HYPOTHESIS-CONDITIONED EVALUATION FOR EFFICIENT HARNESS OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Harness optimization improves an LLM agent by editing its prompts, tools and helper code with model weights fixed. Evaluating every edit on every training task is expensive, but a small random sample can miss the failure that motivated the edit. We introduce *Hypothesis-Conditioned Evaluation* (HCE), which uses the editor's explanation to guide evaluation. Before any candidate rollout, the editor declares an *evaluation contract*: the failure it targets, the tasks it expects to fix or might break, an *activation predicate* specifying where the edit applies, and a runtime event showing that it fired. Conditional prompt rules are inserted only when their predicates hold. The contract's targets and risks are always evaluated; a stratified probability sample fills the remaining budget, including tasks outside the declared scope. An augmented inverse-probability-weighted estimator combines accepted task history with sampled outcomes to compare candidates on the full training-score scale. Retention requires a higher estimated score, observed firing within scope, and improvement on a named target where the edit fired. In two ten-edit comparisons with Agentic Harness Engineering (AHE), using GPT-5.4, HCE used 39.9% fewer scored search-evaluation tokens on Terminal-Bench (30 training, 58 held-out tasks) and 49.3% fewer on Software100 (33 training, 67 held-out). The training-selected harnesses achieved held-out three-attempt mean success rates (mean@3) of 83.33% versus 79.31% on Terminal-Bench, and 78.61% versus 66.17% on Software100. Earlier SWE-bench studies also used fewer evaluations with matching held-out quality. The reported searches establish evaluation savings, but do not establish general quality superiority or isolate the benefit of hypothesis-based task selection.

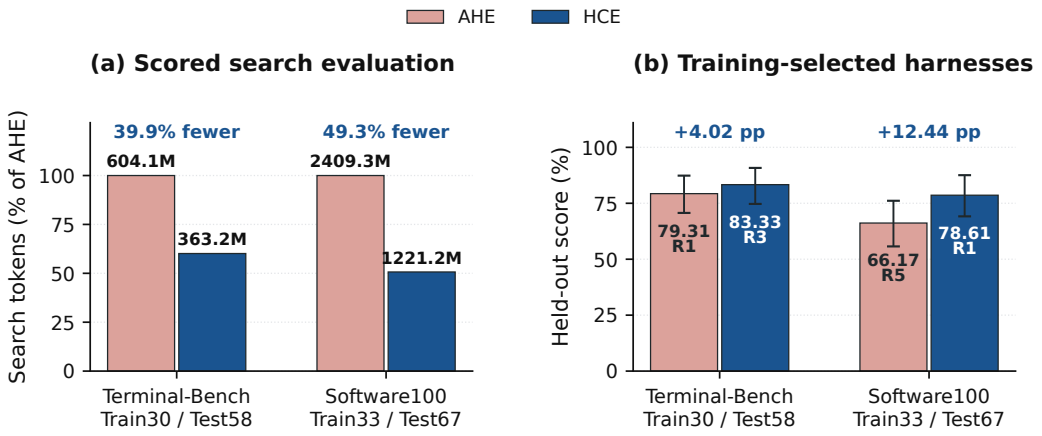

Figure 1: **Search evaluation cost and held-out scores.** (a) Ten-edit evaluation tokens, normalized to AHE; labels give absolute totals. (b) Training-selected harnesses with 95% paired task-bootstrap intervals. Scores use mean@3.

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

1 INTRODUCTION

A coding agent runs a language model in a loop with tools. It reads a task, runs commands in a sandbox and inspects the results. When it stops, an automatic checker determines whether it solved the task. Everything in this loop apart from the model weights is the agent's *harness*: the system prompt, the tool descriptions, helper code that runs between model calls (middleware), and persistent memory. The harness has a large effect on success rates, and recent work improves it automatically (Lin et al., 2026; Lee et al., 2026; Agrawal et al., 2026; Zhang et al., 2026). An *editor* model reads execution traces and proposes changes to harness files; the search keeps a change if it improves performance on the training tasks. Agentic Harness Engineering (AHE) (Lin et al., 2026), for example, runs this loop for ten rounds and raises Terminal-Bench success from 69.7% to 77.0% with the model held fixed.

Evaluating each proposed edit on all training tasks dominates the cost of this loop. A single Terminal-Bench attempt takes minutes of sandbox time and often hundreds of thousands of tokens; a round with 30 tasks and three attempts per task requires 90 runs. A smaller random sample saves runs but can omit the failure that motivated the edit. Neither evaluation strategy alone checks whether the changed prompt or helper ran, so a score increase can lead to retaining an edit that never fired.

In our Terminal-Bench run, the editor proposed the following rule after the agent failed two DNA-related tasks by emitting PCR primers that violated the task constraints:

> **Rule** `primer_contract_check`. **When** the task instruction mentions "primer": reparse the exact primers you emitted and recompute the annealing, orientation and melting checks before finishing.
> **Expected to fix:** `dna-assembly`, `dna-insert`. **Evidence it ran:** the rule is applied in the run.

Hypothesis-Conditioned Evaluation requires such declarations in an *evaluation contract* before any candidate rollout. Conditional prompt rules make the declared scope enforceable: the harness checks the predicate before the agent starts and inserts the rule only for matching tasks. Helper-code edits are observed at runtime and checked against their declared scope.

The contract directs a fixed task budget: named fixes and regression risks are always tested, and the remaining slots are sampled inside and outside the declared scope. Because these subsets differ between edits, an augmented inverse-probability-weighted estimator uses accepted task history and known inclusion probabilities to estimate a common full-training score. Retention requires a score increase along with observed firing, compliance with scope, and improvement on a named target where the edit fired. Section 3 specifies these steps and the conditions under which the score estimate is unbiased.

We evaluate the combined procedure against AHE in two ten-edit comparisons, using GPT-5.4 as both agent and editor. Our AHE baseline follows its optimization loop on shared infrastructure and evaluates every training task (Section 4). On Terminal-Bench, HCE uses 450 rather than 900 scored search rollouts and 39.9% fewer evaluation tokens. On Software100, built from SWE-bench Verified, SWE-bench Pro and DeepSWE, it uses 49.3% fewer tokens. The selected harnesses score 83.33% versus 79.31% and 78.61% versus 66.17% on held-out $\mathrm{mean@3}$, respectively (Figure 1). The checks reject the Terminal-Bench candidate with the highest estimated score because its target does not improve, and a helper that fires outside its declared scope. The analysis uses per-rollout records and frozen selections, with three earlier SWE-bench studies as additional evidence. These comparisons use one search per method on each main benchmark. They show evaluation savings for the combined procedure, without isolating the benefit of task targeting or establishing general quality superiority.

2 RELATED WORK

Automatic agent improvement uses feedback to revise prompts, context or the surrounding harness. TextGrad propagates textual feedback through a computation graph (Yuksekgonul et al., 2024); GEPA evolves prompts by reflecting on execution traces (Agrawal et al., 2026); and ACE maintains an evolving context through generation, reflection and curation (Zhang et al., 2026). SWE-agent

demonstrates the importance of the model–tool interface (Yang et al., 2024). AHE exposes prompts, tools, middleware, skills, sub-agents and memory as editable files, condenses traces, and records the intended effect of each edit in a manifest (Lin et al., 2026). Meta-Harness gives the proposer file-system access to earlier code, scores and traces (Lee et al., 2026), while Evo-Harness organizes experience into transferable patterns (Wei et al., 2026). HCE retains the editor–evaluator loop and uses the declared intent before evaluation: the contract determines the task sample and the evidence required for retention, extending AHE's checks of predictions after evaluation.

Active testing estimates a model's risk by selecting informative test points (Kossen et al., 2021), and tinyBenchmarks estimates benchmark scores from small subsets (Maia Polo et al., 2024); both treat the evaluated system as a black box. Task-CoEvolve adapts the evaluation subset during harness optimization using past outcome variability and corrects subset scores for the sampling design (Miyai et al., 2026). HCE uses the same family of design-based estimators (Horvitz & Thompson, 1952). Its task selection additionally uses the proposed edit's named fixes, risks and activation predicate, and its retention checks require firing and target improvement as well as a higher estimated score.

Regression test selection reruns tests that a change can affect (Rothermel & Harrold, 1997), motivating our use of edit intent. Agent tasks require repeated stochastic runs, and HCE also samples outside the declared scope to check for unintended effects. Algorithm configuration likewise compares a challenger with an incumbent using a limited number of instances (Hutter et al., 2009; López-Ibáñez et al., 2016; Birattari et al., 2002). Where those candidates are parameter vectors, harness edits come with a description of the intended change that can inform evaluation.

3 Hypothesis-Conditioned Evaluation

Objective and score. The search seeks higher scores on a fixed training set $D = \{1, \ldots, N\}$ with less evaluation work per edit. A *rollout* runs the agent with harness H once on task i and returns a verifier outcome $r \in \{0, 1\}$. Each evaluated task receives $K = 3$ rollouts, and its score is the fraction that succeed,

$$x_i(H) = \frac{1}{K} \sum_{j=1}^{K} r_{ij}, \tag{1}$$

which we call $\mathrm{mean@3}$. It estimates the single-attempt success rate; the outcomes $(1, 0, 1)$ contribute $2/3$, not 1. The training score of a harness is the average of x_i over D. The model, the verifier and the execution settings are fixed; only harness files change.

Search state and round order. The initial harness H_0 is evaluated on every training task. The search then maintains a *retained* harness $H^\star$, its recorded score $s^\star$, and per-task results from accepted harnesses. Each round proposes an edit and contract, selects tasks, runs the candidate and records outcomes and firing events, then retains or rolls back the edit (Figure 2). Rejected candidates supply evidence to the editor but do not become the parent of the next edit. Algorithm 1 gives one complete round. The search returns the retained harness.

3.1 The Evaluation Contract

The editor declares the intended effect before running the candidate, so the declaration cannot be fitted to its outcomes. The change manifest lists the edited files. The evaluation contract names the failure, the predicted fixes F and regression risks R, an activation predicate defining the affected task set A, and a runtime observable o. Table 1 shows these fields for the primer example (Section 1) and explains their use. Guaranteed inclusion applies to the contract's sets F and R. Mentioning another target in the manifest does not add it to these sets.

Enforcing prompt scope. Prompt changes are expressed as *conditional rules*: an identifier, a predicate over the task instruction, and text to append to the system prompt. Before the agent starts, the harness evaluates each predicate and appends only the matching rules. For the primer example, the predicate checks whether the instruction contains "primer". An edit may add or remove whole rules, but may not change the base prompt or rewrite an existing rule; a revision receives a new identifier. Where a changed rule's predicate is false, the prompt is unchanged. Each run records the applied rules and a hash of the resulting prompt.

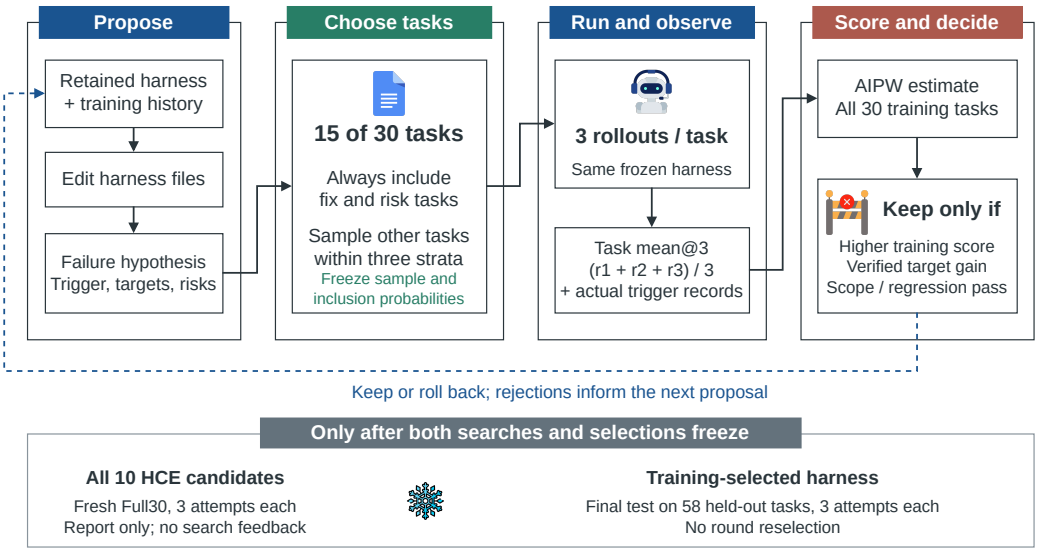

Figure 2: **One round of HCE on Terminal-Bench.** The editor proposes an edit and its contract. The contract fixes a task sample before any outcome is seen: named targets and risks are always included, and the remaining slots are drawn from three strata. The candidate runs three times per selected task, and the runs record both outcomes and which rules or helpers fired. The candidate is kept only if its estimated training score rises *and* its firing, scope, target and regression checks pass. Held-out testing uses only the harness selected at the end of search.

Field	Example (Terminal-Bench, round 1)	Used for
Changed files	`systemprompt.md`	Must equal the actual file diff
Failure hypothesis	Agent checked intermediate candidates, not the primers it emitted	Recorded for the editor and for audit
Activation predicate A	Instruction contains the term "primer"	Guard of the conditional rule; defines the strata; scope check
Predicted fixes F	`dna-assembly`, `dna-insert`	Always evaluated; target-gain check
Regression risks R	(none)	Always evaluated
Observable o	Rule `primer_contract_check` applied	Checks that the edit fired, and only inside A
Scope	Local	A global edit treats every task as affected

Table 1: **The evaluation contract.** The fields support validation, sampling, retention and audit. A task whose activation status cannot be determined is treated as affected.

Helpers (middleware) may also be edited; their scope is checked at runtime. They are loaded for every task and are intended to act only when their declared trigger occurs. An observer wraps the middleware manager and logs each action. For both prompt and helper edits, the logs identify the tasks on which the declared observable occurred.

Validation. Before any rollout, HCE checks that the manifest matches the file diff, predicates use only available task features, and changed files are editable harness components. A candidate that fails is rejected without being run.

3.2 Choosing Tasks

We include every named target in F and risk in R with certainty. The remaining tasks are partitioned by activation status and their most recent accepted score into three strata:

- affected tasks with a latest accepted score below 1, to test for improvement;

- affected tasks with a latest accepted score of 1, to check for regression;
- tasks outside the activation set, to audit for unintended effects.

The remaining budget follows weights 0.5, 0.3 and 0.2 in that order, with slots redistributed when a stratum reaches capacity. Each nonempty residual stratum receives a positive allocation; infeasible plans are rejected (Appendix A). We sample uniformly without replacement within each stratum. Named tasks have inclusion probability $\pi_i = 1$; other tasks have the stratum's sampling fraction, also positive. The budgets are 15 of 30 tasks on Terminal-Bench and 17 of 33 on Software100, with three rollouts per selected task. The sample and its probabilities are fixed before candidate outcomes are observed.

3.3 Scoring Candidates on a Common Scale

An edit with difficult targets can have a lower subset mean than one tested on easier tasks even when its full-training score is higher. We therefore estimate the mean over the same set D for every candidate. Let m_i be the mean of task i's accepted past results, fixed before sampling, and let S be the evaluated subset. The estimate is

$$\widehat{\mu} = \frac{1}{N}\left[\sum_{i \in D} m_i + \sum_{i \in S} \frac{x_i - m_i}{\pi_i}\right]. \tag{2}$$

The first term predicts each task's score from its history. The second corrects those predictions using observed differences $x_i - m_i$, weighted by inverse inclusion probability. This is the augmented inverse-probability-weighted (AIPW), or difference, estimator from survey sampling (Horvitz & Thompson, 1952). Since $\mathbb{E}\big[\mathbf{1}\{i \in S\}/\pi_i\big] = 1$, the historical terms cancel in expectation over the sample. The estimate is therefore unbiased for the full training score of the current outcomes whatever the accuracy of the history, while an accurate history lowers its variance (Appendix B).

3.4 Retain or Roll Back

Retention requires a higher score estimate and observations consistent with the declared intervention. HCE applies five checks in order to the sampled outcomes and firing logs:

(i) **Limit regressions.** At most one sampled task with a score of 1 in every accepted historical evaluation drops to zero successes.

(ii) **The edit ran.** The declared observable occurred. An edit that never fired is rejected even if the score rose.

(iii) **The edit stayed in scope.** It did not fire on sampled tasks outside its activation predicate.

(iv) **The target improved.** At least one contract target has a higher three-rollout mean than its latest accepted mean, and the edit fired in at least one of that task's current rollouts.

(v) **The score rose.** The estimate $\widehat{\mu}$ strictly exceeds the retained score $s^\star$.

Checks (ii)–(iv) require evidence about the declared mechanism; they do not establish that the edit caused the target improvement. On acceptance, the candidate and its estimate replace $H^\star$ and $s^\star$, and its sampled task results join the accepted history. On rejection, the harness files are restored and the editor receives the evidence and failed check. The next round edits the retained harness, so the final selection can differ from the maximum of the plotted score estimates.

Algorithm 1 One round of HCE

Require: retained harness $H^\star$ and score $s^\star$; per-task history m_i; task budget n; strata weights (w_1, w_2, w_3)

1: $(H', \text{manifest}, C) \leftarrow \textsc{Editor}(H^\star, \text{traces, past evidence})$ ▷ $C = (A, F, R, o)$
2: **if** manifest $\neq \textsc{Diff}(H^\star, H')$ **or** C is invalid **then**
3: reject
4: $S \leftarrow F \cup R$; $\pi_i \leftarrow 1$ for $i \in S$ ▷ named targets and risks
5: Split $D \setminus S$ into $U_1 = \{i \in A : \text{latest accepted score} < 1\}$, $U_2 = \{i \in A : \text{latest accepted score} = 1\}$, $U_3 = D \setminus (A \cup S)$
6: Split the remaining $n - |S|$ slots across U_1, U_2, U_3 in proportion to w, moving slots when a stratum runs out
7: Draw n_h tasks uniformly without replacement from each U_h; add them to S with $\pi_i \leftarrow n_h/|U_h|$
8: Run H' three times on every $i \in S$; record outcomes and firing events
9: $x_i \leftarrow \text{mean@3}$ for $i \in S$; $\widehat{\mu} \leftarrow$ Eq. (2)
10: **if** (i) at most one always-passing task drops to 0, (ii) the edit fired, (iii) it fired only inside A,
11: (iv) some $i \in F \cap S$ improved over its last accepted result and the edit fired there, **and** (v) $\widehat{\mu} > s^\star$ **then**
12: $H^\star \leftarrow H'$; $s^\star \leftarrow \widehat{\mu}$; add $\{x_i\}_{i \in S}$ to the history
13: **else**
14: restore $H^\star$; keep the candidate's evidence for the editor

4 Experimental Setup

Benchmarks and splits. *Terminal-Bench*: a fixed Terminal-Bench 2.1 collection (Merrill et al., 2026) split into 30 training and 58 held-out tasks. *Software100*: 100 software-engineering tasks, 50 from SWE-bench Verified, 30 from SWE-bench Pro and 20 from DeepSWE (Jimenez et al., 2024; OpenAI, 2024; Deng et al., 2025; Datacurve, 2026), split into 33 training and 67 held-out tasks. Both splits were frozen before the runs reported here, and held-out tasks never enter search or selection.

Methods and fixed settings. AHE and HCE share the editor, initial NexAU harness and initial evaluation (R0). Each makes ten edits (R1–R10), with one search per method on each benchmark. Both the agent and editor use GPT-5.4, with high reasoning effort for the agent and maximal effort for the editor. Our AHE baseline follows the optimization loop of Agentic Harness Engineering (Lin et al., 2026) on this shared infrastructure; it is not a reproduction of the published AHE system. It evaluates every candidate on all training tasks, three rollouts per task, and retains a candidate only when its training mean@3 strictly exceeds the recorded retained score. HCE evaluates 15 of 30 Terminal-Bench or 17 of 33 Software100 tasks per candidate, three rollouts each, and uses the retention conditions in Algorithm 1.

Additional baselines. The main tables include supplied scores and training-token totals for the seed harness, ReasoningBank, ACE, Meta-Harness, Self-Harness and HarnessLens. Per-task analyses and audited evaluation costs concern the AHE and HCE runs above.

Search cost and held-out quality. Search cost counts scored evaluation rollouts and their tokens; R0, editor calls, diagnostics and held-out testing are counted separately. The *online* training score, available during search, is AHE's full-training average or HCE's estimate in Eq. (2). Quality is the selected harness's held-out mean@3, measured with three rollouts per task after both searches and selections are frozen.

Post-search training measurements. Every HCE candidate also receives a full-training diagnostic after selection. On Terminal-Bench, "Full30" reruns all 30 training tasks with three fresh rollouts each. On Software100, "Full33" combines the 17 sampled tasks' existing results with three new rollouts on each of the 16 unsampled tasks. Full30 is therefore fresh, while Full33 completes the original panel. Neither diagnostic affects selection.

Table 2: **Terminal-Bench 2.1.**
30 training and 58 held-out tasks.

Method	Test score (%) ↑	Training tokens (M) ↓
Baseline		
Seed Harness	77.59	
Self-evolving methods		
ReasoningBank	77.01	160.9
ACE	78.16	158.2
Harness optimization		
AHE	<u>79.31</u>	604.1
Meta-Harness	78.16	515.5
Self-Harness	78.57	563.0
HarnessLens	77.59	150.3
HCE	**83.33**	363.2

Table 3: **Software100.**
33 training and 67 held-out tasks.

Method	Test score (%) ↑	Training tokens (M) ↓
Baseline		
Seed Harness	70.15	
Self-evolving methods		
ReasoningBank	70.65	652.4
ACE	71.64	671.6
Harness optimization		
AHE	66.17	2409.3
Meta-Harness	70.15	3053.2
Self-Harness	<u>72.14</u>	2353.7
HarnessLens	70.15	735.6
HCE	**78.61**	1221.2

Both tables report $\mathrm{mean@3}$ scores. AHE/HCE training tokens count scored evaluations in R1–R10, excluding initialization, editing, diagnostics and testing. The matched cost comparison and percentage savings concern AHE/HCE only; other baselines' supplied token totals are outside that comparison. Blank: no optimization stage.

Uncertainty. We report the 2.5th and 97.5th percentiles of 20,000 paired task-bootstrap samples, resampling task IDs with replacement jointly for both methods. Each sample contains 58 Terminal-Bench or 67 Software100 tasks; the Software100 seed is 20260924. Each task's three-rollout mean stays fixed. The intervals describe variation across tasks, not fresh rollouts or independent searches.

Earlier software studies. We also report three completed studies that used an earlier version of HCE: a 16-task SWE-bench Verified pilot, a 45-task SWE-bench Verified panel, and a 100-task mixed panel. Each has three end-to-end repeats with a shared first edit, one further edit per method, and one held-out attempt per task. Appendix E gives their protocols.

5 Results

Table 2 and Table 3 compare held-out $\mathrm{mean@3}$ scores and training-token costs on Terminal-Bench and Software100.

5.1 Search Uses Fewer Evaluations

On Terminal-Bench, HCE used 450 scored rollouts versus 900 for AHE over ten edits, following its 15-of-30 task budget. Tokens fell by 39.9%; task costs explain the different rollout and token savings. On Software100, HCE used 510 versus 990 rollouts and 49.3% fewer tokens. These counts exclude R0, editing, post-search diagnostics and held-out testing. Incomplete debugger usage records prevent an end-to-end token, dollar or wall-clock comparison. Appendix C gives rollout counts and selected rounds (Table 4) and plots candidate quality against search tokens.

5.2 Quality of the Training-Selected Harnesses

Terminal-Bench. AHE retained R1 and HCE retained R3. On the 58 held-out tasks they scored 79.31% and 83.33%. Per task, HCE's harness was higher on 12 tasks, lower on 6 and equal on 40. The difference of 4.02 points has a paired task-bootstrap interval of $[-2.87, 10.34]$, which includes zero. With one search per method, the comparison does not establish that either method generally finds better harnesses.

Software100. AHE retained R5, which scored 73.74% on the training tasks, and HCE retained R1. On the 67 held-out tasks, AHE scored 66.17% (133/201), and HCE scored 78.61% (158/201), using

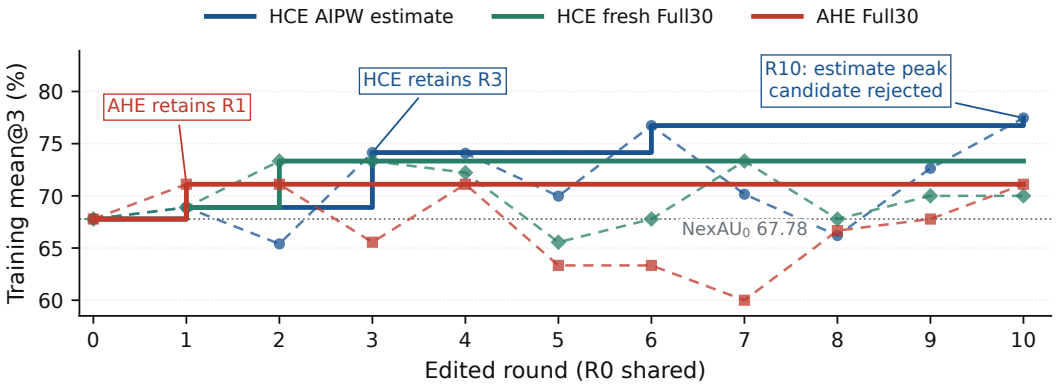

Figure 3: **Terminal-Bench training scores by round.** Dashed lines show per-round scores; solid steps show best-so-far scores, including rejected candidates. Curves show HCE's online estimate, its full training re-evaluation ("Full30"), and AHE's training score. R10's online estimate is highest, but its target did not improve; the selected harness is R3. R0 is shared.

Edited round	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**Total**
Affected, not fully passing	2	1	2	7	2	1	1	1	3	1	**21**
Affected, currently passing	0	0	0	2	0	0	1	1	0	0	**4**
Random audit	13	14	13	6	13	14	13	13	12	14	**125**
Decision	**Keep**	Score	**Keep**	Score	Score	Target	Scope	Target	Target	Target	

2 retained · 3 no score gain · 4 no target gain · 1 scope violation

Figure 4: **Task allocation and decisions by round on Terminal-Bench.** Each round selects 15 tasks. Columns align stratum counts with the candidate's decision: Keep denotes retention; Score, Target and Scope identify the failed check.

mean@3 (Table 3). The 95% task-bootstrap interval for HCE is $[69.15\%, 87.56\%]$. The difference from AHE is 12.44 percentage points, with a 95% interval of $[6.47, 18.41]$.

In the post-search Full33 diagnostic, HCE's highest score was 81.82% at R6 (Figure 6 in Appendix C). That result is for reporting only and does not replace the selected harness.

5.3 Task Allocation and Retention

All ten Terminal-Bench contracts were local, and all 13 primary contract target declarations were evaluated. Small declared scopes left most selected task-rounds in the audit stratum: 21 of 150 were affected tasks with a latest accepted score below 1, 4 were affected tasks with a score of 1, and 125 were outside scope (Figure 4). Of these task-rounds, 18 were contract targets or risks included with certainty.

Five of the ten edits were conditional prompt rules, applying to 2, 1, 2, 9 and 1 of the 30 training tasks according to their instruction predicates. Every observed application matched the predicates across these edits' 225 task batches of search and re-evaluation. R7's helper for checkpointing final artifacts acted on three tasks outside its declared predicate and was rejected.

Of the ten candidates, two were retained, three failed the score condition, four failed the target check and one failed the scope check. R6 and R10 had online estimates of 76.7% and 77.5%, above the retained R3's 74.2%, but neither improved its named target (Figure 3). R10 had the highest estimate in the search. The target check rejected both candidates, which would have been

accepted on score alone at those rounds. Their later Full30 scores were 67.8% and 70.0%, below R3's 73.3%. Appendix D gives every edit, target and decision, including comparisons between search observations and fresh evaluation.

5.4 Earlier SWE-bench Results

On the 45-task SWE-bench Verified panel, HCE reduced post-edit development rollouts from 180 to 72 (60.0%) and tokens from 38.03 to 24.53 million (35.5%), and passed 39/45 held-out task–seed pairs against 37/45 for full evaluation. The 16-task pilot ties at 24/24 and the 100-task mixed panel at 85/150. These studies used earlier protocols, and two of their repeats fell back to full evaluation. Appendix E gives per-seed costs, fallbacks and repairs. These earlier studies used shorter searches and single-attempt held-out scores; the main comparisons use ten edits and $\mathrm{mean@3}$.

6 Limitations

The results describe one search per method on each main benchmark, with small task splits. Their held-out differences do not estimate an average over independent searches. Terminal-Bench held-out tasks were isolated from the reported search, although earlier method development used the full task collection. The earlier SWE-bench panels are also small and were chosen by cached-image availability.

The retention conditions are operational rules rather than statistical tests. An accepted gain can partly reflect noise, and target improvement is measured against an earlier accepted observation rather than a concurrent evaluation of the parent. Firing and scope evidence therefore support the checks without proving that the edit caused an improvement.

The comparisons test the contract, sampling design and retention checks together, without isolating each part's contribution. A random-subset baseline with the same task budget and ablations of the individual checks would be needed to separate those contributions.

7 Conclusion

HCE uses an editor's declared fixes, risks and activation predicate to select tasks and check retention. Conditional prompt rules enforce their scope; sampling audits the remaining tasks, and an AIPW estimate places candidates on a common training-score scale. In two ten-edit comparisons with AHE, the combined procedure reduced scored search-evaluation tokens by 40–49%. Held-out results describe the training-selected harnesses, and three earlier studies also reduced post-edit evaluations.

AI USE STATEMENT

Generative AI tools assisted with software implementation and code editing, literature search and reference checking, interpretation of reported results, figure preparation, translation, and manuscript drafting and revision. The experiments also use language models as agents and editors, as described in Section 4. The authors take responsibility for the final text, claims, code and other research artifacts.

ETHICS STATEMENT

This work studies the evaluation cost and behavior of coding-agent harness optimization on terminal and software-engineering benchmarks. More efficient optimization may make coding agents easier to develop and deploy, while generated changes can still introduce errors or unintended behavior. The retention checks address declared edit mechanisms and benchmark outcomes; they do not establish safety outside the evaluated tasks.

REPRODUCIBILITY STATEMENT

Algorithm 1 provides pseudocode for one round of HCE. Section 3 details the evaluation contract, task sampling, score estimation and retention checks. Section 4 specifies the model settings, task splits, rollout budgets and evaluation protocol. Additional sampling and retention details are given in Appendix A; Appendix C reports round scores and token accounting, and Appendix D documents the attempted edits. We will release the code publicly.

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, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 11–18, 2002. URL https://iridia.ulb.ac.be/~stuetzle/publications/GECCO02.pdf.

Datacurve. DeepSWE benchmark, version 1.1. Dataset, 2026. URL https://huggingface.co/datasets/datacurve/deep-swe. Dataset card accessed September 17, 2026.

Xiang Deng, Jeff Da, Edwin Pan, et al. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025. URL https://arxiv.org/abs/2509.16941.

D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952. doi: 10.1080/01621459.1952.10483446.

Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009. doi: 10.1613/jair.2861.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

Jannik Kossen, Sebastian Farquhar, Yarin Gal, and Tom Rainforth. Active testing: Sample-efficient model evaluation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5753–5763, 2021. URL https://proceedings.mlr.press/v139/kossen21a.html.

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

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

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. doi: 10.1016/j.orp.2016.09.002.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinyBenchmarks: Evaluating LLMs with fewer examples. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 34303–34326, 2024. URL https://proceedings.mlr.press/v235/maia-polo24a.html.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026. URL https://arxiv.org/abs/2601.11868.

Atsuyuki Miyai, Kiyoharu Aizawa, and Toshihiko Yamasaki. Task-CoEvolve: Efficient harness optimization via adaptive validation task selection. *arXiv preprint arXiv:2608.20169*, 2026. URL https://arxiv.org/abs/2608.20169.

OpenAI. Introducing SWE-bench Verified. Technical report, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Gregg Rothermel and Mary Jean Harrold. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, 6(2):173–210, 1997. doi: 10.1145/248233.248262.

Tianxin Wei, Zhan Shi, Minhua Lin, Bing He, Zewen Liu, Yisi Sang, Yuanchen Bei, Xuying Ning, Jiaru Zou, Ting-Wei Li, Xiao Lin, Yanjun Zhao, Chi Wang, Benoit Dumoulin, Dakuo Wang, Jingrui He, and Hanqing Lu. Evo-Harness: Context-to-harness skill compilation for self-evolving agents. *arXiv preprint arXiv:2608.15071*, 2026. URL https://arxiv.org/abs/2608.15071.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R. Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. URL https://arxiv.org/abs/2406.07496.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.04618.

A Sampling and Retention Details

Before evaluation, the change manifest is checked against the Git diff and the contract against known training-task IDs and editable components. The solver model, reasoning configuration and verifier remain fixed. A sampling plan must include every named fix and risk, assign three rollouts to every selected task, and give every training task positive inclusion probability. Otherwise it is rejected.

Named tasks consume slots before the remaining budget is divided among the three strata. Every nonempty residual stratum receives a positive integer quota, following the improvement, regression and audit weights with redistribution at stratum capacity. A named task's inclusion probability is one; other tasks receive the residual sampling fraction of their stratum. Each scored Terminal-Bench candidate uses 15 tasks and 45 rollouts; each Software100 candidate uses 17 tasks and 51 rollouts. Early stopping is not used.

Retention first checks for regressions on historically stable passes, then for observed firing, consistency with the declared scope, improvement on a named target, and a strict increase over the recorded retained score. The gain floor is zero task-equivalents, with numerical tolerance 10^{-9}. The regression check uses all accepted history: at most one task that previously scored 1 every time may drop to zero successes. Sampling strata instead use the latest accepted score. Target improvement compares three-rollout means and requires observed firing in at least one current rollout of that target. These are operational choices, not significance thresholds.

For Terminal-Bench, the common R0 evaluation has 61 successes in 90 attempts: $61/(30 \times 3) = 67.78\%$. The final retained HCE candidate is R3, with commit `2bfc94c762aec9adfb4742e6a4b8d00845f3ce96`. The selected AHE candidate is R1, with commit `979f0a3254b8b629f4929fba500c8d0d2af3b493`. Final-test receipts identify these candidates and their frozen selection records. Later diagnostic maxima do not change the selected harnesses.

B What the Estimator Guarantees

The estimate targets the mean over all training tasks for the current candidate. At round t, condition on the candidate, historical predictions m_{ti}, strata, probabilities and all other information available before sampling. Treat the potential three-attempt means x_{ti} as a fixed vector over the N tasks. With sample-membership indicator $I_{ti} = \mathbf{1}\{i \in S_t\}$ and $\pi_{ti} > 0$,

$$\mathbb{E}_{S_t}[\widehat{\mu}_t] = \frac{1}{N}\sum_{i \in D}\left(m_{ti} + \frac{\mathbb{E}_{S_t}[I_{ti}]}{\pi_{ti}}(x_{ti} - m_{ti})\right) \tag{3}$$

$$= \frac{1}{N}\sum_{i \in D} x_{ti}. \tag{4}$$

Because $\mathbb{E}_{S_t}[I_{ti}] = \pi_{ti}$, each historical prediction cancels in expectation. Correct inclusion probabilities and predictions fixed before outcomes are required; accurate predictions are not. Extending the expectation to fresh rollout randomness additionally assumes that a task's execution distribution is unchanged by its inclusion or by which other tasks are sampled.

This identity is conditional on a candidate and sampling design. It gives no error bound for a single round, does not remove the selection bias of a running maximum, and does not correct a noisy recorded retained score. It also does not establish that the edit caused a target to improve. The reported estimate is unclipped, since clipping would generally change its expectation. Full30 evaluates all tasks with a fresh batch of attempts; it does not reveal the missing entries of the fixed vector in this proof.

C Round Scores and Budget

The cost comparison counts scored search evaluations. On Terminal-Bench, AHE uses 604,118,620 evaluation tokens and HCE uses 363,195,393. Recorded editor totals are 32,409,902 and 26,207,997 tokens, respectively; debugger records contain only partial totals. Superseded attempts that were never scored are excluded, while scored failures and timeouts are included. These totals are not

a complete experiment bill. Table 4 summarizes search budgets and frozen selections for both benchmarks.

		Search evaluation				
Benchmark	Method	Rollouts	Tokens (M)	Edits kept	Selected	Held-out score
Terminal-Bench	AHE	900	604.1	1	R1	79.31% (138/174)
(30 / 58)	HCE	450	363.2	2	R3	83.33% (145/174)
Software100	AHE	990	2409.3	1	R5	66.17% (133/201)
(33 / 67)	HCE	510	1221.2	1	R1	78.61% (158/201)

Table 4: **Search and selection details.** One ten-edit search per method and benchmark; training/held-out task counts in parentheses. Costs count scored search evaluations only. "Selected" is the retained harness at the end of search. Held-out scores use $\mathrm{mean@3}$. Paired task-bootstrap 95% intervals for the difference (HCE minus AHE) are $+4.02$ points $[-2.87, 10.34]$ on Terminal-Bench and $+12.44$ points $[6.47, 18.41]$ on Software100.

Table 5 separates three Terminal-Bench measurements: HCE's raw subset mean, its AIPW estimate, and its fresh Full30 diagnostic. Only the AIPW estimate enters HCE's score check. AHE has one online evaluation of all training tasks per candidate and no independent per-round re-evaluation. The best-so-far curves in Figure 3 take maxima of the corresponding columns, including rejected candidates; the retention column records the actual selections.

Round	AHE Full30	Keep	HCE subset	HCE AIPW	HCE Full30	Keep
0	67.78	yes	67.78	67.78	67.78	yes
1	71.11	yes	55.56	68.89	68.89	yes
2	71.11	no	66.67	65.40	73.33	no
3	65.56	no	71.11	74.15	73.33	yes
4	71.11	no	60.00	74.07	72.22	no
5	63.33	no	71.11	69.97	65.56	no
6	63.33	no	80.00	76.73	67.78	no
7	60.00	no	68.89	70.16	73.33	no
8	66.67	no	71.11	66.20	67.78	no
9	67.78	no	64.44	72.64	70.00	no
10	71.11	no	75.56	77.46	70.00	no

Table 5: **Training scores in percent.** HCE subset is the unweighted mean of the 15 selected task means. HCE AIPW is the estimate used by the gate. HCE Full30 is the fresh post-search diagnostic. "Keep" records actual retention; R0 initializes both arms.

Across edited Terminal-Bench rounds, the mean absolute gap between the online estimate and Full30 is 3.88 points, with root mean square error 4.92. R10, for example, scores 70.00% in Full30 against an online estimate of 77.46%. The gap combines task-sampling variation with variation between rollout batches, so it does not isolate estimator bias.

Figure 5 plots Terminal-Bench scores against cumulative scored search tokens. Full30 scores appear at each candidate's search budget, although they were measured later. Figure 6 instead uses edit number to show Software100's online and completed full-training scores. Section 4 defines both diagnostics.

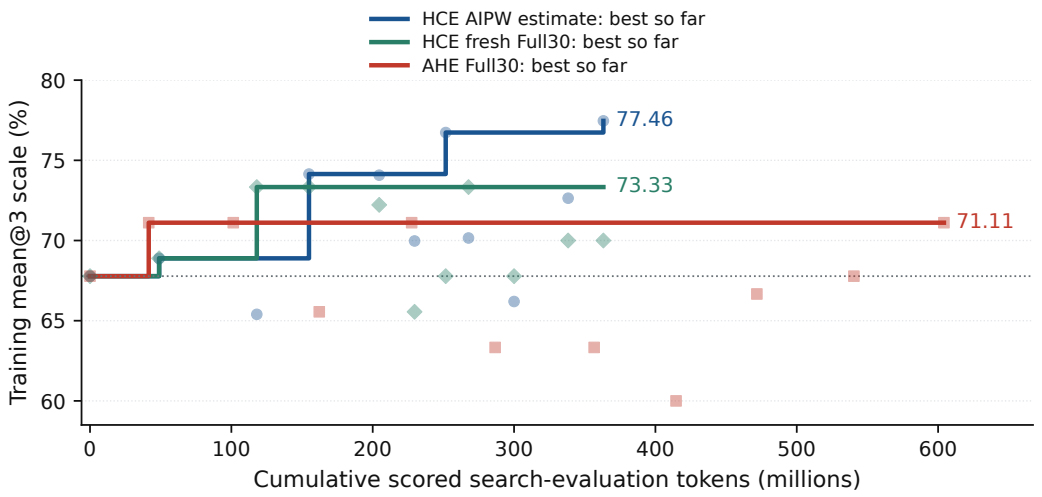

Figure 5: **Search progress against scored evaluation tokens.** Faint points are individual candidates and solid steps show best-so-far scores. One point is a 30-task, three-attempt AHE evaluation, or a 15-task, three-attempt HCE evaluation. The green HCE Full30 score is measured after search and placed at its candidate's search budget for comparison; its 900 extra diagnostic rollouts are *not* included on the horizontal axis. The plot is not a total-compute comparison. The common R0 evaluation is shown at zero additional search tokens.

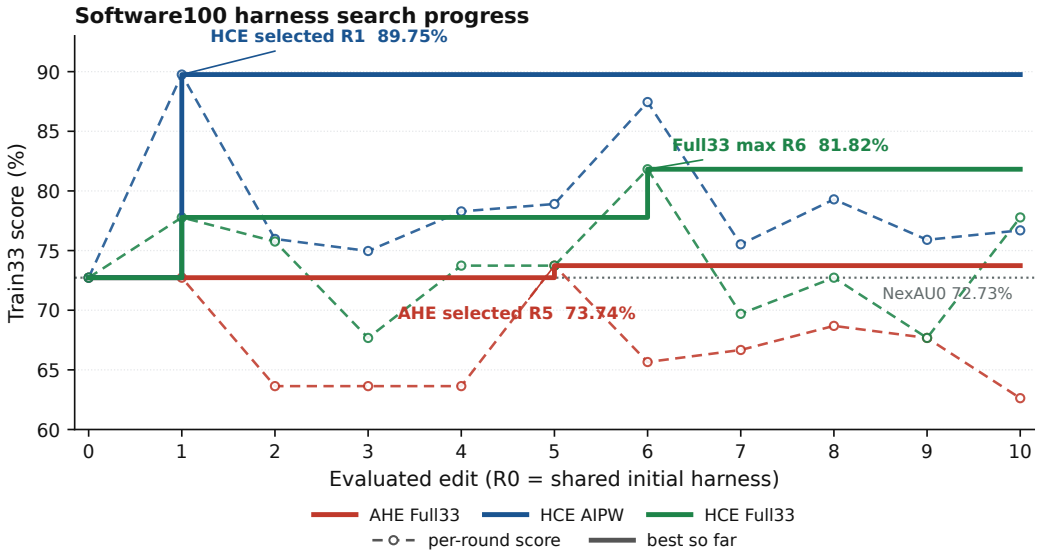

Figure 6: **Software100 candidates and three reporting maxima.** Dashed lines show each round's score; solid steps show each curve's best-so-far score, including rejected candidates. AHE Full33 evaluates all 33 frozen training tasks during search. HCE Full33 combines each round's 17 sampled tasks with later results on the 16 unsampled tasks; both report full-training mean@3. HCE AIPW estimates that score from the sampled panel. The training-selected harnesses are AHE R5 and HCE R1. The later HCE R6 Full33 maximum is a post-search diagnostic and does not change the selection.

D WHICH CHANGES WERE ATTEMPTED?

Table 6 summarizes the proposed Terminal-Bench fixes. Manifests record the editor's diagnosis and intent; they do not prove either. Every candidate from R4 through R10 edits the retained R3 because none of those later candidates is accepted. Per-task measurements are available in Figure 7 and `data/task_changes.csv`.

R	Declared targets	Intended change	Decision
1	`dna-assembly` `dna-insert`	Add a conditional prompt to re-parse and validate the emitted primers.	Retained
2	`fix-code-vulnerability`	Add a conditional prompt for minimal patches and compatibility checks.	No score gain
3	`configure-git-webserver` `make-doom-for-mips`	Add separate conditional curl and stdout validation prompts.	Retained
4	`gcode-to-text` `query-optimize` `train-fasttext`	Add conditional saved/write artifact prompts.	No score gain
5	`dna-assembly`	Add a runtime reminder after suspicious primer-check output.	No score gain
6	`query-optimize`	Add a conditional prompt for early reduction and final-query timing.	No target gain
7	`gcode-to-text`* `query-optimize`	Add middleware that reminds the agent to checkpoint final artifacts.	Scope violation
8	`fix-code-vulnerability`	Add a CWE-specific compatibility reminder after pytest succeeds.	No target gain
9	`query-optimize` `train-fasttext`	Narrow artifact-checkpoint middleware to explicit saved-file instructions.	No target gain
10	`query-optimize`	Add initial SQL timing guidance and a runtime plan-shape reminder.	No target gain

Table 6: **HCE's ten evaluated edits.** Target names include both primary contract targets and additional manifest targets. An asterisk marks the R7 manifest-only target that was not sampled. "No score gain" and "No target gain" refer to different retention conditions.

Relative to NexAU_0, the selected HCE harness changes only `systemprompt.md`. It contains the conditional rules `primer_contract_check`, `stdout_contract_check`, and `curl_runtime_contract_check`. The selected AHE harness changes `systemprompt.md`, `LongTermMEMORY.md`, and `tool_descriptions/run_shell_command.tool.yaml`. All nine later AHE candidates are rejected, including proposals revisiting background-job handling, final files and primer validation.

R7's checkpoint middleware activates on `code-from-image`, `sparql-university` and `vulnerable-secret`, outside its stated condition. R9 narrows that idea to instructions requiring saved files. R10 adds SQL timing and plan-shape guidance, but its primary target stays at 2/3 in online evaluation. The selected harness inherits none of the rejected middleware.

Figure 7 compares the evidence available during search with fresh Full30 results. R5's `dna-assembly` score rises from a historical 0/3 to 1/3 in its search sample, then is 0/3 in the later Full30 batch. Both task scores for R5 use three executions. Across the 13 primary target declarations, fresh comparisons with the retained parent improve in four cases, tie in four and decline in five. The seven HCE rejections after R3 indicate a plateau under this proposal and retention policy, not a proven local optimum.

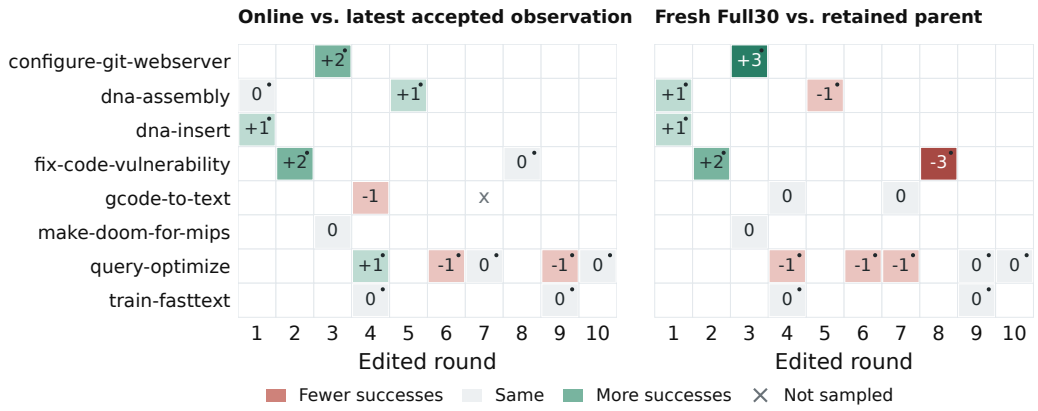

Figure 7: **Declared fixes do not always repeat in fresh evaluation.** Each filled cell is a declared target. Numbers are changes in successful attempts out of three; a dot marks a primary contract target. Left: current search outcomes versus the latest accepted observation. Right: fresh Full30 outcomes versus Full30 outcomes of the retained parent, with R0 supplying the initial reference. Blank means not declared; x means not sampled, as for R7's `gcode-to-text` target.

Figure 8 shows all training tasks. HCE R3 has six zero-success tasks in fresh evaluation: `gcode-to-text`, `make-doom-for-mips`, `sam-cell-seg`, `torch-pipeline-parallelism`, `train-fasttext`, and `video-processing`. Only `torch-pipeline-parallelism` has zero successes in every HCE Full30 round. Improvements on undeclared targets are descriptive observations, not attributed to a particular rule.

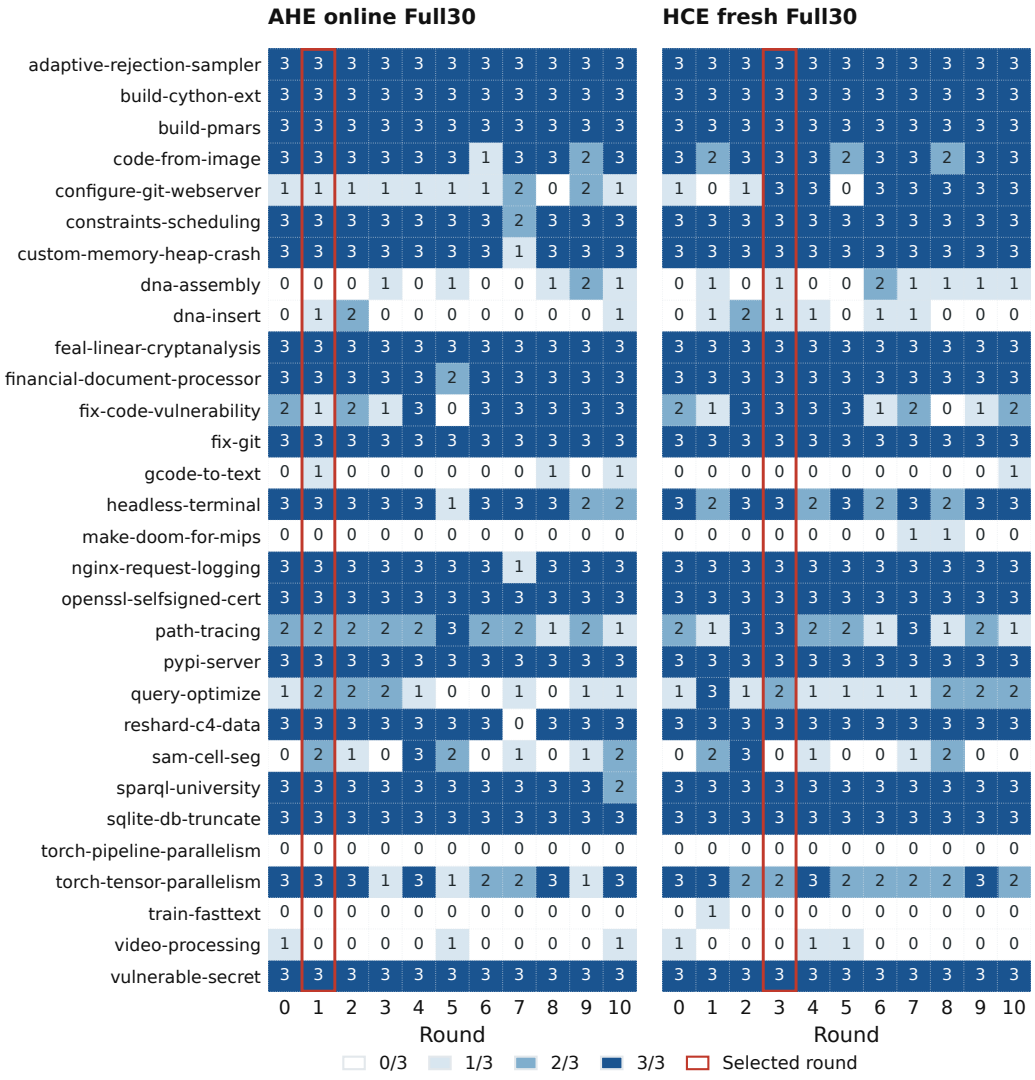

Figure 8: **Every training task, R0–R10.** Each cell is the number of successful attempts out of three. The left panel uses AHE's online evaluations of all training tasks. The right uses fresh post-search HCE evaluations, except the shared R0. Outlined columns identify the training-selected harnesses. No test58 outcome appears in this matrix.

E HISTORICAL SOFTWARE EVIDENCE

The three earlier studies compare an earlier version of HCE with full evaluation. Each has three end-to-end repeats, a shared first edit and one further edit per method. Held-out evaluation uses one attempt per task. These protocols differ from the main ten-edit AHE comparisons, so the results and costs are reported separately.

The 45-task panel covers the cached-image inventory available at registration. It is not a random sample of SWE-bench Verified. Development repositories are matplotlib, requests, pylint and sympy; held-out repositories are astropy, django, seaborn, flask, xarray, pytest, scikit-learn and sphinx. The 16-task pilot uses smaller panels from the same repository groups. The mixed panel's source quotas were chosen using aggregate calibration rates, with calibration tasks excluded from the final panels.

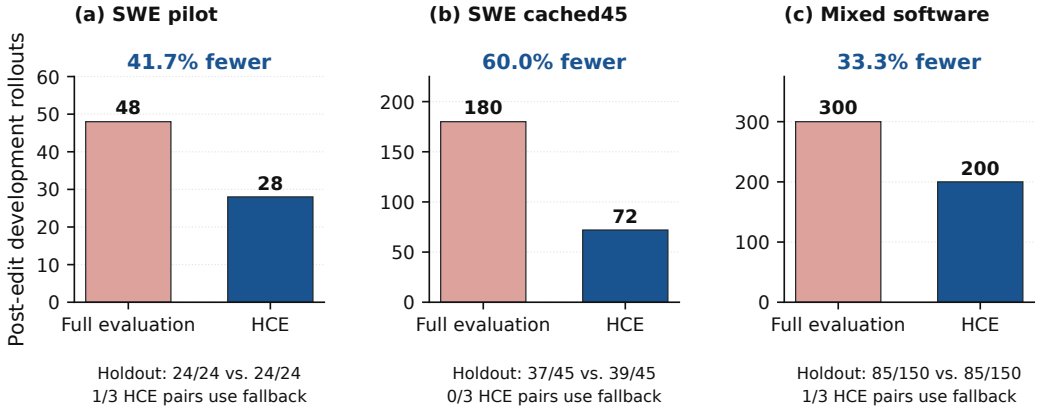

Figure 9: **Earlier software studies, with their original protocols.** Bars count post-edit development rollouts across three end-to-end repeats. Exact holdout pass counts and fallback counts appear below each pair. Holdout evaluation uses one attempt per task, not mean@3. The mixed panel includes SWE-bench Verified, SWE-bench Pro, and DeepSWE; it is not a Verified-only benchmark result.

Figure 9 summarizes all three studies, and Table 7 reports individual seeds, including both fallbacks to full evaluation. In the 45-task panel, seed 2 uses fewer rollouts but 12.2% more tokens (Figure 10).

Across the mixed panel's repeats that used subset evaluation, full evaluation passes 57/100 held-out task–seed pairs and HCE passes 58/100. Six pairs favor HCE, five favor full evaluation and 89 tie. Including the fallback seed, as in the main report, gives 85/150 for each method. Neither comparison supports a quality-superiority claim.

Fallbacks and repairs. The pilot's seed 2 falls back to full evaluation because eight named tasks exceed a four-task budget. For seeds 0 and 1, four historical artifact references were resolved to immutable iteration archives with matching hashes after mutable root paths had been overwritten. This repairs the provenance links and leaves task outcomes unchanged.

The mixed panel's seed 2 falls back because its change manifest does not match the Git diff. In seed 1, a middleware file was repaired before final evaluation. The receipt records zero model generations and zero verifier rewards before the repair. We report the repaired candidate, which was fixed without observing its outcomes.

Panel / seed	Dev. rollouts		Dev. tokens (M)		Holdout passes	
	Full evaluation	HCE	Full evaluation	HCE	Full evaluation	HCE
SWE-bench Verified pilot						
Seed 0	16	6	4.18	0.76	8/8	8/8
Seed 1	16	6	2.97	2.17	8/8	8/8
Seed 2†	16	16	3.04	2.93	8/8	8/8
SWE-bench Verified cached45						
Seed 0	60	25	11.92	5.99	13/15	13/15
Seed 1	60	24	14.71	5.75	13/15	13/15
Seed 2	60	23	11.41	12.79	11/15	13/15
Mixed software panel						
Seed 0	100	56	306.00	219.17	28/50	29/50
Seed 1	100	44	313.53	180.39	29/50	29/50
Seed 2†	100	100	373.26	390.19	28/50	27/50

Table 7: **Earlier software results by optimization repeat.** † marks HCE fallback to full evaluation. Costs cover only the branch's post-edit development evaluation, not common profiling, editing, or holdout testing. Tokens are shown in millions; means are not pooled across different panels.

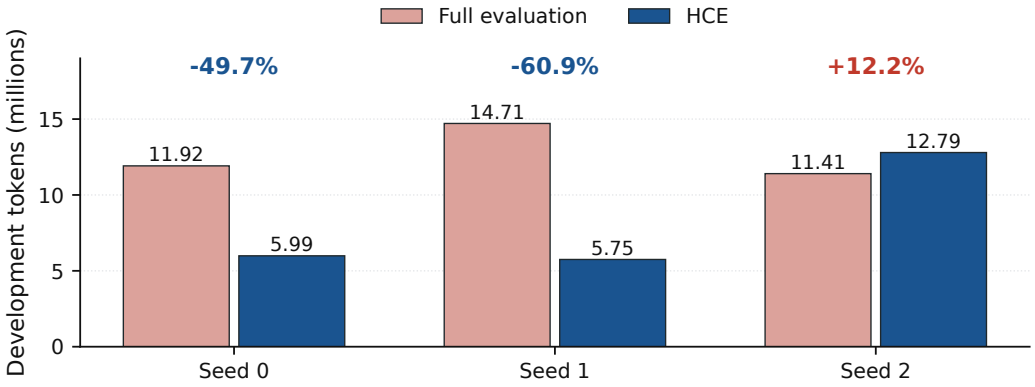

Figure 10: **Fewer rollouts need not mean fewer tokens.** In the 45-task SWE-bench Verified study, full evaluation uses 60 development rollouts per repeat; HCE uses 25, 24, and 23. The percentages show HCE's token change relative to full evaluation in that seed. Seed 2 uses fewer rollouts but 12.2% more tokens.

F Reproducibility and Scope

The manuscript bundle provides the summaries used by the figures and tables. Terminal-Bench records are in `data/terminalbench_overview.json` and `data/terminalbench_training_curve.json`. Software100 summaries and plotted rounds are in `data/software100_summary.json` and `data/software100_training_curve_data.json`. These files retain source paths and hashes for traceability. `data/task_changes.csv` provides Terminal-Bench task-level comparisons across rounds.

Pre-rendered figure PDFs allow the Overleaf manuscript to compile without Docker, Python, external datasets or model-service access. The bundle also includes plotting scripts for the overview, training-curve, sampling-and-decisions and historical software figures. This document build does not rerun the experiments or audit their raw records. We will release the implementation publicly.

The train30/test58 collection omits `extract-moves-from-video`; `video-processing` remains in training. This identifies the evaluated 88-task collection and does not imply coverage of every task in every Terminal-Bench release.