Verify Smarter, Evolve Further: Efficient Harness Evolution through Behavior-Aware Verification

Jinghan Xu[1,2], Yikai Zhang[2,3], Aili Chen[2,3], Weiyuan Li[1,2], Jiaqing Liang[1,2], Deqing Yang[1,2]*
[1]School of Data Science, Fudan University,
[2]Shanghai Key Laboratory of Data Science,
[3]College of Computer Science and Artificial Intelligence, Fudan University
jhxu25@m.fudan.edu.cn, yangdeqing@fudan.edu.cn

Abstract

Agent harnesses shape how language-model agents use instructions, tools, and runtime components, but adapting these harnesses requires costly verification. Existing propose-and-verify methods typically score every candidate on a fixed task set, wasting rollouts on unrelated behaviors and allowing aggregate scores to obscure specific regressions. We introduce HARNESSLENS, a budget-aware framework for automated harness evolution. HARNESSLENS jointly explores the task space and user-configurable components, derives candidate modifications from execution trajectories, and selectively verifies each candidate on behavior-relevant tasks using an attributable-evidence gate. Across three agent harnesses and four benchmarks, HARNESSLENS improves average held-out performance by 7.6–13.6% while consuming substantially less evaluation budget than competing baselines. These results demonstrate that behavior-aware verification with explicit attribution enables more reliable and sample-efficient harness evolution under constrained interaction budgets. Our code is available at https://github.com/jhxu5214/HarnessLens.

1 Introduction

An agent harness determines how a language-model agent perceives tasks, uses tools, and acts in an environment through diverse components such as instructions, skills, tool descriptions, memory, permissions, and agent roles (Lin et al., 2026b; Chen et al., 2026; Ning et al., 2026). It substantially shapes how large language models (LLMs) execute tasks and significantly affects their performance (Lee et al., 2026b; Lin et al., 2026b). However, building an effective harness is usually non-trivial and requires substantial manual effort, while a configuration developed for one model or environment may not be transferable. This raises a critical question: **can an agent harness autonomously and continually evolve from the interaction evidence generated during task execution?**

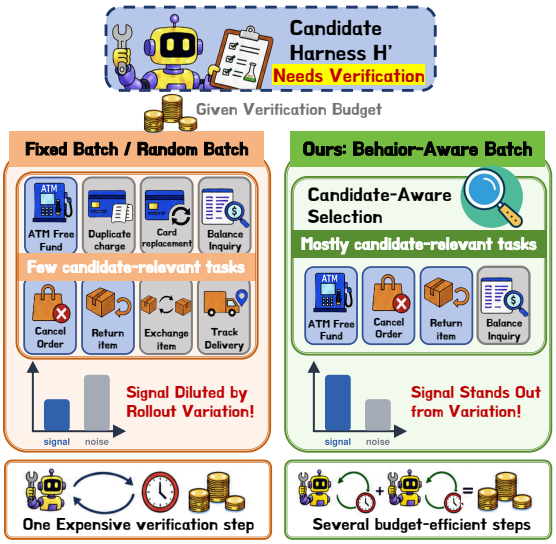

Figure 1: **Comparison of verification paradigms:** Fixed or random batches dilute candidate-specific signals, while our behavior-aware batches enable clearer and more efficient verification.

To solve this problem, existing methods generally follow a *propose-and-verify* paradigm: they propose candidate modifications based on previous interactions and verify them through additional task rollouts. This paradigm is commonly applied to prompt and program optimization (Yang et al., 2024; Opsahl-Ong et al., 2024; Yuksekgonul et al., 2024) and is increasingly applied to richer harness components (Lou et al., 2026; Hao et al., 2026; Lee et al., 2026b; Zhang et al., 2026; Lin et al., 2026b; Chen et al., 2026; Huang et al., 2026). Across these methods, verification typically relies on fixed task sets (Lin et al., 2026b; Chen et al., 2026), randomly sampled minibatches (Agrawal et al., 2026), or task subsets preselected before evolution (Pan et al., 2026), with some methods further adopting staged screening (Luo et al., 2026).

*Corresponding author.

However, these methods suffer from a fundamental limitation: they use the same evaluation tasks for different modifications, even when these modifications target different behaviors. As illustrated in Figure 1, many tasks in such sets may be unrelated to the intended effect of a modification, causing unnecessary rollouts and diluting the modification signal in aggregate metrics. Moreover, verification results may fail to reveal whether the intended behavior has actually emerged when the selected tasks do not cover the affected behaviors. Although recent methods improve trajectory diagnosis (Lin et al., 2026b; Chen et al., 2026) or reduce verification cost through task sampling (Agrawal et al., 2026), preselected task coresets (Pan et al., 2026), or staged screening (Luo et al., 2026), a key limitation persists: verification tasks are still not adapted to the behavioral effect of each modification, and evaluation evidence is not systematically reused to guide subsequent evolution.

To address this gap, we introduce HARNESSLENS, a budget-aware framework for autonomous harness evolution with behavior-aware verification. It selects verification tasks based on supporting trajectories, task patterns, affected components, and regression risks, and compares candidate trajectories to attribute behavioral changes and preserve existing capabilities. HARNESSLENS contains three stages: **Context Exploration** characterizes tasks and user-configurable components, **Trajectory Diagnosis** extracts reusable experiences and deficiencies from rollouts, and **Harness Evolution** constructs and verifies modifications using the collected evidence. Each rollout therefore evaluates current candidates while contributing evidence for future evolution.

We evaluate HARNESSLENS on three agent harnesses across four benchmarks: τ^3-bench Banking Knowledge (Shi et al., 2026), τ^2-bench Retail (Barres et al., 2025), Terminal-Bench 2.0 (Merrill et al., 2026), and the Challenging subset of BIRD Mini-Dev (Li et al., 2024). Under the given budgets, HARNESSLENS improves initial harnesses and outperforms fixed-set verification methods, yielding average performance improvements of 7.6–13.6% across harnesses with a substantially smaller joint rollout-and-analysis budget.

Our contributions are as follows:

- We propose a behavior-aware approach to harness self-evolution that selects verification tasks and allocates rollouts for each modification, instead of evaluating all modifications on the same task set.
- We introduce HARNESSLENS, a budget-aware and harness-independent framework that discovers user-configurable components and their update mechanisms at runtime, diagnoses interaction trajectories, and iteratively proposes, verifies, and reviews harness modifications.
- We evaluate HARNESSLENS across three agent harnesses and four benchmarks. It achieves better test performance than existing methods with substantially fewer rollouts, improving success rates by up to 13.6% on OpenCode, 7.6% on Codex, and 9.2% on Pi.

2 Related Work

Self-improving agents and harness evolution. Self-improving agents use interaction feedback to revise the scaffolds that shape their behavior. Earlier methods operate on relatively lightweight structures, improving prompts (Agrawal et al., 2026), workflow graphs (Zhang et al., 2025), prompt-level harness specifications (Lee et al., 2026a), or domain scaffolds (Hao et al., 2026). Harness evolution expands this scope: some methods synthesize or rewrite harness programs (Lou et al., 2026; Lee et al., 2026b), while others update user-configurable components such as prompts, tools, skills, memory, middleware, and orchestration (Lin et al., 2026b; Zhang et al., 2026; Huang et al., 2026), or accumulate trajectory-derived procedural state (Du et al., 2026). Harness evolution has also been studied for nonstationary task streams (Liu et al., 2026), long-running environments (Karten et al., 2026), and test-time adaptation without labels (Nie et al., 2026). However, candidate-based approaches typically use fixed or shared verification tasks across modifications, wasting rollouts on unaffected behaviors and diluting modification-specific evidence. HARNESSLENS instead selects informative verification tasks for each modification.

Harness verification and rollout allocation. Existing methods use verification rollouts to decide whether to accept candidate modifications. Most evaluate all candidates on a shared, predefined task collection, including fixed or held-out sets (Lee et al., 2026b; Zhang et al., 2026; Huang et al., 2026)

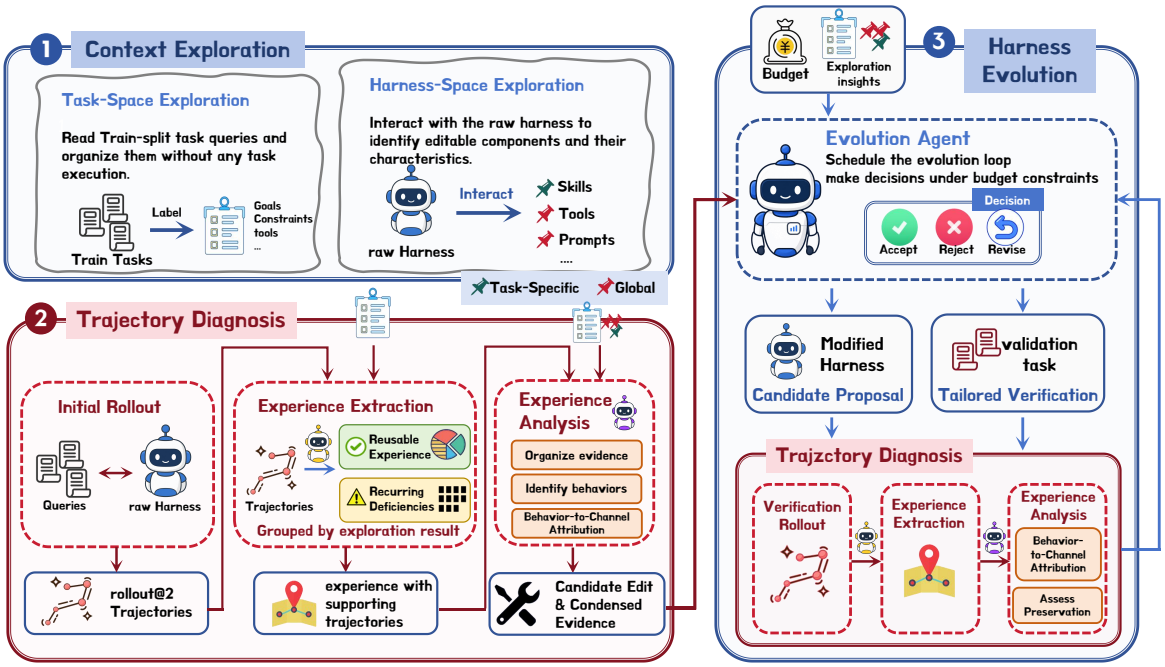

Figure 2: Overview of HARNESSLENS. Context Exploration characterizes available tasks and identifies user-configurable components. Trajectory Diagnosis extracts and analyzes rollout evidence and is reused within Harness Evolution, which constructs and verifies modifications. The entire process is constrained by a fixed interaction budget.

and current evaluation batches (Nie et al., 2026). Other approaches narrow verification to predefined flaw- or failure-related tasks (Chen et al., 2026; Cai et al., 2026) or reduce cost using a globally preselected coreset (Pan et al., 2026). In contrast, HARNESSLENS adapts verification tasks and rollout allocation to each modification's intended behavior, affected components, regression risks, and accumulated evidence, reducing irrelevant rollouts while preserving modification-specific signals.

3 Problem Formulation

We first define the optimization scope of harness evolution, including the user-configurable components of a harness framework, the resulting harness space, and the modifications used to generate candidate harnesses. We then formulate harness evolution as a budget-constrained optimization problem.

3.1 Definitions

Definition 1 (Components and Harness). A harness framework $\mathcal{F}$ exposes a set of user-configurable components,

$$\mathcal{C}_{\mathcal{F}} = \{c_1, \ldots, c_m\}.$$

These components may include instructions, skills, prompt templates, tools and integrations, agent roles, and runtime extensions exposed through predefined interfaces. Each component $c \in \mathcal{C}_{\mathcal{F}}$ has a configuration space $\mathcal{X}_c$, whose elements specify the content and instances configured for that component. A concrete harness is represented as

$$\mathcal{H} = (\mathcal{F}, \mathbf{h}), \quad \mathbf{h} = (h_c)_{c \in \mathcal{C}_{\mathcal{F}}}, \quad h_c \in \mathcal{X}_c.$$

For a fixed framework $\mathcal{F}$, we define the admissible harness space as

$$\mathbb{H}_{\mathcal{F}} \subseteq \{\mathcal{F}\} \times \prod_{c \in \mathcal{C}_{\mathcal{F}}} \mathcal{X}_c.$$

Harnesses in this space may differ in the content and instances configured through the user-configurable components, while sharing the same component types, execution mechanisms, and underlying framework.

Definition 2 (Agent). An agent is a base language model operating within a concrete harness:

$$\mathcal{A}_{\theta,\mathcal{H}} = (M_\theta, \mathcal{H}),$$

where M_θ is a base language model parameterized by θ and $\mathcal{H} \in \mathbb{H}_{\mathcal{F}}$.

Definition 3 (Modification and Candidate). Under a fixed framework $\mathcal{F}$, an admissible harness

modification operates on the joint component configuration:

$$\delta : \prod_{c \in \mathcal{C}_\mathcal{F}} \mathcal{X}_c \to \prod_{c \in \mathcal{C}_\mathcal{F}} \mathcal{X}_c.$$

Given a harness $\mathcal{H} = (\mathcal{F}, \mathbf{h})$, applying δ produces a candidate harness

$$\mathcal{H}' = \big(\mathcal{F}, \delta(\mathbf{h})\big) \in \mathbb{H}_\mathcal{F}.$$

Thus, the modification changes the configuration of one or more user-configurable components while leaving $\mathcal{F}$ unchanged.

Throughout this work, we consider harness evolution under a fixed base model and a fixed harness framework. Thus, θ and $\mathcal{F}$ remain unchanged, while evolution operates only over the admissible component configurations in $\mathbb{H}_\mathcal{F}$.

3.2 Optimization Objective

For a task $x \sim \mathcal{D}$, the agent $\mathcal{A}_{\theta,\mathcal{H}}$ induces a trajectory

$$\tau = (o_0, a_0, o_1, a_1, \ldots, o_T), \ \tau \sim p(\tau \mid x; M_\theta, \mathcal{H}),$$

where o_t and a_t denote the observation and action at step t, respectively. Each trajectory receives a binary reward $R(x, \tau) \in \{0, 1\}$, indicating whether the task is successfully completed. We refer to a *behavior* as a recurring, model-visible pattern within such trajectories: which actions the agent takes, in what order, and under which conditions. Behaviors are identified from trajectories rather than from rewards, so trajectories receiving the same reward may still differ in behavior.

Let $\mathcal{T}_{\text{train}}$ and $\mathcal{T}_{\text{test}}$ be disjoint task sets sampled from the target task distribution $\mathcal{D}$. Only $\mathcal{T}_{\text{train}}$ is accessible during harness evolution, whereas $\mathcal{T}_{\text{test}}$ remains held out for final evaluation.

Starting from an initial harness $\mathcal{H}_0 \in \mathbb{H}_\mathcal{F}$, our objective is to find an evolved harness that maximizes the expected task success rate under a fixed interaction budget:

$$\mathcal{H}^* = \arg\max_{\mathcal{H} \in \mathbb{H}_\mathcal{F}} \mathbb{E}_{x \sim \mathcal{D},\, \tau \sim p(\tau \mid x; M_\theta, \mathcal{H})} \left[R(x, \tau) \right],$$
$$\text{s.t.} \quad C(\mathcal{H}_0 \rightsquigarrow \mathcal{H}) \le B,$$

where B denotes the interaction budget and $C(\mathcal{H}_0 \rightsquigarrow \mathcal{H})$ denotes the number of LLM sessions and task trials consumed along the evolution path from $\mathcal{H}_0$ to $\mathcal{H}$, including the initial rollout, Context Exploration, Trajectory Diagnosis, and Harness Evolution. Here, an LLM session refers to one complete multi-turn execution of a model-based role. Since $\mathcal{T}_{\text{train}}$ participates in harness evolution, generalization performance is measured by the pass rate on the held-out set $\mathcal{T}_{\text{test}}$.

4 Method

Under the budgeted objective in Section 3.2, HARNESSLENS evolves the configurable state $\mathbf{h}_0$ within the fixed harness framework $\mathcal{F}$ under interaction budget B. A deterministic controller coordinates the procedure, tracks budget consumption, isolates candidate executions, manages rollout reuse, and enforces update criteria. Further implementation details appear in Appendix A.1, A.3, and A.4.

As illustrated in Figure 2, HARNESSLENS contains three components: **Context Exploration** (Section 4.1) characterizes tasks and editable components; **Trajectory Diagnosis** (Section 4.2) processes trajectories from initial and verification rollouts; and **Harness Evolution** (Section 4.3) uses its outputs to select proposals, construct candidate harnesses, and verify them before updating the current harness.

4.1 Context Exploration

Context Exploration provides the task organization and editable-component information needed for subsequent behavior-aware verification. *Task-Space Exploration* characterizes task organization from available task context, while *Harness-Space Exploration* identifies the user-configurable components of $\mathcal{F}$ and their editable scope. Their outputs ground Trajectory Diagnosis and focus later rollouts on relevant behaviors and regressions.

Task-Space Exploration. To support behavior-aware verification, *Task-Space Exploration* organizes $\mathcal{T}_{\text{train}}$ according to task goals. For each task, the module inspects only the query, environment instructions and policies, and tool and parameter descriptions. It assigns tasks to groups based on their primary user goals and records additional goals when a task spans multiple groups. This grouping provides a compact view of recurring task patterns and overlaps across goals. The resulting task groups guide verification-task selection and regression checks but are not incorporated into $\mathbf{h}$. At this stage, *Task-Space Exploration* executes no tasks and has no access to trajectories, task outcomes, or $\mathcal{T}_{\text{test}}$.

Harness-Space Exploration. To identify the components available for evolution, *Harness-Space Exploration* examines the configuration, documentation, and runtime behavior of $\mathcal{F}$. For each user-configurable component $c \in \mathcal{C}_{\mathcal{F}}$, the module records how it is exposed to the agent, where its effects apply, how it can be updated, and what behaviors may be affected by a change. Only components that can be reliably identified and updated are retained for subsequent evolution. These component descriptions later help connect diagnosed behaviors to candidate modifications and potential regressions.

4.2 Trajectory Diagnosis

Building on *Context Exploration*, *Trajectory Diagnosis* converts trajectories into behavioral evidence through *Experience Extraction* and *Experience Analysis*. It is applied to the *initial rollout* under $\mathcal{H}_0$ and to each *verification rollout* during *Harness Evolution*.

Experience Extraction. Task rewards alone do not reveal which behaviors should be preserved or improved. The Experience module summarizes trajectories under analysis into *reusable experiences* and *recurring deficiencies*. Each extracted item is linked to the trajectories that support it. To organize this evidence, recurring behaviors are grouped across trajectories, while distinct successful strategies are kept separate. The resulting evidence is passed to *Experience Analysis* for further assessment.

Experience Analysis. To derive candidate modifications from this evidence, the Analyzer combines the extracted experiences and deficiencies with the task groups and identified components from *Context Exploration*, producing a set of modification proposals. Each proposal specifies the targeted behavior, the trajectories that support it, and the user-configurable components that can affect that behavior. The Analyzer then checks whether the supporting trajectories provide sufficient evidence for the proposed modification and removes proposals that lack such support. The resulting proposals and their supporting evidence are passed to *Harness Evolution*, while the extracted experiences remain available for later regression checks.

4.3 Harness Evolution

Using the proposals and supporting evidence from *Trajectory Diagnosis*, *Harness Evolution* iteratively improves the current confirmed harness $\mathcal{H}_j = (\mathcal{F}, \mathbf{h}_j)$. Harness Evolution is carried out by an evolution agent, a model-based role that uses the same harness framework $\mathcal{F}$ as the target agent $\mathcal{A}_{\theta,\mathcal{H}}$ but operates independently from it. At each iteration, the evolution agent selects a proposal in *Candidate Proposal*, chooses its verification tasks in *Behavior-Aware Verification*, and decides whether the resulting candidate should replace $\mathcal{H}_j$ in *Harness Review and Update*.

Candidate Proposal. At each iteration, the evolution agent considers one proposal from *Trajectory Diagnosis*. The proposal is selected based on its supporting evidence, previous attempts, potential regressions, and the remaining budget. The corresponding modification is applied to a copy of the current confirmed harness using the update mechanism identified for the affected component. Let δ_j denote this modification. Applying δ_j to $\mathbf{h}_j$ produces the candidate harness

$$\widetilde{\mathcal{H}}_j = \big(\mathcal{F}, \delta_j(\mathbf{h}_j)\big) \in \mathbb{H}_{\mathcal{F}}.$$

Each new candidate is derived from the current confirmed harness $\mathcal{H}_j$, allowing accepted modifications to accumulate across iterations. Before verification, a lightweight runtime check confirms that the modified component has been successfully applied. Candidates that fail this check do not proceed to verification.

Behavior-Aware Verification. A fixed verification batch may contain few tasks relevant to the behavior targeted by a proposal. To focus verification on that behavior, the evolution agent first selects tasks in $\mathcal{T}_{\text{train}}$ linked to the proposal's supporting trajectories and then selects additional tasks with related goals, constraints, or tool requirements. It also includes tasks that can reveal potential regressions, with broader modifications receiving coverage across more task groups. Each verification batch contains at least five distinct tasks, while the controller fixes the number of trials and ensures that the required verification can be completed within the remaining budget.

The selected tasks are evaluated under both $\mathcal{H}_j$ and $\widetilde{\mathcal{H}}_j$ using matched trial conditions. *Trajectory Diagnosis* is then applied to the resulting trajectories to determine whether δ_j improves the targeted behavior and introduces any regressions.

Harness Review and Update. The evolution agent reviews the behavioral evidence and rollout

metrics to determine whether the observed improvement is supported by the trajectories and whether the candidate introduces regressions.

Candidates with supported improvements and no observed regression undergo an additional confirmation on a new batch constructed by the controller. The batch retains at most two tasks that require further confirmation and fills the remaining positions with previously unused tasks from $\mathcal{T}_{\text{train}}$, prioritizing task groups not covered in the initial verification.

Both $\mathcal{H}_j$ and $\widetilde{\mathcal{H}}_j$ are evaluated on this batch to check whether the observed improvement persists and whether the modification affects other tasks. The final decision considers the behavioral evidence together with the primary metric on the confirmation batch; improvement in the primary metric alone is insufficient.

The harness is updated according to

$$\mathcal{H}_{j+1} = \begin{cases} \widetilde{\mathcal{H}}_j, & \text{if the candidate is accepted,} \\ \mathcal{H}_j, & \text{otherwise.} \end{cases}$$

If the diagnosis identifies a specific issue with δ_j that can be addressed, the evolution agent may adjust the proposal and verify the resulting candidate again. New issues identified during verification may also lead to new proposals.

Evolution ends when no sufficiently supported proposal remains or the remaining budget cannot support another verification cycle.

5 Experiments

5.1 Experiment Setup

Benchmarks and Splits. We evaluate on four environments requiring diverse agent behaviors: τ^2-bench Retail (Barres et al., 2025), τ^3-bench Banking Knowledge (Shi et al., 2026), Terminal-Bench 2.0 (Merrill et al., 2026), and the Challenging subset of BIRD Mini-Dev (Li et al., 2024). For each benchmark, we randomly sample 30 tasks for TRAIN; TEST uses the official split where available and otherwise all remaining tasks. The splits are disjoint, and all information from TEST, including queries, trajectories, and evaluation feedback, is inaccessible throughout harness evolution. Appendix B provides details on the data splits and the full benchmark configuration.

Agent Harnesses and Model. We conduct experiments on the following harnesses: OpenCode (Anomaly, 2026) v1.17.13, Codex CLI (OpenAI, 2026) v0.144.4, and Pi Coding Agent (Zechner, 2025) v0.80.10. All LLM agent and evolution roles use `deepseek-v4-flash-preview` (DeepSeek-AI et al., 2026). To ensure fair evaluation, we disable external information retrieval tools (e.g., web search) and enforce permission controls to prevent answer leakage or unauthorized tool usage.

Baselines. We compare the initial, unmodified harness $\mathcal{H}_0$ with three trajectory-based harness-evolution methods: Self-Harness (Zhang et al., 2026), Meta-Harness (Lee et al., 2026b), and HarnessFix (Chen et al., 2026) [1] . All methods start from $\mathcal{H}_0$ and share the same benchmark runtime and verifier. At their configured maxima, Self-Harness, Meta-Harness, and HarnessFix consume 4,800, 660, and 300 TRAIN rollouts, respectively, whereas HARNESSLENS is capped at 200 total units, including both task rollouts and LLM sessions. Appendix B.4 provides the detailed protocols and the calculation of these configured budget maxima.

5.2 Main Results

HARNESSLENS performs best under the smallest budget. HARNESSLENS attains the best or tied-best pass rate in eight of the twelve harness–benchmark pairs in Table 1, while using the smallest configured budget of any evolving method: two-thirds of HarnessFix's budget and one twenty-fourth of Self-Harness's budget (Table 9). The gains over $\mathcal{H}_0$ are smallest on Retail, where $\mathcal{H}_0$ is already at least 75% for every harness, and largest on Banking and BIRD. The budget gap is understated because HARNESSLENS's budget counts both LLM sessions and task rollouts, whereas the baseline figures count rollouts alone (Appendix B.4).

Harness evolution is stable and effective. HARNESSLENS never falls below $\mathcal{H}_0$, and its worst outcome is an exact tie. Such a tie reflects an unchanged harness rather than a small performance difference: when no proposal accumulates attributable improvement without an attributable regression, the review conditions of Section 4.3 are never satisfied and the run returns $\mathcal{H}_0$. Outcomes thus have only two modes: no edit or an effective

[1] We follow each baseline's original evolution-budget configuration. All methods use the same 30 TRAIN tasks within each harness–benchmark setting, while some baselines further partition them internally for validation.

Harness	Method	τ^2-bench Retail	τ^3-bench Banking	Terminal-Bench 2.0	BIRD Mini-Dev (Challenging)	AVG.
OpenCode	$\mathcal{H}_0$	75.00	20.90	33.90	37.50	41.83
	Self-Harness	80.00	10.45	32.20	37.50	40.04
	Meta-Harness	72.50	13.43	33.90	41.67	40.38
	HarnessFix	80.00	22.39	**35.59**	40.28	44.57
	HarnessLens (Ours)	**85.00**	**25.37**	33.90	**45.83**	**47.53**
Codex	$\mathcal{H}_0$	**80.00**	13.43	32.84	37.50	40.94
	Self-Harness	**80.00**	13.43	**37.29**	34.72	41.36
	Meta-Harness	72.50	**16.42**	30.51	37.50	39.23
	HarnessFix	**80.00**	13.43	35.59	37.50	41.63
	HarnessLens (Ours)	**80.00**	13.43	35.59	**47.22**	**44.06**
Pi	$\mathcal{H}_0$	**85.00**	19.40	**37.29**	40.28	45.49
	Self-Harness	**85.00**	20.90	28.81	33.33	42.01
	Meta-Harness	80.00	16.42	33.90	**48.61**	44.73
	HarnessFix	**85.00**	26.87	35.59	44.44	47.98
	HarnessLens (Ours)	**85.00**	**33.33**	**37.29**	43.06	**49.67**

Table 1: Performance of different harness-evolution methods across agent harnesses and target benchmarks. Results are reported as pass@1 (%) on the held-out TEST split. The best result in each harness–benchmark setting is shown in bold.

edit. Together, Self-Harness and Meta-Harness fall below $\mathcal{H}_0$ in half of their 24 harness–benchmark pairs and lose about ten points at worst. Behavior-aware verification therefore acts as a filter rather than a search accelerator: an uninformative iteration results in a tie rather than a regression.

More verification rollouts do not imply better harnesses. Performance does not increase with verification budget: Self-Harness uses the most rollouts but does not achieve the strongest performance, whereas HarnessFix achieves the strongest baseline results with the smallest budget. The key difference is how candidates are accepted. The baselines rely on aggregate pass rates over fixed task splits, where modification-specific gains can be obscured by unrelated tasks and trial noise. This may allow chance improvements and undetected regressions to accumulate across iterations and become apparent only on TEST. More candidates therefore create more opportunities to accept harmful edits rather than reliably improving the harness.

6 Analysis

To understand how HARNESSLENS achieves effective harness evolution under a limited interaction budget, we analyze two aspects of its behavior-aware verification process. First, we isolate the contributions of behavior-aware batch selection and attributable-evidence gating through ablations. Second, we examine how task-space structure affects the ability to identify modifications that receive consistent, attributable support across related tasks. Appendix C.2 complements these analyses with decision-level evidence on accepted modifications and budget allocation.

6.1 Ablation: Selection and Gating

HARNESSLENS improves verification through two complementary mechanisms: it selects tasks relevant to each candidate modification and accepts the modification only when the observed improvement can be attributed to it. We ablate these mechanisms independently to isolate their contributions.

Settings. All variants use $B = 200$ and modify only the verification-stage configuration of HARNESSLENS; all other settings follow the main experiments. To ablate batch selection, *Fixed Batch*, *Random Batch*, and *RHO-based Batch* (Pan et al., 2026) use a fixed subset, a uniformly resampled subset, and RHO-based task selection, respectively, while retaining the attributable-evidence gate. For these variants, we set the verification batch size to 10, or one third of the 30 TRAIN tasks, yielding a verification fraction comparable to the validation fraction used in HarnessFix (Chen et al., 2026). To ablate the acceptance mechanism, *Metric-Only Gate* retains behavior-aware selection but accepts candidates solely according to the aggregate primary metric.

Results. Table 2 shows that removing either mechanism substantially weakens performance. Fixed, random, and RHO-based batches frequently include tasks unrelated to the candidate modification, introducing variation that the diagnostic anal-

Method	τ^2-bench Retail	τ^3-bench Banking	BIRD Mini-Dev (Challenging)
$\mathcal{H}_0$	75.00	20.90	37.50
Fixed Batch	75.00	20.90	37.50
Random Batch	75.00	20.90	37.50
RHO-based Batch	75.00	20.90	38.89
Metric-Only Gate	80.00	20.90	37.50
HarnessLens (Ours)	**85.00**	**25.37**	**45.83**

Table 2: Ablation of behavior-aware batch selection and attributable-evidence gating on OpenCode.

ysis cannot attribute to the targeted behavior. Consequently, these variants often fail to produce candidates that pass verification. In contrast, *Metric-Only Gate* promotes candidates more readily but also accepts aggregate gains unsupported by attributable evidence; these gains yield no clear held-out improvement on Banking or BIRD. Overall, behavior-aware selection improves the relevance of verification evidence, while attributable-evidence gating prevents noisy or unsupported gains from being retained.

6.2 How Does Task Diversity Affect Evolution?

The gains on Banking and BIRD suggest that evolution benefits from recurring, actionable weaknesses: related tasks can reinforce a modification across examples. Retail instead starts from a strong harness with less potential for further improvement; its trajectories mainly serve as checks for regressions, so retaining the original harness is often desirable.

Figure 3 contrasts the evidence available in BIRD and Terminal-Bench 2.0. In BIRD, several tasks involving extreme-value queries expose the same SQL decision, enabling an attributable recovery. Terminal tasks involve distinct goals and execution paths; the tested broad rule recovers no tasks and causes an attributable regression, so the gate retains the original harness. This is not merely a TRAIN–TEST mismatch: diversity within both splits limits modifications that are supported by TRAIN and broadly applicable to TEST.

This also explains why fixed-set optimizers benefit from repetitive objectives that provide consistent signals toward the same direction. SkillBoost (Lin et al., 2026a) similarly finds that optimized skills transfer most naturally between similar tasks. Since our objective is held-out generalization rather than

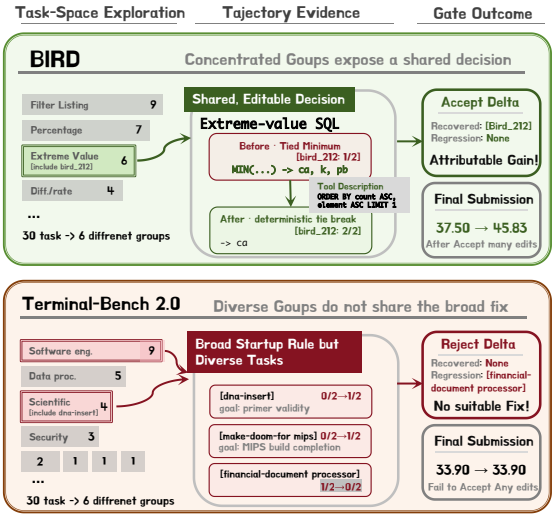

Figure 3: Illustrative contrast between task-space structure and evolution evidence. In BIRD, related tasks expose a shared extreme-value SQL decision, enabling an attributable recovery and an accepted edit. Terminal-Bench 2.0 contains more distinct task goals and execution paths; the tested broad rule yields no recovery and causes a regression, so it is rejected. The BIRD TEST result is cumulative across all accepted edits.

exhaustive TRAIN optimization, high task diversity makes broadly effective modifications harder to identify. Appendix C.1 provides an evolution trace for an OpenCode–BIRD run.

7 Conclusion

We introduce HARNESSLENS, a budget-aware framework that evolves agent harnesses through behavior-aware verification. By selecting tasks related to each modification and accepting changes based on attributable behavioral evidence, HARNESSLENS uses limited rollouts more effectively while avoiding unsupported regressions. Across three harnesses and four benchmarks, it consistently improves or preserves initial performance and achieves average performance improvements of 7.6–13.6%. These results show that reliable harness evolution depends not only on the amount of verification, but also on directing verification toward the behaviors each modification is intended to change.

Limitations

Our evaluation covers one model family, three harnesses, and four public benchmarks. While the results consistently support the effectiveness of behavior-aware verification across these settings, its effectiveness under a broader range of models,

harness architectures, and open-ended deployment environments has not yet been fully validated. In addition, our budget counts LLM sessions and task trials as auditable interaction units, but does not normalize token usage, latency, or monetary cost across roles and benchmarks.

References

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*.

Anomaly. 2026. OpenCode: The open source ai coding agent. https://github.com/anomalyco/opencode. Accessed: 2026-07-26.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-bench: Evaluating conversational agents in a dual-control environment. *Computing Research Repository*, arXiv:2506.07982.

Qianshu Cai, Yonggang Zhang, Xianzhang Jia, Huajiang Zheng, Wei Xue, Jun Song, Xinmei Tian, and Yike Guo. 2026. MOSS: Self-evolution through source-level rewriting in autonomous agent systems. *Computing Research Repository*, arXiv:2605.22794.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, and Qing Wang. 2026. From failed trajectories to reliable LLM agents: Diagnosing and repairing harness flaws. *Computing Research Repository*, arXiv:2606.06324.

DeepSeek-AI et al. 2026. DeepSeek-V4: Towards highly efficient million-token context intelligence. *Computing Research Repository*, arXiv:2606.19348.

Yuetian Du, Yucheng Wang, He Xu, Jiexu Xu, Shanwen Tan, Bing Zhao, Boyu Yang, Zhijie Xu, Ming Kong, Hu Wei, Jie Liu, and Qiang Zhu. 2026. LivingHarness is an interactive-agent evolver. *Computing Research Repository*, arXiv:2607.26598.

Zhezheng Hao, Hong Wang, Jian Luo, Jianqing Zhang, Yuyan Zhou, Qiang Lin, Can Wang, and Hande Dong. 2026. ReCreate: Reasoning and creating domain agents driven by experience. *Computing Research Repository*, arXiv:2601.11100.

Yue Huang, Wenjie Wang, Han Bao, Yuchen Ma, Xiaonan Luo, Yi Nian, Haomin Zhuang, Zheyuan Liu, Yue Zhao, and Xiangliang Zhang. 2026. MemoHarness: Agent harnesses that learn from experience. *Computing Research Repository*, arXiv:2607.14159.

Seth Karten, Joel Zhang, Tersoo Upaa Jr., Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. 2026. Continual harness: Online adaptation for self-improving foundation agents. *Computing Research Repository*, arXiv:2605.09998.

Hyunin Lee, Jinglue Xu, Jeffrey Seely, Donghyun Lee, Matei Zaharia, and Yujin Tang. 2026a. Recursive harness self-improvement. *Computing Research Repository*, 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. *Computing Research Repository*, arXiv:2603.28052.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can LLMs already serve as a database interface? a big bench for large-scale database grounded text-to-SQL. *Advances in Neural Information Processing Systems*, 36.

Hongqiang Lin, Chao Liu, Xiaofan Bai, Xuan Jin, Yuhong Li, Nenggan Zheng, and Xipeng Cao. 2026a. Rethinking self-evolution: A constrained exploration-exploitation process for mitigating skill overfitting. *Computing Research Repository*, arXiv:2607.26643.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, and Hang Yan. 2026b. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *Computing Research Repository*, arXiv:2604.25850.

Zewen Liu, Zhan Shi, Yisi Sang, Bing He, Minhua Lin, Tianxin Wei, Dakuo Wang, Benoit Dumoulin, Wei Jin, and Hanqing Lu. 2026. Adaptive auto-harness: Sustained self-improvement for agentic system deployment on open-ended task streams. *Computing Research Repository*, arXiv:2606.01770.

Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. 2026. AutoHarness: improving LLM agents by automatically synthesizing a code harness. *Computing Research Repository*, arXiv:2603.03329.

Xiaotian Luo, Fengxingyu Wang, Chuanrui Hu, Dizhan Xue, and Yafeng Deng. 2026. Self-evolving agent harnesses via gated semantic quality-diversity. *Computing Research Repository*, arXiv:2607.13683.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, et al. 2026. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *Computing Research Repository*, arXiv:2601.11868.

Jun Nie, Yonggang Zhang, Jun Song, Qianshu Cai, Dahai Yu, Yike Guo, Xinmei Tian, and Bo Han. 2026. TTHE: Test-time harness evolution. *Computing Research Repository*, arXiv:2607.08124.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, Lingjie Chen, Yanjun Zhao, Ke Yang, Bingxuan Li, Cheng Qian, Gaotang Li, Xiao Lin, Zhichen Zeng, Ruizhong Qiu, and 23 others. 2026. Code as agent harness. *Computing Research Repository*, arXiv:2605.18747.

OpenAI. 2026. Codex: A lightweight coding agent. https://github.com/openai/codex. Accessed: 2026-07-26.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.

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. *Computing Research Repository*, arXiv:2606.05922.

Quan Shi, Alexandra Zytek, Pedram Razavi, Karthik Narasimhan, and Victor Barres. 2026. τ-knowledge: Evaluating conversational agents over unstructured knowledge. *Computing Research Repository*, arXiv:2603.04370.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *International Conference on Learning Representations*.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. TextGrad: Automatic "differentiation" via text. *Computing Research Repository*, arXiv:2406.07496.

Mario Zechner. 2025. Pi agent harness. https://github.com/earendil-works/pi. Accessed: 2026-07-26.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. 2026. Self-harness: Harnesses that improve themselves. *Computing Research Repository*, arXiv:2606.09498.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*.

A Implementation and Verification Details

This appendix provides implementation details for the three components of HARNESSLENS described in Section 4: Context Exploration, Trajectory Diagnosis, and Harness Evolution. It specifies the controller procedure, budget accounting, role interfaces, editable harness components, and verification protocol used in our experiments.

A.1 Controller Procedure and Budget Accounting

Algorithm 1 instantiates the procedure in Section 4. Each candidate is constructed from the current confirmed harness, so accepted modifications accumulate and rejected candidates leave the current harness unchanged. All model-based roles run in fresh sessions and communicate only through schema-validated artifacts.

Every task rollout and LLM session consumes one unit of the interaction budget B, as specified in Section 3.2. With 30 TRAIN tasks and $K = 2$ trials per task, the initial rollout costs $30K = 60$ units. For a verification batch of n tasks against the current harness $\mathcal{H}_j$, the required cost is

$$C_1(n, \mathcal{H}_j) = 6 + nK\big(1 + \mathbb{I}[\mathcal{H}_j \neq \mathcal{H}_0]\big) + 2\left\lceil \frac{n}{3} \right\rceil,$$

The fixed six units cover proposal selection, candidate editing, the runtime check, two analyses of intended effects and regressions, and the final review. The rollout term covers candidate trials and, when $\mathcal{H}_j \neq \mathcal{H}_0$, fresh trials under the current harness. The final term covers independently reviewed behavior comparisons in bundles of at most three tasks. The confirmation round always collects fresh trials under both harnesses and costs

$$C_2(n) = 3 + 2nK + 2\left\lceil \frac{n}{3} \right\rceil.$$

Before an iteration starts, the controller checks that the remaining budget covers both rounds and a three-unit retry buffer. Invalid structured outputs may be retried once for exploration and twice for diagnosis or review; every retry remains charged. If the remaining budget cannot support both rounds, the controller does not start another candidate iteration.

For example, with $K = 2$, a five-task verification against $\mathcal{H}_0$ costs 20 units because its initial trajectories are reused, and the corresponding confirmation round costs 27 units. If the current harness differs from $\mathcal{H}_0$, a six-task verification and confirmation round cost 34 and 31 units, respectively. Including the three-unit retry buffer, this six-task iteration requires $34 + 31 + 3 = 68$ available units. The controller starts an iteration only when the complete two-round cycle is affordable and returns the current confirmed harness otherwise.

Algorithm 1 Controller-level implementation of HARNESSLENS.

Require: $\mathcal{H}_0$, $\mathcal{T}_{\text{train}}$, B, $K = 2$
Ensure: Confirmed harness $\mathcal{H}$

Run the initial rollout of $\mathcal{H}_0$ on $\mathcal{T}_{\text{train}}$ with K trials per task
Run Task-Space and Harness-Space Exploration
Apply Trajectory Diagnosis to the initial trajectories to obtain proposals
$\mathcal{H} \leftarrow \mathcal{H}_0$
while a supported proposal and a complete verification cycle remain **do**
  Evolution agent selects proposal p and batch S, $|S| \geq 5$
  $\widetilde{\mathcal{H}} \leftarrow \text{Edit}(\mathcal{H}, p)$
  if the modified component passes the runtime check **then**
    if $\mathcal{H} = \mathcal{H}_0$ **then**
      Reuse the initial trajectories for $\mathcal{H}_0$ on S
    else
      Evaluate $\mathcal{H}$ on S
    end if
    Evaluate $\widetilde{\mathcal{H}}$ on S with matched trials
    Apply Trajectory Diagnosis to the paired trajectories
    Evolution agent reviews the verification evidence and metrics
    if attributable positive evidence and no attributable regression **then**
      Construct confirmation batch S' (Section A.4.1)
      Evaluate both harnesses on S' under fresh matched conditions
      Apply Trajectory Diagnosis to the confirmation trajectories
      Evolution agent reviews the confirmation evidence and metrics
      if supported evidence remains and the primary metric improves **then**
        $\mathcal{H} \leftarrow \widetilde{\mathcal{H}}$
      end if
    end if
  end if
  Add revised or newly diagnosed proposals
  Update the evidence history and remaining budget
end while
return $\mathcal{H}$

A.2 Role Interfaces

Table 3 summarizes the visible inputs, excluded information, and outputs of the model-based roles used by HARNESSLENS. Within Trajectory Diag-

nosis, behavior comparison is performed without access to the candidate diff; the subsequent assessment of intended effects and regressions receives the diff only after that comparison is complete.

Task- and Harness-Space Exploration use at most 30 agent steps, the Harness Editor at most 40, and the Trajectory Diagnosis and evolution-agent roles at most 60. Their complete prompts and output schemas are included in the anonymized software supplement. All roles use the same high-reasoning profile as the evaluated agents, with a 65,536-token context limit and a 24,576-token output limit. Exploration sessions have a 1,800-second wall-clock limit and the remaining roles have a 3,600-second limit.

A.3 Harness Search Spaces

Each candidate is an isolated snapshot of the current configurable state $\mathbf{h}_j$ and the task project. The harness framework, base model, provider, permissions, benchmark tools, and evaluator state remain fixed. Patches to tool and parameter descriptions must bind to exact names already exposed by the environment and therefore cannot introduce new tools.

Before verification, a one-trial load check records the effective startup instructions, discovered skill metadata, tool schema, and lifecycle configuration. A candidate is rejected without rollout if a modified component has no applicable checker or its content is absent from the effective runtime context. This prevents edits to inert files from being credited as behavioral changes.

A.4 Verification and Update Protocol

A.4.1 Task Selection

The evolution agent assigns each task in an initial verification batch one of the four selection labels in Table 5. The labels record each task's role in testing the targeted behavior and detecting potential regressions.

A verification batch contains at least five distinct TRAIN tasks, including at least one conversion task and one task linked to the proposal's supporting trajectory. Remaining positions cover related task groups, affected tools, and preservation risks. The evolution agent selects the tasks and provides their selection labels and evidence links; the controller validates task distinctness, batch size, trial count, and affordability.

The confirmation round has the same batch size but uses mostly fresh tasks. It retains at most $\min(2, n-1)$ verification tasks that require further confirmation and fills the remaining positions with previously unused TRAIN tasks that succeeded under the initial harness. The controller prioritizes task groups not represented in the verification and reuses other verification tasks only when fresh eligible tasks are exhausted.

A.4.2 Paired Rollouts and Review

Each round uses two paired trials per task. Trials under the current and candidate harnesses share benchmark-controlled conditions within a round, while the confirmation round uses fresh trial seeds. For interactive benchmarks, the user-simulator seed is determined by the domain, task ID, and trial index; target-agent seeds additionally include an agent namespace. Pairing reduces avoidable variation but does not assume provider-side determinism.

Based on the binary rewards defined in Section 3.2, behavior comparison assigns each task one of the labels in Table 6.

Comparison reports are independently checked against the source trajectories before verification-stage Trajectory Diagnosis receives the candidate diff. A verification stage advances only if it contains attributable positive evidence and no attributable regression. Positive evidence is either an attributed recovery or an attributed stable success with an increased pass count; preservation alone is insufficient. An update additionally requires improvement under the primary metric on the confirmation batch and a final accept decision. Evidence from an incomplete or unaffordable confirmation round can inform later proposals but cannot update the confirmed harness.

B Experimental Protocol

B.1 Data Splits

Table 7 gives the fixed split sizes and sampling provenance. The exact ordered task IDs, sampling seed (42), and upstream benchmark revisions are included in the anonymized software supplement.

B.2 Models, Harnesses, and Runtime Limits

All evaluated agents and model-based roles use `deepseek-v4-flash-preview`. The evaluated harness versions are OpenCode 1.17.13, Codex CLI 0.144.4, and Pi Coding Agent 0.80.10. Banking Knowledge uses BM25 retrieval.

Role	Visible information	Excluded information	Output
Task-Space Exploration	TRAIN queries, public policies, and tool schemas	Trajectories, rewards, candidates, and TEST	Task groups
Harness-Space Exploration	Harness documentation and runtime observations	Task outcomes and domain-specific proposals	Editable-component inventory
Experience Extraction	Model-visible TRAIN trajectories and outcomes	Candidate diff and TEST	Reusable experiences and recurring deficiencies
Experience Analysis	Extracted evidence, task groups, and component inventory	TEST and hidden references	Evidence-supported modification proposals
Behavior Comparison	Current-harness and candidate trajectories	Candidate diff and hidden evaluator explanations	Task-level behavior changes
Trajectory Diagnosis (verification)	Comparison reports, task groups, and candidate diff	TEST and hidden references	Assessment of intended effects and regressions
Harness Editor	One proposal, the component inventory, and an isolated snapshot of the current harness	TEST, credentials, evaluator state, and fixed runtime limits	Candidate snapshot and file-level diff
Evolution Agent	Proposals, evidence history, task-selection records, and budget state	TEST and hidden references	Proposal selection and candidate acceptance, rejection, or revision

Table 3: Information boundaries between the model-based implementation roles. Budget arithmetic and trial counts are controller-owned rather than accepted from model outputs.

Harness	User-configurable components	Visibility	Main regression risks
OpenCode	Instructions, skills, tool and parameter descriptions, agent definitions, commands, reference sources, and compaction configuration	Startup, tool schema, skill loading, or lifecycle events	Broad prompt effects, context growth, inert files, or authority changes
Codex	Developer and project instructions, skills, lifecycle-hook context, tool and parameter descriptions, and compaction prompt	Startup, tool schema, skill loading, or lifecycle events	Broad instructions, trigger failure, repeated injection, or context loss
Pi	System-prompt append, project instructions, skills, tool and parameter descriptions, and compaction configuration	Startup, tool schema, on-demand reading, or lifecycle events	Native-prompt replacement, trust dependence, trigger failure, or context loss

Table 4: User-configurable components identified by Harness-Space Exploration for the three evaluated harnesses.

Selection label	Criterion
Conversion	Initial failure that directly exercises the proposed behavior
Positive control	Initial success directly linked to supporting evidence
Preservation	Other initial success used to probe collateral regression
Diagnostic	Other initial failure used to test the proposal's scope

Table 5: Controller records used to construct a verification batch.

Status	Definition
Recovered	The current harness has no successful trial and the candidate has at least one
Stable success	The current harness has a successful trial and all candidate trials succeed
Regressed	The current harness has a successful trial and the candidate has none
Still failing	Neither harness has a successful trial
Mixed	All remaining split or partial outcomes

Table 6: Task-level labels used by behavior comparison.

For τ^2 interaction, the agent may make at most 10 tool calls per conversation turn, 40 conversation turns, and 60 tool calls overall; the per-turn timeout is 180 seconds. The user simulator uses temperature 0.3 and is reset with the stable pairing seed for each trial. Terminal-Bench uses at most 50 agent steps, a 600-second agent timeout, and a 1,800-second verifier timeout. BIRD uses at most 30 tool calls per round, a 600-second agent timeout, a 5-second query timeout, and a 30-second grader timeout.

Evaluator-owned overrides are applied after candidate-native configuration is loaded. They fix

Benchmark	TRAIN	TEST	Sampling record
τ^2-bench Retail	30	40	Official 40-task TEST split; TRAIN drawn with seed 42 from the other 74 tasks
τ^3-bench Banking Knowledge	30	67	TRAIN drawn with seed 42 from all 97 tasks; TEST is the remainder
Terminal-Bench 2.0	30	59	TRAIN drawn with seed 42 from all 89 tasks; TEST is the remainder
BIRD Mini-Dev (Challenging)	30	72	TRAIN drawn with seed 42 from the 102 challenging items in official JSONL order; TEST is the remainder

Table 7: Fixed splits used for harness evolution and blind evaluation. Exact ordered IDs and upstream revisions are distributed in the anonymized software supplement.

the model, provider, task tools, inference limits, and permissions, so a modification cannot change the model, enlarge its own step or token budget, enable external information retrieval, or alter task tools.

B.3 Metrics and Blind TEST Protocol

TRAIN uses $K = 2$ trials per task. Its controller metric is the trial-level pass rate

$$\mathrm{PassRate}_{\mathrm{TRAIN}} = \frac{1}{NK} \sum_{i=1}^{N} \sum_{r=1}^{K} R_{i,r}, \quad (1)$$

which is used only for update decisions alongside the attributable-evidence gate. Table 1 instead reports held-out TEST pass@1: one fresh trial per task under the final selected harness. We use the term pass@1 only for this single-trial TEST metric.

Initial and evolved harnesses are evaluated on the same ordered TEST split, with the same runtime limits and pairing offsets. TEST runs through a separate entry point after evolution completes. All information and services associated with TEST—including task IDs, trajectories, evaluation feedback, and rollout services—are unavailable to every exploration, diagnosis, proposal, verification, and review role.

B.4 Baseline Adaptation and Budget Comparability

All methods use the same initial harness, 30-task TRAIN split, benchmark runtime, verifier, and mechanisms for applying changes to user-configurable components. Adaptations are restricted to environment interfaces and do not change a baseline's proposal or selection policy. Table 8 gives the configured protocols; their actual consumption can be lower because of early stopping, retries, or method-specific validation.

For Self-Harness, 20 iterations each evaluate the current harness and three candidates on all 30 tasks with two trials, giving $20 \times 4 \times 30 \times 2 = 4{,}800$ TRAIN rollouts. Meta-Harness evaluates the initial harness once and one candidate in each of ten iterations, giving $11 \times 30 \times 2 = 660$ rollouts. HarnessFix performs three iterations; each evaluates the current harness and candidate on 30 tasks and rechecks at most two repairs on 20 tasks, giving $3 \times (2 \times 30 + 2 \times 20) = 300$ rollouts.

B.5 Artifact Release and Responsible Reporting

The anonymized supplement includes our implementation, configurations, prompts, split identifiers, and derived run metadata. Third-party agent source code, benchmark data, model weights, and full trajectories are not redistributed and should be obtained from their original releases.

The benchmark dependencies are used under their respective licenses: MIT for τ-bench, Apache License 2.0 for Terminal-Bench, and CC BY-SA 4.0 for BIRD Mini-Dev. The agent software and model APIs remain subject to their respective licenses and terms of use.

AI Assistance Disclosure. The authors used general-purpose AI assistants for literature search, implementation support, and manuscript drafting and revision. The authors reviewed, edited, and verified all resulting code, citations, analyses, and prose, and remain responsible for the final paper.

C Supporting Analyses

C.1 OpenCode–BIRD Evolution Trace

Figure 4 shows the accepted and rejected modifications from an illustrative OpenCode–BIRD run. The first instruction edit produced no attributable improvement and was rejected. Four later edits

Method	Internal split	Trials/eval.	Max. iterations	Candidates/iter.	Additional rule
Self-Harness	20/10	2	20	3	Released held-in/held-out protocol
Meta-Harness	30/0	2	10	1	All TRAIN tasks used for validation
HarnessFix	20/10	1	3	1	At most two modification retries
HARNESSLENS	30 (targeted)	2	–	–	Total budget $B = 200$ units

Table 8: Configured evolution protocols. The internal split is carved from the same 30 TRAIN tasks; TEST is excluded from every evolution budget.

Method	Configured calculation	Maximum
Self-Harness	20 iterations $\times$ 4 evaluations $\times$ 30 tasks $\times$ 2 trials	4,800
Meta-Harness	11 evaluations $\times$ 30 tasks $\times$ 2 trials	660
HarnessFix	3 iterations $\times$ [2 full evaluations $\times$ 30 tasks + 2 repair rechecks $\times$ 20 tasks]	300
HARNESSLENS	$B = 200$ total units, including 60 initial trials and all LLM sessions	≤ 200

Table 9: Configured TRAIN cost. HARNESSLENS uses strictly fewer than 200 task rollouts. The displayed maxima compare auditable interaction budgets, not matched total compute costs.

added deterministic tie handling, a conditional-aggregation pattern for two-entity differences, an output-column restriction, and native precision guidance. For each edit, both verification rounds supported the targeted improvement without an attributable regression, and the primary metric improved on the confirmation batch; the edit was therefore accumulated into the confirmed harness. The final skill candidate was not invoked during verification and produced no attributable change, so it was rejected. Because successive candidates use different modification-specific batches, their local verification rates should not be interpreted as a performance curve. The final harness improves held-out TEST pass@1 from 27/72 (37.50%) to 33/72 (45.83%), an 8.33 percentage-point increase.

C.2 Decision-Level Analysis of the OpenCode Runs

C.2.1 Scope

This appendix examines the decisions within the four evolution runs that produce the OpenCode results in Table 1: one run with $B = 200$ for each benchmark. Fixing the target harness to OpenCode follows the setting of Section 6.1 and avoids differences in component inventory across harnesses.

Table 10 lists the four runs. Three accepted at least one modification, and one returned $\mathcal{H}_0$. Together they contain 21 candidate iterations, 19 of which completed a paired verification comparison against the current confirmed harness. Evidence about TEST outcomes comes from Tables 1 and 2, not from this analysis.

Benchmark	Iter.	Units	$\mathcal{H}_0$	Final
Retail	6	196	75.00	85.00
Banking	6	197	20.90	25.37
BIRD	5	197	37.50	45.83
Terminal	4	181	33.90	33.90

Table 10: Summary of the four OpenCode evolution runs, one per benchmark. **Iter.** reports the number of candidate iterations, and **Units** reports the interaction-budget units consumed from the total budget of 200. $\mathcal{H}_0$ and **Final** report held-out TEST pass@1 (%) before and after evolution, respectively. The Terminal-Bench run accepted no modification and therefore returned $\mathcal{H}_0$.

C.2.2 Re-scoring the Recorded Decisions

For every iteration, the controller recorded the candidate and the current confirmed harness under identical tasks and trial conditions. We recompute the paired difference in verification TRAIN pass rate and ask which iterations a metric-only rule ($\Delta > 0 \Rightarrow$ accept) would decide differently from the attributable-evidence gate. Nothing is re-executed: both rules are applied to the same stored evidence, so the comparison is exact for these runs.

The two rules differ on 7 of the 19 paired iterations (Figure 5): the metric-only rule would accept 10 modifications, whereas the attributable-evidence gate accepted 5. Six iterations improved the verification pass rate but were not accepted. In the

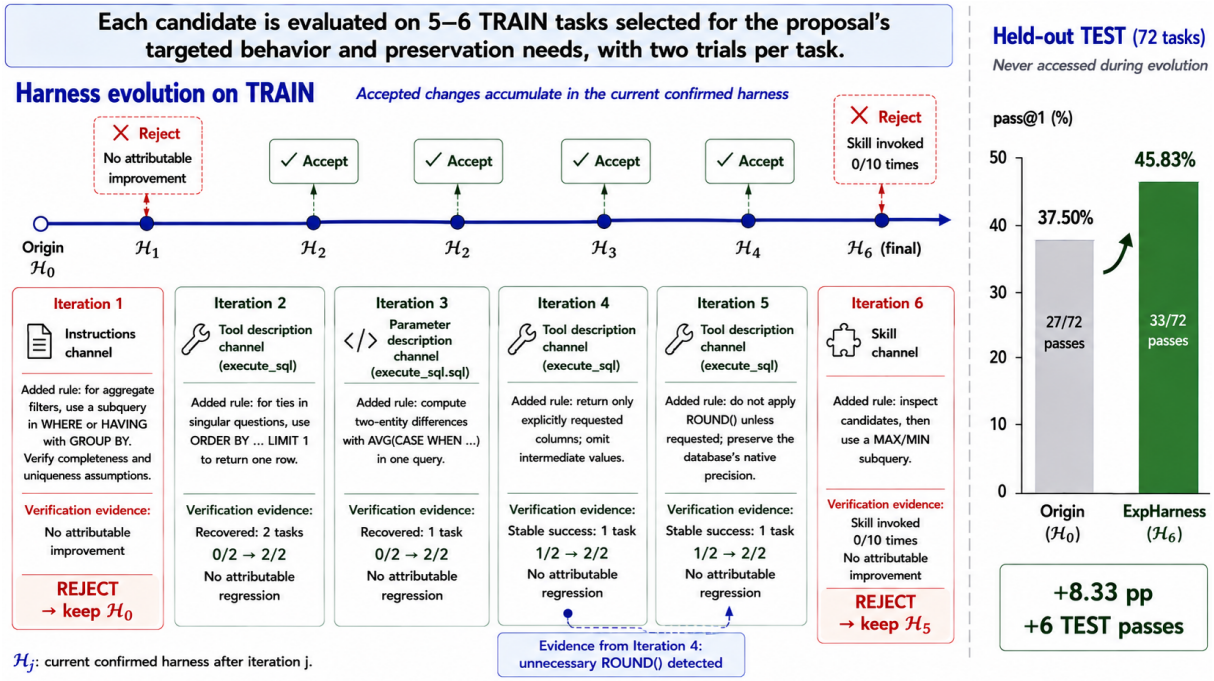

Figure 4: Illustrative OpenCode–BIRD evolution trace. Four modifications with attributable improvements are accumulated, while two modifications without attributable evidence are rejected. The final harness improves held-out TEST pass@1 by 8.33 percentage points.

largest disagreement, a candidate gained 50.0 percentage points on its verification batch, but Trajectory Diagnosis found no task recovery attributable to a modified component. Conversely, one iteration advanced despite a 10.0-point verification-batch decrease because it showed an attributable recovery of the targeted behavior with no attributable regression; its final acceptance still required improvement on the confirmation batch. The Terminal run gives the clearest example of rejection: its only iteration with a positive paired difference gained 40.0 points without an attributable recovery, and the run therefore returned $\mathcal{H}_0$, as reported in Table 1.

This re-scoring shows that the two rules select different modifications from the same within-run evidence; it does not show how the counterfactual metric-only harnesses would perform because those harnesses were not constructed in these runs. The separate *Metric-Only Gate* ablation in Table 2 provides the corresponding TEST result.

C.2.3 Which Components Were Edited

We identify the native component modified in each iteration from the workspace diff captured after the editor ran, rather than from the component proposed by Experience Analysis. The resulting record illustrates how task properties and component visibility shape iteration, although one run per benchmark does not support a rate estimate. Because every run uses OpenCode, all four draw from the same component inventory; nevertheless, the components modified differ across benchmarks. In the BIRD run included in this decision-level analysis, all five iterations modify benchmark tool or parameter descriptions, whereas the three completed Terminal edits modify general instructions. Retail and Banking edits center on skills, with fewer instruction and tool-description edits. General instruction components are attempted most often but are seldom accepted in these runs, whereas skill and tool-schema edits do.

C.2.4 Where the Budget Goes

Aggregating the interaction-budget ledger described in Section A.1 for each run yields Figure 6. Initial and verification rollouts, behavior comparison, and post-rollout Trajectory Diagnosis account for 82.2% of all units charged across the four runs and between 79.0% and 83.8% within individual runs. Context Exploration, proposal selection, and candidate editing account for the remainder. These values are direct sums from the budget ledger. This accounting makes the comparison in Table 9 conservative because HARNESSLENS's 200-unit budget counts both task trials and LLM sessions, while the baseline figures count task rollouts only and

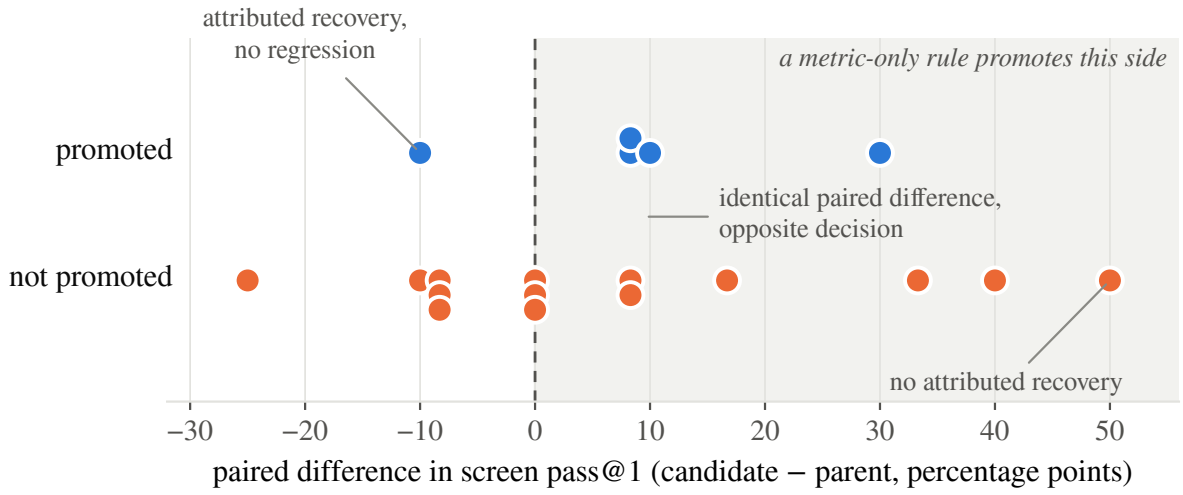

Figure 5: Candidate-minus-current-harness differences in TRAIN pass rate on matched tasks and trials for 19 completed initial-verification comparisons across four OpenCode runs. Rows indicate the attributable-evidence decisions, and the shaded region marks positive differences that a metric-only rule would accept.

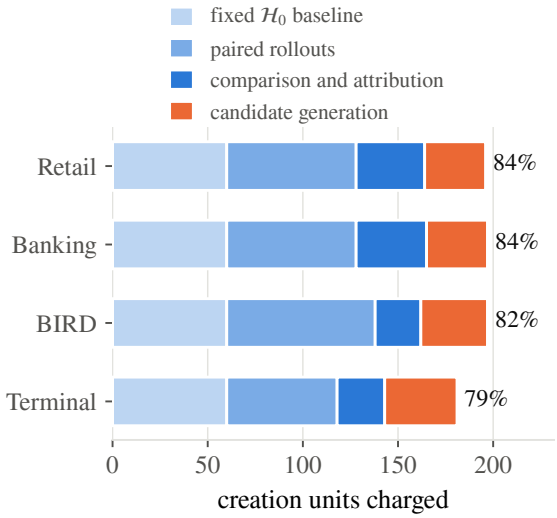

Figure 6: Breakdown of interaction-budget consumption across the four OpenCode runs. Each stacked bar shows the units spent on the initial rollout, paired verification rollouts, comparison and attribution, and candidate generation. Percentage labels give the combined share of the first three categories.

exclude non-rollout LLM sessions.