CAN HARNESS OPTIMIZATION IMPROVE AGENTS? GENERALIZABLE GAINS VERSUS EVALUATION NOISE

Anonymous authors
Paper under double-blind review

ABSTRACT

Harness optimization methods revise an agent's prompts, tools, memory, and runtime control without updating model weights. We ask which of their improvements can be distinguished from evaluation noise, and where the reliable ones come from. We evaluate Meta-Harness, Agentic Harness Engineering, and Retrospective Harness Optimization on Terminal-Bench 2.1, DeepSWE, and FrontierCS, eight method–benchmark evaluations in all. In each, the selected harness and its seed harness receive repeated independent runs on every held-out task under the same model, budgets, and environment. The benchmark-level effect is an equally weighted mean over tasks, and its confidence interval reflects run-to-run variation. Two of the eight selected harnesses improve reliably on their seed harnesses, five effects cannot be distinguished from evaluation noise, and one is a reliable regression. In an additional Meta-Harness search on Terminal-Bench 2.1, most of the selected harness's held-out improvement comes from tasks on which the seed harness's shell tends to crash. The selected harness restarts the shell after a crash; on the other tasks, there is no clear evidence of improvement. Repairing such an environment failure can raise held-out performance without showing that the agent solves problems better. We analyze change targets and observed failure causes to distinguish evidence of environment repair from evidence of improved problem solving.

1 INTRODUCTION

An agent's behavior depends on its harness as well as its language model: the prompts, tools, memory, and runtime control around the model. With the model fixed, the design of these components alone can change task performance, as work on interleaving reasoning with actions (Yao et al., 2023) and on agent–computer interfaces (Yang et al., 2024) has shown. The harness is therefore an optimization target in its own right, not only the infrastructure that calls the model. Harness optimization methods revise these components using task traces and evaluation rewards or self-evaluations. What they edit ranges from textual instructions (Agrawal et al., 2025) and agent code (Hu et al., 2024) to whole executable harnesses (Lee et al., 2026; Lin et al., 2026; Pan et al., 2026). Because they leave the model weights untouched, an agent could keep improving after its model is fixed. The improvements these methods report suggest that agents could improve their own harnesses: experience on the source tasks of a search would carry over to held-out tasks it never sees.

Two questions decide what such improvements mean. The first is whether an observed improvement can be distinguished from evaluation noise. Coding and tool-using agents are stochastic even when the model, task, harness, and budget are fixed: repeated runs of the same task can end differently (Yao et al., 2024). The methods we study select the harness they return from comparisons of such runs, and none requires a confidence interval to exclude zero. Selection can then favor changes that benefited from noise (Cawley & Talbot, 2010). Evaluating the selected harness on held-out tasks avoids reusing the source-task runs behind its selection, but with only a few runs per task the held-out estimate is uncertain as well. A positive mean difference alone therefore does not establish that a candidate harness improves on its *seed harness*, the configuration from which it was generated. Beyond the standardized evaluation conditions that agent comparisons need (Kapoor et al., 2024), establishing an improvement takes a confidence interval on the difference between the two harnesses.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

The second question is where a reliable improvement comes from. A harness can raise an agent's score by helping it find and check solutions. It can also remove failures in how the agent interacts with its environment, such as a shell that crashes or a tool that is missing. An improvement from restoring execution does not by itself establish an improvement in the agent's problem-solving strategy. Evaluating on held-out tasks does not tell the two apart. Source and held-out tasks share the harness's interface with the environment, so a failure of that interface can recur on both, and a change that removes it raises held-out scores as well. What a held-out improvement shows therefore depends on the failures that the harness changes remove, not only on the tasks the search used.

We evaluate Meta-Harness, Agentic Harness Engineering (AHE), and Retrospective Harness Optimization (Retro Harness) on Terminal-Bench 2.1, DeepSWE, and FrontierCS (Merrill et al., 2026; Huang et al., 2026; Mang et al., 2025). AHE cannot be run on FrontierCS (Section 3.1), which leaves eight method–benchmark evaluations. In each, the method searches on the source tasks and returns one selected harness, frozen before any held-out run. The selected harness and its seed harness then receive $J_b = 8$ independent runs per held-out task under the same model, reasoning effort, budgets, and environment policy. Each held-out task yields a task-level comparison of the two harnesses. We average the task-level effects over held-out tasks, weighting every task equally, into a benchmark-level effect whose confidence interval reflects run-to-run variation on these tasks. We report each selected harness's improvement over its own seed and do not use these effects to rank the optimization methods.

Seven of the eight selected harnesses have a positive benchmark-level effect, but only two improve reliably on their seed harnesses: their confidence intervals lie above zero (Figure 1). Both are Meta-Harness evaluations; each method starts from its own seed harness, so this does not rank the methods. Five intervals include zero, and one lies below it. The five effects whose intervals include zero, $+0.5$ to $+1.6$ percentage points, are small relative to run-to-run variation. Using the seed-harness variance, the extrapolated 95% range of repeat-evaluation differences falls below these observed effects at approximately 140 to 2,200 runs per task, against four to five for the other three (Table 1). Most task-level intervals include zero (Figure 2). At either level, an interval that includes zero does not show that the two harnesses perform alike; the runs only fail to distinguish the effect from zero.

To see where a reliable improvement comes from, we conduct an additional Meta-Harness search on Terminal-Bench 2.1 (Section 4.1). This search generates its own five candidates using four runs per source task and a modified candidate-selection rule. Its returned harness also improves reliably on held-out tasks: $+11.2$ percentage points, with 95% interval $[+4.3, +18.1]$. For the held-out analysis, we classify tasks as crash-prone or not using seed-harness runs from the main MH/TB evaluation, outside the case-study comparison. The held-out improvement comes mostly from the crash-prone tasks: $+27.1$ percentage points, against $+3.8$ on the others, whose interval includes zero. Two of the returned harness's three changes restart the terminal session after a crash, so a run that the seed harness would lose continues. In the held-out evaluation, 24 of the seed harness's 152 runs end in a shell crash, and none of the returned harness's. We call such a change *environment repair*. It restores execution after a failure in the seed harness's interaction with its environment.

Environment repair can raise held-out performance without showing that the agent solves problems better. A difference between groups of tasks also does not show by itself which change caused it. We therefore define a descriptive split of the effect based on run-level failure labels (Section 4.2). We separately label the failures targeted by harness changes using search-phase material. Table 2 reports the decomposition of all eight evaluations.

Our contributions are:

- A repeated held-out evaluation of three harness optimization methods on three benchmarks, with independent runs under matched evaluation conditions, task-level comparisons, and benchmark-level confidence intervals (Sections 3.1 and 3.2, Appendix A). It separates two reliable improvements and one reliable regression from five effects that cannot be distinguished from evaluation noise.
- An analysis of where the reliable improvements come from. In one case, most of the held-out improvement comes from crash-prone tasks, where the selected harness repairs the environment (Section 4.1). We define a descriptive effect decomposition and report it for all eight evaluations alongside change targets and observed failure causes (Section 4.2, Table 2).

The study asks which improvements are reliable and what evidence explains them.

2 Related Work

Automatic harness optimization. Harness optimization keeps the language model fixed and optimizes the components around it. Methods differ in which components they may edit and in the feedback that guides each edit. GEPA edits textual instructions: it reflects on execution traces to revise prompts and combines complementary candidates (Agrawal et al., 2025). Automated Design of Agentic Systems (ADAS) searches over agent code instead: its Meta Agent Search programs new agents from an archive of earlier designs (Hu et al., 2024). Meta-Harness edits executable harnesses and gives its proposer direct access to the search history. The proposer reads the code, scores, and traces of earlier candidates through a filesystem (Lee et al., 2026). Agentic Harness Engineering (AHE) structures this process around explicit editable components, layered evidence from traces, and a prediction of each change's effect on individual tasks (Lin et al., 2026). The four methods above learn from externally scored task outcomes, whereas Retrospective Harness Optimization (Retro Harness) needs no external grading. It derives updates from its own past traces through self-validation and self-consistency, and it selects among candidates by pairwise self-preference (Pan et al., 2026). Two of these works analyze what their searches found. Meta-Harness examines environment bootstrapping, which its search discovered on Terminal-Bench: the harness puts a snapshot of the sandbox into the agent's first prompt (Lee et al., 2026). AHE measures the effects of individual components and their interactions (Lin et al., 2026). Our analysis complements theirs. It separates *environment repair*, which removes a failure in the seed harness's interaction with its environment, from improvements in problem solving (Section 4.1). Such a repair can produce a reliable held-out improvement without showing that the agent solves problems better. We use this distinction to define the analysis in Section 4.2. Change labels record the failures targeted by a harness change; run-level labels separately describe the observed causes of shortfalls in held-out runs.

Statistical reliability of agent evaluation. A stochastic agent's consistency across repeated runs is distinct from the uncertainty in its estimated performance and in its comparisons with other systems. τ-bench measures consistency with pass^k, which requires all k runs of a task to succeed (Yao et al., 2024). Kapoor et al. (2024) identify inadequate held-out sets, inconsistent evaluation procedures, and insufficient attention to cost as obstacles to meaningful agent comparisons. When evaluations also guide candidate selection, the problem becomes one of statistical model selection. Cawley & Talbot (2010) show that optimizing a noisy selection criterion can overfit the criterion itself and favor candidates whose estimates are too optimistic. This bias affects the estimates used for selection. An independent held-out evaluation does not reuse them, but it still carries evaluation noise from a finite number of runs. Reusing the same data to guide later proposals also makes the analysis adaptive. Rogers et al. (2020) provide confidence intervals that remain valid under such adaptive queries. We focus on the uncertainty in a selected harness's improvement over its seed harness. We freeze each selected harness before any held-out run. We then compare it with its own seed harness in repeated independent runs under the same model, budgets, and environment (Section 3.1). The benchmark-level interval weights every held-out task equally. Like the task-level intervals, it quantifies run-to-run variation and is conditional on the evaluated tasks. Intervals on source tasks describe the search rather than test it (Section 4.1). The held-out intervals decide which improvements are reliable, and the failure analysis of Section 4 asks what the reliable ones change.

3 Which Harness Improvements Are Reliable?

3.1 Experimental Setup

Methods. We evaluate Meta-Harness (MH), AHE, and Retro Harness (RH), which optimize different components of an agent harness. We use three benchmarks covering terminal-based tasks (Terminal-Bench 2.1; TB), long-horizon software engineering (DeepSWE; DS), and open-ended computer-science problems (FrontierCS; FCS) (Merrill et al., 2026; Huang et al., 2026; Mang et al., 2025). Each benchmark is randomly split into source tasks and approximately 20% held-out tasks. For the eight primary evaluations, each method runs one search per benchmark, using only the source tasks to select one harness. Section 4.1 reports an additional MH/TB search for the case study. AHE requires binary rewards and is therefore evaluated only on TB and DS.

Table 1: Seed noise range against the benchmark-level effect, in percentage points. Values at $J = 64$ and 256 are extrapolated. MH/DS results are provisional (Section 3.1).

		Noise ($\pm$) at J runs per task				Runs to reach
Evaluation	$\widehat{\Delta}$	$J = 1$	8	64	256	$\lvert\widehat{\Delta}\rvert$
MH/TB	$+14.5$	25.7	9.1	3.2	1.6	4
MH/DS	$+1.6$	21.3	7.5	2.7	1.3	170
MH/FCS	$+5.0$	11.1	3.9	1.4	0.7	5
AHE/TB	$+1.3$	16.5	5.8	2.1	1.0	160
AHE/DS	$+0.5$	25.4	9.0	3.2	1.6	2200
RH/TB	$+1.3$	23.8	8.4	3.0	1.5	330
RH/DS	$+1.6$	19.2	6.8	2.4	1.2	140
RH/FCS	-4.4	7.9	2.8	1.0	0.5	4

Each method starts from the same initial (*seed*) harness across benchmarks, chosen to match its implementation's assumptions about which harness components can be modified. The one exception is Meta-Harness on DeepSWE, which starts from mini-swe-agent. Appendix A summarizes these choices and permitted modifications. Because seeds differ across methods, we evaluate improvements relative to each method's own seed, without ranking or pooling improvements across methods.

The primary protocol uses `gpt-5.6-terra` with high reasoning effort for all task-execution runs. For each comparison, the seed and selected harnesses use identical task environments, verifiers, and execution budgets; only the permitted harness components may change. Search budgets and method-specific setup details are provided in Appendix A.

Statistical Plan. Each held-out task has $J_b = 8$ independent runs per harness. For method m, benchmark b, and task t in the held-out task set $\mathcal{T}_b$, the *task-level effect* $d_{mbt} = \bar{r}^{\text{sel}}_{mbt} - \bar{r}^{\text{seed}}_{mbt}$ is the difference between the two harnesses' mean rewards. The *benchmark-level effect* $\widehat{\Delta}_{mb} = |\mathcal{T}_b|^{-1} \sum_{t \in \mathcal{T}_b} d_{mbt}$ is their equally weighted stratified mean, with each held-out task as one stratum. A *task-level comparison* is a significant improvement or regression when its 95% interval excludes zero, and statistically indistinguishable otherwise. For binary rewards we use the $N - 1$ χ^2 test with the Miettinen–Nurminen interval (Campbell, 2007; Miettinen & Nurminen, 1985); for continuous rewards, Welch's t-test. The standard error of $\widehat{\Delta}_{mb}$ combines the within-task variances of the task-level effects, $\mathrm{SE}(\widehat{\Delta}_{mb}) = \big(\sum_{t \in \mathcal{T}_b} \mathrm{SE}(d_{mbt})^2\big)^{1/2} / |\mathcal{T}_b|$. Its 95% interval $\widehat{\Delta}_{mb} \pm 1.96\,\mathrm{SE}$ therefore reflects run-to-run variation on these tasks. A selected harness shows a *reliable benchmark-level improvement* when the interval's lower endpoint L is above zero. We apply no multiplicity correction. The *seed noise range* $\pm 1.96\, s \sqrt{2/(J|\mathcal{T}_b|)}$ is the 95% range of the difference between two evaluations of the seed harness with J runs per task. Here s is the root mean square over held-out tasks of the per-task sample standard deviation of the seed harness's reward. We extrapolate the range beyond J_b, assuming independent runs, and report the number of runs per task at which it would fall to $|\widehat{\Delta}_{mb}|$.

3.2 Benchmark-Level Effects

Table 1 sets the benchmark-level effects of Figure 1 against the seed noise range, how far apart two evaluations of the same seed harness fall by chance. We measure it in each of the eight evaluations separately rather than once per benchmark. Pooling would average over different seed harnesses and their within-task variances. Across the eight evaluations, s is 17.6 to 43.9 percentage points. Assuming similar within-task variability under the two harnesses, the root-mean-square task-level standard error for one run per harness is approximately $\sqrt{2}\, s$, 24.9 to 62.1 percentage points. The one-run seed noise range is ± 7.9 to ± 25.7 percentage points. Splitting the observed runs into single-run evaluations of the seed harness reproduces this spread. The range shrinks as $1/\sqrt{J}$: using the observed effects as reference magnitudes, it falls below them at 4 to 2,200 runs per task, against the eight we run.

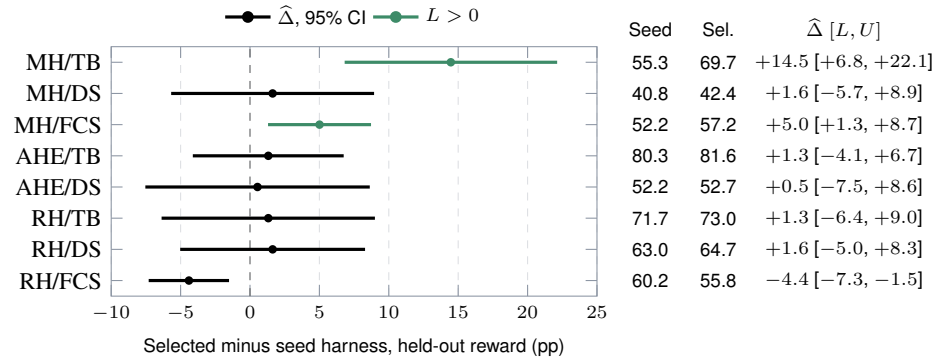

Figure 1: Benchmark-level effect of each selected harness over its seed harness. Seed, Sel.: mean held-out reward (%) of the two harnesses. MH/DS results are provisional (Section 3.1).

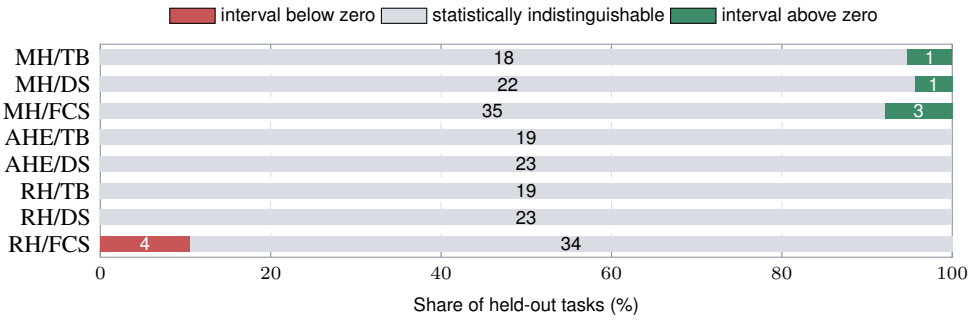

Figure 2: Task-level comparisons of the selected and seed harnesses. Numbers on the segments are task counts. MH/DS results are provisional.

The eight benchmark-level effects (Figure 1) range from -4.4 to $+14.5$ percentage points, and five of their confidence intervals include zero. Seven of the estimates are positive, and two, both from Meta-Harness, meet the criterion for a reliable benchmark-level improvement, $L > 0$; the interval of RH/FCS lies below zero.

The effects fall into two groups. The two reliable improvements and the reliable regression are large relative to the seed noise range, which falls below them at four to five runs per task. For the five effects whose intervals include zero, the seed-variance extrapolation falls below the observed effect magnitudes at approximately 140 to 2,200 runs per task. This is a precision reference, not a sample-size calculation at a specified detection power.

3.3 Task-Level Effects

A benchmark mean can combine task-specific improvements and regressions. Figure 2 classifies the task-level comparisons of all eight evaluations, as counts and as shares of held-out tasks. The share of tasks whose interval lies above zero ranges from 0% to 7.9%, and the share whose interval lies below zero from 0% to 10.5%. The rest are statistically indistinguishable.

With eight runs per harness, task-level intervals on binary rewards exclude zero only for large differences in observed success rates. Most task-level effects remain unresolved at this run count, so we interpret individual-task results cautiously.

In each bar, most task-level comparisons are statistically indistinguishable. The available runs cannot resolve the direction of the effect for these tasks. The task-level results locate large observed improvements, regressions, and unresolved effects alongside the benchmark mean.

4 WHERE DO RELIABLE HARNESS IMPROVEMENTS COME FROM?

Section 3.2 found reliable improvements in two evaluations. We now ask where such an improvement comes from and what it depends on: first in one case (Section 4.1), then in every evaluation (Section 4.2).

4.1 CASE STUDY: META-HARNESS ON TERMINAL-BENCH 2.1

Setting. For the case study, we conduct an additional Meta-Harness search on Terminal-Bench 2.1. Relative to the main MH/TB evaluation in Figure 1, this search uses four runs per source task and a modified candidate-selection rule. It generates its own five candidates. Four runs allow each candidate's source-task effect to be estimated within groups of tasks. Among the candidates whose 99% lower confidence bound on source tasks is above zero, it returns the one with the largest mean difference from the seed harness. The returned candidate, candidate 5, also has the largest mean difference of the five, so the point-estimate criterion would have returned it as well. We evaluate candidate 5 on the 19 held-out tasks with $J_b = 8$ runs per task and harness, scoring a run that ends in a shell crash as zero. It scores 66.4% against 55.3% for the seed harness: $\widehat{\Delta} = +11.2$ percentage points with 95% interval $[+4.3, +18.1]$, a reliable improvement. The $+14.5$-point effect in Figure 1 belongs to the harness selected by the main search.

Crash-prone tasks. The seed harness, Terminus-KIRA, drives a shell through a terminal multiplexer (tmux). When a command kills the terminal session, the harness cannot send another command: the run ends in a shell crash and scores zero. For the held-out analysis, we classify a task as crash-prone if at least two of its eight seed-harness runs in the main MH/TB evaluation ended in a shell crash. Those runs are outside the case-study comparison. This gives 6 crash-prone tasks among the 19 held-out tasks. The source-task analysis uses a provisional classification of 32 of the 70 tasks; its provenance remains to be reconciled with the seed-run records. The seed harness's shell crashed in 49% of its runs on crash-prone tasks and in 4% on the others.

Where the improvement arises. Figure 3 follows the effect through the search. Over all 70 source tasks (top row), the mean difference from the seed harness ranges from $+3.6$ percentage points for candidate 1 to $+14.3$ for candidate 5; candidate 1's 95% interval includes zero, and those of candidates 2 to 5 lie above it. The bottom row splits the same comparisons by crash propensity. On the 32 crash-prone source tasks, every candidate's effect lies between $+10.9$ and $+25.8$ points and its interval lies above zero; on the other 38 source tasks, every effect lies between -4.0 and $+4.6$ points and its interval includes zero. On held-out tasks (right), candidate 5's effect is $+27.1$ points $[+13.8, +40.4]$ on the six crash-prone tasks and $+3.8$ points $[-4.1, +11.8]$ on the other thirteen, whose interval includes zero. Both tasks whose task-level interval lies above zero are crash-prone. With thresholds of one or three crashes in eight instead of two, the crash-prone group's interval still lies above zero and the other group's still includes zero. Most of the held-out improvement comes from the crash-prone tasks; on the others there is no clear evidence of improvement.

Environment repair. Two of candidate 5's three changes restart the terminal session and its shell after a crash and let the run continue: one, from iteration 2, handles a crash during a command, and the other, from iteration 5, a crash noticed between commands. The third change, from iteration 4, retries a model call that stalls. In the held-out evaluation, 24 of the seed harness's 152 runs ended in a shell crash, against none of candidate 5's. After a shell crash, the seed harness terminates the run with reward zero, whereas candidate 5 attempts to restart the session and continue. We call a change of this kind *environment repair*. It removes a failure in the seed harness's interaction with its environment. These changes restore execution after a shell failure. The resulting performance gain does not by itself establish an improvement in the agent's problem-solving strategy. The main MH/TB evaluation's selected harness contains the same repair, which its search also introduced in iteration 2. The difference between groups of tasks does not by itself show that the repair produced the improvement, since crash-prone tasks may differ from the others in more than crashes.

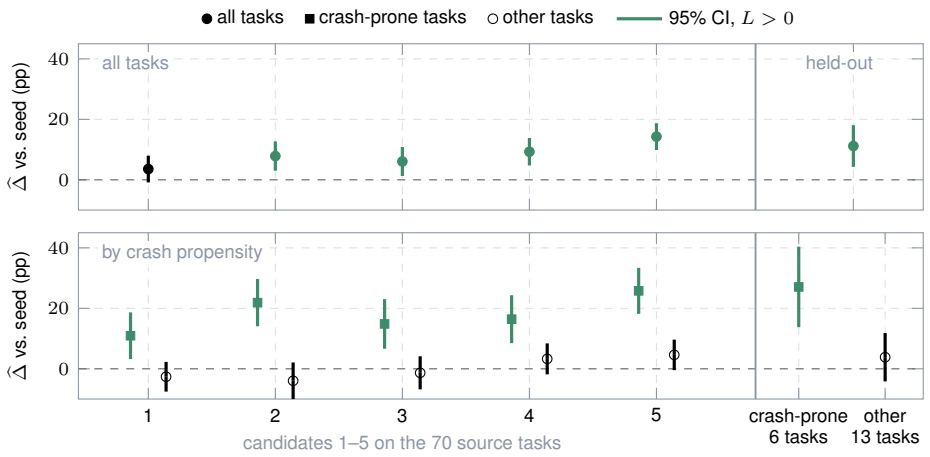

Figure 3: Additional Meta-Harness search on TB with four runs per source task: effect of each candidate on source tasks (left) and of the returned candidate on held-out tasks (right), over all tasks (top) and within crash-prone and other tasks (bottom). The source-task grouping is provisional; the held-out grouping uses separate seed runs from the main evaluation.

4.2 Environment Repair or Problem Solving across Evaluations

The case motivates a descriptive analysis across evaluations: how does the observed effect divide between changes in the frequency of environment-related failures and changes in conditional reward? We report each evaluation separately, since the methods start from different seed harnesses. We split each selected harness into harness changes, coherent edits relative to its seed harness that each address a single failure. For Meta-Harness, a change is a code edit described in the candidate's change notes. For Retro Harness, it is a notes file, or a section of one whose sections address different failures; a script or template counts as one change. For AHE, it is an entry of the change manifest.

Environment or problem. A harness change, or the shortfall of a run below the maximum reward, is *environment-related* when the environment or the harness interface blocks or slows the agent. This covers a crashed process or shell, an unavailable or failing tool or command, a long-running background process, time and resource limits, and the form of delivery (a commit, a branch, an output path). It is *problem-related* when it concerns how the agent searches for a solution and checks the solution itself: strategy, decomposition, verifying correctness, and domain knowledge. A check that the solution is correct is problem-related, and a check that it was delivered is environment-related. A timeout is classified by what the time went to: waiting on the environment, or searching for a solution. Time and memory limits imposed by the grader on a submitted solution are part of the problem. This analysis concerns scored outcomes, including harness-attributable execution failures; external infrastructure failures are rerun under the protocol in Appendix A. Environment repair is an environment-related change directed at a failure observed under the seed harness.

Change targets and observed run outcomes. We label changes and runs separately. A change is labeled from its target, the failure it addresses, using search-phase material alone: the failure pattern for AHE, and the change notes and the candidate's summary for the other methods. This label records what the method attempted. We separately classify observed failure causes in held-out runs. Every run of either harness that falls short of the maximum reward is labeled by the cause of its shortfall. A crash of the agent or its shell is environment-related by rule. Every other run is labeled once by a language model (Claude Fable 5), which reads a condensed trace of the run and applies the definitions above. The trace gives the task, each action and its output, how the run ended, and the grader's output. The model is not told which harness produced the run, although the trace can reveal it. The definitions, the model's instructions, and all labels are frozen before any split below is computed.

Table 2: Environment and problem parts of each benchmark-level effect, with 95% intervals. The problem part is a descriptive residual. Changes: number of environment-/problem-related changes. The MH/DS total effect is provisional (Section 3.1).

			Split of $\widehat{\Delta}$	
Evaluation	Changes (E/P)	$\widehat{\Delta}$	Environment part	Problem part (residual)
MH/TB	3/0	$+14.5$	$+15.5$ [$+10.6$, $+20.4$]	-1.0 [-6.8, $+4.7$]
MH/DS	0/3	$+1.6$	$+1.4$ [-1.0, $+3.8$]	$+0.2$ [-6.8, $+7.2$]
MH/FCS	3/1	$+5.0$	$+1.6$ [-0.3, $+3.6$]	$+3.4$ [$+0.4$, $+6.3$]
AHE/TB	5/2	$+1.3$	$+2.0$ [$+0.2$, $+3.8$]	-0.7 [-5.8, $+4.5$]
AHE/DS	1/4	$+0.5$	0.0 [0.0, 0.0]	$+0.5$ [-7.5, $+8.6$]
RH/TB	0/7	$+1.3$	$+0.8$ [-2.8, $+4.4$]	$+0.5$ [-6.1, $+7.1$]
RH/DS	1/7	$+1.6$	0.0 [0.0, 0.0]	$+1.6$ [-5.0, $+8.3$]
RH/FCS	0/4	-4.4	0.0 [0.0, 0.0]	-4.4 [-7.1, -1.7]

Splitting the effect. Within a task, a harness's mean reward is $e + p\,(q - e)$. Here p is the share of scored runs not labeled as environment-related failures, including fully successful runs; q is their mean reward, and e is the mean reward of environment-labeled runs, zero on the binary benchmarks. Write Δ for selected minus seed and a bar for the average over the two harnesses. The task-level effect splits as

$$d_{mbt} = \underbrace{(\bar{q} - \bar{e})\,\Delta p}_{\text{environment part}} + \underbrace{\bar{p}\,\Delta q + (1 - \bar{p})\,\Delta e}_{\text{problem part (residual)}}.$$

On binary benchmarks, $e = 0$, so the residual is the change in success rate among runs not labeled as environment-related failures, weighted by $\bar{p}$. On FrontierCS, it also includes changes in the mean reward of environment-labeled runs, $(1 - \bar{p})\Delta e$. Thus the label "problem part" denotes a residual in this descriptive split, not an isolated contribution of problem-solving ability.

When a conditional group is empty ($p = 0$ or $p = 1$), its undefined mean takes the other harness's corresponding mean, or zero if both are undefined. We apply this convention in the point estimates and in each bootstrap resample. For each task, we independently resample each harness's scored runs 2,000 times, recomputing both parts to estimate their standard errors. We average each part equally over tasks and combine its within-task variances as for $\widehat{\Delta}$ to obtain a normal-approximation 95% interval. The two point estimates add up to the benchmark-level effect. This decomposition does not identify which change caused an effect: an environment-related change that saves time can raise the residual, and a problem-related change can prevent environment-related failures.

The current change inventory contains 41 changes, of which 13 are labeled environment-related. Table 2 reports the decomposition of each evaluation.

4.3 Implications and Limits

The MH/TB case study combines a reliable held-out improvement, fewer shell-crash terminations, and code changes that restore execution after a crash. Together, these observations support environment repair as an explanation of the gain, while the task-group comparison does not isolate the effect of an individual change. An improvement from environment repair is real on the benchmark. Its size, however, depends on how often the seed harness fails in the environment as well as on what the search added. Reporting where an effect arises, next to its size and its interval, therefore matters as much as showing that it is not evaluation noise. The case-study evidence concerns one additional MH/TB search. The available results therefore support conclusions about the reported cases; broader conclusions about improved problem-solving strategies require the remaining analyses.

5 Conclusion

We asked which improvements from harness optimization can be distinguished from evaluation noise, and where the reliable ones come from. Across three methods and three benchmarks, two of the eight selected harnesses improve reliably on their seed harnesses, and one is a reliable regression. The other five effects cannot be distinguished from evaluation noise at eight runs per task. In the

additional Meta-Harness search on Terminal-Bench 2.1, most of the held-out improvement comes from tasks on which the seed harness's shell tends to crash, a failure the selected harness repairs by restarting the shell. Environment repair of this kind raises held-out performance without showing that the agent solves problems better. The case study motivates reporting observed failure causes alongside effect sizes and confidence intervals.

AI USE STATEMENT

In this work, we used generative AI tools and AI coding agents to assist with the implementation of software and experimental code based on methods and designs specified by the authors. We also used generative AI tools during paper writing to assist with drafting and editing, primarily to improve sentence-level wording, transitions, and clarity.

We also used a language model to assign run-level failure labels according to an author-defined rubric (Section 4.2). The authors developed the research hypotheses, core ideas, methodology, experimental design, literature review, figures, and analysis procedures, and interpreted the experimental results. The substantive organization and argumentation of the paper, including the structure and logic of individual paragraphs, were also written by the authors rather than generated by AI tools.

All AI-assisted code was reviewed and tested by the authors, and all AI-assisted text was reviewed and revised to ensure that it accurately reflected the authors' intended arguments and the underlying experimental results. We take responsibility for the final content of this work, including text, code, and other artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We will open-source all code and data necessary to reproduce our experiments upon publication.

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. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Ian Campbell. Chi-squared and Fisher–Irwin tests of two-by-two tables with small sample recommendations. *Statistics in Medicine*, 26(19):3661–3675, 2007. doi: 10.1002/sim.2832.

Gavin C. Cawley and Nicola L. C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(70):2079–2107, 2010.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring frontier coding agents on original, long-horizon engineering tasks. *arXiv preprint arXiv:2607.07946*, 2026.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

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

Qiuyang Mang, Wenhao Chai, Zhifei Li, Huanzhi Mao, Shang Zhou, Alexander Du, et al. FrontierCS: Evolving challenges for evolving intelligence. *arXiv preprint arXiv:2512.15699*, 2025.

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

Olli Miettinen and Markku Nurminen. Comparative analysis of two rates. *Statistics in Medicine*, 4 (2):213–226, 1985. doi: 10.1002/sim.4780040211.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference. *arXiv preprint arXiv:2606.05922*, 2026.

Ryan Rogers, Aaron Roth, Adam Smith, Nathan Srebro, Om Thakkar, and Blake Woodworth. Guaranteed validity for empirical approaches to adaptive data analysis. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pp. 2830–2840, 2020.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

A SETUP DETAILS

Section 3.1 summarizes the experimental setup; this appendix gives the details and the reasons behind each choice.

Methods. We run each method's upstream implementation, pinned to a fixed commit, behind one interface. The method receives the seed harness, the source tasks, and an evaluation callback restricted to source tasks, and it returns one selected harness. For the eight primary evaluations, each method is called once per benchmark and owns all of its internal iterations. Section 4.1 describes the additional MH/TB search. Meta-Harness runs five iterations; in each, a proposer (Claude Opus 4.8) writes one candidate, which is run once on every source task. Retro Harness runs one round of probing, coreset selection, group rollouts, and pairwise self-preference over three candidate notes. AHE runs five evolution iterations with $k = 2$ rollouts per source task, where its paper reports ten. We shorten the iteration count rather than k, because k supplies the pass-rate signal AHE selects on; the iteration count is also the axis its own limitations section names.

Benchmarks and splits. The benchmarks cover terminal tasks (Terminal-Bench 2.1), long-horizon repository engineering (DeepSWE), and algorithmic problems with test-case verifiers (FrontierCS) (Merrill et al., 2026; Huang et al., 2026; Mang et al., 2025). We keep each benchmark's outcome measure and report reward differences in percentage points. A seeded hash ranking of task identifiers splits each benchmark into source and disjoint held-out tasks: 70 and 19 on Terminal-Bench 2.1, 90 and 23 on DeepSWE, and 150 and 38 on FrontierCS. Methods may inspect source-task statements, traces, and rewards but never see the split registry. Before held-out evaluation we freeze the selected harness and record its content digest; the evaluation refuses a harness whose digest has changed.

Seed harnesses. Seed harnesses are chosen to match each method's upstream implementation, with a benchmark-specific exception for Meta-Harness on DeepSWE (Table 3). Meta-Harness starts from Terminus-KIRA, the parent its proposer prompt names, except on DeepSWE, where it starts from mini-swe-agent. Retro Harness starts from the mini-swe-agent host loop, whose observations its trajectory digest is written against. AHE starts from the NexAU0 workspace required by its upstream implementation. The pinned implementations do not support a common seed without modification. Meta-Harness evolves the agent class and refuses a seed whose agent resolves to installed code, while Retro Harness and AHE evolve files around a fixed runner that *is* installed code. Each seed fixes the entire agent configuration, including model, reasoning effort, command timeouts, and network policy, and is frozen as a content-addressed directory before any search reads it. The selected and seed harnesses differ only in the component the method declares; for Retro Harness, the seed harness mounts the method's initial empty notes behind the same instruction preamble. Candidates that change anything else are rejected before they are run. Because the seed harnesses differ, we never subtract, rank, or average the effects of different method–benchmark evaluations. A summary that spans them counts how many of the eight behave a given way.

Table 3: The eight method–benchmark evaluations and their seed harnesses.

Method × benchmark	Seed harness	Candidate may change
Meta-Harness × TB2.1	Terminus-KIRA	`harness.py`
Meta-Harness × DeepSWE	mini-swe-agent	`harness.py`
Meta-Harness × FrontierCS	Terminus-KIRA	`harness.py`
AHE × TB2.1	NexAU0 workspace snapshot	`workspace/`
Retro Harness × TB2.1	mini-swe-agent + empty notes	`notes/`
Retro Harness × DeepSWE	mini-swe-agent + empty notes	`notes/`
Retro Harness × FrontierCS	mini-swe-agent + empty notes	`notes/`
AHE × DeepSWE	NexAU0 workspace snapshot	`workspace/`

AHE's objective is defined on a binary reward: rollouts are scored $r \in \{0, 1\}$, and the acceptance rule compares pass@1. FrontierCS's fractional outcome therefore lies outside the method's domain, and we mark AHE on FrontierCS not applicable rather than missing. On DeepSWE, AHE's agent

loop runs inside the task container. We mount its prebuilt runtime read-only, so the container installs nothing. The mount is identical for the two harnesses and gives the agent nothing it could use to solve a task.

Matched conditions. Under the primary protocol, both harnesses use the task model `gpt-5.6-terra` at high reasoning effort, the same task images and verifiers, and the same budgets. Every seed manifest states the effort, and a candidate that changes it, including through AHE's workspace, is rejected before any held-out run. The effort is a large effect in its own right: in an earlier run, one method's candidate raised it from inside its own code and captured most of the apparent improvement. High is also the setting Retro Harness's own meta roles are forced to, so the solver is not reasoning less than the judge that grades it. Every budget is set by the task or by the seed harness, never by the method. No step, token, or monetary-cost cap binds in these runs.

Runs and failures. For method m, benchmark b, and held-out task t, let r^{sel}_{mbtj}, $j = 1, \ldots, n^{\mathrm{sel}}_{mbt}$, and r^{seed}_{mbti}, $i = 1, \ldots, n^{\mathrm{seed}}_{mbt}$, denote the rewards of the scored runs under the selected and seed harnesses. The primary protocol specifies $J_b = 8$ scored runs per task and harness. The runs are independent: the two harnesses' runs share no random seeds, run indices are not paired, and no run is reused across harnesses or iterations. The number of attempted runs may differ because external infrastructure failures are rerun. After replacement and failure scoring, $n^{\mathrm{sel}}_{mbt} = n^{\mathrm{seed}}_{mbt} = J_b$ in the primary analysis. The provisional evaluations described in Section 3.1 await completion of this protocol. For each task, the task-level effect is the difference between the two harnesses' means,

$$d_{mbt} = \frac{1}{n^{\mathrm{sel}}_{mbt}} \sum_{j=1}^{n^{\mathrm{sel}}_{mbt}} r^{\mathrm{sel}}_{mbtj} - \frac{1}{n^{\mathrm{seed}}_{mbt}} \sum_{i=1}^{n^{\mathrm{seed}}_{mbt}} r^{\mathrm{seed}}_{mbti},$$

and the benchmark-level effect is its equally weighted stratified average over the held-out task set $\mathcal{T}_b$, one stratum per task:

$$\widehat{\Delta}_{mb} = \frac{1}{|\mathcal{T}_b|} \sum_{t \in \mathcal{T}_b} d_{mbt}.$$

A run that ends without a reward is not automatically a failure of the harness. We therefore classify its cause before scoring it, without reference to any outcome. External infrastructure failures (model-gateway errors; container, image, or network failures; setup steps that time out under host load) are outside the harness under test. We replace those runs until each task and harness has J_b scored runs. Failures the harness can cause or prevent (the agent exhausts its own budget, crashes the shell it works in, or submits nothing) are real outcomes of that harness and score zero. A method that removes them earns the resulting improvement. The environment-related analysis in Section 4.2 concerns these retained, scored outcomes. A verifier timeout can reflect a budget that is tight for that task or an artifact that makes verification hang. We separate the two by rerunning the affected runs rather than by assumption.

Statistical inference. Statistical conclusions about reward differences are based on confidence intervals. We report no p-value, since an interval excluding zero already is the two-sided test at 0.05, and apply no multiplicity correction at either level.

For task-level inference on binary outcomes, we use the unpaired $N - 1$ chi-squared test (Campbell, 2007) with the Miettinen–Nurminen score interval for the difference in proportions (Miettinen & Nurminen, 1985). For continuous outcomes, we use Welch's t-test with Welch–Satterthwaite degrees of freedom and the matching interval. A task-level comparison is a significant improvement or significant regression when its 95% interval excludes zero, and statistically indistinguishable otherwise. At $J_b = 8$ only large shifts are detectable (Section 3.3), so the indistinguishable category covers both negligible effects and effects the available runs cannot resolve; it does not establish equality.

For benchmark-level effects, each held-out task is a stratum with equal weight. The variance of $\widehat{\Delta}_{mb}$ is the sum of the within-task variances of the d_{mbt} divided by $|\mathcal{T}_b|^2$. Each within-task variance is the squared standard error of the task-level comparison: unpooled for binary rewards, Welch for continuous ones, and zero for a task with no variation under either harness. We report the 95%

interval $\widehat{\Delta}_{mb} \pm 1.96\,\mathrm{SE}(\widehat{\Delta}_{mb})$. Equal weights give every held-out task the same influence on the benchmark-level estimate. The interval describes run-to-run variation on the held-out tasks and is conditional on those tasks. A selected harness shows a reliable benchmark-level improvement over *its own* seed harness when the interval's lower endpoint is above zero. We do not correct across the eight method–benchmark evaluations. Each asks whether one method's selected harness improves on its own seed harness, and we report all eight intervals, so a reader can apply a stricter threshold.