

When the Critic Falls Behind: Co-Evolving Rubrics and Harnesses for Continual Agent Self-Improvement

Hao Chen¹, Yukun Yan^{2*}, Ziyu Han¹, Ye He¹, Chuanzhe Guo¹, Shangzhan Li¹,
Qi Shi², Qingfu Zhu¹, Maosong Sun², Wanxiang Che^{1*}

¹Research Center for Social Computing and Interactive Robotics, Harbin Institute of Technology

²Department of Computer Science and Technology, Institute for AI, Tsinghua University

Abstract

Autonomous agents increasingly depend on harnesses that govern interaction, memory, tools, and execution around large language models (LLMs). While recent work has begun to evolve these harnesses from execution experience, the supervision used to guide evolution often remains relatively static, making it difficult to track changing behavioral weaknesses across rounds and potentially limiting generalization to unseen tasks. To address this issue, we propose RuHarness, a rubric–harness co-evolution framework for continual agent self-improvement. RuHarness continually discovers and reassesses behavioral criteria from execution trajectories, prioritizes the most relevant unresolved weaknesses, and uses them to guide targeted harness updates while keeping the task objective fixed. Extensive experiments across diverse agent tasks and backbone models show that RuHarness outperforms strong harness-evolution baselines, particularly on unseen held-out tasks. Further analyses show that adaptive rubric prioritization tracks shifting weaknesses and supports more sustained improvement and stable generalization across evolution rounds.

1 Introduction

As autonomous agents become increasingly prevalent, their performance depends not only on the capabilities of the underlying large language model (LLM) but also on the surrounding harness that governs interaction, memory, tools, and execution (Luo et al., 2026; Pan et al., 2026). Recent work has begun to evolve the harness itself from execution experience, enabling agents to improve without updating model parameters (Zhang et al., 2026a; Lin et al., 2026b; Lee et al., 2026a). While such approaches can yield substantial gains, recent controlled evaluations suggest that these improvements may weaken on unseen tasks (Wang et al., 2026c),

* Corresponding Authors.

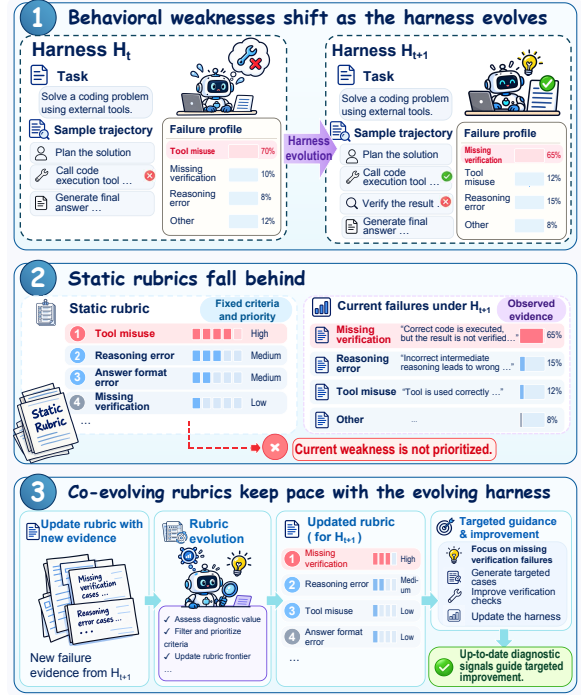


Figure 1: Illustration of rubric–harness co-evolution. Static rubrics may fall behind as behavioral weaknesses shift during harness evolution, while co-evolving rubrics track emerging failures and guide harness improvement.

raising questions about whether harness optimization produces generalizable improvement.

One challenge behind this limited generalization is that repeated harness optimization may gradually overfit to task-specific failure patterns exposed during evolution (Srikanth et al., 2026). Existing methods optimize harness against task-level feedback, which indicates whether the agent succeeds but provides limited information about the behavioral deficiencies underlying failure (Zhang et al., 2026a). This limitation becomes increasingly important during continual evolution: as the harness improves, previously dominant failures may be mitigated, while subtler or previously unseen failure patterns emerge (Nie et al., 2026). Figure 1 il-

illustrates that when the guidance used for evolution remains coarse-grained and relatively static, its discriminative and guiding value may gradually diminish (Kim et al., 2025; Tian et al., 2026). Consequently, the system lacks an explicit mechanism for tracking how the importance of different weaknesses changes throughout evolution. Continual harness evolution requires adaptive diagnosis that continually reassesses and reprioritizes behavioral weaknesses while keeping the underlying task objective fixed (Dou et al., 2026; Zhang et al., 2026c).

To this end, we propose RuHarness (**Rubric-Harness Co-Evolution**), a framework for continual agent self-improvement through joint rubric and harness evolution. RuHarness continually derives behavioral criteria from execution trajectories, reassesses their relevance under the current harness, and prioritizes those that best capture unresolved weaknesses. The resulting rubric frontier guides targeted and coordinated updates to the corresponding harness components, while candidate harnesses are selected using task performance on an independent validation set. By adapting the diagnostic guidance as agent behavior changes while keeping the task objective fixed, RuHarness continually focuses harness optimization on the weaknesses that matter most at each stage of evolution.

Extensive experiments across diverse agent tasks and backbone models demonstrate that RuHarness consistently outperforms strong harness-evolution baselines, with particularly clear gains on unseen held-out tasks. The analyses also show that the rubric frontier adapts to shifting behavioral weaknesses, concentrates on diagnostically relevant criteria, and supports more sustained improvement across evolution rounds. These results indicate that co-evolving rubrics with the harness provides adaptive supervision for continual agent improvement, supporting both continued evolution across evolution rounds and stronger generalization.

2 Related Work

Self-Evolving Agents. Agent self-evolution has gradually shifted from storing interaction experience to optimizing persistent external state. Reflection- and memory-based methods refine experience to improve future decisions (Shinn et al., 2023; Zhong et al., 2023), while skill evolution treats reusable skills as editable knowledge that can be refined from execution feedback (Zhu et al., 2026; Yang et al., 2026). These paradigms

extend naturally to the broader agent harness, where the optimization target expands from a single external component to the execution infrastructure, including prompts, tools, memory, and control logic (Zhang et al., 2026d). Recent harness-evolution methods seek to diagnose failures from trajectories and iteratively revise this infrastructure (Jiang et al., 2026b), with subsequent work broadening the editable space and making the evolution process more observable and verifiable (Lee et al., 2026b; Lin et al., 2026a). More broadly, recent studies have explored recursive self-improvement and harness-model co-evolution across repeated adaptation rounds (Xia et al., 2026; Yu et al., 2026; Chen et al., 2026b; Fan and Huang, 2026). Yet these approaches still largely treat the diagnostic signals guiding evolution as fixed, leaving how such guidance should adapt to shifting behavioral weaknesses underexplored.

Rubrics for Self-Improvement. Rubrics have evolved from structured evaluation criteria into a flexible interface between evaluation and optimization (Chen et al., 2026a). By decomposing complex objectives into explicit behavioral criteria, they provide fine-grained supervision and have increasingly been incorporated into reward modeling and reinforcement learning (Gunjal et al., 2025; Li et al., 2026a). More recent work makes the rubric itself adaptive: EvoLM (Li et al., 2026b), EvoRubrics (Ding et al., 2026), and DynamicRubric (Wang et al., 2026a) co-evolve evaluative criteria with the optimized policy (Guan et al., 2026), while Wang et al. (2026b) extends this idea to a self-evolving loop over responses, rubrics, and policy. These studies suggest that effective supervision should evolve with the behavior it guides (Jiang et al., 2026a). However, existing work mainly co-evolves rubrics with policies or responses (Li et al., 2026d), leaving rubric adaptation for continual harness evolution largely unexplored.

3 Method

In this section, we first present the overall design of RuHarness (§3.1), then describe how rubric evolution tracks the changing behavioral distribution (§3.2) and guides subsequent harness evolution (§3.3), as illustrated in Figure 2.

3.1 Overview of the Framework

RuHarness establishes a co-evolution process between the diagnostic rubric and the agent harness

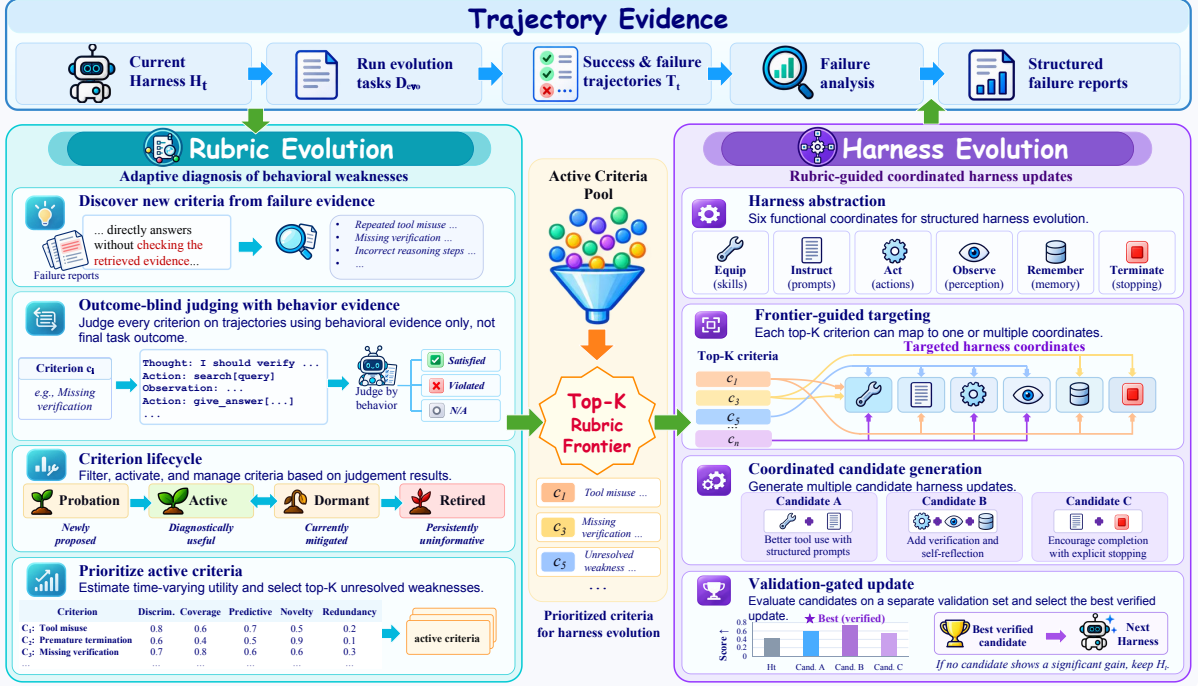


Figure 2: Overview of the Rubric-Harness Co-Evolution Framework. Rubric evolution adaptively identifies and prioritizes unresolved behavioral weaknesses from trajectory evidence, while harness evolution performs targeted updates accordingly, yielding new behaviors that drive the next round of evolution.

through iterative interaction. At evolution round t , the system state is represented by $\mathcal{S}_t = (H_t, R_t)$, where H_t denotes the current harness and R_t the maintained rubric pool. RuHarness updates the diagnostic signal from the behavior exposed by the current harness and uses the resulting signal to guide harness updates, thereby alternating between rubric evolution and harness evolution.

Alternating evolution process. At each round, the current harness H_t is executed on a set of evolution tasks D_{evo} to collect agent trajectories $\mathcal{T}_t$. These trajectories are analyzed to identify remaining failure patterns, which drive the rubric update:

$$R_{t+1} = \mathcal{E}_R(R_t, \mathcal{T}_t), \quad (1)$$

where $\mathcal{E}_R$ discovers new behavioral criteria and reassesses existing ones from $\mathcal{T}_t$ detailed in §3.2. The updated rubric pool is then prioritized to obtain a set of high-priority criteria $\hat{R}_{t+1}$, which captures the behavioral weaknesses most relevant to the current harness. These criteria guide the rubric-conditioned harness update:

$$H_{t+1} = \mathcal{E}_H(H_t, \hat{R}_{t+1}), \quad (2)$$

where $\mathcal{E}_H$ denotes the harness evolution procedure detailed in §3.3. The updated harness induces a

new behavioral distribution in the next round, providing fresh evidence for subsequent rubric evolution and closing the co-evolution loop. The two processes are therefore coupled through agent behavior: the harness shapes the weaknesses that emerge, while the rubric determines which weaknesses are prioritized to guide subsequent evolution.

3.2 Rubric Evolution: Adapting to Shifting Agent Weaknesses

Given trajectories from H_t , RuHarness summarizes each failed run into a structured failure report and aggregates these reports to identify recurring cross-task behavioral patterns. These patterns are converted into new criteria $\mathcal{R}_{t+1}^{\text{new}}$, which capture recurring weaknesses across tasks while remaining verifiable from trajectory evidence.

Outcome-blind behavioral assessment. Newly discovered criteria are evaluated together with existing criteria on trajectories generated by the current harness. For each criterion c and trajectory τ , an outcome-blind judge assigns:

$$J(c, \tau) \in \{\text{Satisfied}, \text{Violated}, \text{N/A}\}, \quad (3)$$

where the judge has access only to behavioral evidence in the trajectory and is blinded to the final task outcome. This prevents criterion assess-

ment from directly leaking success or failure labels. Here, *N/A* indicates that the behavior specified by the criterion does not occur, whereas *Satisfied* indicates that it occurs and is handled correctly.

Criterion admission and lifecycle. Based on these assessments, newly discovered criteria first undergo a *birth check* stage before entering the maintained rubric pool. Criteria with insufficient diagnostic value are filtered at this stage, including those that are rarely applicable, fail to identify violations on unsuccessful trajectories, or trigger indiscriminately across outcomes. Criteria that pass this admission stage enter a four-state lifecycle: *probation*, *active*, *dormant*, and *retired*. Newly admitted criteria enter probation and are promoted to active once they exhibit sufficient diagnostic value. Active criteria become dormant when their associated failures are no longer observed, and may be reactivated upon recurrence, while persistently uninformative criteria are eventually retired. Only *active* criteria are subsequently prioritized and used to guide harness evolution.

Time-varying diagnostic utility. To determine which active criteria are most useful for the current harness, we assign each criterion a time-dependent diagnostic utility:

$$U_{t+1}(c) = \alpha D_{t+1}(c) + \beta C_{t+1}(c) + \gamma P_{t+1}(c) + \delta N_{t+1}(c) - \lambda \text{Red}_{t+1}(c), \quad (4)$$

where $D_{t+1}(c)$ denotes *discrimination* between successful and failed trajectories, $C_{t+1}(c)$ measures the fraction of current failures *covered* by the criterion, $P_{t+1}(c)$ measures *predictive validity* with respect to task failure, $N_{t+1}(c)$ captures *novel* failures identified by the criterion, and $\text{Red}_{t+1}(c)$ penalizes behavioral *redundancy* with existing criteria. These statistics are derived from the preceding behavioral assessments and the corresponding task outcomes, and are recomputed for each harness state. Consequently, $U_{t+1}(c)$ evolves with the harness: criteria associated with mitigated failures decrease in priority, while those capturing newly prominent failures receive higher utility.

Adaptive rubric frontier. Since statistics can be uncertain under limited observations, we rank active criteria using a conservative posterior estimate of diagnostic utility. Specifically, we estimate a Dirichlet posterior from the joint task-outcome and criterion-assessment counts and use the lower q -quantile of the induced utility distribution:

$$\tilde{U}_{t+1}(c) = Q_q[p(U_{t+1}(c) \mid \mathcal{T}_t)], \quad (5)$$

where $\tilde{U}_{t+1}(c)$ denotes the conservative utility of criterion c , and $Q_q(\cdot)$ is the q -th posterior quantile. This reduces the influence of criteria whose high utility is supported by only a few observations. We select the top- K criteria from the active rubric pool:

$$\hat{R}_{t+1} = \text{TopK}_{c \in R_{t+1}^{\text{active}}} \tilde{U}_{t+1}(c), \quad (6)$$

where R_{t+1}^{active} denotes the active criteria and $\hat{R}_{t+1}$ the resulting rubric frontier. The resulting frontier prioritizes unresolved behavioral weaknesses that remain both diagnostically informative and statistically reliable under the current harness.

Further details of rubric evolution are provided in Appendix A.2–A.4.

3.3 Harness Evolution: Optimizing against the Moving Frontier

To structure the harness search space, we represent the editable harness using six functional coordinates: Equip, Instruct, Act, Observe, Remember, and Terminate. These coordinates respectively capture resource provisioning, behavioral instruction, action policy, observation processing, context management, and completion control. Each coordinate defines a corresponding set of editable components in the agent system.

Frontier-guided targeting. Given the current rubric frontier $\hat{R}_{t+1}$, RuHarness identifies the harness coordinates that can directly address each prioritized behavioral weakness. The criterion-to-coordinate mapping is not constrained to be one-to-one, since a single failure pattern may involve multiple parts of the harness. Instead, each criterion may target multiple coordinates, allowing the corresponding interventions to be planned jointly.

Coordinated candidate generation. Based on these criterion-to-coordinate assignments, RuHarness jointly plans a small set of candidate harness updates. Each candidate specifies a concrete hypothesis for addressing the diagnosed weaknesses and may modify multiple related coordinates jointly. We represent each candidate as a coordinated set of coordinate-level modifications:

$$\Delta_j = \left\{ \delta_j^{(k)} \right\}_{k \in M_j}, \quad (7)$$

where M_j is the set of modified coordinates and $\delta_j^{(k)}$ denotes the corresponding modification. The coordinated modifications are applied to the current harness through the update operator $\oplus$, yielding the candidate harness:

$$H_t^{(j)} = H_t \oplus \Delta_j. \quad (8)$$

Each candidate is treated as a single multi-coordinate update, preserving the interactions among jointly planned modifications. Before evaluation, each candidate undergoes a lightweight verification step to check whether the proposed modifications are consistent with the diagnosed weaknesses and supporting evidence. The verifier only accepts or rejects the candidate and does not rewrite it or estimate task performance.

Validation-gated update. Candidates that pass verification are evaluated together with the current harness on a separate set of validation tasks D_{val} , used exclusively for candidate selection and excluded from both evolution and final evaluation. This separation decouples candidate selection from the trajectories used to drive evolution. Let $S(H, D_{\text{val}})$ denote the average validation score of harness H . Let $\mathcal{H}_t^{\text{cand}}$ denote the set of candidate harnesses generated at round t . The candidate with the highest validation score is selected as:

$$H_t^* \in \arg \max_{H \in \mathcal{H}_t^{\text{cand}}} S(H, D_{\text{val}}), \quad (9)$$

and commit the update only if its improvement over the current harness exceeds a threshold ϵ :

$$H_{t+1} = \begin{cases} H_t^*, & S(H_t^*, D_{\text{val}}) - S(H_t, D_{\text{val}}) > \epsilon, \\ H_t, & \text{otherwise.} \end{cases} \quad (10)$$

A candidate is advanced to the next evolution round only if its validation improvement exceeds the acceptance threshold, with the current harness retained otherwise.

4 Experimental Methodology

This section presents our experimental setup, covering models and environment, benchmarks, metrics, baselines, and implementation details.

Models and Environment. We conduct experiments with Qwen3.5-35B-A3B (Team, 2026) and GLM-4.7-Flash (Team et al., 2025) as the backbone models. The agent is instantiated with DeepAgent (Li et al., 2026c), which provides the basic execution framework for tool use, filesystem interaction, and multi-step task solving. Based on this framework, we construct the initial harness and expose its configurable components for subsequent evolution, while keeping the underlying model fixed throughout the experiments. All stages of task execution and evolution are performed using the corresponding backbone model itself.

Benchmarks and Metrics. Evaluation is conducted on EvoAgentBench (Gao et al., 2026), a multi-domain benchmark for long-horizon agent self-evolution. It covers five agent domains: information retrieval based on BrowseComp-Plus (Chen et al., 2025), mathematical reasoning based on OmniMath (Gao et al., 2024), software engineering based on SWE-Bench (Jimenez et al., 2024), code implementation based on LiveCodeBench (Jain et al., 2024), and knowledge work based on GDP-Val (Patwardhan et al., 2025). We use Pass@1 as the primary metric and evaluate each domain with its benchmark evaluator (Tang et al., 2026).

Baselines. For comprehensive comparison, we compare RuHarness against representative self-evolving agent methods that optimize different external states. ACE evolves persistent contextual knowledge from interaction experience (Zhang et al., 2026b), while EvoSkill discovers and refines reusable skills through failure analysis (Alzubi et al., 2026). For harness evolution, we include Self-Harness, which mines model-specific weaknesses and proposes targeted updates (Zhang et al., 2026a); Meta-Harness, which searches executable harness code using accumulated candidates and trajectories (Lee et al., 2026b); and Agentic Harness Engineering (AHE), which improves harnesses through observable and verifiable component updates (Lin et al., 2026a). Together, these baselines cover context, skill, and full-harness evolution.

Implementation Details. We use disjoint evolution (*held-in*) and validation splits, with the test set (*held-out*) reserved for final reporting only. At each round, RuHarness generates up to three coordinated harness candidates and accepts at most one update when its validation improvement exceeds $\epsilon = 0.05$. For diagnostic utility, we set $\alpha = \gamma = 1.0$ and $\beta = \delta = \lambda = 0.5$. Rubric guidance is obtained by ranking criteria with the q -th posterior quantile ($q = 0.2$) and selecting the Top- K criteria with $K = 10$. Each candidate modifies at most three harness components. More experimental details are provided in the Appendix A.1.

5 Results and Analysis

In this section, we first evaluate the performance of RuHarness and conduct ablation studies. We then analyze the evolution of the rubric frontier and how adaptive prioritization guides harness improvement. Finally, we compare evolution trajectories to assess sustained improvement and generalization.

Method	Information Retrieval		Knowledge Work		Software Engineering		Code Implementation		Math Reasoning		Average	
	Held-in	Held-out	Held-in	Held-out	Held-in	Held-out	Held-in	Held-out	Held-in	Held-out	Held-in	Held-out
<i>Qwen3.5-35B-A3B</i>												
Vanilla Agent	28.0	20.0	51.7	40.0	40.0	50.0	83.6	65.1	69.5	46.0	54.6	44.2
ACE (2026b)	31.0	21.5	56.7	40.0	46.7	55.6	86.7	69.8	67.9	45.0	57.8	46.4
EvoSkill (2026)	<u>36.0</u>	<u>27.7</u>	53.3	42.0	44.4	47.2	85.9	<u>68.6</u>	70.3	47.0	58.0	46.5
Self-Harness (2026a)	34.0	<u>27.7</u>	53.3	44.0	<u>51.1</u>	55.6	90.6	67.4	72.9	47.0	60.4	48.3
AHE (2026a)	33.0	24.6	55.0	<u>50.0</u>	53.3	<u>58.3</u>	88.3	66.3	71.1	47.0	60.1	<u>49.2</u>
Meta-Harness (2026b)	34.0	26.2	<u>58.3</u>	36.0	<u>51.1</u>	52.8	89.1	<u>68.6</u>	71.1	<u>49.0</u>	<u>60.7</u>	46.5
RuHarness (ours)	37.0	30.8	60.0	52.0	53.3	63.9	<u>89.8</u>	69.8	<u>72.1</u>	52.0	62.4	53.7
<i>GLM-4.7-Flash</i>												
Vanilla Agent	13.0	6.2	45.0	32.0	31.1	33.3	49.2	43.0	29.5	8.0	33.6	24.5
ACE (2026b)	14.0	4.6	53.3	42.0	37.8	44.4	54.7	47.7	26.8	11.0	37.3	29.9
EvoSkill (2026)	10.0	<u>9.2</u>	50.0	36.0	44.4	38.9	50.0	46.5	29.2	7.0	36.7	27.5
Self-Harness (2026a)	15.0	<u>9.2</u>	50.0	44.0	<u>46.7</u>	<u>47.2</u>	61.7	<u>51.2</u>	29.7	<u>12.0</u>	<u>40.6</u>	<u>32.7</u>
AHE (2026a)	15.0	7.7	46.7	36.0	42.2	41.7	60.2	50.0	29.2	10.0	38.7	29.1
Meta-Harness (2026b)	<u>16.0</u>	7.7	<u>56.7</u>	<u>46.0</u>	40.0	38.9	59.4	47.7	<u>30.0</u>	15.0	40.4	31.1
RuHarness (ours)	18.0	13.8	58.3	56.0	48.9	52.8	<u>60.9</u>	52.3	30.5	15.0	43.3	38.0

Table 1: Performance comparison across five domains on the held-in and held-out sets. Average denotes the macro-average across all five domains. The best result is shown in **bold**, and the second-best result is underlined.

5.1 Overall Performance

We evaluate RuHarness across five agent domains on EvoAgentBench using Qwen3.5-35B-A3B and GLM-4.7-Flash, as shown in Table 1. Overall, RuHarness outperforms all the baselines, achieving the best average performance on both held-in and held-out sets across the two backbone models.

Overall Effectiveness. On Qwen3.5-35B-A3B, RuHarness improves the strongest baseline by 1.7 points on held-in and 4.5 points on held-out. The gains are even larger on GLM-4.7-Flash, reaching 2.7 and 5.3 points, respectively. These results indicate that rubric-harness co-evolution provides stable improvements beyond a specific backbone.

Generalization to Unseen Tasks. The advantage of RuHarness is more pronounced on the held-out sets. Although several baselines achieve competitive or even better results on individual held-in tasks, RuHarness attains the highest average held-out performance for both models and exhibits a smaller held-in-held-out gap. This suggests that its improvements are less tied to the evolution tasks and transfer more effectively to unseen tasks.

Robustness across Tasks and Models. RuHarness performs consistently well across heterogeneous tasks and both backbone models, achieving the strongest average held-out performance and competitive results across individual domains. This suggests that its gains are not tied to a specific task type or backbone.

5.2 Ablation Study

We further examine the contribution of key components of RuHarness, as summarized in Table 2.

Necessity of the Rubric Module. Removing the rubric module lowers the average held-in score from 62.4 to 57.1 and the held-out score from 53.7 to 45.8. This indicates that directly evolving the harness from task feedback and trajectories provides insufficient behavioral diagnosis, highlighting the role of the rubric in translating task outcomes into actionable weaknesses.

Effect of Rubric Evolution. To isolate the effect of rubric evolution, we retain the initial rubric as a fixed diagnostic rubric throughout the evolution process. Surprisingly, this variant performs even worse than removing the rubric module entirely, suggesting that a static rubric can become counterproductive as the harness evolves. When previously dominant failures are mitigated, outdated criteria may continue to steer optimization toward less relevant behaviors. Further analysis of this experimental process is provided in Appendix A.6.

Role of the Rubric Frontier. Using the entire ranked rubric pool instead of the selected frontier also leads to performance degradation. This suggests that more rubrics do not necessarily provide better guidance, as focusing on relevant criteria reduces interference from low-value signals and yields more effective harness updates. A more detailed examination is provided in Appendix A.7.

Method	Information Retrieval		Knowledge Work		Software Engineering		Code Implementation		Math Reasoning		Average	
	Held-in	Held-out	Held-in	Held-out	Held-in	Held-out	Held-in	Held-out	Held-in	Held-out	Held-in	Held-out
RuHarness	37.0	30.8	60.0	52.0	53.3	63.9	89.8	69.8	72.1	52.0	62.4	53.7
w/o Rubric Module	31.0	23.1	51.7	44.0	48.9	52.8	82.8	65.1	71.1	44.0	57.1	45.8
w/o Rubric Evolution	35.0	29.2	55.0	34.0	44.4	55.6	76.6	57.0	71.3	46.0	56.5	44.4
w/o Rubric Frontier	35.0	30.8	56.7	50.0	46.7	50.0	84.4	67.4	72.6	47.0	59.1	49.0

Table 2: Ablation study of RuHarness on Qwen3.5-35B-A3B across five domains.

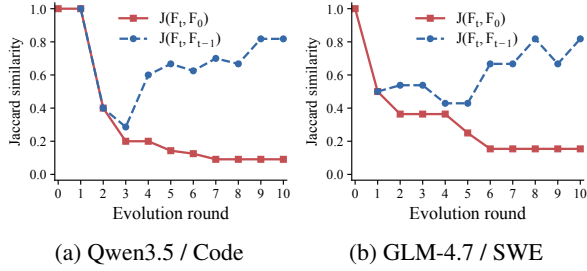
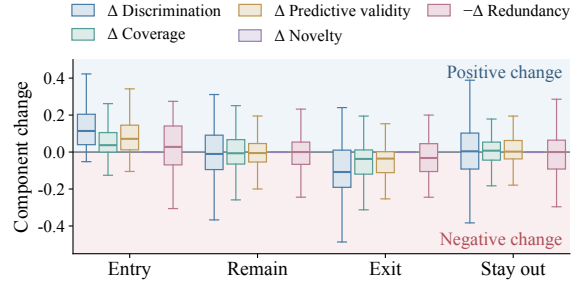


Figure 3: Rubric similarity across evolution rounds.

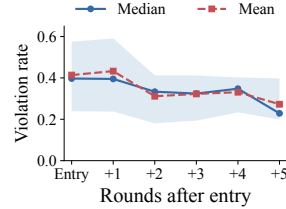
5.3 Analysis of Adaptive Rubric Tracking

To examine how the diagnostic focus changes throughout harness evolution, we measure the Jaccard similarity between rubric frontiers across rounds. Specifically, each frontier is represented by its selected criterion IDs, and we compare the frontier at round t with both the initial frontier F_0 and the frontier from the preceding round F_{t-1} . We conduct this analysis on two representative settings, Qwen3.5-35B-A3B on Code Implementation and GLM-4.7-Flash on Software Engineering.

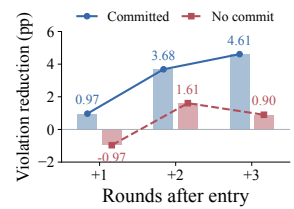
As shown in Figure 3, the similarity to the initial frontier decreases sharply in the early rounds and remains low thereafter. By contrast, the similarity between consecutive frontiers first declines, then gradually increases and stabilizes at a relatively high level. These two trends reveal a characteristic evolution pattern: the diagnostic frontier progressively departs from its initial configuration, while changes between adjacent rounds become increasingly gradual. This behavior suggests that early harness updates substantially reshape the set of weaknesses that require attention, whereas later evolution proceeds through more localized adjustments. Overall, the results confirm that the behavioral weaknesses most relevant to harness improvement are not static, supporting the need for an evolving rubric that continuously tracks changes in the agent’s failure landscape. A finer-grained visualization of criterion-level state transitions across evolution rounds is provided in Figure 6.



(a) Diagnostic changes across frontier transitions.



(b) Violation rates after criteria become prioritized.



(c) Violation reduction with and without harness updates.

Figure 4: Analysis of adaptive rubric prioritization and subsequent harness improvement.

5.4 Effectiveness of Adaptive Rubric Prioritization on Harness Evolution

To examine whether adaptive rubric prioritization translates into subsequent behavioral improvement, we analyze criterion-level dynamics around frontier transitions and after frontier entry. Specifically, we first compare how the diagnostic components of criteria change when they enter, remain in, exit, or stay outside the frontier. We then align newly prioritized criteria at their frontier-entry round and track how their violation rates evolve over subsequent rounds. To further relate this trend to harness evolution, we additionally distinguish cases with and without committed harness updates.

Frontier transitions track diagnostic relevance. To characterize how diagnostic relevance changes with frontier membership, we group criteria by their transitions between two adjacent evolution rounds: entering the frontier, remaining in the frontier, exiting the frontier, or staying outside

the frontier. For each group, we measure the round-to-round changes in the diagnostic utility components defined in Section 3.2, as shown in Figure 4a. As expected, criteria entering the frontier show increases in discrimination, coverage, and predictive validity together with reduced redundancy, while exiting criteria exhibit the opposite pattern. Criteria that remain inside or outside the frontier change substantially less. These results confirm that frontier turnover reflects changes in criterion-level diagnostic relevance rather than arbitrary replacement.

Prioritized weaknesses are progressively mitigated. To assess whether frontier prioritization translates into subsequent behavioral improvement, Figure 4b examines what happens after a criterion enters the frontier. The violation rate remains nearly unchanged in the first round after entry, but decreases noticeably over the following rounds. This delayed reduction suggests that newly prioritized weaknesses are not necessarily resolved immediately, but are progressively mitigated as subsequent harness evolution proceeds. The trend therefore links frontier prioritization with later behavioral improvement, suggesting that the evolving rubric does more than reorganize diagnostic signals and helps focus subsequent optimization on weaknesses that can be progressively reduced.

Committed harness updates yield stronger longer-horizon mitigation. To more directly relate behavioral improvement to harness evolution, Figure 4c compares frontier-entry criteria under committed and non-committed harness updates, where *committed* denotes a candidate update that passes the validation gate and is adopted for subsequent evolution. The committed group shows a clear and sustained increase in violation reduction over subsequent rounds, whereas the non-committed group fluctuates without a consistent improvement trend. This contrast indicates that frontier-guided prioritization is more likely to translate into persistent behavioral gains when the proposed harness update is successfully validated and adopted.

Taken together, these results show that adaptive rubric prioritization tracks changing diagnostic relevance and helps focus subsequent harness evolution on weaknesses that are progressively mitigated.

5.5 Evolution Progress and Generalization

To examine performance across evolution rounds, Figure 5 compares the evolution trajectories of different harness evolution methods on the Qwen3.5-35B-A3B model with the knowledge work task.

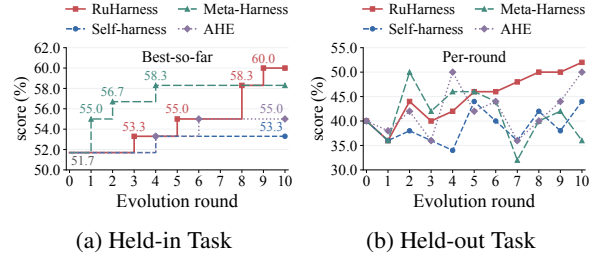


Figure 5: Evolution trajectories of different methods.

We report best-so-far held-in performance to reflect cumulative progress, while held-out performance is evaluated at each round to track generalization.

As shown in Figure 5a, on the held-in evolution tasks, several baselines improve quickly in early rounds but soon plateau, whereas RuHarness continues to gain in later rounds and achieves the highest performance. This suggests that dynamically updating the diagnostic focus helps expose new optimization opportunities after previously dominant weaknesses have been addressed.

The held-out trajectories in Figure 5b show that baseline performance fluctuates more substantially across rounds, while RuHarness exhibits a more stable upward trend in later stages. This suggests that the improvements acquired during evolution transfer more consistently beyond the held-in tasks. Together, these results indicate that adaptive rubric guidance supports both sustained improvement on held-in tasks and more stable generalization to held-out tasks throughout the evolution process.

6 Conclusion

This paper presents RuHarness, a rubric-harness co-evolution framework for continual agent self-improvement. RuHarness dynamically adapts diagnostic criteria to the agent’s evolving behavioral weaknesses and uses the resulting rubric frontier to guide targeted harness optimization. Extensive experiments across diverse tasks and backbone models show that RuHarness achieves strong gains in both held-in performance and generalization to unseen held-out tasks. Further analyses show that the rubric frontier tracks shifting diagnostic relevance and supports sustained harness improvement over repeated evolution rounds. These results highlight the importance of jointly adapting diagnostic supervision and the agent harness for more sustained and generalizable agent evolution.

Limitations

Although RuHarness is evaluated across multiple tasks and backbone models, all experiments are conducted within a single underlying agent framework. Evaluating the six-coordinate harness abstraction on a broader range of agent architectures would further establish its generality. In addition, due to computational resource constraints, our experiments are conducted on benchmark tasks under a fixed evolution horizon. Examining longer-term and more open-ended adaptation settings would provide a broader view of the resulting co-evolution dynamics.

References

- Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. 2026. [Evoskill: Automated skill discovery for multi-agent systems](#). *Preprint*, arXiv:2603.02766.
- Hao Chen, Ziyu Han, Yukun Yan, Qingfu Zhu, Maosong Sun, and Wanxiang Che. 2026a. [From holistic evaluation to structured criteria: Rubrics across the evolving llm landscape](#). *Preprint*, arXiv:2606.08625.
- Zhengyu Chen, Teng Xiao, Huaisheng Zhu, Yige Yuan, Luan Zhang, and Jingang Wang. 2026b. [Co-harness: Co-evolving harnesses and model weights for llm agents](#). *Preprint*, arXiv:2607.22688.
- Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifmoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. 2025. [Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent](#). *Preprint*, arXiv:2508.06600.
- Hongxin Ding, Baixiang Huang, Yue Fang, Weibin Liao, Zheng Li, Jinyang Zhang, Zhijing Wu, Junfeng Zhao, and Yasha Wang. 2026. [EvoRubrics: Dynamic rubrics as rewards via adversarial co-evolution for llm reinforcement learning](#). *Preprint*, arXiv:2606.23038.
- Shihan Dou, Haoxiang Jia, Shichun Liu, Feng Chen, Chenhao Huang, Yujiong Shen, Shaofan Liu, Jiayi Chen, Jiahang Lin, Honglin Guo, Qianyu He, Minghao Guo, Ziyi Ye, Pluto Zhou, Tao Gui, Qi Zhang, and Xuanjing Huang. 2026. [Agents in the large: Perception-centered architecture for persistent agents](#). *Preprint*, arXiv:2608.30478.
- Tianyu Fan and Chao Huang. 2026. [Helix: Model-harness co-evolution for recursive self-improvement](#). *Preprint*, arXiv:2608.13951.
- Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. 2024. [Omni-math: A universal olympiad level mathematic benchmark for large language models](#). *Preprint*, arXiv:2410.07985.
- Xingze Gao, Chuanrui Hu, Hongda Chen, Pengfei Yao, Zhao Wang, Yi Bai, Zhengwei Wu, Yunyun Han, Xiaofeng Cong, Jie Gui, Yafeng Deng, and Teng Li. 2026. [Evoagentbench: Benchmarking agent self-evolution via ability transfer](#). *Preprint*, arXiv:2607.05202.
- Xin Guan, Xiaomeng Hu, Shen Huang, Zhenyi Wang, Bo Zhang, Zijian Li, Pengjun Xie, Bo Liu, and Jiaxin Cao. 2026. [EvoRubric: Self-evolving rubric-driven rl for open-ended generation](#). *Preprint*, arXiv:2605.29847.
- Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. 2025. [Rubrics as rewards: Reinforcement learning beyond verifiable domains](#). *Preprint*, arXiv:2507.17746.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. [Livecodebench: Holistic and contamination free evaluation of large language models for code](#). *Preprint*, arXiv:2403.07974.
- Haoxiang Jiang, Zihan Dong, Tianci Liu, Wanying Wang, Ran Xu, Tony Yu, Linjun Zhang, and Haoyu Wang. 2026a. [Rubric-arrow: Alternating pointwise rubric reward modeling for llm post-training in non-verifiable domains](#). *Preprint*, arXiv:2605.29156.
- Wen Jiang, Mingmin Chu, Yimeng Tian, Qianxin Zhang, Haofei Yang, Rui Yang, Yang Liu, Tao Lv, and Fangming Li. 2026b. [Harnessesolve: Learning from reference trajectories for reliable agent self-evolution](#). *Preprint*, arXiv:2609.00829.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. [Swe-bench: Can language models resolve real-world github issues?](#) *Preprint*, arXiv:2310.06770.
- Sunghwan Kim, Dongjin Kang, Taeyoon Kwon, Hyunjoo Chae, Dongha Lee, and Jinyoung Yeo. 2025. [Rethinking reward model evaluation through the lens of reward overoptimization](#). *Preprint*, arXiv:2505.12763.
- Hyunin Lee, Jinglue Xu, Jeffrey Seely, Donghyun Lee, Matei Zaharia, and Yujin Tang. 2026a. [Recursive harness self-improvement](#). *Preprint*, arXiv:2607.15524.
- Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. 2026b. [Meta-harness: End-to-end optimization of model harnesses](#). *Preprint*, arXiv:2603.28052.

- Gaotang Li, Bhavana Dalvi Mishra, Zifeng Wang, Jun Yan, Yanfei Chen, Chun-Liang Li, Long T. Le, Rujun Han, George Lee, Hanghang Tong, Chen-Yu Lee, and Tomas Pfister. 2026a. [Rubricem: Meta-rl with rubric-guided policy decomposition beyond verifiable rewards](#). *Preprint*, arXiv:2605.10899.
- Shuyue Stella Li, Rui Xin, Teng Xiao, Yike Wang, Rulin Shao, Zoey Hao, Melanie Sclar, Sewoong Oh, Faeze Brahman, Pang Wei Koh, and Yulia Tsvetkov. 2026b. [Evolm: Self-evolving language models through co-evolved discriminative rubrics](#). *Preprint*, arXiv:2605.03871.
- Xiaoxi Li, Wenxiang Jiao, Jiarui Jin, Guanting Dong, Jiajie Jin, Yinuo Wang, Hao Wang, Yutao Zhu, Jirong Wen, Yuan Lu, and 1 others. 2026c. Deepagent: A general reasoning agent with scalable toolsets. In *Proceedings of the ACM Web Conference 2026*, pages 2219–2230.
- Zhicong Li, Lingjie Jiang, Yulan Hu, Xingchen Zeng, Yixia Li, Xiangwen Zhang, Guanhua Chen, Zheng Pan, Xin Li, and Yong Liu. 2026d. [No more stale feedback: Co-evolving critics for open-world agent learning](#). *Preprint*, arXiv:2601.06794.
- Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. 2026a. [Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses](#). *Preprint*, arXiv:2604.25850.
- Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, Dakuo Wang, Xiang Zhang, Benoit Dumoulin, Cihang Xie, Yuyin Zhou, Suhang Wang, and Hanqing Lu. 2026b. [Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents](#). *Preprint*, arXiv:2605.30621.
- Xiaotian Luo, Dizhan Xue, Fengxingyu Wang, Chuanrui Hu, and Yafeng Deng. 2026. [Harnessbank: Semantic gene-bank search with gated verification for agent-harness self-evolution](#). *Preprint*, arXiv:2607.13683.
- Jun Nie, Yonggang Zhang, Jun Song, Qianshu Cai, Dahai Yu, Yike Guo, Xinmei Tian, and Bo Han. 2026. [Tthe: Test-time harness evolution](#). *Preprint*, arXiv:2607.08124.
- Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. 2026. [Evolving agents in the dark: Retrospective harness optimization via self-preference](#). *Preprint*, arXiv:2606.05922.
- Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubei, Phoebe Thacker, Lorraine Fauconnet, Natalie S. Kim, Patrick Chao, Samuel Miserendino, Gildas Chabot, David Li, Michael Sharman, Alexandra Barr, Amelia Glaese, and Jerry Tworek. 2025. [Gdpval: Evaluating ai model performance on real-world economically valuable tasks](#). *Preprint*, arXiv:2510.04374.
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. [Reflexion: Language agents with verbal reinforcement learning](#). *Preprint*, arXiv:2303.11366.
- Dhruv Srikanth, Bingchen Zhao, Dixing Xu, Yuxiang Wu, and Zhengyao Jiang. 2026. [Recursive self-improvement of ai research agents](#). *Preprint*, arXiv:2609.26457.
- Bo Tang, Yang Zhang, Guomian Zhuang, Wenqiang Wei, Gaoyang Zheng, Lindong Xie, Yanchao Tan, Feiyu Xiong, Qingyu Yang, Edward Chung, and Zhiyu li. 2026. [From memory to skills: Evidence-grounded co-evolution governance for long-horizon llm agents](#). *Preprint*, arXiv:2607.16621.
- 5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, and 152 others. 2025. [Glm-4.5: Agentic, reasoning, and coding \(arc\) foundation models](#). *Preprint*, arXiv:2508.06471.
- Qwen Team. 2026. [Qwen3.5: Towards native multi-modal agents](#).
- Zihang Tian, Jingsen Zhang, Rui Li, Xiaohe Bo, Yuanzi Li, and Xu Chen. 2026. [Arco: Adaptive rubrics with co-evolution for multi-step llm-based agents](#). *Preprint*, arXiv:2606.21262.
- Beining Wang, Weihang Su, Hongtao Tian, Hao Kong, Tao Yang, Ting Yao, Qingyi Pan, Yueyue Wu, Qingyao Ai, Min Zhang, and Yiqun Liu. 2026a. [Co-evolving llm evaluators and policies via dynamic rubric](#). *Preprint*, arXiv:2607.20083.
- Jianze Wang, Kunwang Zheng, Ying Liu, Yu Cao, Qilong Zhang, Jinlong Chen, Hua Yang, and Qianglong Chen. 2026b. [Serp: Self-evolving rubric policy optimization for open-ended test-time reinforcement learning](#). *Preprint*, arXiv:2607.26873.
- Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Noah A. Smith, Pradeep Dasigi, and Teng Xiao. 2026c. [Rethinking the evaluation of harness evolution for agents](#). *Preprint*, arXiv:2607.12227.
- Peng Xia, Rujun Han, Zifeng Wang, Yanfei Chen, Yufan Zhuang, Yoonho Lee, Chengsong Huang, Han Yu, Zhongying CuiZhu, Yifei Ming, Huaxiu Yao, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. 2026. [Rrsi: Regularized recursive self-improvement of agent harnesses](#). *Preprint*, arXiv:2609.24972.

- Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. 2026. [Skilopt: Executive strategy for self-evolving agent skills](#). *Preprint*, arXiv:2605.23904.
- Zhou Yu, Bin Bi, Shiva Kumar Pentiyala, Shubham Mehrotra, Sougata Chaudhuri, Shilpa Bhagavath, Zeyuan Chen, Ran Xu, Phil Mui, James Zhu, and Sitaram Asur. 2026. [Co-evolving harnesses and models: On-policy correction helps weaker models catch up where imitation fails](#). *Preprint*, arXiv:2609.09134.
- Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. 2026a. [Self-harness: Harnesses that improve themselves](#). *Preprint*, arXiv:2606.09498.
- 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. 2026b. [Agentic context engineering: Evolving contexts for self-improving language models](#). *Preprint*, arXiv:2510.04618.
- Wenlong Zhang, Zhengbo Jiao, Chenxu Zhang, Lekang Jiang, SiYuan Ma, Qituan Zhang, Guo Chen, and Linfeng Zhang. 2026c. [Recursive harness self-improvement for frontier reasoning data synthesis](#). *Preprint*, arXiv:2610.03548.
- Xuanliang Zhang, Dingzirui Wang, Keyan Xu, Qingfu Zhu, and Wanxiang Che. 2026d. [Scaling laws for agent harnesses via effective feedback compute](#). *Preprint*, arXiv:2605.29682.
- Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2023. [Memorybank: Enhancing large language models with long-term memory](#). *Preprint*, arXiv:2305.10250.
- Jiayin Zhu, Kelong Mao, Yudong Guo, Dengbo He, Sulong Xu, Simiu Gu, and Yutao Yue. 2026. [Skillcoach: Self-evolving rubrics for evaluating and enhancing agentic skill-use](#). *Preprint*, arXiv:2607.01874.

A Appendix

A.1 Experimental Details

In this subsection, we provide the dataset splits, harness abstraction and editable interfaces, and prompt templates used throughout the experiments.

Dataset Statistics and Settings. We conduct experiments on five agent tasks derived from EvoAgentBench (Gao et al., 2026), covering information retrieval, knowledge work, software engineering, code implementation, and mathematical reasoning. EvoAgentBench reorganizes existing benchmarks into train and test sets to evaluate whether improvements acquired during evolution transfer to unseen tasks. For our experiments, we further split the original training set into evolution and validation subsets, while preserving the original task distribution. The evolution subset is used to drive rubric-harness evolution, whereas the validation subset is used exclusively for candidate selection. We also remove a small number of tasks that cannot be reliably executed in our experimental environment. The resulting dataset statistics are summarized in Table 3.

We report performance on the evolution split as *held-in* performance. The original test split is never accessed during evolution or candidate selection and is reserved exclusively for *held-out* evaluation. All splits remain fixed throughout the experiments and are kept identical across all baseline methods to ensure a fair comparison. Evolution rounds, candidate budgets, and validation-based selection protocols are matched across harness-evolution methods unless otherwise specified.

Harness Abstraction and Editable Interfaces. The editable harness space is organized according to the six functional coordinates introduced in Section 3.3. To instantiate this abstraction in the agent framework, each coordinate is aligned with the concrete framework interfaces that control the corresponding functionality, such as skill and tool configuration, instruction prompts, action logic, observation processing, memory handling, and termination behavior. This alignment determines where each proposed harness modification is applied in the underlying agent system, as summarized in Table 4. All methods start from the same initial harness H_0 for each benchmark and backbone model. The benchmark environment, task specification, tool APIs, and evaluation protocol remain fixed throughout evolution.

Prompt Templates. We provide the key prompt

Domain	Evol. (Held-in)	Val.	Test (Held-out)
Information Retrieval	100	54	65
Knowledge Work	60	36	50
Software Engineering	45	26	36
Code Implementation	128	54	86
Math Reasoning	380	98	100

Table 3: Dataset statistics for the evolution, validation, and test splits across five evaluation domains.

Coordinate	Functional Scope	Editable Content
Equip	Capability provisioning	Skills, tools, and reusable modules
Instruct	Behavioral guidance	System and task instructions
Act	Action execution	Tool-use and action strategies
Observe	Observation processing	Parsing, filtering, and summarization
Remember	Context management	Memory and context handling
Terminate	Completion control	Stopping and finalization

Table 4: Unified harness coordinates, their functional scopes, and editable contents.

templates used in the co-evolution procedure. The trajectory-analysis prompt is shown in Figure 9, the rubric proposal and outcome-blind judging prompts in Figures 10–11, and the harness intervention planning prompt in Figure 12.

A.2 Details of Diagnostic Utility

In this subsection, we provide the detailed definitions of the five criterion-level statistics used to compute the diagnostic utility in Section 3.2.

For each criterion c , the outcome-blind judge assigns every trajectory one of three labels: VIOLATED, SATISFIED, or N/A. Let v_{pass} , s_{pass} , and n_{pass} denote the numbers of violated, satisfied, and non-applicable trajectories among successful tasks. Similarly, let v_{fail} , s_{fail} , and n_{fail} denote the corresponding counts among unsuccessful tasks. Based on these statistics, the five components are defined as follows.

Discrimination (D): Measures how well a criterion distinguishes unsuccessful trajectories from successful ones by comparing its violation rates across the two groups:

$$D(c) = \frac{v_{\text{fail}}}{v_{\text{fail}} + s_{\text{fail}}} - \frac{v_{\text{pass}}}{v_{\text{pass}} + s_{\text{pass}}}. \quad (11)$$

Coverage (C): Measures the proportion of unsuccessful trajectories for which the criterion iden-

tifies a violation:

$$C(c) = \frac{v_{\text{fail}}}{v_{\text{fail}} + s_{\text{fail}} + n_{\text{fail}}}. \quad (12)$$

Predictive Validity (P): Measures whether observing a criterion violation increases the likelihood of task failure relative to the criterion’s applicable population:

$$P(c) = \frac{v_{\text{fail}}}{v_{\text{fail}} + v_{\text{pass}}} - \frac{v_{\text{fail}} + s_{\text{fail}}}{v_{\text{pass}} + s_{\text{pass}} + v_{\text{fail}} + s_{\text{fail}}}. \quad (13)$$

Equivalently,

$$P(c) = \Pr(\text{fail} \mid \text{violated}) - \Pr(\text{fail} \mid \text{applicable}). \quad (14)$$

Novelty (N): Measures the proportion of failed trajectories uniquely identified by criterion c . Let $\mathcal{U}_c$ denote the set of failed trajectories violated only by criterion c , and let N_F denote the total number of failed trajectories:

$$N(c) = \frac{|\mathcal{U}_c|}{N_F}. \quad (15)$$

Redundancy (Red): Measures the behavioral overlap between criterion c and the remaining criteria. Let $\mathcal{V}(c)$ denote the set of trajectories violated by criterion c :

$$\text{Red}(c) = \max_{c' \neq c} \text{Jaccard}(\mathcal{V}(c), \mathcal{V}(c')). \quad (16)$$

A.3 Detail of Rubric Admission and Lifecycle

In this subsection, we provide the detailed admission criteria and lifecycle transition rules used to maintain the evolving rubric pool introduced in Section 3.2.

Birth Check. Before entering the maintained rubric pool, each newly discovered criterion undergoes a birth check designed to remove criteria that are unlikely to provide useful diagnostic signals. We reject a criterion under any of the following conditions: (i) *never applies*, when more than 80% of its assessments are N/A; (ii) *never fires*, when it produces no violations on unsuccessful trajectories; or (iii) *fires on all*, when its violation rate exceeds 90% overall and also exceeds 90% on successful trajectories. These checks remove criteria that are respectively inapplicable, insensitive to actual failures, or insufficiently discriminative. Criteria that pass the birth check enter the lifecycle in the *probation* state. Since lifecycle transitions are evaluated within the same evolution round, a newly admitted criterion may already appear as *active* in the

Current State	Condition	Next State
Probation	$P > 0$	Active
Probation	Untriggered for 4 rounds	Retired
Active	Trigger rate $< 5\%$ for 3 rounds	Dormant
Active	N/A drift for 3 rounds	Dormant
Active	$P < 0.10$ for 3 rounds	Retired
Dormant	Trigger rate $\geq 15\%$	Active
Dormant	Remains dormant for 4 rounds	Retired

Table 5: Transition rules for the rubric lifecycle. Trigger rate is the violation frequency of a criterion, and N/A drift reflects a sustained rise in its not-applicable rate.

round-end visualization if it immediately satisfies the promotion condition.

Lifecycle Transition Mechanism. Each admitted criterion is maintained in one of four states: *probation*, *active*, *dormant*, or *retired*. State transitions are evaluated after each evolution round according to the criterion’s observed diagnostic statistics.

Probation. A newly admitted criterion first enters the *probation* state. It is promoted to *active* once it exhibits positive predictive validity ($P > 0$). If it remains untriggered for four consecutive rounds, it is instead moved directly to *retired*.

Active. An *active* criterion is eligible for frontier construction. It becomes *dormant* when its trigger rate remains below 5% for three consecutive rounds or when the predefined N/A-drift condition persists for three consecutive rounds. It is directly retired when its predictive validity remains below 0.10 for three consecutive rounds.

Dormant. A *dormant* criterion is retained for monitoring but excluded from frontier construction. It is reactivated when its trigger rate rises to at least 15%. If it remains dormant for four consecutive rounds without reactivation, it is moved to *retired*.

Retired. The *retired* state is terminal. Retired criteria no longer participate in criterion prioritization or frontier construction. Table 5 summarizes all lifecycle transition rules.

Visualization of Lifecycle Dynamics. Figure 6 visualizes the lifecycle dynamics on Qwen3.5-35B-A3B for the Knowledge Work domain. Each column shows criterion states at the end of an evolution round, while the flows trace transitions between states. “Not yet born” denotes criteria introduced only in later rounds. For visualization, the *retired* category also includes criteria rejected at birth as bookkeeping, although such criteria do not formally enter the lifecycle.

The figure reveals substantial turnover through-

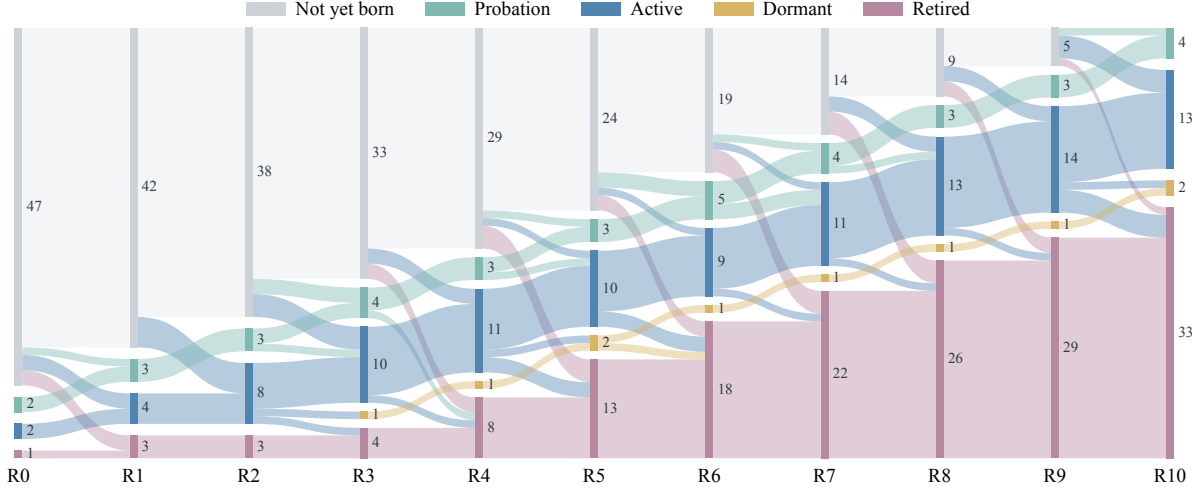


Figure 6: Rubric lifecycle across evolution rounds on the Qwen3.5-35B-A3B model with the knowledge work task. Each column shows criterion states at the end of a round, with flow widths proportional to the number of criteria. “Not yet born” denotes criteria introduced in later rounds; “Retired” is cumulative and includes candidates rejected at birth.

out evolution. New criteria continue to enter the lifecycle and become active as they satisfy the promotion condition, while previously active criteria move to dormant or retired states as their diagnostic relevance decreases. Dormant criteria may also return to the active set when the corresponding failures re-emerge. Consequently, the active criterion set is repeatedly reorganized rather than monotonically accumulated.

This turnover provides a mechanistic explanation for the frontier dynamics observed in the main analysis. As harness updates alter the behavioral distribution, some criteria lose diagnostic relevance while others become newly informative, continually reshaping the active set available for frontier construction.

A.4 Details of Posterior Estimation and Conservative Ranking

In this subsection, we provide the details of the conservative ranking strategy introduced in Section 3.2. Directly ranking criteria by their point-estimated utility can be unreliable when only a small number of trajectories support a criterion, since a few observations may produce an artificially high score. We therefore rank criteria using a lower posterior estimate of their diagnostic utility.

For each criterion, we construct a 2×3 contingency table over task outcome (*pass* or *fail*) and criterion assessment (VIOLATED, SATISFIED, or N/A). Since discrimination, coverage, and predictive validity are all computed from this same table,

we estimate them jointly. We place a symmetric Jeffreys prior over the six cells, corresponding to adding 0.5 to each observed count, and obtain a Dirichlet posterior over the joint outcome distribution.

We draw $B = 2000$ samples from this posterior. For each sample, we recompute discrimination, coverage, and predictive validity, while keeping novelty and redundancy fixed because they are computed from trajectory-level set statistics. This produces a posterior distribution over the diagnostic utility. We then use its lower q -quantile as the conservative score:

$$\tilde{U}(c) = \text{Quantile}_q \left(\left\{ U^{(b)}(c) \right\}_{b=1}^B \right), \quad (17)$$

where $q = 0.2$ in all experiments.

This lower-quantile ranking favors criteria whose diagnostic value remains high under uncertainty, while reducing the influence of criteria supported by only a few observations. The resulting scores are used to rank active criteria, from which the Top- K criteria are selected to form the rubric frontier.

A.5 Robustness to Random Seeds

To assess the robustness of different evolution methods to stochasticity, we repeat all methods with three independent random seeds on two representative settings: Qwen3.5-35B-A3B on Knowledge Work and GLM-4.7-Flash on Software Engineering. Dataset splits, initial harnesses, and all hyperparameters are kept fixed across runs. Table 6

Method	Held-in	Held-out
<i>Qwen3.5-35B-A3B on Knowledge Work</i>		
Vanilla Agent	51.1 $\pm$ 1.0	38.0 $\pm$ 2.0
ACE	55.6 $\pm$ 1.0	42.0 $\pm$ 2.0
EvoSkill	54.4 $\pm$ 1.0	41.3 $\pm$ 1.2
Self-Harness	52.2 $\pm$ 1.9	41.3 $\pm$ 3.1
AHE	55.0 $\pm$ 1.7	47.3 $\pm$ 3.1
Meta-Harness	57.8 $\pm$ 2.5	38.0 $\pm$ 2.0
RuHarness	58.9 $\pm$ 1.0	50.0 $\pm$ 2.0
<i>GLM-4.7-Flash on Software Engineering</i>		
Vanilla Agent	31.1 $\pm$ 2.2	31.5 $\pm$ 3.2
ACE	37.8 $\pm$ 0.0	45.3 $\pm$ 1.6
EvoSkill	42.2 $\pm$ 2.2	39.8 $\pm$ 1.6
Self-Harness	45.9 $\pm$ 1.3	45.4 $\pm$ 3.2
AHE	42.9 $\pm$ 1.3	39.8 $\pm$ 1.6
Meta-Harness	40.0 $\pm$ 0.0	38.0 $\pm$ 1.6
RuHarness	48.9 $\pm$ 2.2	51.9 $\pm$ 1.6

Table 6: Robustness to random seeds on two representative settings. Results are reported as mean $\pm$ standard deviation over three independent runs.

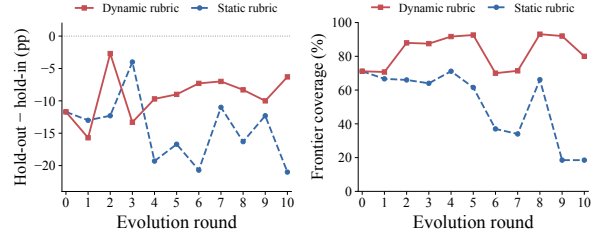
reports the mean and standard deviation over the three runs.

RuHarness maintains the best average performance across both held-in and held-out evaluations under three independent random seeds. Importantly, the gains remain consistent across different random seeds, while the observed variance is comparable to that of competing methods. This trend is particularly clear on held-out tasks, where RuHarness maintains a clear advantage under both backbone–task combinations. These results indicate that the observed gains are robust to stochasticity in the evolution process.

A.6 Effectiveness of Dynamic Evolution

In this subsection, we further analyze how dynamic rubric evolution affects generalization during harness optimization. We compare the full RuHarness framework with a static-rubric variant in which the rubric pool is fixed after initialization, while the remaining evolution procedure is kept unchanged. Figure 7 reports the round-wise generalization gap and frontier coverage on the Qwen3.5-35B-A3B Knowledge Work setting.

Dynamic rubrics maintain a smaller generalization gap. We define the generalization gap as held-out performance minus held-in performance at each evolution round. As shown in Figure 7a, although both variants can improve held-in performance during evolution, the static-rubric variant exhibits substantially larger fluctuations and increasingly negative gaps in later rounds. In con-



(a) Generalization gap across evolution rounds. (b) Frontier coverage across evolution rounds.

Figure 7: Effectiveness of dynamic rubric evolution on the Qwen3.5-35B-A3B model with the knowledge work task. Compared with a static rubric, the dynamic rubric maintains a smaller held-out–held-in generalization gap and substantially higher frontier coverage over the course of evolution.

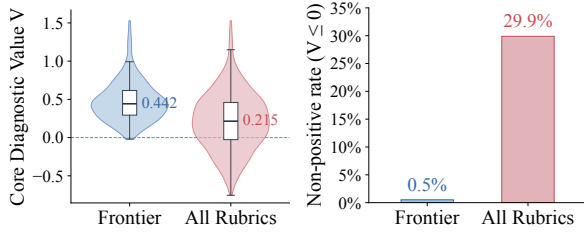
trast, the dynamic-rubric variant remains relatively stable across evolution and ends with a markedly smaller discrepancy between held-in and held-out performance. This suggests that harness optimization guided by an evolving rubric transfers more effectively to unseen tasks.

Dynamic rubrics preserve coverage of current failures. Figure 7b further shows a clear divergence in frontier coverage, defined as the proportion of failed trajectories captured by at least one criterion in the currently prioritized rubric frontier. As evolution proceeds, the dynamic rubric maintains high coverage across most rounds, whereas the static rubric progressively loses coverage, particularly in later stages. This indicates that a fixed rubric becomes increasingly misaligned with the weaknesses exposed by the evolving harness, while continual rubric updates allow the frontier to track newly emerging failure patterns.

Taken together, these results provide additional evidence for the role of rubric evolution in continual harness optimization. A static rubric can still support improvements on the evolution tasks, but its diagnostic relevance gradually weakens as agent behavior changes. By continually updating and reprioritizing behavioral criteria, RuHarness preserves coverage of unresolved failures and consequently achieves more stable generalization throughout evolution.

A.7 Effectiveness of Rubric Frontier

To understand why harness evolution benefits from a prioritized rubric frontier rather than the entire rubric pool, we compare the diagnostic composition of the frontier with that of all eligible rubrics. Figure 8 reports the distribution of core diagnos-



(a) Distribution of core diagnostic value. (b) Rate of non-positive diagnostic value.

Figure 8: Effectiveness of rubric frontier prioritization. Compared with the complete eligible rubric pool, the frontier concentrates criteria with substantially higher core diagnostic value and contains far fewer criteria with non-positive diagnostic value.

tic value and the proportion of criteria with non-positive diagnostic value. Here, the core diagnostic value is defined as the conservative posterior estimate:

$$V(c) = Q_{0.2}[D(c) + 0.5C(c) + P(c)], \quad (18)$$

which captures the principal diagnostic signals used for rubric prioritization.

The frontier concentrates diagnostically valuable criteria. As shown in Figure 8a, criteria in the frontier exhibit a substantially higher diagnostic-value distribution than the complete rubric pool. The median value increases from 0.215 for all eligible rubrics to 0.442 for the frontier. Since the full pool contains the frontier itself, this shift indicates that prioritization substantially increases the concentration of informative criteria presented to harness evolution, rather than simply reducing the number of criteria.

Using all rubrics introduces substantial low-value guidance. Figure 8b further shows that only 0.5% of frontier criteria have non-positive diagnostic value, compared with 29.9% when considering the complete rubric pool. Thus, providing all rubrics to the proposer introduces a considerable fraction of weak diagnostic signals alongside the informative ones. This helps explain the degradation observed when frontier selection is removed: the frontier serves as a focused diagnostic filter that preserves high-value guidance while avoiding dilution by criteria that provide limited information about the agent’s current weaknesses.

A.8 Case Study of Rubric–Harness Co-Evolution

To examine how rubric evolution progressively reshapes harness optimization, we trace two repre-

sentative evolution trajectories in Table 7. Rather than repeatedly targeting the same failure pattern, the diagnostic focus becomes increasingly specific across rounds, and the corresponding harness updates evolve from coarse checks to more operational, explicit, and structured interventions.

Knowledge Work. The first trajectory exhibits a clear progression in verification requirements. It begins with a basic question of *whether generated scripts are actually executed*, leading to an OBSERVE-level rule that requires execution and error checking. The next stage moves beyond detecting failure to asking *why the failure occurred*, prompting explicit analysis of returned error messages. The focus then shifts from execution correctness to *deliverable correctness*: the agent must verify that the expected output artifact exists and matches the required filename. This requirement is subsequently strengthened from a verbal confirmation into *programmatically verification*, making the check directly executable. In the final stage, verification is moved earlier in the workflow, from post-hoc confirmation to *pre-access checking* before downstream file use. The trajectory therefore progresses from execution, to diagnosis, to artifact verification, to executable verification, and finally to proactive verification.

Information Retrieval. The second trajectory shows a parallel progression from coarse evidence checking to structured auditing. It starts by requiring *full-document retrieval* instead of relying on search snippets, establishing a stronger evidence basis for candidate matching. The next refinement moves this requirement closer to the retrieval step itself, preventing premature judgments immediately after observing a snippet. The focus then advances from merely verifying a candidate to making the verification *explicit and auditable*, requiring the agent to state which constraints are satisfied and provide supporting evidence. Finally, the process becomes *systematic*: individual evidence statements are organized into a constraint-satisfaction matrix so that all constraints are checked collectively rather than independently. Thus, the diagnostic trajectory evolves from source verification, to immediate verification, to explicit evidence grounding, and finally to structured constraint coverage.

These trajectories illustrate that rubric evolution progressively raises the granularity of behavioral supervision. As earlier requirements are incorporated into the harness, later rubrics shift attention to finer-grained weaknesses, which in turn

Round	Observed Pattern	Rubric Diagnosis	Target Coordinate	Harness Response
Case I: Knowledge Work — From Generic Checking to Executable Verification				
R1	<i>Execution</i>	Generated scripts should not be treated as valid merely because they were written; successful execution must be verified.	OBSERVE	Require every generated script to be executed and explicitly checked for errors before subsequent actions.
R3	<i>Error Analysis</i>	Detecting a tool failure is insufficient if the underlying cause is not examined.	OBSERVE	Require the agent to inspect the error output and identify its cause before proceeding with unrelated commands.
R4	<i>Deliverable Check</i>	Successful execution does not guarantee that the expected deliverable has actually been produced.	TERMINATE +EQUIP	Add a completion-stage check that confirms the output file exists and matches the required filename.
R5	<i>Programmatic Check</i>	Declarative confirmation remains underspecified for reliable verification.	TERMINATE	Strengthen the check by requiring programmatic verification of the output filename, e.g., through <code>os.path.basename</code> .
R8	<i>Pre-access Check</i>	File validity should be established before the file is consumed downstream.	REMEMBER	Move verification earlier by requiring file existence to be checked before reading reference files.
Case II: Information Retrieval — From Candidate Retrieval to Auditable Verification				
R1	<i>Full-text Check</i>	Search snippets provide only partial evidence and should not be sufficient to establish a candidate match.	REMEMBER +TERMINATE	Require the full document to be retrieved before claiming that a candidate satisfies the target constraints.
R5	<i>Early Verification</i>	The agent may form a premature match judgment immediately after seeing a search result.	OBSERVE	Explicitly mark search results as snippets and require full-text confirmation before accepting a match.
R7	<i>Explicit Evidence</i>	Verification remains difficult to audit when the response does not expose how individual constraints are satisfied.	OBSERVE +EQUIP +TERMINATE	Require the agent to state satisfied constraints and attach supporting evidence for each claim.
R9	<i>Structured Audit</i>	Individual evidence statements may still leave overall constraint coverage unclear.	TERMINATE +REMEMBER	Require a structured constraint-satisfaction matrix and continue retrieval until all constraints are verified.

Table 7: Case studies of rubric–harness co-evolution with Qwen3.5-35B-A3B. Across evolution rounds, diagnostic focus progressively shifts from coarse behavioral checks to more specific and operational requirements, which are translated into targeted modifications of the corresponding harness coordinates.

induce more precise and structured modifications across the relevant harness coordinates. This creates a staged refinement process in which diagnosis and intervention become progressively more operational as evolution proceeds.

Failure Trajectory Analysis

You are analyzing ONE agent trajectory to produce a factual failure report. Another system will read many of these reports together and design evaluation criteria from them. Your job is to record WHAT HAPPENED, precisely and verifiably -- not to judge the agent, and not to state principles.

VERIFIER RESULT (authoritative):
outcome: {outcome}
exception: {exception_type}
failed tests: {failed_tests}
verifier output (tail): {verifier_tail}
steps: {n_steps}
wall clock: {wall_clock_min} min
repeated-action ratio: {repetition_ratio}

TRAJECTORY:
{trajectory_text}

Produce ONE JSON object:

```
{
  "goal": "<what the task required, one sentence>",
  "final_state": "<where the run actually ended, observable facts only>",
  "divergence_step": <int>,
  "divergence_evidence": "<VERBATIM excerpt from that step: the actual action, command, and/or error text. Copy it; do not paraphrase.>",
  "failure_mode": "<exactly one of: {failure_modes}>",
  "agent_attributable": <true|false>,
  "attribution_note": "<one sentence>",
  "missed_opportunity": "<the specific alternative action available AT divergence_step -- name the action, tool, file, or operation the agent could have used instead. If none, write 'none'.>"
}
```

Rules:

- Every quote must appear verbatim above. Do not invent output.
- divergence_step is where progress STOPPED, usually much earlier than where the run ended.
- agent_attributable is false ONLY when no available action could have changed the result -- infrastructure broke, the agent harness malfunctioned, or the model API failed.
- An environment obstacle that the agent responded to badly is still attributable (true).
- missed_opportunity must be concrete and executable.
- "Should have been more careful", "should have verified the result", and "should have planned better" are rejected -- if that is all you can say, write "none".

Output ONLY the JSON object.

Figure 9: Prompt template for failure trajectory analysis.

Rubric Proposal

You are designing evaluation criteria for an agent rubric.

A criterion is one sentence describing an OBSERVABLE agent behavior. A judge will later read a trajectory and answer satisfied / violated / not_applicable against it. A criterion is only useful if that judge would reach DIFFERENT answers on failing runs and succeeding runs.

EXISTING RUBRIC POOL -- do not restate these:
{pool_summary}

FAILURE REPORTS ({n_fail} runs):
{failure_reports}

SUCCESS REPORTS ({n_pass} runs):
{success_reports}

BLIND SPOTS -- no existing criterion fired on these failures. Prioritize them:
{uncovered_list}

TOO BROAD -- these existing criteria fired on nearly every run this round, so they cannot separate good runs from bad ones. Where one names a real concern, write a sharper criterion that captures the same concern in a checkable way; it will replace the vague version. Where it names nothing real, ignore it:
{overfiring_list}

Propose {n} new criteria. Each must:

1. DISCRIMINATE -- describe behavior visible in the SUCCESS reports and rare or absent in the FAILURE reports. Derive it from the failures: read what went wrong, then write the action that would have gone right. Before you commit to a criterion, check it against the success reports: if a judge would mark it VIOLATED there too, it carries no information. Discard it and write a different one.

2. BE AFFIRMATIVE -- state what the agent DOES, never what it does not do. "The agent X", or "Before Y, the agent X". Never "the agent does not X", "the agent avoids X", or "the agent never X". A judge rules "satisfied" when the agent did the right thing, so the sentence has to name the right thing. If the failure you are describing is an action the agent should not take, write the action it should take instead.

A contrastive clause is fine and often sharper -- "targets a different mechanism RATHER THAN another variant of it" names a positive action and contrasts it. What is forbidden is a negated MAIN clause.

3. BE OBSERVABLE -- an action taken, a tool call issued, a check performed or skipped, or a response to an error. Not an internal state like "understands" or "is careful".

4. BE FALSIFIABLE -- a run must be able to clearly satisfy it. A criterion that nearly every run violates is worthless.

5. GENERALIZE WITHOUT GOING VAGUE -- no task-specific nouns. General does NOT mean abstract: prefer a concrete behavioral rule over a virtue.

6. NOT DUPLICATE the pool, or each other.

REJECTED -- unfalsifiable, fires on nearly every run:

"The agent validates its solution against task requirements before completion."

"The agent adapts its strategy when previous actions fail."

ACCEPTED -- same generality, but observable and discriminating:

"After the same action fails twice with the same error, the agent's next action targets a different mechanism rather than another variant of it."

"Before declaring completion, the agent executes or inspects the artifact it produced and verifies the resulting output."

REJECTED -- prohibitions. A judge can rule these "satisfied" for a run that did nothing at all, and they tell a reader modifying the harness what to remove rather than what to install:

"The agent does not issue more than three consecutive search actions without retrieving a document."

"The agent does not output a final answer while a constraint is unverified."

ACCEPTED -- the same two concerns, stated as the behavior that should happen:

"Within three search actions of starting a line of inquiry, the agent retrieves the full content of one of the results."

"Before outputting a final answer, the agent states, for each constraint in the goal, the retrieved evidence that satisfies it."

Output ONLY a JSON array of {n} strings. No markdown, no commentary:

```
[  
  "First criterion sentence.",  
  "Second criterion sentence.",  
  ...  
]
```

Figure 10: Prompt template for rubric proposal.

Outcome-Blind Rubric Judge

You are evaluating whether the following criterion holds for an agent trajectory.

CRITERION:
{nl_description}

TRAJECTORY (task {task_id}):
{trajectory_text}

Judge the behavior described by the criterion on its own terms. Do not try to infer whether the run ultimately succeeded, and do not let a guess about that influence the verdict.

Verdicts:
"satisfied" - the trajectory shows the criterion being met
"violated" - the trajectory shows the criterion being broken
"not_applicable" - the situation the criterion describes never arose

Output ONLY a single JSON object with these fields:

```
{
  "verdict": "satisfied" | "violated" | "not_applicable",
  "reason": "<one-sentence justification>",
  "step_index": <0-based int of the offending step, or null if not violated>,
  "quoted_action": "<VERBATIM excerpt from that step, copied not paraphrased, or null if not violated>"
}
```

Do not include any other text - no markdown, no commentary. Just the JSON.

Figure 11: Prompt template for outcome-blind rubric judging.

Harness Intervention Planning

You are designing interventions that make an agent satisfy rubric criteria it currently violates.

You plan at the level of the WHOLE harness, not one component at a time. A plan is one coordinated change: a theory of change, the criteria it targets, the trajectory evidence it was inferred from, and the components it edits - each with a distinct role. A plan that edits one component is a complete answer; a plan that edits three is only worth it when the evidence says the three have to move together.

THIS ROUND'S BEHAVIORAL FRONTIER - {n_criteria} rubric criteria that this round's runs still violate AND that best separate passing runs from failing ones. A judge rules each run "satisfied" or "violated" against each sentence; runs it rules "satisfied" pass substantially more often than runs it rules "violated". They are ordered by evidential value, NOT by how often they fire, so the first is not necessarily the most common or most severe problem:

{criteria_list}

{evidence_section}

MODIFIABLE COMPONENTS - these are the only things you can change:

{components_list}

Produce up to {n_candidates} plans. Each plan must:

- state a falsifiable hypothesis: what you believe is going wrong, and why these edits should stop it. "Improves quality" is not a hypothesis.
- cite the evidence it rests on, as concrete (task_id, step indices) pairs drawn from the runs above. Cite the steps that actually show the problem.
- name the criteria it targets, by id, from the frontier above.
- assign ONE ROLE PER COMPONENT, and never the same component twice in one plan. Roles must not duplicate or contradict each other.
- use at most {max_components} components. Fewer is usually right: every extra component is

another way for the bundle to fail to load and another variable that cannot be attributed afterwards.

- state what observable behavior should change if the hypothesis is right, so a reader can tell later whether it was.

Plans must differ from each other in evidence, mechanism, target criteria, or component set. If you cannot construct {n_candidates} genuinely distinct plans, return fewer. A duplicate plan costs a full evaluation sweep and returns noise; an empty slot costs nothing.

Note on self-created obstacles: a common failure is a change that makes the agent's own job harder - for example, injecting control text into a query that only an external retrieval system sees, or adding a rule that blocks the very calls the agent needs. Before proposing an edit, ask what the agent loses by it.

Output ONLY a single JSON object:

```
{
  "plans": [
    {
      "hypothesis": "<falsifiable claim about the failing mechanism>",
      "target_criterion_ids": ["<criterion id from the frontier>", ...],
      "evidence": [
        {
          "task_id": "<task id exactly as shown above>",
          "step_indices": [<int>, ...],
          "explanation": "<what these steps show>"
        }
      ],
      "expected_behavior": "<what should observably change>",
      "component_roles": [
        {
          "component": "<component name>",
          "role": "<what this component does in the plan>"
        }
      ],
      "risks": "<how this plan could backfire>"
    }
  ]
}
```

Do not include any other text - no markdown, no commentary. Just the JSON.

Figure 12: Prompt template for harness intervention planning.