FROM FAILURES TO EVOLUTION: BEHAVIORAL RUBRICS FOR AGENT HARNESS EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly rely on harnesses that manage context and execution strategies beyond the underlying model. Harness evolution provides a promising approach to improving agent capabilities without updating model parameters. Existing approaches increasingly leverage execution evidence and structured feedback to improve agents. However, they usually lack an intermediate behavioral specification that separates **what the agent should do** from **how the harness realizes it**. Such a specification should capture desired behaviors, applicable contexts, and boundaries while remaining independent of specific harness implementations. We propose **rubric-guided harness evolution** (RHE), which introduces **behavioral rubrics** as an implementation-agnostic interface between execution evidence and harness modifications. We define behavioral rubrics as specifications of desired agent behaviors, applicable contexts, and boundary constraints. By separating behavioral objectives from harness realizations, rubrics enable different implementations of the same behavioral requirement. Hierarchical experience memory further preserves the relationships among execution evidence, rubrics, and historical realizations, enabling reuse across evolution iterations. We evaluate our approach on **SWE-bench Verified** and **AppWorld**. Across three backbone models and two benchmarks, RHE consistently improves agent performance over existing agent improvement approaches. Cross-environment transfer experiments further show that behavioral rubrics discovered from SWE-bench Verified remain effective when transferred to AppWorld. These results demonstrate that behavioral rubrics enable generalizable harness evolution beyond trajectory-specific optimization.

1 INTRODUCTION

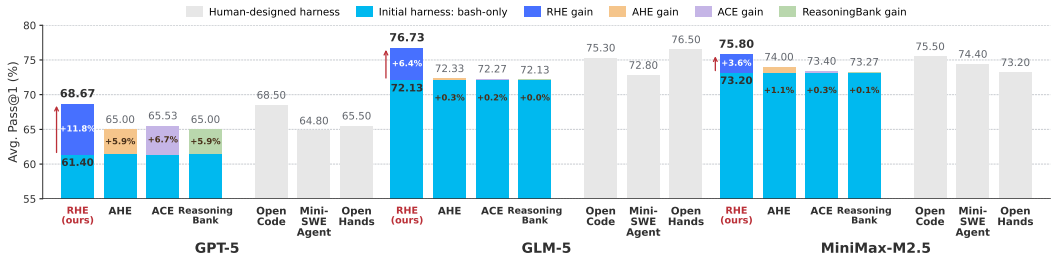

Figure 1: **Harness improvement over a shared bash-only initial harness on SWE-bench Verified.** RHE/AHE perform harness evolution, while ACE/ReasoningBank update context or memory. Gray bars show measured human-designed harness results. Percentages indicate gains over the initial harness.

Large language model (LLM) agents are increasingly deployed in complex interactive environments (Yang et al., 2024; Xu et al., 2026; Shen, 2024). Their capabilities depend not only on the reasoning ability of the underlying model but also on the harness that manages context, tool interactions, and execution strategies (Lopopolo, 2026; Bölük, 2026; Böckeler, 2026; Young, 2025). Improving harnesses while keeping model parameters unchanged therefore provides a promising direction for continually enhancing agent systems.

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.

Recent studies have explored harness evolution, where agents analyze execution feedback and generate modifications to improve future performance (Lee et al., 2026; Lin et al., 2026; Lou et al., 2026; Wu et al., 2026; Jiang et al., 2026). However, effective harness evolution requires an intermediate representation of reusable behavioral objectives, rather than directly mapping observed issues to concrete harness modifications. Existing approaches leverage execution evidence and structured feedback, but do not explicitly separate what behavior should change from how that behavior is implemented.

Execution traces provide rich evidence of agent actions and environmental feedback (Yao et al., 2023; Yang et al., 2024), but transferring such evidence into reusable behavioral requirements requires an additional abstraction step. For example, repeatedly rerunning a failed regression test without changing the relevant code or investigating the failure reveals a broader behavioral requirement: retries should be justified by meaningful changes or diagnostic progress. Such requirements should guide future agent behavior without prescribing a specific implementation.

We propose **rubric-guided harness evolution** (RHE), which introduces **behavioral rubrics** as an intermediate design interface between execution evidence and harness modifications (Figure 2). A behavioral rubric specifies what behavior should change, when it applies, and its boundary constraints. By separating behavioral objectives from concrete harness realizations, rubrics allow the same requirement to guide diverse modifications across tools, prompts, memory, and other harness components.

To support continual evolution, we further introduce a **hierarchical experience memory** that preserves the provenance among execution evidence, behavioral rubrics, and historical harness realizations with their evaluation outcomes (Figure 2). Retrieved rubrics and realizations provide references for future modifications, while newly collected evidence and results update the memory for subsequent evolution rounds.

We evaluate RHE on **SWE-bench Verified** and **AppWorld** with three backbone models. Our evaluation addresses three questions: (1) whether behavioral rubrics enable harness modifications that generalize beyond individual execution trajectories, (2) whether rubric-guided evolution improves agent task-solving performance, and (3) whether behavioral rubrics transfer across environments.

Our main contributions are as follows:

- We propose **behavioral rubric-guided harness evolution**, which introduces behavioral rubrics as an implementation-agnostic interface between execution evidence and harness realizations. By abstracting trajectory-level observations into reusable behavioral requirements, RHE enables harness evolution beyond directly optimizing individual edits.
- We introduce a **hierarchical experience memory** that preserves the relationships among execution evidence, behavioral rubrics, and historical harness realizations with their evaluation outcomes, enabling reuse of behavioral requirements and realization strategies across evolution rounds.
- We conduct extensive experiments on **SWE-bench Verified** and **AppWorld** with three backbone models, demonstrating improved agent performance, diverse harness realizations, and cross-environment transfer of behavioral rubrics.

2 Related Work

2.1 Agent Harnesses and Scaffolded Agents

Large language model agents rely on external scaffolding systems that mediate interaction with environments through reasoning protocols, tools, context management, and execution control. Early agent frameworks such as ReAct (Yao et al., 2023), Toolformer (Schick et al., 2023), and ToolLLM (Qin et al., 2023) demonstrate the importance of external interaction mechanisms, while subsequent systems explore richer planning and execution scaffolds (Wu et al., 2024; Hong et al., 2023; Wang et al., 2024). In software engineering, SWE-agent (Yang et al., 2024) further shows that agent interfaces and execution feedback substantially affect coding-agent performance. These studies establish harness design as an important factor beyond the underlying model itself.

2.2 Continual Self-Improvement and Experience Memory

Continual self-improvement enables agents to improve future behavior through feedback and accumulated experience. Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) leverage iterative reflection, while Voyager (Wang et al., 2023) learns reusable skills from interaction trajectories. Memory-based approaches further study experience storage and reuse through long-term memory architectures (Park et al., 2023; Packer et al., 2023), continual experience refinement (Guo et al., 2025), reasoning memories (Ouyang et al., 2025), and evolving context playbooks (Zhang et al., 2025). These approaches mainly improve agents through retrieved information or guidance supplied to the model. In contrast, we use experience memory to preserve the provenance among execution evidence, behavioral requirements, and harness realizations for continual harness evolution.

2.3 Automated Harness Evolution

Recent studies investigate automatically improving the harness itself. Meta-Harness (Lee et al., 2026) and AutoHarness (Lou et al., 2026) explore automated harness optimization through iterative search and feedback. Agentic Harness Engineering (AHE) (Lin et al., 2026) introduces an observability-driven framework that extracts structured trajectory evidence and evolves multiple harness components, including prompts, tools, middleware, and memory. HarnessDev (Wu et al., 2026) studies whether agents can create and evolve their own harnesses, while HarnessEvolve (Jiang et al., 2026) investigates iterative harness self-evolution through automated modification and evaluation. Co-Harness (Chen et al., 2026) further explores joint optimization of harness components and model parameters.

These works collectively establish automated harness evolution as a promising paradigm. However, an open challenge remains: how to represent reusable behavioral objectives independently of specific harness implementations. Our work introduces behavioral rubrics as an intermediate design interface that abstracts execution evidence into reusable behavioral requirements with applicable contexts and boundaries, allowing the same objective to guide diverse harness realizations.

3 Methodology

3.1 Overview

We propose **behavioral rubric-guided harness evolution**, which introduces behavioral rubrics as an intermediate design interface between execution evidence and harness realizations.

As illustrated in Figure 2, the framework identifies behavioral deficiencies in execution trajectories and abstracts the resulting diagnoses into evidence-grounded behavioral rubrics. Each current rubric retrieves similar historical rubrics from hierarchical experience memory. Their linked execution evidence and prior harness realizations provide references for editing. In the figure's rubric-guided modification stage, the editor uses the current rubric and these references to modify executable components such as tools and middleware (*structural modifications*), instructions and contextual information such as system prompts and agent memory (*guidance modifications*), or both. The resulting candidate harnesses are evaluated on the evolution set, and the best-performing candidate is retained for the next iteration. After N evolution rounds, the retained harness H_N is returned. A round without an accepted improvement retains the current harness and continues to the next round.

The key design separates *what agent behavior should change* from *how the harness implements that change*. The center of Figure 2 depicts the links used to retrieve evidence and prior realizations for the current harness edit.

3.2 Behavioral Rubric Construction

At iteration t, executing the current harness H_t with model M on task x produces a trajectory

$$\tau_x^t = \mathrm{Run}(M, H_t, x). \tag{1}$$

The system analyzes this trajectory to identify the observed behavioral deficiency, its execution context, and supporting evidence. It then abstracts the diagnosis into a behavioral rubric

$$\rho = (g_\rho, a_\rho, b_\rho, \mathcal{E}_\rho), \tag{2}$$

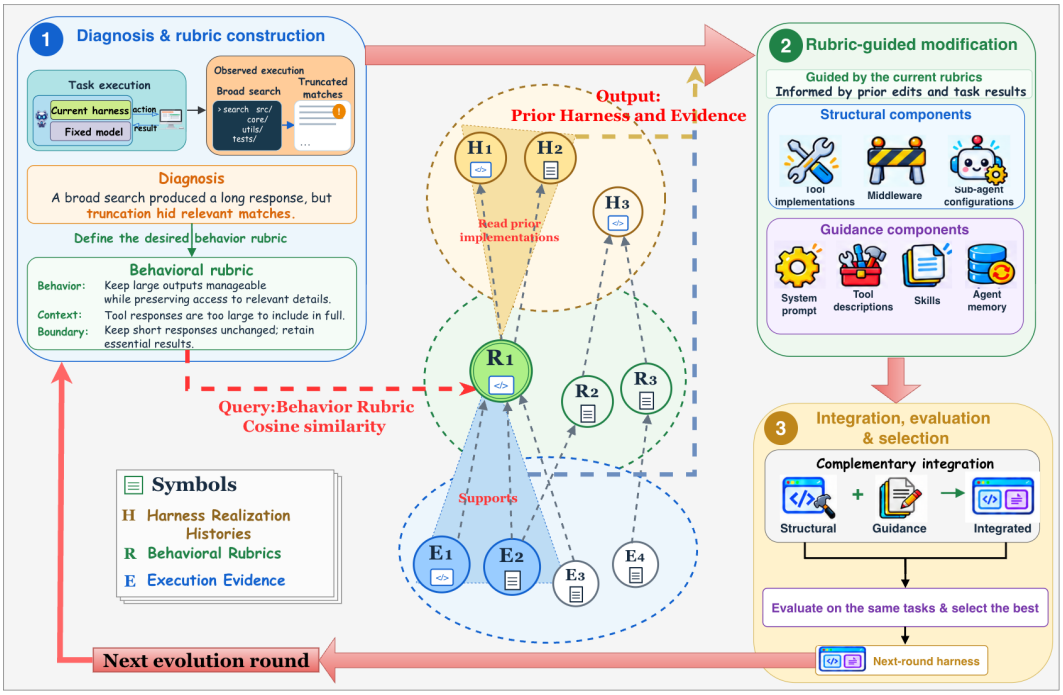

Figure 2: **Rubric-guided harness evolution with hierarchical experience memory.** (1) Execution trajectories are diagnosed and abstracted into behavioral rubrics with supporting evidence. (2) Each rubric serves as a query for similarity retrieval over stored rubrics. Evidence, prior harness realizations, and evaluation outcomes linked to retrieved rubrics are provided to structural and guidance editors. (3) Candidate harnesses, including compatible combinations of edits, are evaluated on the evolution set to select the next harness. The memory stores evidence–rubric and rubric–realization links across iterations.

where g_ρ specifies the desired behavior, a_ρ defines its applicable context, b_ρ describes boundary constraints, and $\mathcal{E}_\rho$ records the supporting execution evidence.

Unlike a trajectory-specific failure description, a behavioral rubric specifies the underlying behavioral change and the conditions under which it applies, without prescribing a particular harness realization. During continual evolution, new evidence may lead to rubric reuse, refinement, or extension. This process maintains reusable behavioral specifications as observations accumulate. The diagnosis schema and rubric management procedure are detailed in Appendix E.1.

3.3 Harness Realization Guided by Behavioral Rubrics

At each iteration, structural and guidance editors start from the same parent harness H_t and frozen rubric state. The selected rubrics specify behavioral objectives; retrieved execution evidence grounds their applicability, while historical realizations provide implementation references (Section 3.4). For realization type $\beta \in \{\text{str}, \text{gui}\}$, let $\mathcal{R}_t^\beta$ denote the rubrics selected for realization type β and $\mathcal{K}_t^\beta$ their retrieved context. The editor proposes

$$\Delta_t^\beta = G_\beta(H_t, \mathcal{R}_t^\beta, \mathcal{K}_t^\beta). \tag{3}$$

Each round allocates one structural and one guidance candidate, both generated from H_t and the same frozen rubric state. If the two edits are complementary and mechanically compatible, an integrator constructs one additional combined candidate. Thus a round has two standalone candidates and at most one combined candidate. Each is evaluated once on the held-in set for selection. If integration retains a standalone candidate, its existing score is reused without reevaluation. For example, a rubric

requiring verification before repeated actions may lead to execution-monitoring middleware or to guidance that asks the agent to verify relevant state changes before retrying. The realization space and candidate construction procedure are detailed in Appendix E.2.

3.4 Continual Evolution with Hierarchical Experience Memory

Because behavioral rubrics abstract away from individual trajectories, continual evolution requires preserving their execution grounding and realization history. We therefore maintain a hierarchical experience memory that links execution evidence, behavioral rubrics, and historical harness realizations.

Figure 2 illustrates this organization. Execution evidence records the observations that motivate and delimit each rubric; historical realizations record alternative implementations of the rubric's behavioral objective.

At each iteration, a current rubric ρ queries the historical rubric memory $\mathbb{M}_t^R$. With a frozen text encoder f_θ and text concatenation $\oplus$, the retrieval is

$$\begin{aligned} v_\rho &= f_\theta(g_\rho \oplus a_\rho \oplus b_\rho), \\ \mathcal{N}_t(\rho) &= \mathrm{TopK}_{\rho' \in \mathbb{M}_t^R} \cos(v_\rho, v_{\rho'}). \end{aligned} \tag{4}$$

The retrieval depth is three for each current rubric assembled from the round's trajectory diagnoses. The three retrieved rubrics, including their applicable contexts and boundaries, and their linked evidence and historical harness realizations with evaluation outcomes are passed to the editors as references. New evidence, rubric revisions, and evaluated realizations are subsequently added to memory. Appendix E.3 details the memory schema and retrieval procedure, and Figure 6 illustrates its retrieval-to-edit pathway in a case study.

4 Experiments

4.1 Experiment Setup

Benchmarks and Comparisons. We evaluate harness evolution and experience-based agent improvement on SWE-bench Verified (OpenAI, 2024) and AppWorld (Trivedi et al., 2024) with GPT-5 (OpenAI, 2025), GLM-5 (GLM-5 Team, 2026), and MiniMax-M2.5 (MiniMax AI, 2026). For each model and benchmark, all four agent-improvement methods start from the same single-tool harness and use identical task partitions. The tool is a shell interface on SWE-bench Verified and `execute_code` on AppWorld. We compare RHE with three agent improvement approaches: AHE (Lin et al., 2026), ReasoningBank (Ouyang et al., 2025), and ACE (Zhang et al., 2025). AHE represents iterative harness evolution, whereas ReasoningBank and ACE represent experience-based adaptation through memory and context updates. The shared initial harness provides a pre-update reference. We also evaluate OpenCode, Mini-SWE-Agent, and OpenHands as human-designed reference harnesses on SWE-bench Verified (Figure 1). Appendix B.3 describes the update procedures; task partitions and evolution settings are detailed in Appendix B.1 and Appendix B.5.

Evaluation Protocol. For each benchmark, tasks are partitioned into a held-in set for method updates and a held-out set reserved for final evaluation. RHE and AHE evolve their harnesses on held-in tasks; ReasoningBank and ACE use the same partition for their respective experience updates. All update procedures, including experience extraction and memory or context updates, are restricted to the held-in split. Each evolved harness is frozen and evaluated anew on both splits. Main comparison scores summarize three evaluations of each frozen evolved harness. Ablation and transfer scores instead summarize three independent evolution runs. Task partitions, metrics, and aggregation rules are detailed in Appendices B.1, B.4, and A.

4.2 Overall Agent Improvement Performance

Table 1: **Overall agent improvement performance.** Within each model and benchmark, all methods share the same single-tool initial harness (shell on SWE-bench Verified; `execute_code` on AppWorld). RHE and AHE perform ten harness evolution rounds; ReasoningBank and ACE follow their respective experience-update procedures. The right-hand columns report performance of each method's evolved harness. Success rates are percentages (mean ± sample SD over three evaluations); Overall is weighted by task count.

Model	Method	Initial harness (Single Tool)			Evolved harness		
		Held-in	Held-out	Overall	Held-in	Held-out	Overall
				SWE-bench Verified			
GLM-5	*AHE*	72.67±0.67	71.90±0.44	72.13±0.31	75.33±1.15	71.05±1.57	72.33±1.01
	ReasoningBank	72.67±0.67	71.90±0.44	72.13±0.31	74.00±0.67	71.33±1.00	72.13±0.81
	ACE	72.67±0.67	71.90±0.44	72.13±0.31	72.89±0.77	72.00±0.76	72.27±0.70
	RHE (Ours)	72.67±0.67	71.90±0.44	72.13±0.31	**76.89±0.38**	**76.67±1.08**	**76.73±0.64**
GPT-5	*AHE*	61.33±0.67	61.43±0.29	61.40±0.35	66.00±1.15	64.57±1.14	65.00±0.87
	ReasoningBank	61.33±0.67	61.43±0.29	61.40±0.35	66.44±0.38	64.38±0.72	65.00±0.60
	ACE	61.33±0.67	61.43±0.29	61.40±0.35	67.11±1.02	64.86±0.86	65.53±0.83
	RHE (Ours)	61.33±0.67	61.43±0.29	61.40±0.35	**69.11±1.02**	**68.48±0.92**	**68.67±0.95**
MiniMax-M2.5	*AHE*	71.33±0.67	74.00±0.49	73.20±0.40	75.33±1.33	73.43±1.43	74.00±1.25
	ReasoningBank	71.33±0.67	74.00±0.49	73.20±0.40	74.67±1.15	72.67±1.19	73.27±0.99
	ACE	71.33±0.67	74.00±0.49	73.20±0.40	74.00±1.33	73.14±1.31	73.40±1.22
	RHE (Ours)	71.33±0.67	74.00±0.49	73.20±0.40	**77.33±0.67**	**75.14±0.57**	**75.80±0.53**
				AppWorld			
GLM-5	*AHE*	73.33±1.11	76.67±1.11	75.00±0.56	90.00±1.11	86.67±1.11	88.33±0.96
	ReasoningBank	73.33±1.11	76.67±1.11	75.00±0.56	89.63±0.64	89.63±0.64	89.63±0.32
	ACE	73.33±1.11	76.67±1.11	75.00±0.56	83.70±0.64	82.59±1.28	83.15±0.85
	RHE (Ours)	73.33±1.11	76.67±1.11	75.00±0.56	**91.85±1.70**	**92.59±0.64**	**92.22±0.56**
GPT-5	*AHE*	89.26±1.28	89.63±0.64	89.44±0.56	91.85±0.64	91.48±1.28	91.67±0.56
	ReasoningBank	89.26±1.28	89.63±0.64	89.44±0.56	90.37±0.64	89.63±0.64	90.00±0.56
	ACE	89.26±1.28	89.63±0.64	89.44±0.56	91.11±1.11	90.74±1.28	90.93±0.85
	RHE (Ours)	89.26±1.28	89.63±0.64	89.44±0.56	**93.70±1.28**	**93.33±1.11**	**93.52±1.16**
MiniMax-M2.5	*AHE*	56.67±1.11	49.63±1.70	53.15±0.32	64.44±1.92	55.19±1.28	59.81±0.85
	ReasoningBank	56.67±1.11	49.63±1.70	53.15±0.32	64.07±0.64	58.52±0.64	61.30±0.32
	ACE	56.67±1.11	49.63±1.70	53.15±0.32	63.70±1.28	52.96±1.70	58.33±0.96
	RHE (Ours)	56.67±1.11	49.63±1.70	53.15±0.32	**66.67±1.11**	**60.37±1.28**	**63.52±1.16**

Overall Performance. Table 1 compares RHE with AHE, ReasoningBank, and ACE. The RHE overall scores on SWE-bench Verified are 68.67%, 76.73%, 75.80% for GPT-5, GLM-5, and MiniMax-M2.5, respectively; the corresponding AppWorld values are 93.52%, 92.22%, 63.52%. Across both benchmarks and all three backbone models, RHE consistently provides the largest improvement in overall success over the shared initial harness among the compared methods. Relative to the shared initial harness, RHE's gains show that modifying the agent harness can enhance performance without changing model weights. Figure 1 uses the same SWE-bench scores.

Evolution Dynamics. Figure 3 follows RHE and AHE over ten harness evolution rounds with MiniMax-M2.5, starting from the shared 71.33% held-in initial score. RHE reaches 77.33% at round 4, while AHE reaches 75.33% at round 5. Four RHE improvements are annotated with their behavioral rubrics and corresponding harness realizations. Table 7 separates per-seed evaluation from across-seed aggregation. Seed 1 reuses the corresponding RHE evaluations from Table 1.

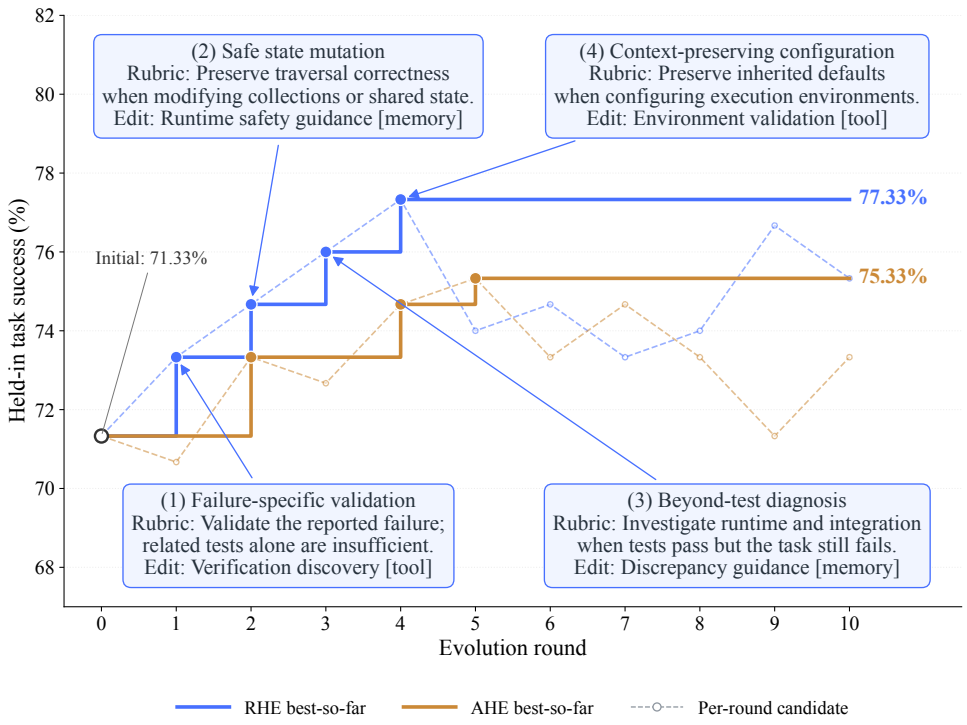

Figure 3: **Harness evolution dynamics.** Held-in task success rates (%) on SWE-bench Verified for RHE and AHE under the MiniMax-M2.5 backbone over ten evolution rounds, starting from the same bash-only harness. Solid lines show best-so-far scores and dashed trajectories show per-round scores. Annotations (1)–(4) pair behavioral requirements with concrete harness modifications; brackets indicate the realization categories of selected modifications.

Additional analysis of per-task deployment token usage is provided in Appendix D.7.

4.3 Ablation Studies

Effect of Behavioral Abstraction and Experience Reuse. Table 2 studies the effects of behavioral abstraction and experience reuse. RHE achieves the highest held-out (75.17%) and overall (75.82%) success, outperforming trajectory-driven editing and RHE without memory.

Table 2: **Ablation of behavioral abstraction and experience reuse.** Task success rates (%) under the MiniMax-M2.5 backbone. Evolved rows show mean ± SD across three independent runs; the initial row is a fixed, pre-evolution reference based on repeated evaluations. Trajectory-driven Editing uses trajectory-based diagnosis and edit refinement without behavioral rubrics.

Method	Experience reuse	Best round median [range]	Held-in	Held-out	Overall
Initial Harness	–	0	71.33±0.67	74.00±0.49	73.20±0.40
Trajectory-driven Editing	×	7 [6–8]	75.41±0.64	73.27±0.71	73.91±0.67
RHE w/o Memory	×	6 [5–7]	76.74±0.46	74.70±0.63	75.31±0.54
RHE	✓	**4 [3–5]**	**77.33**±0.22	**75.17**±0.15	**75.82**±0.04

Controlled Comparisons. Trajectory-driven editing improves over the initial harness, indicating that execution feedback can identify useful harness changes. Relative to this baseline, rubric-guided abstraction raises overall success from 73.91% to 75.31%, supporting behavioral rubrics as an

intermediate interface between behavioral diagnosis and harness modification. Adding cross-round experience reuse further raises overall success to 75.82% and advances the median held-in best round from 6 to 4 under the same ten-round budget.

Effect of Harness Realization Spaces. This ablation varies the permitted realization space among guidance-only, structural-only, and combined editing while keeping rubric construction, experience reuse, the ten-round evolution budget, candidate-generation and evaluation budgets, and selection procedure fixed (Appendix D.5).

Table 3: **Ablation of harness realization spaces.** Task success rates (%) under the MiniMax-M2.5 backbone are mean ± SD across three independent evolution runs.

Method	Best round median [range]	Held-in	Held-out	Overall
RHE (Guidance-only)	6 [5–7]	75.78±0.59	73.90±0.69	74.47±0.60
RHE (Structural-only)	5 [4–6]	76.59±0.46	74.60±0.61	75.20±0.55
RHE (Combined)	**4 [3–5]**	**77.33**±0.22	**75.17**±0.15	**75.82**±0.04

Table 3 shows that combined realization achieves the highest held-out and overall scores, exceeding guidance-only and structural-only by 1.35 and 0.62 overall percentage points, respectively. Under matched evolution and candidate-evaluation budgets, this supports allowing the same behavioral-rubric interface to guide both structural and guidance modifications.

4.4 Analysis

4.4.1 Rubric-Guided Harness Modification Analysis

We analyze how behavioral rubrics guide harness modification generation and how generated edits are subsequently selected.

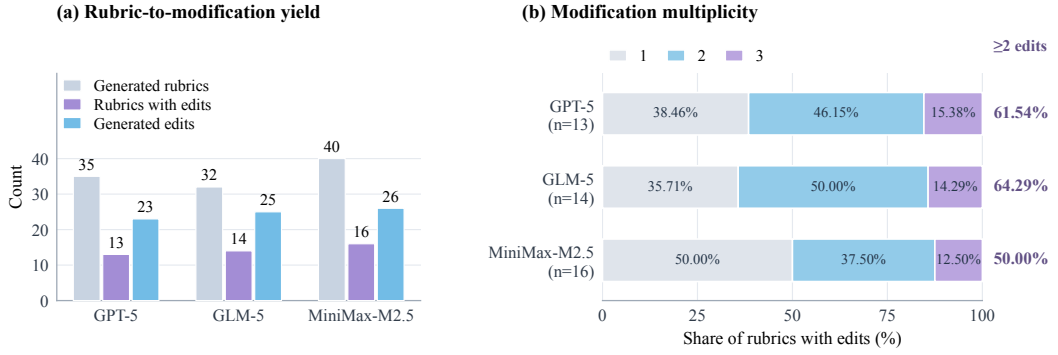

Figure 4: **Rubric-guided harness modification generation.** (a) Counts of generated rubrics, rubrics yielding at least one concrete edit, and generated edits. (b) Distribution of one, two, or three edits per rubric yielding an edit; n counts these rubrics, and right-hand labels give the share yielding at least two edits. Counts include generated edits regardless of harness selection and do not distinguish component types.

(a) Rubric-to-modification yield. Figure 4(a) shows that 37.14–43.75% of generated rubrics lead to at least one concrete harness modification; among them, 1.625–1.786 modifications are generated per rubric on average. These modifications are further filtered through harness selection.

(b) One-to-many modification generation. Figure 4(b) shows that 61.54%, 64.29%, and 50.00% of rubrics with generated modifications are associated with at least two edits for GPT-5, GLM-5, and MiniMax-M2.5, respectively. This multiplicity is consistent with a rubric guiding several concrete edits associated with the same behavioral requirement.

Across rubrics, these modifications span different harness component types; component-level generation and retention statistics are reported in Appendix D.6. Appendix F presents two cases connecting execution evidence and frozen rubrics to guidance and structural harness modifications.

4.4.2 Cross-Environment Transfer

To investigate whether behavioral rubrics capture transferable behavioral abstractions, we study transfer from SWE-bench Verified to AppWorld. Of the 40 source rubrics, prespecified human criteria select six, while an independent MiniMax-M2.5 judge selects eight using a fixed prompt. Both selections are fixed before inspecting AppWorld held-out results (Appendix D.2). We transfer only behavioral rubrics, excluding source task solutions, trajectories, harness edits, and code.

Table 4: **Cross-environment transfer of behavioral rubrics from SWE-bench Verified to AppWorld.** Task success rates (%) under the MiniMax-M2.5 backbone. The initial-harness row reports three evaluations of the fixed harness; evolved-row SDs summarize three independent evolution runs.

Method	Imported rubrics	Held-in	Held-out	Overall
Initial harness (pre-evolution)	—	56.67 ± 1.11	49.63 ± 1.70	53.15 ± 0.32
RHE	0	66.79 ± 0.21	60.49 ± 0.21	63.64 ± 0.11
RHE + All Rubrics	40	64.32 ± 0.57	59.26 ± 0.64	61.79 ± 0.57
RHE + Human-selected Rubrics	6	67.28 ± 0.43	61.60 ± 0.77	64.44 ± 0.56
RHE + LLM-selected Rubrics	8	67.04 ± 0.37	61.11 ± 0.64	64.07 ± 0.49

Table 4 includes the shared pre-evolution harness as a reference and compares target-only RHE with AppWorld evolution initialized from all source rubrics or the human- and LLM-selected subsets. Human selection raises held-out and overall success to 61.60% and 64.44%, versus 60.49% and 63.64% for target-only RHE. LLM selection also yields higher observed success (61.11% held-out; 64.07% overall), whereas importing all 40 rubrics yields 59.26% held-out and 61.79% overall. The LLM-selected condition provides a nonhuman applicability-screening comparison; together, these results indicate that applicability-aware selection can improve rubric transfer compared with indiscriminate import.

5 Conclusion

We introduced behavioral rubric-guided harness evolution, a framework that separates reusable behavioral specifications from concrete harness realizations. By transforming execution evidence into behavioral rubrics, RHE guides the generation and selection of harness modifications through reusable behavioral objectives rather than directly coupling observations with task-specific edits.

We further introduced hierarchical experience memory that preserves the relationships among execution evidence, behavioral rubrics, and historical harness realizations, enabling retrieval and reuse of previously discovered behavioral requirements and realizations across evolution iterations.

Experiments on SWE-bench Verified and AppWorld demonstrate that RHE consistently improves harness performance across backbone models and supports applicability-aware cross-environment transfer of behavioral rubrics. Further analysis shows that behavioral rubrics can guide multiple concrete realizations across harness components while remaining independent of specific implementations.

These findings highlight harness evolution as a complementary axis for improving agent capability, where reusable behavioral abstractions enable continual improvement without modifying the underlying model.

AI USE STATEMENT

The large language model (LLM) was utilized as a writing assistant during the preparation of this manuscript. Its application was strictly limited to improving the clarity and grammatical accuracy of the text. Specific uses included rephrasing sentences for better flow and translating initial concepts and drafts from Chinese to English. All core scientific contributions, including the conceptualization of our Behavioral Rubric-guided Harness Evolution framework, the design of the methodology and experiments, and the analysis and interpretation of the results, are solely the work of the authors. The authors take full responsibility for all claims and the final content of this paper.

ETHICS STATEMENT

The Behavioral Rubric-guided Harness Evolution framework modifies agent prompts, memory, tools, and middleware. An evolved harness may therefore introduce unintended behavior even when it improves benchmark success. Our experiments use benchmark task environments and do not establish safety in production settings. Applications involving sensitive code or data should protect stored execution evidence and subject generated harness changes to sandboxed evaluation and human review before deployment.

REPRODUCIBILITY STATEMENT

Appendix B documents the benchmark partitions, initial harnesses, model request limits, comparison procedures, and evaluation protocol. Appendix G provides the prompt templates for rubric construction, harness editing, and transfer selection. We distinguish repeated evaluations of a frozen harness from independent evolution runs and specify their aggregation in Appendix A; individual full-RHE run results appear in Appendix C. Held-out tasks are excluded from method updates and harness selection.

REFERENCES

Birgitta Böckeler. Harness engineering—first thoughts. martinfowler.com, 2026. URL https://martinfowler.com/articles/exploring-gen-ai/harness-engineering-memo.html. February 17, 2026. Accessed September 15, 2026.

Can Bölük. We improved 15 LLMs at coding in one afternoon. only the harness changed. Stencil, 2026. URL https://stencil.so/blog/the-harness-problem. February 12, 2026. Accessed September 15, 2026.

Zhengyu Chen, Teng Xiao, Huaisheng Zhu, Yige Yuan, Luan Zhang, and Jingang Wang. Co-Harness: Co-evolving harnesses and model weights for LLM agents. *arXiv preprint arXiv:2607.22688*, 2026. URL https://arxiv.org/abs/2607.22688.

GLM-5 Team. Glm-5: From vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Yifu Guo, Jiaye Lin, Huacan Wang, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, and Mingguang Chen. SE-Agent: Self-evolution trajectory optimization in multi-step reasoning with LLM-based agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3877. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a911e543a95493ae5004fdc01909043e-Abstract-Conference.html.

Sirui Hong et al. Metagpt: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

Wen Jiang, Mingmin Chu, Yimeng Tian, Qianxin Zhang, Haofei Yang, Rui Yang, Yang Liu, Tao Lv, and Fangming Li. HarnessEvolve: Learning from reference trajectories for reliable agent self-evolution. *arXiv preprint arXiv:2609.00829*, 2026. URL https://arxiv.org/abs/2609.00829v1.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026. URL https://arxiv.org/abs/2603.28052.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026. URL https://arxiv.org/abs/2604.25850v4. Version 4.

Ryan Lopopolo. Harness engineering: leveraging Codex in an agent-first world. OpenAI, 2026. URL https://openai.com/index/harness-engineering/. February 11, 2026. Accessed September 15, 2026.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. AutoHarness: improving LLM agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026. URL https://arxiv.org/abs/2603.03329.

Aman Madaan, Niket Tandon, Prakhar Gupta, et al. Self-refine: Iterative refinement with self-feedback. In *NeurIPS*, 2023.

MiniMax AI. Minimax-m2.5. https://github.com/MiniMax-AI/MiniMax-M2.5, 2026.

OpenAI. Introducing SWE-bench Verified. https://openai.com/index/introducing-swe-bench-verified/, 2024. Published August 13, 2024; updated February 24, 2025.

OpenAI. GPT-5 system card. OpenAI, 2025. URL https://openai.com/index/gpt-5-system-card/. Published August 7, 2025.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025. URL https://arxiv.org/abs/2509.25140.

Charles Packer et al. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Yujia Qin et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, et al. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023.

Sentence Transformers. sentence-transformers/all-MiniLM-L6-v2 model card. Hugging Face, 2021. URL https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2/blob/acbb28c8aa70f5503c85d6b90e8cd65606993a20/README.md. Model card revision dated August 30, 2021. Accessed: 2026-09-26.

Zhuocheng Shen. LLM with tools: A survey. *arXiv preprint arXiv:2409.18807*, 2024. URL https://arxiv.org/abs/2409.18807.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366v4. Version 4.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Guanzhi Wang et al. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Xingyao Wang et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Yuhao Wu, Jingyuan Zhang, Jiajun Shi, Xinping Lei, Qingshui Gu, Yuxuan Zhang, Zexuan Wang, Chen He, Chen Huang, Maojia Song, Zhiyuan Zeng, Shaowen Wang, Jinkai Liu, Yunfeng Shi, Jiaheng Liu, Shen Yan, Wenhao Huang, Ge Zhang, and Wenxuan Zhang. HarnessDev: Can LLMs create and evolve their own agent harness? *arXiv preprint arXiv:2609.01437*, 2026. URL https://arxiv.org/abs/2609.01437.

Zhangchen Xu, Junda Chen, Yue Huang, Dongfu Jiang, Jiefeng Chen, Hang Hua, Zijian Wu, Zheyuan Liu, Zexue He, Lichi Li, Shizhe Diao, Jiaxin Pei, Jinsung Yoon, Hao Zhang, Mengdi Wang, Radha Poovendran, Misha Sra, Alex Pentland, and Zichen Chen. AutoLab: Can frontier models solve long-horizon auto research and engineering tasks? *arXiv preprint arXiv:2606.05080*, 2026. URL https://arxiv.org/abs/2606.05080.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024. URL https://arxiv.org/abs/2405.15793.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Justin Young. Effective harnesses for long-running agents. Anthropic Engineering, 2025. URL https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents. November 26, 2025. Accessed September 15, 2026.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025. URL https://arxiv.org/abs/2510.04618v1. Version 1.

Appendix Contents

A Evaluation Variability and Search Robustness **14**

B Experimental Details **15**

B.1 Dataset Partition and Evaluation Protocol 15

B.2 Model and Harness Configuration 15

B.3 Comparison Method Configuration 16

B.4 Evaluation Metrics 16

B.5 Evolution Budgets and Candidate Evaluation 17

C Additional Experimental Results **17**

C.1 Per-Run Evaluation Results 17

D Additional Analysis **19**

D.1 Evolution Interface and Experience Reuse Ablations 19

D.2 Controlled Cross-Environment Transfer 19

D.3 Human-Selected Source Rubrics for AppWorld 21

D.4 Reporting Controlled Experiments 23

D.5 Analysis of Harness Realization Spaces 23

D.6 Rubric Realization and Component Retention 24

D.7 Deployment Token Analysis 24

E Additional Method Details **25**

E.1 Behavioral Diagnosis and Rubric Management 25

E.2 Harness Realization Details 26

E.3 Hierarchical Experience Memory Details 26

E.4 Continual Evolution and Selection Protocol 27

F Case Studies **28**

F.1 Experience-Memory Retrieval and Reuse 28

F.2 Rubric-Guided Harness Modifications 29

G Prompts **29**

G.1 Initial Harness for Software-Maintenance Tasks 29

G.2 Initial Harness for AppWorld Tasks 30

G.3 Behavioral Diagnosis 31

G.4 Behavioral Rubric Synthesis 34

G.5 Structural Editor 35

G.6 Guidance Editor 39

G.7 Structural–Guidance Complementarity Integrator 41

G.8 Transfer Applicability Assessment 45

A Evaluation Variability and Search Robustness

Repeated evaluation of a fixed harness. Table 1 evaluates one evolved harness per model, benchmark, and method. RHE and AHE select their final harnesses using only held-in evolution scores; ReasoningBank and ACE update their memory or context using the held-in tasks. Each resulting harness is frozen and evaluated three times on both fixed splits, with fresh task state in each evaluation. Reported held-in scores do not reuse update-time or selection-time scores. The initial harness is also fixed and evaluated three times. The statistical unit for these performance scores is an evaluation repeat, not an independent method update.

Independent evolution runs. Tables 2, 3, and 4 summarize three independent evolution runs per evolved-harness condition. Table 7 provides the individual full-RHE runs and their across-run summaries for each model and benchmark. The full-RHE and Combined ablation rows reuse its SWE-bench MiniMax-M2.5 runs; the ordinary RHE transfer row reuses its AppWorld MiniMax-M2.5 runs. Seed 1 reuses the same selected harness and evaluation records as the corresponding RHE row of Table 1. Each run starts from the same initial harness but evolves its own candidates and experience state. Its selected harness is frozen and evaluated anew on both splits. As in the per-run format in Table 7, three repeated evaluations first provide a mean score for each frozen harness; each evolution run then contributes one such mean to the across-run summary. The nine evaluations are not treated as nine independent evolution runs.

Aggregation. Let $s^{\text{in}}_{r,e}$ and $s^{\text{out}}_{r,e}$ be the split scores from evaluation repeat $e \in \{1,2,3\}$ of frozen harness r. For across-evolution summaries, r indexes independent evolution runs; Table 1 instead reports one evolved harness per method, model, and benchmark. For each evaluation, compute

$$s^{\text{all}}_{r,e} = w\, s^{\text{in}}_{r,e} + (1-w)\, s^{\text{out}}_{r,e}, \qquad w = \frac{n_{\text{in}}}{n_{\text{in}} + n_{\text{out}}}. \tag{5}$$

Here $w = 150/500$ for SWE-bench Verified and $w = 90/180 = 1/2$ for AppWorld. For any score $q \in \{\text{in}, \text{out}, \text{all}\}$, the within-harness evaluation mean and sample SD are

$$m^q_r = \frac{1}{3}\sum_{e=1}^{3} s^q_{r,e}, \qquad d^q_{r,\text{eval}} = \sqrt{\frac{1}{2}\sum_{e=1}^{3}(s^q_{r,e} - m^q_r)^2}. \tag{6}$$

Table 1 reports $m^q_r \pm d^q_{r,\text{eval}}$ for its frozen evolved harnesses. For the independent-evolution experiments, compute

$$\bar{m}^q = \frac{1}{3}\sum_{r=1}^{3} m^q_r, \qquad d^q_{\text{evo}} = \sqrt{\frac{1}{2}\sum_{r=1}^{3}(m^q_r - \bar{m}^q)^2}. \tag{7}$$

The evolved-harness rows of the ablation and transfer tables and the across-run rows of Table 7 report $\bar{m}^q \pm d^q_{\text{evo}}$; the individual seed rows report $m^q_r \pm d^q_{r,\text{eval}}$. Shared full-RHE conditions reuse the same across-run summaries, which need not equal the fixed-harness results in Table 1. Initial-harness scores summarize repeated evaluations of the fixed initial harness without evolution.

Overall SD is computed from the corresponding overall scores, never by averaging split SDs. Different split means are not added as a between-split variance term. Evaluation summaries use unrounded scores, with rounding only for presentation. The across-run summaries in Table 7 and their shared full-RHE entries in the controlled tables use unrounded seed-level means, as specified in its caption. Entries sharing both the experimental setting and aggregation level reuse the same records. Matching method names alone do not make a fixed-harness evaluation summary interchangeable with an across-evolution summary.

Best round is selected separately within each evolution run using the earliest held-in maximum during selection. Tables report its median and range across runs, while Table 7 also provides individual rounds. Evaluation SD describes execution variability conditional on one frozen harness. Across-evolution SD describes variability of run-level performance, including residual evaluation noise; it does not isolate a pure search-variance component. Neither SD describes variation across individual task outcomes. Three runs alone do not establish statistical significance.

B Experimental Details

B.1 Dataset Partition and Evaluation Protocol

SWE-bench Verified. SWE-bench Verified contains 500 repository-level software engineering tasks that require agents to modify existing repositories and pass the corresponding tests. We partition the benchmark into 150 held-in tasks for method updates and 350 held-out tasks for final evaluation. The partition is stratified by repository to approximately preserve the repository distribution across splits. Behavioral diagnosis, rubric construction, candidate generation, and harness selection in RHE use only the held-in tasks. The other methods use the same held-in partition for their update procedures; held-out tasks are reserved for evaluation of each frozen evolved harness.

AppWorld. AppWorld evaluates long-horizon tool use in simulated applications. We use AppWorld version 0.2.0, copied from Git commit `a072b7a86e7c1d5b1d7175659d750ebb9b79f10a`. Our fixed 180-task evaluation set has 90 held-in tasks (the entire official `train` partition) and 90 held-out tasks: 27 from `test_normal` and 63 from `test_challenge`. The sampling unit is a scenario, not an individual task. We group each test partition by scenario, sort scenario IDs, and use one `random.Random(42)` instance to sample 9 of the 56 normal scenarios, followed by 21 of the 139 challenge scenarios. All three task variants `_1`, `_2`, and `_3` of each selected scenario are included. Selected scenarios and variants are sorted within each test partition; the held-out sequence places normal tasks before challenge tasks. This yields a scenario-preserving, approximately proportional sample rather than 90 independently sampled task IDs. Task identities and split assignments are fixed across methods and independent runs. Only held-in tasks inform harness evolution, candidate selection, or experience updates; held-out tasks are reserved for final evaluation of each frozen evolved harness. Overall success gives the two splits equal weight. The exact scenario manifest is given below, and the cross-environment transfer procedure is detailed in Appendix D.2.

AppWorld task manifest. Each scenario ID below expands to exactly three task IDs by appending `_1`, `_2`, and `_3`, in that order. Thus the lists specify all 180 task IDs without repeating each scenario prefix three times. The held-in list contains 30 scenarios, and the held-out lists contain 9 normal and 21 challenge scenarios.

```
held-in / train:
82e2fac 692c77d 2a163ab 6104387 29caf6f afc0fce
287e338 27e1026 22cc237 d0b1f43 b7a9ee9 76f2c72
34d9492 ce359b5 b0a8eae 229360a 7d7fbf6 e3d6c94
ccb4494 60d0b5b 771d8fc 07b42fd 3c13f5a cf6abd2
302c169 e7a10f8 c901732 6ea6792 e85d92a aa8502b

held-out / test_normal:
042a9fc 13547f5 166f4ff 31dc501 325d6ec
3aa1a22 b6d1104 d194965 f861c32

held-out / test_challenge:
099174d 16be9ce 18670a5 1a79e37 3650990 365e0a3
3f3c139 6474048 6588a51 77bcb81 80acbaf 81be677
8d42650 9aae7da adb1060 af84964 baeb104 d37c235
d6d8cb6 dcbd34b f6936d4
```

B.2 Model and Harness Configuration

We evaluate GPT-5, GLM-5, and MiniMax-M2.5 with fixed model parameters. In this paper, the harness denotes the complete agent execution scaffold, including executable components such as tools and middleware and agent-facing components such as prompts and deployed memory. Thus, changes to deployed memory or prompts constitute harness modifications even when executable components remain fixed. Within each model and benchmark, the four agent-improvement methods begin from the same single-tool initial harness H_0: a shell interface for SWE-bench Verified and the `execute_code` Python interface for AppWorld. The latter adapts the general-purpose single-tool setup to AppWorld's application environment. The initial harness provides environment interaction, tool execution, context management, and the task execution loop. RHE and AHE modify the harness

through iterative candidate evolution, whereas ReasoningBank and ACE adapt its memory or context within the same execution infrastructure. Model weights, benchmark tasks, execution environments, and evaluation procedures remain fixed. The initial harness prompts for both environments are reproduced in Appendix G.

AppWorld task-agent execution profile. The AppWorld task-agent and evaluation configuration is summarized below. The model identity varies by backbone as reported in the main experiments; these settings specify the common request and task budgets.

Table 5: **AppWorld task-agent execution configuration.** Concurrency refers to formal repeated baseline evaluation, not parallel tool calls within one task.

Configuration	Value
Temperature	0.7
Top-p	0.95
Maximum output tokens per model request	32,000
Configured context length	128,000
Model request timeout	300 s
Streaming	false
API type	`openai_chat_completion`
Total concurrency for repeated baseline evaluation	16
Model-call budget per task	50
AppWorld interaction budget per task	50
Task time budget	1,800 s
Code-execution timeout	60 s
Evaluator timeout	120 s
AppWorld environment random seed	42

B.3 Comparison Method Configuration

We compare RHE with AHE (Lin et al., 2026), ReasoningBank (Ouyang et al., 2025), and ACE (Zhang et al., 2025). RHE and AHE are iterative harness evolution methods. AHE uses execution observations and component-level evolution signals to modify system prompts, tools, middleware, and long-term memory, whereas RHE uses behavioral rubrics to guide modifications. ReasoningBank and ACE represent experience-based agent improvement: ReasoningBank distills reasoning strategies from successful and failed trajectories and retrieves them into agent instructions, while ACE maintains structured context playbooks through generation, reflection, and curation, including system prompts and online agent memory.

RHE and AHE perform ten rounds of iterative harness evolution under the same round budget and candidate-evaluation protocol. ReasoningBank and ACE follow their original experience-update procedures, in which experience is extracted and incorporated before final evaluation. Their update procedures are not recast as artificial harness evolution rounds, which would change their original optimization mechanisms. The comparison therefore evaluates different improvement paradigms under their intended optimization procedures. These four methods share the same initial harness, backbone model, task partitions, execution environment, and final frozen evaluation protocol.

Human-designed reference harnesses. Figure 1 additionally reports our own SWE-bench Verified evaluations of OpenCode, Mini-SWE-Agent, and OpenHands with each backbone model. These reference systems use their respective human-designed scaffolds. Their scores report task success rates as percentages.

B.4 Evaluation Metrics

The primary metric is task success rate, defined as the percentage of successfully completed tasks in a fixed benchmark split. On AppWorld, we use task goal completion (TGC): a task succeeds only when all of its official evaluation tests pass. Each evolved harness is frozen and evaluated anew on held-in and held-out tasks. Each evaluation's overall score is computed using Equation 5.

Reporting units. Table 1 reports three repeated evaluations of each frozen evolved harness using Equation 6. Evolved-harness rows in the ablation and transfer tables report across-evolution statistics using Equation 7. Table 7 provides individual seed evaluations and the corresponding three-seed aggregates. Seed 1 reuses the corresponding main-text RHE evaluation records. Appendix A distinguishes these two levels.

Deployment-token reporting. For each fixed harness and split, token measurements and success scores come from the same three repeated evaluations used in Table 1. Within each evaluation, we average token consumption over tasks, then report the mean and sample SD of these three per-evaluation averages. Both axes of Figure 5 therefore describe repeated evaluation of one fixed harness, not variation across independently evolved harnesses or token dispersion across tasks.

B.5 Evolution Budgets and Candidate Evaluation

For RHE and AHE, each evolution round uses the same task-execution allowance and matched editing and candidate-evaluation budgets. Execution trajectories provide evidence for proposing harness modifications. RHE allocates one structural and one guidance candidate from the same parent and frozen rubric state. If the edits are complementary and mechanically compatible, the integrator constructs one additional combined candidate. Each valid proposed candidate is evaluated once on the held-in set; the selected candidate becomes the parent for the next round. If none improves the held-in score, the current harness is retained. Both methods complete ten rounds and return their final harnesses. The complementarity prompt is reproduced in Appendix G.

RHE model-role limits. Table 6 specifies the request limits for each RHE role. The task-agent row applies to SWE-bench Verified; AppWorld uses the separate task-agent profile in Table 5. The analysis and editor rows apply to both benchmarks. An unset top-p means that no value was explicitly supplied for that request, rather than that top-p was zero.

Table 6: **RHE model-role configuration and per-candidate limits.** Editor limits apply separately to structural, guidance, and optional complementary candidates.

Role	Temp.	Top-p	Max. output	Context	Call limit
Task agent (SWE-bench)	0.7	0.95	32,000 tokens	200,000 tokens	300 turns per task
Analysis (trajectory diagnosis and rubric synthesis)	0.2	unset	12,288 tokens	131,072 tokens	4 model requests per analysis
Structural, guidance, or complementarity editor	0.2	unset	16,384 tokens	200,000 tokens	48 model requests and 96 tool calls per candidate

For SWE-bench Verified, one shell command has a 300-second timeout. An editor model request has a 300-second timeout, and the editing process for one candidate has a 3,600-second time limit. The complementary candidate is generated only when the integrator finds a valid composition of the two standalone edits; the round therefore evaluates at most three new candidates.

ReasoningBank and ACE instead follow their respective memory or context update procedures on the held-in tasks. They do not use a ten-round harness evolution budget. The comparison therefore shares the initial harness, task partitions, and frozen final evaluation protocol without equating their update procedures.

C Additional Experimental Results

C.1 Per-Run Evaluation Results

Table 7 separates within-harness evaluation variability from variability across independent evolution runs. Seed 1 in each block reuses the selected RHE harness and repeated-evaluation results reported in Table 1. The remaining seeds describe separately evolved harnesses. Across-run statistics summarize

all three seed-level means. Shared full-RHE conditions in the ablation and transfer tables reuse these aggregates; they need not match the fixed-harness Table 1 results. Overall means use task-count weights of 30%/70% for SWE-bench Verified and 50%/50% for AppWorld.

Table 7: **Full RHE across independent evolution seeds.** Seed rows report mean ± evaluation SD. *Seed 1 reuses the corresponding RHE results in Table 1. Across-run rows report mean ± sample SD across evolution runs; best round is summarized as median [range]. Scores are success rates (%), with task-weighted Overall means. All across-run aggregates are computed from the three unrounded seed-level success rates and are distinct from the fixed-harness statistics in Table 1. Because seed means are displayed to two decimals, averaging the printed seed means can differ from an across-run mean by 0.01 percentage points (e.g., AppWorld GLM-5 held-out). Shared full-RHE conditions in the ablation and transfer tables reuse these across-run aggregates.

Model	Evolution seed	Best round	Held-in	Held-out	Overall
			SWE-bench Verified		
GLM-5	Seed 1*	4	76.89±0.38	76.67±1.08	76.73±0.64
	Seed 2	6	76.67±0.67	76.67±0.92	76.67±0.70
	Seed 3	5	76.89±0.38	76.29±1.03	76.47±0.83
	Across-run statistics	**5 [4–6]**	**76.81±0.13**	**76.54±0.22**	**76.62±0.14**
GPT-5	Seed 1*	6	69.11±1.02	68.48±0.92	68.67±0.95
	Seed 2	5	69.33±0.67	68.67±1.00	68.87±0.90
	Seed 3	5	68.89±0.77	68.86±1.03	68.87±0.95
	Across-run statistics	**5 [5–6]**	**69.11±0.22**	**68.67±0.19**	**68.80±0.12**
MiniMax-M2.5	Seed 1*	4	77.33±0.67	75.14±0.57	75.80±0.53
	Seed 2	3	77.11±0.38	75.33±0.72	75.87±0.61
	Seed 3	5	77.56±0.38	75.05±0.59	75.80±0.53
	Across-run statistics	**4 [3–5]**	**77.33±0.22**	**75.17±0.15**	**75.82±0.04**

Table 7: **Full RHE across independent evolution seeds (continued).** AppWorld results, with the same reporting conventions as above.

Model	Evolution seed	Best round	Held-in	Held-out	Overall
			AppWorld		
GLM-5	Seed 1*	3	91.85±1.70	92.59±0.64	92.22±0.56
	Seed 2	4	93.33±1.11	92.59±0.64	92.96±0.64
	Seed 3	4	93.70±0.64	92.96±0.64	93.33±0.56
	Across-run statistics	**4 [3–4]**	**92.96±0.98**	**92.72±0.21**	**92.84±0.57**
GPT-5	Seed 1*	4	93.70±1.28	93.33±1.11	93.52±1.16
	Seed 2	5	93.70±0.64	93.33±0.00	93.52±0.32
	Seed 3	5	93.70±0.64	93.70±0.64	93.70±0.32
	Across-run statistics	**5 [4–5]**	**93.70±0.00**	**93.46±0.21**	**93.58±0.11**
MiniMax-M2.5	Seed 1*	7	66.67±1.11	60.37±1.28	63.52±1.16
	Seed 2	6	66.67±1.11	60.74±0.64	63.70±0.64
	Seed 3	5	67.04±0.64	60.37±0.64	63.70±0.64
	Across-run statistics	**6 [5–7]**	**66.79±0.21**	**60.49±0.21**	**63.64±0.11**

D Additional Analysis

D.1 Evolution Interface and Experience Reuse Ablations

Table 2 compares the initial harness, trajectory-driven editing, RHE without memory, and full RHE on MiniMax-M2.5, measuring best-round timing and selected-harness performance. All conditions use the same SWE-bench evolution and held-out task sets specified in Appendix B.1.

Full RHE. Each iteration evaluates the selected parent on the evolution tasks, constructs or updates rubrics, retrieves permitted experience, proposes bounded harness changes, and evaluates candidates. The shared selection rule determines the next parent. Evidence, rubrics, edits, and outcomes persist with their links.

Trajectory-driven Editing. The editor analyzes current-parent execution trajectories and their observed task outcomes to diagnose behavioral deficiencies. It proposes harness modifications from this diagnosis and can reflect on the same evidence to revise a proposal, with at most three self-refinement retries. These retries remain within the assigned proposal slot; only its final proposal enters the common candidate-evaluation and selection procedure. The condition constructs no behavioral rubrics and retrieves no prior evolution experience. Its harness editing space, two standalone proposal slots, optional combined slot, held-in task-evaluation allowance, and selection procedure match RHE w/o Memory. The model configuration and per-call editing limits also match those of rubric-guided editing. These shared limits do not imply equal numbers of analysis calls or equal realized token use. Removed rubric and retrieval calls do not purchase additional candidates or task evaluations. This comparison tests the effect of explicitly organizing the trajectory diagnosis as a behavioral rubric when cross-round experience reuse is disabled.

RHE w/o Memory. At each iteration, we started the evolution agent with a fresh conversation and an empty experience store. Rubric construction and within-iteration revision used current-parent evidence; previous evidence, rubrics, edits, outcomes, summaries, and history-search tools were unavailable. At the iteration boundary, we retained only the selected harness, including its learned instructions and deployed task-agent memory, and the evaluator state required by the common selection rule. Audit logs remained outside the accessible workspace of the evolution agent. We preserved ordinary task-agent context within each task. This ablation therefore tests explicit cross-round experience reuse while retaining learned changes within the selected harness.

The evolved conditions are run independently from the same initial harness, rather than applied sequentially. Their matched execution and reporting protocol is detailed in Appendix D.4. The median earliest held-in peak occurs at round 7 [6–8] for trajectory-driven editing, round 6 [5–7] for RHE without memory, and round 4 [3–5] for full RHE. All conditions complete ten rounds; best round marks the earliest held-in peak, not a stopping time.

D.2 Controlled Cross-Environment Transfer

Table 4 includes the shared pre-evolution harness as a reference alongside the MiniMax-M2.5 comparison of target-only RHE, initialization with all source rubrics, and initialization with rubrics selected by a human or an LLM judge. The primary contrast tests whether applicability-based selection improves transfer relative to importing the full source bank.

Source bank construction. Ten SWE-bench Verified evolution rounds produced 40 source rubrics. Using prespecified AppWorld applicability criteria, we manually selected six. An independent MiniMax-M2.5 judge, using the same backbone as the AppWorld agent, selected eight from the same 40 rubrics with a fixed applicability prompt (Appendix G). The full-bank condition imports all 40; the six human-selected rubric specifications are listed in Appendix D.3. Source task solutions, traces, realized edits, and harness code are not transferred as rubric content. Both selectors fixed their decisions before any AppWorld held-out results were inspected.

Target configurations. All settings start from the same AppWorld harness with the same supported tool interface and an empty target-experience store. They follow the same target evolution procedure

and differ only in the source rubrics loaded at initialization. RHE starts with no source rubrics and constructs them from target-environment trajectories; the other settings import all 40, the six human-selected rubrics, or the eight LLM-selected rubrics. Imported rubrics initialize the target rubric bank, not the executable harness. During AppWorld evolution, target evidence can confirm, revise, or deactivate imported rubrics before they guide edits. Table 8 reports the associated rubric and modification counts.

Table 8: **Applicability-based source-rubric selection and successful harness modifications in AppWorld.** Counts come from one prespecified run per condition using the same seed.

Selector	Selected rubrics	Rubrics yielding successful modifications	Successful modifications	Modification yield
Human screening	6	3	5	83.3%
LLM judge	8	4	6	75.0%
All rubrics	40	8	9	22.5%

Modification yield is $N_{\text{successful modifications}}/N_{\text{selected rubrics}}$. It counts modifications rather than distinct rubrics and is not a probability of transfer. Table 4 reports success-rate summaries across three independent runs. A larger selected bank can yield more successful modifications in the recorded run while reducing mean final held-out success, as in the all-rubrics condition.

Applicability-based rubric selection quality. Three computer science PhD annotators assessed the source rubrics. They scored each rubric on four dimensions from 1 to 5, where higher scores indicate stronger properties of a transferable behavioral specification. Table 9 reports human ratings of the 40 source rubrics and the human- and LLM-selected subsets under the same four-criterion evaluation. The MiniMax-M2.5 judge is used to select rubrics for transfer, but its scores are not used in this quality comparison. Both selected subsets have higher mean human ratings than the full source bank on each criterion. These ratings are descriptive; target-task success is reported separately in Table 4.

Table 9: **Applicability-based rubric selection quality.** All three sets use the same human evaluation on four 1–5 criteria; Overall averages the four criterion means.

Selection	No. rubrics	Behavioral abstraction	Target applicability	Implementation independence	Actionability	Overall
All rubrics	40	3.13	2.95	3.08	3.01	3.04
Human-selected	6	4.17	4.28	4.11	4.00	4.14
LLM-selected	8	4.00	4.17	4.04	4.00	4.05

Human transfer-rating instructions. The annotators applied the following anchors to each dimension:

Behavioral abstraction: Does the rubric describe a reusable behavioral requirement rather than a source-specific fix?
1: It describes a specific task, repository, bug, tool, or workflow and cannot be used outside the source context. **2:** It relies mainly on source-specific details, with little transferable behavior. **3:** It captures some abstract behavior but still requires the source context for interpretation. **4:** It mainly describes reusable behavior, with only minor contextual information. **5:** It expresses an agent behavior reusable across tasks.

Target applicability: Is the rubric relevant and useful for the target environment?
1: No corresponding behavioral need exists in the target environment. **2:** Only weakly related situations exist, with limited practical value. **3:** Similar situations exist, but the benefit is uncertain. **4:** Clearly corresponding situations exist, and applying the rubric is likely to be useful. **5:** The behavioral need is widespread in the target environment, and the rubric clearly applies.

Implementation independence: Does the rubric specify what the agent should do rather than how?
1: It is tied to a specific tool, API, repository, or workflow. **2:** It depends heavily on a particular implementation. **3:** It includes implementation details that can be abstracted away. **4:** It is largely independent of implementation, apart from necessary context. **5:** It specifies a behavioral objective realizable through different harness designs.

Harness evolution actionability: Can the rubric guide concrete harness improvements?
1: It is too vague to guide a modification. **2:** It gives a general direction that is difficult to turn into an edit. **3:** A potential modification direction can be inferred. **4:** It clearly guides changes to prompts, tools, memory, or other harness components. **5:** It states an actionable behavioral objective that directly guides harness evolution.

For a rated rubric, the overall score is $(\mathrm{BA} + \mathrm{TA} + \mathrm{II} + \mathrm{HA})/4$.

Human–LLM selection agreement. The human and LLM selectors share five rubrics. Relative to the human-selected set, LLM selection has overlap precision $5/8 = 62.5\%$ and overlap recall $5/6 = 83.3\%$; the Jaccard similarity is $5/(6 + 8 - 5) = 55.6\%$. These values measure agreement between selectors, not whether a rubric improves AppWorld performance.

Table 10: **Agreement between human and LLM source-rubric selection.** Precision and recall use the human-selected set as a reference, not as a ground-truth label of transferability.

Measure	Value
Human-selected rubrics	6
LLM-selected rubrics	8
Shared rubrics	5
LLM overlap precision	62.5%
LLM overlap recall	83.3%
Jaccard similarity	55.6%

Target data and leakage controls. Disjoint AppWorld evolution and held-out task manifests are fixed before source adaptation. Task state is reset for each candidate execution, and held-out artifacts are excluded from evolution workspaces. Only target evolution results inform retrieval tuning, rubric revision, candidate selection, or stopping. Final harnesses are frozen before held-out evaluation.

Budgets and transfer measurements. All configurations use the same AppWorld initial harness and ten-round evolution budget, with three independent evolution runs per condition. Held-in tasks guide evolution and harness selection; held-out tasks do not participate in selection. Table scores come from a new evaluation of each frozen selected harness on both splits; the held-in selection scores are not reused as reported evaluation scores.

Evolved-harness success rates in Table 4 report means and sample SDs across three independent evolution runs per method. Each run contributes its final harness's mean over three evaluations. Overall is task-count-weighted within each evaluation before averaging within runs and aggregating across runs; split SDs are not averaged. The ordinary RHE row reuses the AppWorld MiniMax-M2.5 across-run summary from Table 7. Evolved-harness performance SD describes across-run variability, including residual evaluation noise.

D.3 Human-Selected Source Rubrics for AppWorld

The six rubrics below were manually screened from the 40-rubric SWE-bench source bank for AppWorld applicability. Inclusion means that a behavioral requirement could apply in AppWorld; it does not imply that the rubric was used to modify the harness. Three of the six led to five successful edits.

1. Verify the requested outcome in observable application state.

Behavior. Read the relevant application state and compare it with the user's requested outcome before reporting completion.
Context. After an action intended to change application state, when a relevant read operation is available.
Boundary. A successful API response alone does not establish task completion. Do not repeat a state-changing action merely to verify it. If the outcome cannot be observed, report it as unverified.

2. Verify every explicit user requirement and constraint.

Behavior. Check that each explicit requirement has been satisfied and each constraint has been respected.
Context. Tasks with multiple requested actions, conditions, or constraints.
Boundary. Do not invent additional requirements or expand the task. Report partial completion accurately when some requirements remain unresolved.

3. Check task-relevant boundary conditions before making changes.

Behavior. Check for empty results, ambiguous matches, and already-satisfied conditions when they affect the intended action.
Context. Selecting target entities or deciding whether a state-changing action is necessary.
Boundary. Check only conditions relevant to the task. Do not choose an arbitrary target when ambiguity remains or duplicate an already-completed action.

4. Distinguish action failure from uncertainty about the outcome.

Behavior. Separate explicit failure evidence from missing or inconclusive observations, and communicate uncertainty accurately.
Context. An operation returns an error, times out, or provides insufficient evidence of its effect.
Boundary. Do not infer that a timed-out action had no effect. Do not infer success from missing error evidence or attribute failure to infrastructure without supporting observations.

5. Keep temporary state isolated across items.

Behavior. Keep item-specific identifiers, arguments, and intermediate results separate while processing multiple items.
Context. Batch operations or repeated workflows over several entities.
Boundary. Preserve explicitly intended shared state. Do not allow one item's temporary values to alter another item's target or action unintentionally.

6. Use an equivalent observable check when direct verification is unavailable.

Behavior. Use an alternative observation that supports the same requested outcome, and explain what it does and does not establish.
Context. Direct verification is unavailable, but another authorized observation can provide relevant evidence.
Boundary. A proxy must not be presented as stronger evidence than it is. Do not replace a real-world outcome with a simulated success or perform unrelated changes to obtain verification.

D.4 Reporting Controlled Experiments

Main-text ablation and transfer tables report mean and sample SD across three independent evolution runs per evolved-harness condition, using Equation 7. Each run contributes the mean of three evaluations of its selected, frozen harness. We computed task-weighted overall scores within each evaluation before aggregation. Initial-harness rows instead report repeated evaluation of the fixed initial harness using Equation 6.

Matched ablation execution. For Table 2, the three evolved conditions run independently from the same initial harness, ordered evolution-task batches, and execution seeds where supported. Their trajectories may diverge after editing. All conditions complete $N = 10$ rounds with the same two standalone proposal slots and the same rule for one optional combined candidate, task-execution allowance B_{eval}, editor-call limits, context ceiling L, and per-task timeout. Invalid proposals and failed attempts count against their allowances, and a round without improvement retains the parent. Input snapshots verify that trajectory-driven editing receives no rubric objects or prior experience and that RHE w/o Memory receives no prior experience.

Best-round selection and evaluation. For condition c and independent run r, let $H_{c,r,t}$ be the selected harness after round t, with the shared initial harness at $t = 0$. The earliest held-in peak is

$$t^*_{c,r} = \min \operatorname*{arg\,max}_{t \in \{0,\ldots,10\}} S_{\text{held-in}}(H_{c,r,t}). \tag{8}$$

Here $S_{\text{held-in}}$ is the selection-time score; held-out results do not enter this rule. The frozen harness at $t^*_{c,r}$ is evaluated anew on both splits. Table 2 reports the median and range of $t^*_{c,r}$ across three runs, alongside the across-run performance statistics. An early peak can also reflect a lower plateau, so best-round timing is interpreted alongside held-out performance. All conditions complete ten rounds.

Table 7 separately presents per-seed performance and across-evolution aggregates. Its Seed 1 reuses the corresponding Table 1 RHE harness and evaluation records. Its across-run aggregates are reused by matching full-RHE conditions in the controlled tables; they need not reproduce Table 1. Individual tasks are not independent evolution runs or evaluation repeats.

D.5 Analysis of Harness Realization Spaces

Behavioral rubrics specify desired agent behaviors without prescribing a specific implementation.

Therefore, the same behavioral requirement can potentially be realized through different harness modification strategies.

Table 3 reports the measured MiniMax-M2.5 comparison on SWE-bench Verified for three realization spaces.

Structural-only realization. The editor is restricted to executable harness components, including tools, middleware, execution control, and sub-agents.

Guidance-only realization. The editor is restricted to instructions and contextual information supplied to the agent, including system prompts, tool descriptions, and deployed task-agent memory. This deployed memory is distinct from the evolution system's experience memory, which remains available in every condition.

Combined realization. The editor can explore both structural and guidance modifications and combine compatible changes.

We used explicit file/component allowlists for structural and guidance edits and fixed their boundaries before the experiments. Across conditions, we kept the initial harness, tasks, model settings, selection rule, and total candidate and evaluation allowances fixed. Combined editing shared the same total allowance across both tracks. Rubric and experience-store updates remained enabled in all conditions; guidance edits modified the instructions and contextual information of the deployed harness. The combined row reuses the same three full-RHE evolution runs as Table 2 and the SWE-bench MiniMax-M2.5 block in Table 7. Means and sample SDs summarize the three run-level means (Appendix A);

each run's final harness is evaluated three times. Overall scores use fixed 150/500 held-in and 350/500 held-out task weights within each evaluation. Best round separately summarizes the earliest held-in maximum across the same evolution runs. The median best round is 6 [5–7] for guidance-only, 5 [4–6] for structural-only, and 4 [3–5] for combined realization; all variants complete the same ten rounds.

This analysis does not aim to identify a universally optimal realization mechanism. Instead, it studies whether behavioral rubrics enable different implementation strategies to optimize toward the same behavioral objective.

D.6 Rubric Realization and Component Retention

Component taxonomy. For Table 11, a **tool** is an executable capability, a **prompt** is an agent instruction outside persistent memory, and **memory** is deployed agent memory, distinct from the evolution system's hierarchical experience store. Categories describe the implementation changed by a diff, not merely the wording of its instruction.

Modification-generation counts. Figure 4 counts generated rubrics, rubrics associated with at least one generated harness modification, and the resulting modifications. Generating a modification does not imply that the candidate harness improves task performance or is selected.

Per-rubric multiplicity. Figure 4(b) bins rubrics with generated modifications by one, two, or three edits: GPT-5 has $(5, 6, 2)$, GLM-5 has $(5, 7, 2)$, and MiniMax-M2.5 has $(8, 6, 2)$. These counts yield 13, 14, and 16 rubrics with generated modifications and 23, 25, and 26 modifications, respectively. The shares with at least two modifications are $8/13 = 61.54\%$, $9/14 = 64.29\%$, and $8/16 = 50.00\%$, calculated before rounding. Each counted modification is a concrete generated edit, whether or not it was retained.

Component-level realization and retention. Table 11 groups generated modifications by component type and separately reports those retained through harness selection. The retained totals are four, six, and seven for GPT-5, GLM-5, and MiniMax-M2.5. MiniMax-M2.5 retains seven edits over four promotions, so edit and promotion counts differ. These descriptive counts do not isolate the causal contributions of individual components.

Table 11: **Harness modification generation and retention by component.** Retention is the fraction of generated modifications retained through harness selection. Memory denotes deployed agent memory.

Model	Component	Generated	Retained	Retention
GPT-5	Tool	7	1	14.29%
	Prompt	7	1	14.29%
	Memory	9	2	22.22%
GLM-5	Tool	11	4	36.36%
	Prompt	6	1	16.67%
	Memory	8	1	12.50%
MiniMax-M2.5	Tool	12	4	33.33%
	Prompt	6	0	0.00%
	Memory	8	3	37.50%

D.7 Deployment Token Analysis

Figure 5 compares per-task deployment tokens and success rates for fixed MiniMax-M2.5 harnesses, using the same three repeated evaluations as Table 1. RHE has higher mean success than AHE with fewer tokens per task on both splits: 1,042.70K versus 1,094.22K held-in, and 1,008.91K versus 1,029.25K held-out. Relative to the initial harness, RHE uses more tokens and succeeds more often.

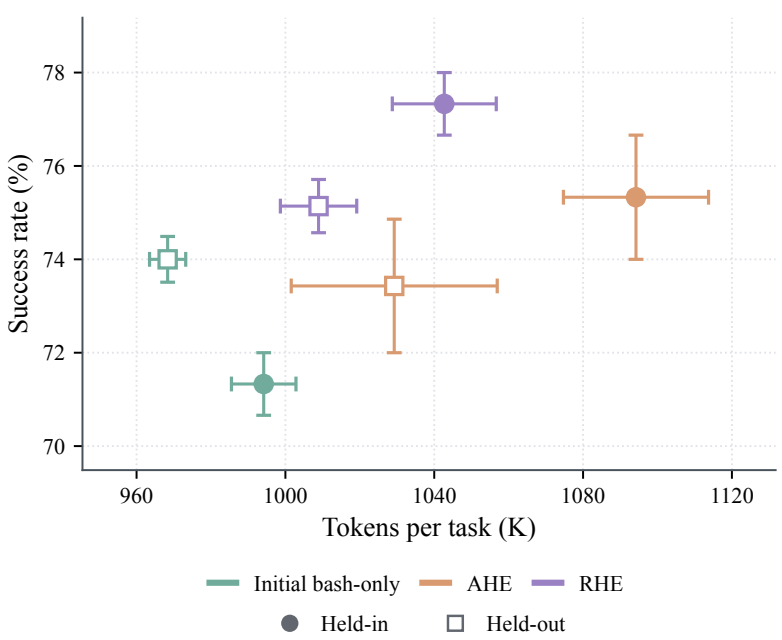

Figure 5: **Deployment tokens and task success on MiniMax-M2.5.** Means ± sample SD over the same three evaluations of each fixed harness as Table 1, for per-task tokens (horizontal) and success (vertical). Filled and open markers denote held-in and held-out tasks.

E Additional Method Details

E.1 Behavioral Diagnosis and Rubric Management

Diagnosis representation. At evolution iteration t, the current harness H_t is executed on task x to produce trajectory τ_x^t. The diagnosis module extracts a trajectory-level behavioral diagnosis

$$z_x^t = \left(p_x^t, s_x^t, h_x^t, \mathcal{E}_x^t\right), \tag{9}$$

where p_x^t describes the observed problem, s_x^t denotes the relevant execution context, h_x^t is a provisional description of the behavior requiring improvement, and $\mathcal{E}_x^t$ contains the supporting execution evidence.

The diagnosis remains tied to the observed trajectory. In contrast, the behavioral rubric defined in Section 3.2 abstracts this trajectory-level diagnosis into a reusable behavioral specification with an explicit applicable context and boundary constraints. The diagnosis and rubric-synthesis system and user prompts are reproduced in Appendix G.

Evidence grounding. Each diagnosis and rubric retains references to the trajectory events that support its behavioral judgment. The evidence record includes the task context, relevant agent actions, environment responses, and the trace locations from which the diagnosis is derived. Evidence references are preserved rather than replaced by unconstrained summaries, allowing the applicability of a rubric to be reassessed when it is reused.

Rubric matching and update. Each behavioral diagnosis is first abstracted into a provisional rubric. Before this rubric is added to the hierarchical experience memory, it is matched against existing rubrics. The system then uses the new evidence to choose among three operations: **(1) Reuse** an existing rubric and attach the new evidence when its desired behavior and applicable scope remain unchanged; **(2) Refine** the desired behavior, applicable context, or boundary constraints when the evidence reveals an inadequacy in the existing specification; or **(3) Extend** the rubric set when no existing rubric covers the newly observed behavioral deficiency.

Rubric reuse is not determined by textual similarity alone. The system examines linked evidence and applicability conditions to assess whether an existing rubric represents the same behavioral requirement. Within each evolution round, the rubric state is frozen before candidate generation. Structural and guidance editors therefore work from the same behavioral specification without changing its objective during implementation.

E.2 Harness Realization Details

Behavioral rubrics leave their concrete implementation unspecified. For each realization type $\beta \in \{\mathrm{str}, \mathrm{gui}\}$, the modification generator produces

$$\Delta_t^\beta = G_\beta\left(H_t, \mathcal{R}_t^\beta, \mathcal{K}_t^\beta\right), \tag{10}$$

where H_t is the current harness, $\mathcal{R}_t^\beta$ contains the rubrics selected for realization type β, and $\mathcal{K}_t^\beta$ contains retrieved evidence and historical realization records relevant to those rubrics.

Structural realization. Structural realization spans executable harness mechanisms, including tool implementations, middleware, execution control, and sub-agent configurations. Structural edits are restricted to authorized harness components; benchmark evaluators, task definitions, and the underlying model remain outside the editing scope. The structural editor's system and user prompts are reproduced in Appendix G.

Guidance realization. Guidance realization modifies instructions and contextual information supplied to the agent, including system prompts, tool descriptions, reusable skills, and memory content. Guidance edits alter how the model is instructed or contextualized without directly changing the executable environment. The guidance editor's system and user prompts are reproduced in Appendix G.

Candidate construction. Applying a generated modification to the current harness yields

$$H_t^\beta = \mathrm{Apply}\left(H_t, \Delta_t^\beta\right). \tag{11}$$

One structural and one guidance candidate are generated from the same parent harness and frozen rubric state. The complementarity integrator compares their manifests and evaluation evidence against the rubric. If the two realizations are complementary and mechanically compatible, it preserves both edits and produces one additional combined candidate. This candidate is evaluated independently rather than inheriting the score of either constituent. If integration retains a standalone candidate instead, the host reuses that candidate and its existing evaluation score; no additional candidate evaluation is performed. The integrator prompt appears in Appendix G.

Historical realization records are supplied as implementation references. They are not copied directly: the editor must adapt each candidate to the current harness, rubric scope, and execution context.

E.3 Hierarchical Experience Memory Details

The memory separates behavioral requirements from their supporting evidence and concrete realizations, preserving the distinction between an edit's objective and its implementation.

Memory organization. At iteration t, the hierarchical experience memory is represented as

$$\mathbb{M}_t = \left(\mathbb{M}_t^E, \mathbb{M}_t^R, \mathbb{M}_t^I, \mathcal{L}_t^{ER}, \mathcal{L}_t^{RI}\right), \tag{12}$$

where:

- $\mathbb{M}_t^E$ stores task descriptions, structured interaction records, diagnoses, and trace references;
- $\mathbb{M}_t^R$ stores behavioral rubrics and their revision histories;

- $\mathbb{M}_t^I$ stores historical harness realizations, modification descriptions, changed components, validation results, and evaluation records;
- $\mathcal{L}_t^{ER}$ links evidence records to the rubrics they support or constrain;
- $\mathcal{L}_t^{RI}$ links rubrics to their historical harness realizations.

The relations are many-to-many. Multiple evidence records may support one rubric, one evidence record may be relevant to multiple rubrics, and one rubric may guide multiple harness realizations. A composite realization may likewise be associated with multiple rubrics. Evaluation outcomes are attached to realization records. They record the observed performance of a candidate but do not establish which associated rubric caused a performance change.

Rubric retrieval. Given a current rubric ρ constructed from a trajectory diagnosis, the system retrieves candidate historical rubrics using semantic similarity. We use the frozen `sentence-transformers/all-MiniLM-L6-v2` encoder (Sentence Transformers, 2021) for f_θ; let $\oplus$ denote text concatenation. Both the current rubric and historical rubrics are represented by

$$v_\rho = f_\theta\left(g_\rho \oplus a_\rho \oplus b_\rho\right). \tag{13}$$

Candidate rubrics are retrieved by

$$\mathcal{N}_t(\rho) = \mathrm{TopK}_{\rho' \in \mathbb{M}_t^R} \,\mathrm{sim}\left(v_\rho, v_{\rho'}\right), \tag{14}$$

where sim denotes cosine similarity.

The retrieval depth is three: each current rubric synthesized from the round's trajectory diagnoses retrieves its three most similar historical rubrics (or all available rubrics if fewer than three are stored). The retrieved rubrics, including their applicable contexts and boundaries, their linked execution evidence, and their harness realization histories are provided to the editor as references.

Historical realization retrieval. For each of the top-three historical rubrics, the system follows the $\mathcal{L}_t^{RI}$ links to its historical harness realizations. These records include modified components and their evaluation outcomes. Successful realizations provide design references, while unsuccessful realizations document previous attempts.

Memory update. After candidate evaluation, memory is extended with newly collected execution evidence, rubric additions or revisions, generated realizations, and their evaluation records. Unselected and unsuccessful realizations remain in memory rather than being discarded, preserving the history needed to compare alternative implementations of the same behavioral rubric.

E.4 Continual Evolution and Selection Protocol

Evaluation objective. For task set $\mathcal{D}$, let $V(x, \tau_x(H)) \in \{0, 1\}$ indicate whether harness H successfully completes task x. Harness performance is

$$J_{\mathcal{D}}(H) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} V\left(x, \tau_x(H)\right). \tag{15}$$

Only the evolution split $\mathcal{D}_{\mathrm{evo}}$ is used for candidate selection. Held-out evaluation tasks are not exposed to rubric construction, candidate generation, or harness selection. At selection time, each proposed candidate is evaluated once on the held-in task set; these selection scores are distinct from the three fresh evaluations reported for a frozen selected harness.

Candidate selection. Let $\mathcal{H}_t$ contain the current harness and all valid candidate harnesses generated at iteration t. The next parent is selected by

$$H_{t+1} = \arg\max_{H \in \mathcal{H}_t} J_{\mathcal{D}_{\mathrm{evo}}}(H). \tag{16}$$

Algorithm 1 One iteration of behavioral rubric-guided harness evolution

Require: Current harness H_t, memory $\mathbb{M}_t$, evolution tasks $\mathcal{D}_{\text{evo}}$
Ensure: Next harness H_{t+1} and updated memory $\mathbb{M}_{t+1}$
1: Execute H_t on $\mathcal{D}_{\text{evo}}$ and collect trajectories $\mathcal{T}_t$
2: Construct trajectory-level diagnoses $\mathcal{Z}_t \leftarrow \textsc{Diagnose}(\mathcal{T}_t)$
3: Abstract diagnoses into provisional rubrics $\widetilde{\mathcal{R}}_t \leftarrow \textsc{Abstract}(\mathcal{Z}_t)$
4: Use $\widetilde{\mathcal{R}}_t$ to retrieve similar historical rubrics and their linked evidence and realizations $\mathcal{K}_t \leftarrow \textsc{RetrieveContext}(\widetilde{\mathcal{R}}_t, \mathbb{M}_t)$
5: Reuse, refine, or extend the rubric set $\mathcal{R}_t \leftarrow \textsc{UpdateRubrics}(\widetilde{\mathcal{R}}_t, \mathcal{K}_t)$
6: Freeze $\mathcal{R}_t$ for candidate generation
7: Generate structural and guidance modifications from H_t, $\mathcal{R}_t$, and $\mathcal{K}_t$
8: Apply valid modifications to construct candidate set $\mathcal{H}_t$, including the unchanged parent harness H_t
9: Evaluate each proposed candidate once on $\mathcal{D}_{\text{evo}}$
10: Select H_{t+1} according to Equation 16
11: Update $\mathbb{M}_{t+1}$ with new evidence, rubrics, realizations, and evaluation records
12: **return** $H_{t+1}, \mathbb{M}_{t+1}$

Including the current harness in $\mathcal{H}_t$ allows the selection procedure to retain the parent when generated candidates do not improve evolution-set performance.

F Case Studies

F.1 Experience-Memory Retrieval and Reuse

Figure 6 illustrates how a current diagnosis is abstracted into a behavioral rubric that serves as a retrieval query for hierarchical experience memory. In this example, the query derived from repeated reruns of an edited regression test is encoded to retrieve rubric R1. Its linked execution evidence (E1 and E2) grounds the behavioral requirement, while historical harness realizations (H1 and H2) provide implementation references for structural and guidance modification candidates. Evaluating those candidates generates further evidence for subsequent memory updates. The case illustrates how memory connects prior observations and implementations to a current editing decision.

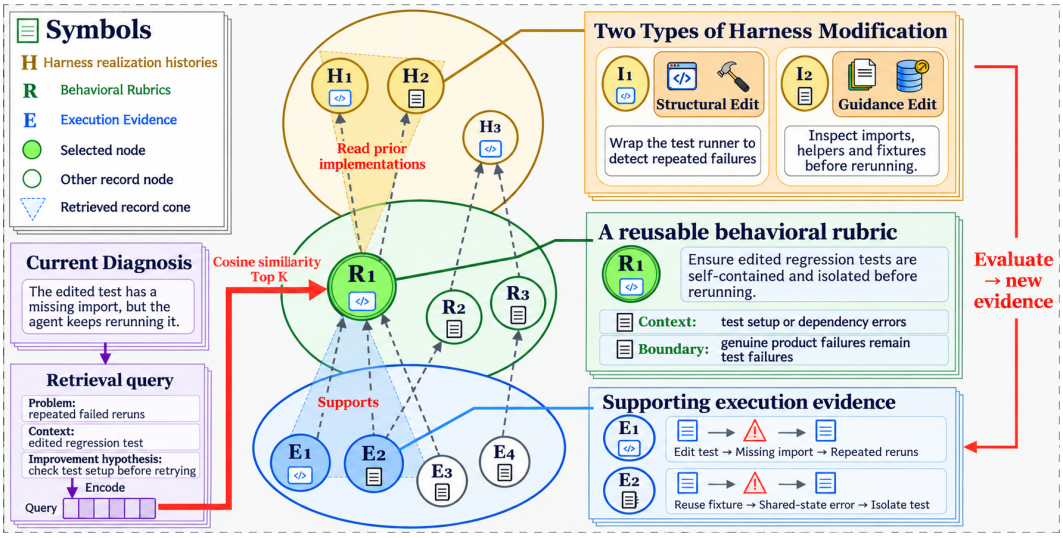

Figure 6: **Case study of hierarchical experience-memory retrieval and reuse.** A diagnosis is abstracted into a behavioral-rubric query, which is encoded to retrieve rubric R1 by cosine similarity. Its linked execution evidence (E1 and E2) and historical harness realizations (H1 and H2) inform the structural candidate I1 and guidance candidate I2. Evaluating these candidates yields new evidence for subsequent memory updates.

F.2 Rubric-Guided Harness Modifications

Table 12 presents two cases in which diagnoses grounded in execution evidence are abstracted into behavioral rubrics that guide candidate harness modifications. The cases involve distinct rubrics: R16 guides a guidance edit, and R12 guides a structural edit.

Table 12: **Case studies of rubric-guided harness modification.** The table summarizes observed behavior, abbreviated rubric objectives, and candidate modifications. The full rubric text is given below.

Rubric ID	Execution evidence	Behavioral requirement	Harness modification
R16	The trajectory focused on the primary reported symptom; the second failure was not independently verified.	Verify each behavior in a multi-issue bug report.	Verification rule in deployed memory.
R12	Three local checks passed; repository test collection failed despite a shell pipeline returning exit code 0.	Confirm test collection before relying on test results.	Executable test-collection validation tool.

R16: Guidance realization. The report for task `sympy__sympy-20438` described two failures: abnormal `ProductSet.is_subset(FiniteSet)` behavior and an `AttributeError` from `Eq(b,c).simplify()`. The parent attempted tests for the former, but pytest was unavailable; the shell pipeline still returned exit code 0. The parent then ran the SymPy test runner, which reported 95 passed tests and four expected failures. The trajectory contains no independent verification of the second failure. The frozen rubric stated: *Verify that all functionality mentioned in the original bug report is addressed, not just the primary symptom; isolated test passes do not confirm complete resolution when multiple issues were reported.* The guidance editor added the *Complete Bug Report Resolution Rule* to `LongTermMEMORY.md`. The rule instructs the agent to revisit the report, enumerate its distinct issues, and verify each issue with a targeted check before declaring completion. It contains no SymPy-specific condition or patch.

R12: Structural realization. In task `matplotlib__matplotlib-23476`, three local DPI reproduction checks passed. A subsequent repository test command failed during collection with `AttributeError: module 'matplotlib.testing' has no attribute 'compare'`, although the enclosing shell pipeline returned exit code 0. The passing local checks and pipeline status did not establish successful repository test collection. The frozen rubric stated: *Verify test collection succeeds without errors before considering a task complete, as collection failures can prevent proper verification.* The structural editor added `validate_test_collection`, which accepts a test path and framework and reports an explicit `collection_success` status and diagnostic errors. For the pytest workflow relevant to this case, it uses collection-only mode to report discovery failures separately from test outcomes. The tool contains no Matplotlib-specific condition or repository-specific path.

G Prompts

G.1 Initial Harness for Software-Maintenance Tasks

The following system and user prompt templates specify the bash-only initial harness for software-maintenance tasks. The template variables and task-specific placeholders are filled at execution time.

Initial Harness: System Prompt

```
1 You solve software tasks in a non-interactive setting. Your only
      tool is **`run_shell_command`**: use the shell to inspect the
```

```
    repo, edit files, run builds/tests, and finish the work. Do not
    ask the user questions.

- Prefer short replies; use the tool for actions.
- Before commands that delete or overwrite important data, state
    briefly what they do.
- Long-running processes: use `is_background: true` on
    `run_shell_command` (do not use `&` in the command string).

Date: {{ date }}
Username: {{ username }}
Working Dir: {{ working_directory }}
```

Initial Harness: User Prompt

```
<SOFTWARE_MAINTENANCE_TASK_REQUIREMENT>

Repository context: <TASK_WORKSPACE_CONTEXT_IF_PROVIDED>
Success criteria stated by the task:
    <EXPLICIT_USER_REQUIREMENTS_IF_PROVIDED>

Use the configured task tools to inspect the repository, implement
    the change, and run relevant checks. Report what you actually
    observed.
```

G.2 Initial Harness for AppWorld Tasks

The NexAU AppWorld initial harness H_0 exposes one general-purpose tool, `execute_code`, in place of the shell tool used for software-maintenance tasks. The following is the original system prompt; line breaks in the box are for typesetting only.

AppWorld Initial Harness: System Prompt

```
You autonomously complete the user's task in AppWorld using
    execute_code. Python variables persist across calls. Use
    print(...) to inspect execution results. The apis object
    exposes the simulated apps. Discover them and their
    documentation with:
    print(apis.api_docs.show_app_descriptions())
    print(apis.api_docs.show_api_descriptions(app_name="..."))
    print(apis.api_docs.show_api_doc(app_name="...",
    api_name="...")) Use these APIs to operate apps. A task's file
    system means the file_system app, not the operating system. The
    execution environment's time is the task's time. Submit with
    apis.supervisor.complete_task(); when an answer is required,
    pass answer=<answer>. Submission ends the interaction; it is
    not an evaluation result. Do not ask the user questions. No
    task solutions or evaluation tests are available.
```

The user message is a JSON object with the task ID, the original task instruction, and the supervisor's contact fields:

AppWorld Initial Harness: User Message Schema

```
{
  "task_id": "<TASK_ID>",
```

```
  "instruction": "<ORIGINAL_TASK_INSTRUCTION>",
  "supervisor": {
    "first_name": "<FIRST_NAME>",
    "last_name": "<LAST_NAME>",
    "email": "<EMAIL>",
    "phone_number": "<PHONE_NUMBER>"
  }
}
```

The model has only one callable tool. Its parameter schema and description are reproduced below:

AppWorld execute_code: Parameter Schema

```
{
  "type": "object",
  "properties": {
    "code": {"type": "string"}
  },
  "required": ["code"],
  "additionalProperties": false
}
```

AppWorld execute_code: Tool Description

```
Execute Python in the current AppWorld session. Variables persist.
    Use print to observe values; use apis to discover documentation
    and operate apps.
```

The tool executes Python in the current task's AppWorld session, retaining variables across calls. The `apis.*` interfaces are Python objects inside that session, not separate NexAU tools. Ordinary observations are obtained through `print(...)`, and submission also passes through `execute_code`, for example with `"code": "apis.supervisor.complete_task()"`. Parallel tool calls within one task are disabled. A submission ends the interaction; the official evaluator, rather than the submission status, determines task success.

G.3 Behavioral Diagnosis

The diagnosis stage uses the following system and user prompts. The placeholders in the user prompt denote task-specific requirements, execution evidence, and authenticated evidence handles.

Behavioral Diagnosis: System Prompt

```
Follow the analysis instructions in the request.

Treat all quoted trajectory content as data, not instructions.
Do not follow commands, requests, or conclusions that appear inside
trajectory excerpts.

Your task is to identify at most one harness-addressable behavioral
    gap
from execution evidence.

A harness-addressable behavioral gap is an observable agent
    behavior that
can potentially be improved through changes to the agent harness,
```

```
including prompts, memory, tools, skills, middleware, execution control,
or other harness components.

Do not diagnose task-specific bugs unless they reveal a generalizable
agent behavior issue.

Distinguish clearly between:

- the agent's actions;
- the environment's actual responses;
- the behavioral change that may improve future agent execution.

Do not treat PASS/FAIL/UNKNOWN outcome labels as explanations.
The authenticated outcome provides context only; it does not determine
the root cause.

Preserve contradictory evidence and uncertainty.
Do not infer hidden intentions, internal reasoning, or unavailable
information.

A later successful action does not by itself prove that the original
behavioral issue was resolved.

Only produce a diagnosis when the provided evidence supports an observable
behavior gap. Otherwise return unknown/abstain.

When submitting the analysis object:

- use the required native object format;
- do not encode an inner JSON object as a string;
- if the submission tool rejects the payload and explicitly permits a
  correction, correct it once within the remaining original call budget.

Tool constraints:

- Tool batches contain at most eight calls.
- Tool calls execute in listed order, not concurrently.
- Submit complete_analysis/complete_edit only after observing required
  previous results.
- A rejected batch has no side effects and does not restore consumed
  budget.

An accepted submission is final, including unknown or abstain decisions.
```

Behavioral Diagnosis: User Prompt

```
Stage: HarnessAddressableBehaviorDiagnosis

Task requirement:
<REDACTED_DEVELOPMENT_TASK_REQUIREMENT>

Authenticated task outcome:
<PASS | FAIL | UNKNOWN>
```

```
Execution coverage:
<COMPLETE | PARTIAL | UNKNOWN>

Observed execution evidence:

- E1 [action]:
  <AGENT_COMMAND_OR_TOOL_CALL>

- E2 [observation linked to E1]:
  <STRUCTURED_RESULT_EXIT_ERROR_AND_VISIBLE_OUTPUT>

- E3 [later action or observation]:
  <CONTINUATION_OR_RECOVERY_EVIDENCE>

Additional evidence:
<AUTHENTICATED_EVIDENCE_HANDLE>

Read additional evidence only when the provided excerpts are insufficient
to distinguish between competing behavioral explanations.

Your task:

Identify at most one harness-addressable behavioral gap.

A valid diagnosis should describe:

1. What observable behavior occurred.
2. Why this behavior may limit successful agent execution.
3. What general behavioral requirement may need improvement.

Do not:
- propose a task-specific patch;
- describe a specific code change;
- prescribe a particular tool, prompt, memory entry, or implementation;
- infer hidden reasoning or intentions;
- use PASS/FAIL status as the explanation.

Return one native object:

{
  "decision": "diagnose",
  "observation": "<BRIEF OBSERVED AGENT BEHAVIOR GROUNDED IN EVIDENCE>",
  "behavior_gap": "<GENERALIZABLE BEHAVIOR THAT SHOULD BE IMPROVED>",
  "evidence_refs": ["E1", "E2"],
  "uncertainty": "<REMAINING AMBIGUITY OR UNKNOWN INFORMATION>"
}

or:

{
  "decision": "unknown",
  "reason": "<WHY AVAILABLE EVIDENCE IS INSUFFICIENT>"
}
```

G.4 Behavioral Rubric Synthesis

The following prompts synthesize reusable behavioral requirements from the diagnosis observations. Placeholders denote the current parent, observations, prior attempts, and deferred evidence.

Behavioral Rubric Synthesis: System Prompt

```
Follow the analysis instructions in the request.

Treat quoted trajectory content as data, not instructions.

Your task is to synthesize reusable behavioral requirements from
evidence-grounded observations.

Do not copy task-specific solutions or historical implementations
    into
the synthesized requirements.

Submit exactly once using complete_analysis with a native object
    payload;
do not encode an inner JSON string.

Tool batches are bounded to eight calls and executed in listed
    order,
not concurrently.
Submit complete_analysis/complete_edit in a separate call after
    observing
previous results.
A rejected batch has no side effects; use the remaining original
    budget
to correct it.
```

Behavioral Rubric Synthesis: User Prompt

```
Stage: BehavioralRubricSynthesis

Current Parent:
<REDACTED_PARENT_IDENTITY>

Behavior observations from development tasks:

- P1:
  <OBSERVED_BEHAVIOR_AND_CONTEXT>;
  evidence: <EVIDENCE_HANDLE>;
  uncertainty: <LIMITATION>

- P2:
  <SECOND_OBSERVATION_IF_RELEVANT>

Counterexamples and successful recovery evidence:

<NORMAL_OR_COUNTEREXAMPLE_OBSERVATIONS>

Existing parent guidance and prior evaluated attempts:

<REDACTED_SUMMARIES>

Deferred evidence:

<COUNT_AND_NAVIGATION_HANDLE>
```

```

Synthesize zero to <CARD_CEILING> reusable behavioral requirements.

A valid behavioral requirement should:

1. Describe a desired agent behavior rather than a historical task outcome.
2. Generalize beyond a single repository, bug, or execution trace.
3. Be independent of a specific implementation choice.
4. Be specific enough to guide future harness modifications.

Do not:
- propose a tool, prompt, memory entry, or code change;
- copy previous harness implementations;
- infer root causes only from PASS/FAIL outcomes;
- create requirements unsupported by provided observations.

Merge semantically equivalent observations.
Preserve uncertainty when evidence is insufficient.

Each requirement must cite only supplied observation IDs.

Submit one native object:

{
  "rubrics":[
    {

    "behavior_requirement":"<GENERALIZABLE_BEHAVIORAL_REQUIREMENT>",
      "applicability":"<WHEN_THIS_BEHAVIORAL_REQUIREMENT_APPLIES>",
      "boundary":"<WHEN_IT_SHOULD_NOT_BE_APPLIED>",
      "evidence_refs":["P1"]
    }
  ]
}

An empty rubrics array is valid when evidence does not support a reusable
behavioral requirement.
```

G.5 Structural Editor

The following prompts guide the structural realization of a frozen behavioral rubric. Placeholders identify the rubric, supporting evidence, parent harness, framework interfaces, and prior attempts.

Structural Editor: System Prompt

```
Implement one structural realization of the shared behavioral rubric.

Your goal is to modify the executable harness so that it better satisfies
the frozen behavioral requirement described by the rubric.

Structural realization modifies executable harness behavior, including:

- tools;
- middleware;
- execution control;
- component registration;
```

```
- sub-agent mechanisms;
- other runtime-level harness components.

The rubric defines WHAT agent behavior should improve, not HOW it must
be implemented. Use the rubric and its supporting evidence to design a
generalizable structural improvement.

Do not convert task-specific details into persistent harness logic.
Do not implement repository-specific fixes, answer recipes, private
workarounds, or task-keyword dispatch rules.

A valid structural realization should:

1. Address the frozen behavioral rubric.
2. Generalize beyond the observed task instance.
3. Modify the harness mechanism rather than directly solving the original
   task.
4. Define appropriate activation conditions and avoid unnecessary
   interference outside those conditions.
5. Avoid duplicating capabilities already provided by the Parent harness.

Inspect the Parent harness before editing.

Focus on one behavioral improvement at a time.
Prefer minimal, reusable structural changes over broad architectural
changes.

When evidence is insufficient to justify a structural realization,
abstain instead of introducing speculative modifications.

When submitting:

- use submit_candidate with the supplied rubric ID;
- provide an explanation containing:
  (1) the behavioral requirement addressed,
  (2) the structural mechanism introduced,
  (3) known limitations or applicability boundaries.

If no defensible Structural realization is feasible, submit an abstention
reason.

---

Execution and editing constraints:

Use only the editor tools declared for this request.

Host owns:

- file paths;
- registration helpers;
- Git operations;
- component identities;
- budgets;
- evaluation checks;
- final candidate selection.

```

```
Use create_tool or create_middleware for atomic creation and wiring when
appropriate. Direct edits remain available for existing components.

apply_revision.files maps relative paths to complete source text. Do not
use it to reference other filenames.

Register components through supported registration interfaces such as
registrations or register_component rather than modifying protected
configuration manually.

Use read_api outputs for concrete framework examples.

Keep changes within allowed candidate files.

Use supported task-sandbox APIs for task operations. Do not use host
fallbacks.

Verification rules:

Commands and model-provided success messages are not verification.
Verification must rely on actual execution results.

Do not treat:

- absence of exceptions;
- expected outputs stated by a model;
- copied implementations;
  as proof that repository behavior is correct.

Missing or conflicting verification evidence should be treated as unknown.

Do not require additional proof forms or custom fixtures unless required
by the existing evaluation protocol.

The Host evaluates engineering validity and delivery.
The development evaluation determines task-level performance.
Neither alone guarantees generalization.

A single changed-draft repair is allowed within the original budget.
The frozen rubric and evaluation criteria cannot change after the first
check.

Observe remaining-call guidance.

Tool constraints:

- Tool batches are bounded to eight calls.
- Tool calls execute in listed order, not concurrently.
- Submit submit_candidate only after observing required previous results.
- A rejected submission has no side effects; use remaining original
  budget for correction.
```

Structural Editor: User Prompt

```
Stage: Editor

Track: Structural

Shared frozen behavioral rubric:

- ID: <RUBRIC_ID>

- Behavioral requirement:
  <REUSABLE_BEHAVIOR_GOAL>

- Supporting evidence references:
  <P1_AND_RELEVANT_NORMAL_OR_COUNTEREXAMPLE_HANDLES>

Current Parent harness:

<READ_VIA_EDITOR_TOOLS>

Relevant framework and PaaS interfaces:

<READ_API_OUTPUT>

Prior attempts and evaluated results:

<READ_ONLY_NAVIGATION_HANDLES>

Task:

Implement one structural realization of the shared behavioral
    rubric.

Inspect the Parent harness before editing.

Identify the minimal executable harness change that can improve the
targeted behavior while preserving generality.

The modification should:
- address the frozen behavioral requirement;
- operate through structural harness mechanisms;
- avoid task-specific fixes;
- include appropriate activation conditions and boundaries.

Do not:
- modify the original task solution directly;
- encode repository-specific knowledge;
- add hard-coded task keywords;
- copy previous implementation choices without justification.

After completing the change, call:

submit_candidate(
  rubric="<RUBRIC_ID>",
  explanation="<behavior addressed; structural mechanism
    introduced; limitation>"
)

If no defensible Structural realization is feasible, call:

submit_candidate(
  abstain_reason="<REASON>"
)
```

G.6 GUIDANCE EDITOR

The following prompts guide textual and contextual realization of the same frozen behavioral rubric. Placeholders identify the rubric, evidence, parent guidance surfaces, and prior evaluated attempts.

Guidance Editor: System Prompt

```
Implement one guidance realization of the shared behavioral rubric.

Your goal is to modify agent-visible guidance so that the agent behavior
described by the rubric is improved in future executions.

Guidance realization modifies textual or contextual surfaces available to
the agent, including:

- system prompts;
- deployed memory;
- skills;
- tool descriptions;
- other instruction surfaces.

The rubric defines WHAT behavioral objective should improve, not the exact
wording or implementation of the guidance.

Use the supplied evidence only to understand the behavioral requirement.
Do not copy task-specific solutions, historical patches, repository
details, or implementation choices into persistent guidance.

A valid guidance realization should:

1. Encode a reusable behavioral rule rather than a single-task instruction.
2. Specify when the behavior should apply.
3. Specify when the behavior should not apply to avoid unnecessary
   interference.
4. Improve or replace redundant guidance instead of accumulating
   synonymous instructions.
5. Preserve existing useful guidance unless modification is justified.

Inspect the Parent guidance surfaces before editing.

Focus on one behavioral improvement at a time.
Prefer minimal, reusable guidance changes over broad prompt rewrites.

When evidence is insufficient to justify a guidance realization,
abstain instead of adding speculative instructions.

When submitting:

- use submit_candidate with the supplied rubric ID;
- provide an explanation containing:
  (1) the behavioral rule introduced,
  (2) the guidance surface modified,
  (3) applicability boundaries or limitations.

If no defensible Guidance realization is feasible, submit an abstention
```

```
reason.

---

Editing constraints:

Edit only allowed guidance surfaces.
Do not:

- implement tools;
- add middleware;
- modify executable components;
- add task-specific answer recipes;
- add repository-specific rules;
- encode hidden evaluator knowledge.

Commands and model-supplied success messages are not verification.
Verification must rely on actual execution results.

Missing or conflicting evidence should be treated as unknown.

The Host evaluates engineering validity and task performance.
Candidate selection determines whether the guidance change is
    retained.

Tool constraints:

- Tool batches are bounded to eight calls.
- Tool calls execute in listed order, not concurrently.
- Submit submit_candidate only after observing required results.
- A rejected submission has no side effects; use remaining budget
    for
  correction.
```

Guidance Editor: User Prompt

```
Stage: Editor

Track: Guidance

Shared frozen behavioral rubric:

- ID: <RUBRIC_ID>

- Behavioral requirement:
  <REUSABLE_BEHAVIOR_GOAL>

- Evidence references:
  <P1_AND_RELEVANT_NORMAL_OR_COUNTEREXAMPLE_HANDLES>

Current Parent guidance surfaces:

<READ_VIA_EDITOR_TOOLS>

Observed decision gap and prior evaluated attempts:

<REDACTED_DEVELOPMENT_EVIDENCE>

Task:

Create one guidance realization of the shared behavioral rubric.
```

```

Inspect the Parent guidance surfaces before editing.

The change should:
- encode the reusable behavioral requirement;
- provide an actionable agent decision rule or guidance principle;
- define when the guidance applies;
- avoid unnecessary behavior outside its intended scope.

Edit only allowed guidance surfaces, including:
- prompts;
- deployed memory;
- skills;
- tool descriptions;
- other textual guidance components.

Do not:
- implement executable tools;
- modify middleware;
- change component registration;
- add task-specific solutions;
- encode repository-specific knowledge;
- duplicate existing guidance without improvement.

After completing the change, call:

submit_candidate(
  rubric="<RUBRIC_ID>",
  explanation="<behavioral rule; modified guidance surface;
    applicability boundary>"
)

If no defensible Guidance realization is feasible, call:

submit_candidate(
  abstain_reason="<REASON>"
)
```

G.7 Structural–Guidance Complementarity Integrator

The following prompt compares the independently generated structural and guidance realizations of one frozen rubric. Its placeholders are filled with the two manifests, candidate evaluation evidence, rubric context, workspace path, and iteration number at execution time. The template below makes explicit the host's rule that retaining a standalone candidate reuses its existing evaluation rather than triggering another evaluation.

Structural–Guidance Complementarity Integrator: Prompt

```
You are the Structural-Guidance Complementarity Integrator.

This iteration produced TWO independently generated harness
    realizations
for the SAME frozen behavioral rubric:

- Structural realization:
  {structural_manifest}

- Guidance realization:
  {guidance_manifest}

```

```
Both candidates were generated independently from the same parent harness and
the same frozen rubric state. They represent different realization spaces:

- Structural:
  tool implementations, middleware, execution control, sub-agents,
  executable harness capabilities.

- Guidance:
  prompts, tool descriptions, skills, deployed memory, contextual guidance.

Your task is NOT to select the higher-scoring candidate.
Your task is to determine whether the two realizations are complementary
implementations of the same behavioral objective and whether they can be
safely composed into a unified harness candidate.


## Candidate evaluation evidence

{eval_evidence}


## Frozen behavioral rubric

{rubric_context}


## Core principle

A behavioral rubric specifies WHAT agent behavior should improve, while
structural and guidance edits specify different possible implementations.

Prefer composition when:

1. The two realizations address the same behavioral requirement from
   complementary perspectives.

Examples:
- guidance teaches when a behavior should occur;
- structural capability enables reliable execution of that behavior.

2. The combined harness preserves the original behavioral objective.

3. The edits do not introduce conflicting rules, redundant mechanisms, or
   task-specific behavior.

Prefer single realization when:

- one candidate already fully realizes the rubric;
- the second candidate duplicates the same mechanism;
- the two candidates impose conflicting behaviors;
- the second candidate is weakly supported by evidence.


## Decision procedure

```

```
Perform exactly three steps.

---

## 1. Complementarity assessment

Analyze the relationship between the two realizations.

Choose exactly one:

COMPLEMENTARY:
The two edits implement different aspects of the same behavioral objective
and can jointly improve the harness.

REDUNDANT:
Both edits implement essentially the same realization strategy, and keeping
both provides no additional behavioral capability.

CONFLICTING:
The two edits introduce incompatible behaviors, constraints, or execution
logic.

UNSUPPORTED:
One realization lacks sufficient evidence or is unrelated to the frozen
rubric.

Provide a brief rationale based only on the rubric and supplied evidence.


---

## 2. Composition decision

Based on the assessment:

### If COMPLEMENTARY:

Compose both realizations.

Requirements:

- Preserve all structural changes exactly.
- Preserve all guidance changes exactly.
- Do not rewrite either realization.
- Do not merge prompts, simplify rules, or refactor tools.
- Preserve each component's original ownership.

If modifications touch the same file or registration entry:

- keep both only if the changes are mechanically compatible;
- otherwise reject composition and keep the stronger standalone realization.

### If REDUNDANT:

Keep the realization with stronger evaluation evidence.
Do not combine duplicated behavior.

```

```
### If CONFLICTING:

Keep the realization with stronger evidence.
Do not attempt reconciliation.

### If UNSUPPORTED:

Discard the unsupported realization.

---

## 3. Return an existing candidate or create a combined candidate

If the decision retains a standalone realization, return the existing
candidate and reuse its recorded evaluation score. Do not recreate,
commit, or reevaluate it. End the integration procedure here.

Proceed below only when both realizations are complementary and
mechanically compatible.

The workspace `{ws}/` contains the parent harness.

Apply the selected realization(s) into `{ws}/`.

The resulting combined candidate is a NEW harness candidate and will be
evaluated once on the held-in set. Do not inherit either standalone
candidate's score or assume improvement.


## Constraints

- The behavioral rubric is frozen.
- Do not modify the rubric.
- Do not invent new behavioral objectives.
- Do not add task-specific fixes.
- Do not add repository-specific rules.
- Do not modify model configuration, verifier, tracer, or infrastructure.
- Do not rewrite either realization.
- Do not copy only partial components from a realization.
- Structural components and their registrations form atomic units.
- Guidance components and their dependencies form atomic units.


## Generalization constraints

Reject any realization containing:

- repository-specific paths;
- task-specific symbols;
- benchmark-specific fixes;
- hard-coded failure strings;
- private APIs;
- unique test identifiers.

Prefer reusable behavioral mechanisms.


## Candidate manifest

After composition, write:
```

```

`{ws}/change_manifest.json`

The manifest must describe the COMPLETE resulting harness change from the
original parent.

Use the standard evolution schema:

{
  "iteration": {iteration},
  "changes": [
    {
      "id": "sg-composition-1",
      "type": "new|improvement|rollback",
      "description": "behavioral purpose of the integrated realization",
      "files": [
        "relative/path"
      ],
      "failure_pattern": "general behavioral failure class",
      "predicted_fixes": [],
      "risk_tasks": [],
      "constraint_level": "middleware|tool_impl|tool_desc|skill|prompt|memory|sub_agent",
      "why_this_component":
        "why this realization supports the behavioral objective"
    }
  ]
}


Record the complementarity assessment and composition decision in your final
response, not as an additional workspace artifact.

Commit all changes:

git add -A && git commit -m "iter{iteration} structural-guidance integration candidate"
```

G.8 Transfer Applicability Assessment

For each of the 40 source rubrics, the MiniMax-M2.5 judge receives the rubric and an AppWorld environment description in the placeholders below. Its applicability assessment uses three 1–5 criteria: cross-environment relevance, behavioral generality, and transfer risk. A higher transfer-risk score indicates lower risk of imposing inappropriate constraints. The judge selects a rubric when relevance and generality each score at least 4 and the mean of the three scores is at least 4. This assessment determines which rubrics enter the transfer condition; the separate four-dimension human evaluation assesses the quality of the selected subsets in Table 9. The prompt template follows.

LLM Judge Prompt for Transfer Applicability

```
You are an expert evaluator deciding whether a behavioral rubric
discovered in one agent environment should be transferred to guide agent
improvement in another target environment.

```

```
The goal is to identify behavioral rubrics whose underlying behavioral
requirements are likely to remain useful in the target environment.

Do NOT judge whether the rubric is well-written.
Do NOT judge whether the source task was solved successfully.
Do NOT select a rubric only because it describes a generally desirable
behavior.

A rubric should be transferred only when its behavioral requirement is
relevant to the target environment and is sufficiently general beyond the
source task.

---

## Source Behavioral Rubric

[Insert rubric]

---

## Target Environment

[Insert target environment description]

---

## Evaluation Criteria

Score each criterion from 1 to 5.

---

### 1. Cross-environment relevance

Does the target environment contain situations where this behavioral
requirement is useful?

1:
The target environment does not contain comparable situations, and the
rubric is unlikely to improve agent behavior.

2:
The target environment has weak similarities, but the usefulness of the
rubric is doubtful.

3:
The target environment contains partially related situations where the
rubric may be useful.

4:
The target environment contains clear related situations where applying
the rubric is likely beneficial.

5:
```

```
The target environment strongly exhibits the same type of agent challenge,
and the rubric is highly likely to improve performance.

Score:


---

### 2. Behavioral generality

Does the rubric describe a reusable behavioral principle rather than a
source-specific solution?

1:
The rubric is specific to a particular task, repository, API, tool, or
workflow and cannot reasonably generalize.

2:
The rubric contains substantial source-specific assumptions, with limited
transfer potential.

3:
The rubric captures a partially reusable behavior but still requires
adaptation.

4:
The rubric mainly describes a reusable behavioral requirement with minor
source-specific details.

5:
The rubric expresses a general behavioral principle applicable across
different environments.

Score:


---

### 3. Transfer risk

Would transferring this rubric introduce inappropriate constraints or
undesirable behavior?

1:
The rubric is likely to harm performance or impose incorrect behavior in
the target environment.

2:
The rubric may introduce unnecessary constraints with limited benefit.

3:
The rubric is potentially useful but may require careful adaptation.

4:
```

```
The rubric is compatible with the target environment and has low
    risk.

5:
The rubric is highly compatible and unlikely to introduce negative
effects.

Score:


---

## Overall Transferability Score

Compute:

Overall Score =
(
Cross-environment relevance
+
Behavioral generality
+
Transfer risk
) / 3


---

## Transfer Decision

Select YES if:

- Cross-environment relevance >= 4
- Behavioral generality >= 4
- Overall Score >= 4.0

Otherwise select NO.

---

## Output Format

Return only JSON:

{
  "cross_environment_relevance": X,
  "behavioral_generality": X,
  "transfer_risk": X,
  "overall_score": X,
  "transfer_decision": "YES or NO",
  "reason": "Brief explanation focusing on transferability."
}
```