. If an instance cannot be resolved or its geometry is no longer valid, the skill directs the calling procedure toward re-observation or replanning rather than falling back to guessed coordinates.

Recover Grasp Lost

Purpose and Invocation. This workflow is used when monitoring has already identified a lost or unstable grasp, meaning that the manipulated object appears no longer attached to or controlled by the gripper. It is a planning harness for tool use rather than a fixed recovery script: the workflow constructs a situation-specific sequence from the tools currently available and remains within the supplied tool budget.

Inputs. The workflow receives the grasp-loss monitor signal, the interrupted subtask, an observation summary containing object visibility, gripper state, and robot posture, previous recovery tool calls and outcomes, the runtime-provided tool set, the maximum tool-call budget, and the allowed post-recovery intents.

Core Procedure.

1. Assess whether the object remains visible and reachable, whether the gripper is still closed on anything, and whether the arm is near a collision or obstruction.
2. Select only the tools needed for the observed state. The workflow may open the gripper to clear an unstable grasp, retreat or lift the arm to create clearance, move toward an already grounded instance for bounded retry setup, or refresh observation after physical recovery.
3. Repeat observation only when an intermediate physical tool has changed the scene or robot posture.
4. Select retry when the target remains visible and reachable after recovery, replan when the target moved or the original subtask is no longer valid, and abort when recovery tools fail or the scene is unsafe.

Output Contract. The workflow returns JSON only. It produces an ordered list of situation-specific tool calls, together with a post-recovery intent and an explicit stop condition.

```
{
  "selected_workflow": "recover-grasp-lost",
  "tool_calls": [
    {
      "tool_name": "<available runtime tool>",
      "args": {},
      "reason": "<reason for this tool call>"
    }
  ],
  "post_recovery_intent": "retry | replan | abort",
  "stop_condition": "<workflow stop condition>"
}
```

The repository provides reference plans such as opening the gripper, retreating, and re-observing, but these plans are examples rather than mandatory sequences.

Key Constraints.

- The workflow assumes that monitoring has already classified grasp loss and does not perform that classification itself.
- It does not decide whether the global task or subtask has succeeded; it only prepares the runtime for retry, replanning, or abort.
- Only tools present in the runtime-provided tool list may be used, and the complete plan must respect the supplied tool budget.
- Uncertain object or arm state should not trigger repeated grasp-related physical recovery without new evidence.
- When useful for safety, an unstable grasp should be released before a large retreat motion.
- Grounded-instance motion is used only when scene memory already contains the finite 3D instance required for the retry setup.

- If opening the gripper fails, repeated forceful motion is avoided; if retreat fails, lifting is attempted only when that tool is available and safe.
- If the target is no longer visible after re-observation, the workflow returns control to replanning.

Interaction with Other Components. Monitoring identifies the grasp-loss condition, after which the routing layer may select this workflow. Recover Grasp Lost then composes concrete calls from the runtime-provided primitive tools, such as gripper control, retreat, lifting, grounded motion, and re-observation. Its output returns either retry, replan, or abort as the intended control state after the tool sequence.

E.7 Effect Verification Skills

The Effect Verification category contains a single runtime skill for assessing the immediate physical outcome of an executed action or tool sequence from before-and-after evidence. This runtime verification is distinct from the AFK evidence verdict in Section 3.4: the former evaluates the current execution and subtask transition, whereas the latter binds an execution event to persistent knowledge and classifies its evidence as support, opposition, or unverified.

Action Effect Verification

Purpose and Invocation. This skill is invoked after a tool sequence has executed and a fresh observation is available. It determines whether the observed physical effect is consistent with the current subtask and the intended action. The verifier is the after-action authority for the current subtask transition: a pre-execution post-action intent is treated only as a hint, while the resulting subtask status is determined from fresh evidence. The verifier does not introduce a new runtime memory schema; its result is compressed back into the existing recovery history and recent observation summary.

Inputs. The verifier receives the global task, current subtask, proposed post-execution intent, expected outcome, reason for the preceding tool sequence, and the executed tool calls together with their arguments and runtime results. It additionally receives before- and after-action evidence containing robot state, scene memory, observation summaries, and runtime evaluation, as well as recent recovery history. For placement operations, the runtime may additionally provide deterministic placement validation covering support and occupancy, target position, gripper state, end-effector clearance, and cross-observation stability.

Core Procedure.

1. Infer the intended physical effect from the current subtask and executed tool sequence rather than from task-specific object-class rules or planner prose.
2. Compare before- and after-action evidence using current grounded geometry, track continuity, robot and gripper state, monitor signals, and measured state changes. Current object positions are taken from the current observation fields; history-smoothed positions are used for identity continuity rather than measuring the displacement caused by the latest action.
3. Apply effect-specific verification when required. For grasp or carry, compare object motion with end-effector motion and their relative coupling. For placement, require the runtime placement validator to report a verified placement before accepting release or subtask completion. For contact-based transient actions, verify a complete grounded contact-and-release cycle from the same arm rather than inferring repetitions from controller steps or traveled distance.
4. Classify the action effect as verified, contradicted, or unverified according to the available evidence.
5. Determine the current subtask status from fresh physical evidence and select the corresponding control recommendation for continued execution, retry, or replanning.

Output Contract. The verifier returns JSON only and never returns additional tool calls.

```
{
  "effect_verified": "true | false | unverified",
  "effect_type":
    "grasp | place | release | push | press | move |
     align | open | close | cover | uncover | unknown",
  "confidence": 0.0,
  "evidence_summary": "<short evidence-based summary>",
  "failure_reason": "",
  "next_constraint": "<constraint for the next plan>",
  "memory_update": "<compact recovery-history update>",
  "subtask_status":
    "in_progress | completed | failed | uncertain",
  "recommended_control": "continue | retry | replan"
}
```

The memory update is kept short and action-oriented rather than copying raw observations or scene-memory records into persistent memory.

Key Constraints.

- Successful tool execution is not sufficient evidence that the intended physical effect occurred.
- A closed gripper or a successful close command does not by itself establish that an object is being held.
- Perception-query metadata, task wording, and planner rationales describe intent and must not be used as direct evidence for physical relations such as holding, contact, release, coverage, or clearance.
- Current action effects are measured from current before-and-after grounding. Historical reference poses and history-smoothed positions must not substitute for the current object position when measuring instantaneous motion.
- Placement is verified only when the runtime placement validator reports `verified=true`. A successful final move or gripper-open command alone is insufficient.
- If placement validation reports that recovery is required after release, the subtask is treated as failed and control is returned for replanning rather than repeatedly requesting observation.
- An empty tool sequence with a direct replanning intent does not imply that a new physical effect occurred. The verifier evaluates whether current evidence supports the claimed handoff state.
- One verified transient contact cycle counts as exactly one event; controller steps, internal repetitions, or traveled distance do not create additional verified repetitions.
- `subtask_status=completed` requires fresh evidence of the current subtask's physical stop condition. Requested intent or successful tool return is insufficient.
- Missing, occluded, contradictory, or insufficient evidence produces an *unverified* result rather than being forced into success or failure.
- The runtime environment-success signal remains authoritative for global task completion and may bypass this verifier once global success has already been established.

Interaction with Other Components. This skill is applied after an action or tool sequence has executed and fresh post-action evidence has been collected. Its immediate result is used by the runtime to update the local subtask state and determine whether execution should continue, retry, or return to planning. The verification record may also become part of the execution event subsequently used by the AFK evidence-assessment pipeline; AFK maintenance separately determines whether that event supports, opposes, or leaves unverified a persistent knowledge entry.