TASK-COEVOLVE: EFFICIENT HARNESS OPTIMIZATION VIA ADAPTIVE VALIDATION TASK SELECTION

Anonymous authors
Paper under double-blind review

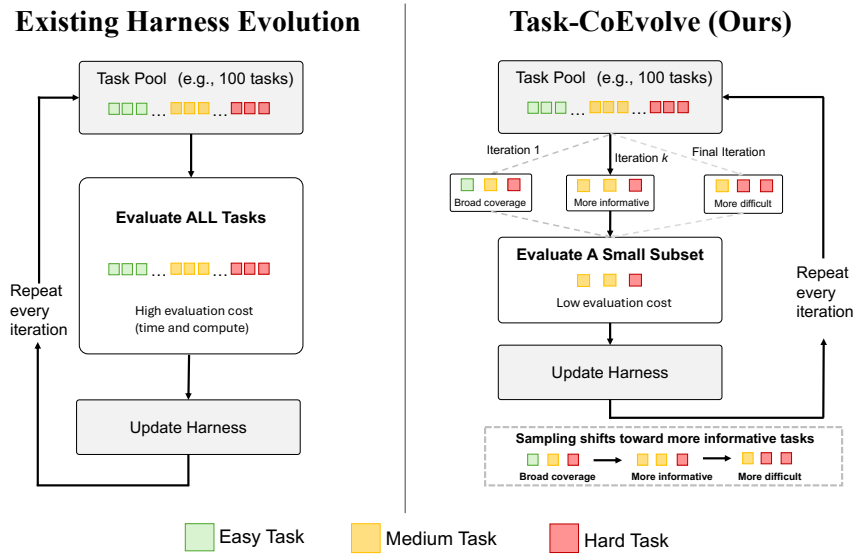

Figure 1: **Comparison between existing harness optimization and our Task-CoEvolve.**

ABSTRACT

We present a novel approach to efficient LLM harness optimization through adaptive validation task selection. Harness optimization iteratively rewrites the harness code based on validation performance, enabling substantial performance gains without updating the underlying model weights. Existing approaches, however, evaluate a fixed validation set in full at every iteration, incurring substantial evaluation costs even on tasks that become less discriminative as the harness evolves. We propose **Task-CoEvolve**, which *co-evolves* the validation tasks with the harness by addressing two challenges: selecting informative tasks and estimating full-set performance from partial evaluations. Task-CoEvolve builds on the observation that tasks on which candidate harnesses disagree are more informative for distinguishing among them than tasks that are consistently solved or failed. It uses variance-weighted sampling based on past outcomes to focus evaluation on tasks near the capability frontier, with the sampling distribution adapting as the harness evolves. It then estimates full-set scores from the sampled tasks by accounting for their sampling probabilities, enabling comparisons on a common estimated full-set scale across iterations despite evaluating different subsets. Experiments on online text classification and Terminal-Bench 2.1 show that Task-CoEvolve improves average performance over subset-based baselines and achieves a similar pass rate on the same 89-task benchmark of full-set search while reducing the number of evaluations during optimization by 80%.

1 INTRODUCTION

When deploying LLM agents to real-world tasks, the design of the *harness*, the code that determines what to store, what to retrieve, and what to present to the model, is critical. Indeed, simply changing the harness around a fixed LLM has been shown to yield up to a 6× difference in performance on the same benchmark (Tian et al., 2026), suggesting that the harness can matter as much as the

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.

underlying model itself. Traditionally, harnesses have been designed by hand (Zhang et al., 2026c; Ye et al., 2026; Merrill et al., 2026; KRAFTON AI & Ludo Robotics, 2026), but their vast design space makes this process costly and reliant on extensive trial and error. Recently, *automated harness optimization* has begun to attract considerable attention, where a meta-level agent iteratively rewrites the harness code, evaluates the resulting candidates on a benchmark, and retains changes that improve performance (Lee et al., 2026; Lin et al., 2026; Zhang et al., 2026a; Wang et al., 2026; Guo et al., 2026; Weng, 2026).

Existing work on automated harness optimization, however, has largely adopted a naive evaluation strategy: at every iteration, each candidate harness is evaluated on the entire *fixed* set of validation tasks (Lee et al., 2026; Lin et al., 2026; Zhang et al., 2026a; Wang et al., 2026). While simple to implement and straightforward for comparing candidates, this strategy has two key limitations. First, it is expensive. Every iteration requires inference over the full validation set, and when individual tasks are costly to execute, such as long-horizon terminal tasks that occupy a sandbox environment for tens of minutes (Merrill et al., 2026), evaluation can dominate the cost of the entire optimization loop. Second, it is *static*. As the harness evolves, the tasks that meaningfully discriminate among candidates also change. Tasks that every candidate can already solve, or that no candidate can yet solve, continue to consume the evaluation budget while providing little signal for optimization. We therefore need to reduce evaluation cost while preserving the ability to distinguish between candidate harnesses and adapting the evaluated tasks as the harness evolves.

In this paper, we study efficient harness optimization using only a subset of validation tasks at each iteration. We propose **Task-CoEvolve** that *co-evolves* the validation task set alongside the harness (Figure 1). Task-CoEvolve addresses two challenges: (i) selecting tasks that best discriminate among candidate harnesses and (ii) making evaluations comparable across iterations despite using different subsets. For (i), we observe that tasks where candidates have different outcomes are more informative, while tasks that are always solved or failed provide little information for ranking candidates. We measure this using the Bernoulli variance of each task's historical success rate, which becomes large when success and failure are balanced. As the harness evolves, the sampling distribution also changes to focus on informative tasks for the current harness. For (ii), we use task inclusion probabilities to estimate full-set scores from each sampled subset, providing a common evaluation criterion across iterations. These components concentrate evaluation on informative tasks while enabling fair comparison and final selection without evaluating the full validation set at every iteration.

Following prior work (Lee et al., 2026), we evaluate Task-CoEvolve on online text classification (Fei et al., 2024; Gretel AI, 2023; Schneider et al., 2016) and Terminal-Bench 2.1 (Merrill et al., 2026), a benchmark for long-horizon terminal agents. On text classification, Task-CoEvolve approaches full-set search even with a per-candidate evaluation budget of only 7%. With a 20% evaluation budget, Task-CoEvolve even achieves a higher mean held-out test accuracy than full-set search in our runs. On Terminal-Bench 2.1, Task-CoEvolve matches the performance of full-set search using only 20% of the evaluations, while reducing the overall search cost by 67-80%. Our contributions are summarized as follows:

- **Problem Setting.** We introduce the problem of optimizing *which tasks* are used to evaluate candidate harnesses. This direction is orthogonal to prior efficiency approaches that reduce the number of candidates, instead reducing the evaluation cost per candidate.
- **Task-CoEvolve.** We propose Task-CoEvolve, which combines adaptive task selection based on discriminative power with full-set score estimation from partial evaluations (Figure 1). The former concentrates the evaluation budget near the capability frontier, while the latter provides a common evaluation criterion across different subsets.
- **Empirical Findings.** On text classification, Task-CoEvolve approaches full-set search with only 7% of the evaluation budget and surpasses it with 20% (Table 1). On Terminal-Bench 2.1, it achieves comparable performance while reducing search costs by 67–80%.

2 Related Work

Automatic Optimization of Harnesses. Research on recursive self-improvement has traditionally focused on improving the model itself by updating its weights (Huang et al., 2023; Yuan et al., 2024; Zweiger et al., 2025; Zelikman et al., 2022). More recently, attention has shifted toward improving



Figure 2: **Overview of the Task-CoEvolve procedure.** Task-CoEvolve first selects a subset of tasks from the full validation set using variance-weighted sampling and evaluates the candidate harness on the selected tasks. Task-CoEvolve estimates the full-set accuracy from the sampled evaluations. Finally, The resulting outcomes are used to update the validation-task statistics.

the harness while keeping the underlying model fixed (Lee et al., 2026; Lin et al., 2026; Nie et al., 2026; Zhang et al., 2026a). A representative example is Meta-Harness (Lee et al., 2026), which evaluates multiple harness candidates on a given task set and leverages their execution traces and performance histories to iteratively improve the harness, ultimately aiming to maximize performance over the target task distribution. A common limitation of these approaches (Lee et al., 2026; Lin et al., 2026; Zhang et al., 2026a), however, is that they repeatedly evaluate candidate harnesses on the full validation task set at every iteration.

Efficient Harness Optimization. Recent work has explored improving the efficiency of harness and program optimization from several directions. DemoEvolve (Che et al., 2026) incorporates human demonstrations to provide more informative feedback in sparse-reward settings, while ShinkaEvolve (Lange et al., 2026) and TurboEvolve (Yang et al., 2026) improve the sample efficiency of LLM-driven evolutionary search through more efficient candidate generation and selection strategies. HarnessCompass (Zhang et al., 2026b) further improves harness evolution through constrained, feedback-guided, and component-wise optimization. These approaches primarily improve efficiency on the search side, such as by generating, selecting, or refining candidate harnesses more effectively.

Sample-Efficient Model Evaluation. Prior work has reduced evaluation costs by estimating model performance from selected test examples. Active testing selectively evaluates informative test examples with importance-weighted performance estimation (Kossen et al., 2021). tinyBenchmarks (Maia Polo et al., 2024) and AcTracer (Huang et al., 2026) further improve sample-efficient LLM evaluation through compact benchmark subsets and model-informed sampling, respectively. These approaches primarily focus on efficient performance estimation of a fixed model. In contrast, Task-CoEvolve selects tasks to discriminate among evolving harness candidates, while accounting for non-uniform sampling when estimating full-set performance.

3 Method

3.1 Problem Statement

We consider a fixed LLM and a *harness* h, *i.e.*, the control code surrounding the model. Let $\mathcal{T} = \{1, \ldots, N\}$ denote a set of validation tasks, and let $x_t(h) \in [0, 1]$ denote the average success rate of harness h on task $t \in \mathcal{T}$ over r trials. We define the true performance of a harness h as its full-set score

$$S(h) = \frac{1}{N} \sum_{t \in \mathcal{T}} x_t(h). \tag{1}$$

In harness optimization, a meta-level agent proposes a candidate harness h_k at each iteration $k = 1, \ldots, K$ based on previous evaluation results, and feeds the new results back into subsequent

iterations. The goal is to select, among all candidates generated during optimization, the harness with the highest $S(h)$. In this paper, we consider achieving this goal under a limited evaluation budget, measured by the total number of task executions.

Importantly, our task selection and full-set estimation procedures depend only on observed task outcomes, *i.e.*, the history of successes and failures, and make no assumptions about either the task content or the harness code.

3.2 Review of Meta-Harness

Meta-Harness (Lee et al., 2026) is a representative framework for automated harness optimization. At each iteration, the meta-agent receives previous harness candidates and their evaluation results as context and generates the code for a new harness h_k. The candidate h_k is then evaluated on the *entire* validation task set $\mathcal{T}$, and its average score, $\frac{1}{N}\sum_{t\in\mathcal{T}} x_t(h_k)$, is recorded. After K iterations, the candidate with the highest recorded score is selected as the final harness. This procedure requires a total of $K \times N \times r$ task executions, incurring an evaluation cost proportional to N at every iteration.

3.3 Proposed Approach: Task-CoEvolve

A natural way to reduce evaluation cost is to evaluate each candidate on only a subset $\mathcal{S}_k \subset \mathcal{T}$ of size $m = \lceil \rho N \rceil$, which reduces the task executions from $K \times N \times r$ to $K \times m \times r$. This raises two challenges. Reusing a fixed subset invites overfitting, as the meta-agent keeps favoring modifications that work on the same tasks. Resampling it every iteration avoids this, but the raw subset mean then depends on the difficulty of the sampled tasks, so scores from different iterations are no longer comparable.

Overall Concept. Task-CoEvolve addresses these two challenges with two components. First, *variance-weighted task selection* selects a new validation subset at each iteration based on past evaluation outcomes, focusing the evaluation budget on informative tasks. Second, *full-set score estimation* estimates the full-set performance from the sampled tasks while accounting for their sampling probabilities (Horvitz & Thompson, 1952). This allows us to compare candidates evaluated on different subsets using the same scale. Figure 2 shows the overall procedure.

Phase 0: Initialization. Before the search starts, we evaluate two starting harnesses on the full task set $\mathcal{T}$. These runs are not an extra cost of our method: harness optimization needs them in any case, because the meta-agent writes its first candidate on top of them. We reuse their outcomes as the initial history, so that every task already has a success rate $\bar{p}_t$ when the first subset is drawn.

Phase 1: Variance-Weighted Task Selection. We use a simple criterion to measure how informative each task is: the Bernoulli variance of its past outcomes. This variance becomes large when previous harnesses have different outcomes on the task. At iteration k, let $\bar{p}_t$ and n_t denote the mean and the number of outcomes observed for task t in previous iterations. We define the sampling weight for task t as follows:

$$w_t = \max\big(\bar{p}_t(1-\bar{p}_t),\ \ell_t\big) + \frac{\lambda}{\sqrt{n_t}}. \tag{2}$$

The first term represents the Bernoulli variance, which is largest at $\bar{p}_t = 0.5$ and becomes zero when the task is always solved or always failed. The floor ℓ_t equals a small positive constant ℓ for tasks that have never been solved and 0 otherwise, because a task nobody has solved may still become solvable as the harness improves. The second term, $\lambda/\sqrt{n_t}$, gives a larger weight to tasks with fewer observations, so that they are not excluded based on only a few early outcomes.

Phase 2: Sampling-Aware Full-Set Estimation. To enable comparisons on a common estimated full-set scale across iterations, we estimate full-set performance from the sampled tasks. The key idea is to account for each task's inclusion probability $\pi_t = \Pr[t \in \mathcal{S}_k]$ under the sampling design (Horvitz & Thompson, 1952), which we estimate via Monte Carlo simulation. However, we find that the appropriate form of full-set estimation depends on the structure of the task pool in each benchmark (refer to Section A for detailed experiments). Therefore, guided by the Phase-0 evaluations and the type of task set, we use one of the following two estimators for each benchmark.

Hájek estimation when within-pool mean success rates are near 0 ***or*** 1**.** Weighting each sampled outcome by $1/\pi_t$ gives the Hájek estimator (Hájek, 1964)

$$\hat{S}(h) = \frac{\sum_{t \in \mathcal{S}_k} x_t(h)/\pi_t}{\sum_{t \in \mathcal{S}_k} 1/\pi_t}. \tag{3}$$

This form suits a task set that (i) splits into several pools, each sampled and estimated separately and then averaged, and (ii) has a mean success rate close to 0 or 1 within every pool, since a rarely sampled task then carries an outcome close to its pool mean.

Anchored difference estimation when they sit near the middle. Otherwise a consistently solved task with a tiny π_t is occasionally sampled and its term $x_t(h)/\pi_t$ dominates Equation (3). We therefore weight not the outcome but its deviation from an *anchor* $\bar{p}_t$, the historical success rate of task t:

$$\hat{S}(h) = \frac{1}{N} \sum_{t \in \mathcal{T}} \bar{p}_t \; + \; \frac{1}{N} \sum_{t \in \mathcal{S}_k} \frac{x_t(h) - \bar{p}_t}{\pi_t}. \tag{4}$$

The anchor is fixed before sampling, so it cancels in expectation.

Phase 3: Final Selection. After optimization, we select the candidate with the highest estimated full-set score $\hat{S}(h)$ as the final harness, breaking exact ties in favor of the earliest iteration. We provide additional experiments on the tie-breaking rule in Section B.

4 Experiments

Following Lee et al. (2026), we evaluate our method in two settings. We use online text classification for rigorous verification of Task-CoEvolve, precisely quantifying its gains through comprehensive experiments. We use Terminal-Bench 2.1 (Merrill et al., 2026) to test whether our findings generalize to a realistic, compute-intensive setting.

4.1 Setup on Online Text Classification

Benchmark and Implementation Details. We use the online text classification setup of Zhang et al. (2026c); Ye et al. (2026); Lee et al. (2026). The LLM receives labeled examples sequentially and updates its memory (harness), with performance measured on a held-out test set. We use `GPT-OSS-120B` (Agarwal et al., 2025) with temperature 0 as the classifier LLM, following Lee et al. (2026). We consider three datasets: **LawBench** (Law) (Fei et al., 2024), a 215-class criminal charge classification dataset; **Symptom2Disease** (S2D) (Gretel AI, 2023), a 22-class disease classification dataset; and **USPTO-50k** (Schneider et al., 2016), a 180-class precursor reactant prediction dataset. The train/validation/test splits are 200/50/100 for Law, 200/50/212 for S2D, and 50/30/100 for USPTO, giving 130 validation and 412 test examples in total.

For harness optimization, we use Claude Opus 4.6 (Anthropic, 2026) as the meta-agent and follow the Meta-Harness protocol with 20 evolution iterations and three candidate harnesses per iteration (60 candidates in total). We evaluate sampling rates of $\rho \in \{7\%, 20\%\}$. Each setting is repeated three times, and we report the macro-average held-out test accuracy of the selected harness across the three datasets.

Baselines and Comparison Methods. We use Meta-Harness (Lee et al., 2026) as the basic search framework. Meta-Harness is a representative framework for automatic harness optimization (Section 3.2). Its simple structure and high extensibility make it suitable for comparing different evaluation designs. Following the official implementation of Meta-Harness, all evolutionary protocols start from two initial harnesses: zero-shot (direct prompting without memory) and few-shot (all) (putting all training examples into the context).

We consider three evolutionary methods: **(1) Meta-Harness (Full Search)** is the original protocol (Lee et al., 2026), which evaluates the full validation set ($\rho = 100\%$) at every iteration; **(2) Naive** samples a fixed subset at the beginning of search and reuses the same subset for all iterations. The candidate with the highest raw subset score is selected. **(3) Random-Resample** resamples a subset $\mathcal{S}_k$ randomly from $\mathcal{T}$ at every iteration, but uses the subset score for final selection. This baseline helps separate the effect of simply changing the evaluation subset to reduce overfitting from

Table 1: **Online text classification.** Held-out test accuracy of the harness selected by each protocol under evaluation budget ρ. "Val/iter": validation samples per iteration (USPTO/S2D/Law); "Evals": total sample evaluations during search. Task-CoEvolve nearly matches full-set search at ρ=7% and surpasses it at ρ=20%.

Method	ρ	n	Val/iter	Evals	USPTO	S2D	Law	Avg
Hand-designed harnesses								
Zero-shot	–	1	–	–	13.0	66.0	9.0	29.3
Few-shot	–	1	–	–	15.0	84.9	25.0	41.6
Evolved harnesses (harness search)								
Meta-Harness (Full search)	100%	3	30/50/50	7,800	17.0±1.0	88.5±0.7	40.3±3.8	48.6±0.8
Naive	7%	3	2/3/3	480	11.0±7.8	85.7±2.0	39.0±3.6	45.2±3.2
Random-Resample	7%	3	2/3/3	480	18.3±3.1	87.1±0.3	35.7±9.9	47.0±2.2
Task-CoEvolve (ours)	7%	3	2/3/3	480	16.3±2.3	86.2±1.8	40.3±0.6	**47.6±0.9**
Naive	20%	3	6/10/10	1,560	14.3±0.6	86.8±1.7	40.3±0.6	47.2±0.6
Random-Resample	20%	3	6/10/10	1,560	16.0±1.0	87.7±1.2	41.0±3.0	48.2±0.5
Task-CoEvolve (ours)	20%	3	6/10/10	1,560	19.3±1.2	86.2±1.0	42.3±2.5	**49.3±0.8**

the effects of discriminative selection and sampling-aware estimation in our method. Scores here are estimated per dataset, and the rule of Section 3.3 assigns the Hájek estimator (Equation (3)) to this setting (Table C).

4.2 Results on Online Text Classification

We show the results in Table 1. The main findings are as follows.

Task-CoEvolve achieves the highest accuracy at both budgets. Task-CoEvolve achieves 49.3% at ρ=20% and 47.6% at ρ=7%, outperforming Naive by 2.1 and 2.4 points, respectively. With ρ=7%, Task-CoEvolve improves the few-shot accuracy from 41.6% to 47.6% while using approximately 16 times fewer validation-task evaluations than full-set search.

Task-CoEvolve outperforms Meta-Harness at $\rho = 20\%$. Surprisingly, Task-CoEvolve outperforms Meta-Harness by about 1% even when using only 20% of the validation set. One possible reason is that Meta-Harness overfits to the validation set during search, leading to lower performance on the test set. This result suggests that changing the evaluation samples across iterations can reduce the risk of overfitting.

4.3 Setup on Terminal-Bench 2.1

Benchmark and Implementation Details. Terminal-Bench 2.1 (Merrill et al., 2026), a minor evaluation update to Terminal-Bench-2, evaluates LLM agents on 89 challenging tasks that require long-horizon, fully autonomous execution under complex dependencies and substantial domain knowledge. Following Lee et al. (2026), we use the same 89 tasks for both search and final evaluation. This is because the benchmark is small and expensive enough that introducing a separate split would substantially weaken the search signal (Lee et al., 2026).

We use GPT-5.6 Luna (OpenAI, 2026) and Qwen3.6-35B-A3B (Qwen Team, 2026) to balance performance and inference cost, as running large-scale evaluations on Terminal-Bench 2.1 is particularly expensive. We set the number of runs per task to $r = 1$ (one rollout per task per iteration) and run 10 evolution iterations, generating one candidate harness per iteration and 10 candidates in total. A small fraction of sandboxed executions can fail randomly due to infrastructure issues unrelated to the harness, introducing noise into the evaluation. We therefore retry such failures at most twice based on a fixed list of errors, using the same rule for all experiments.

Baselines and Comparison Methods. Following Lee et al. (2026), we initialize the search from two strong open baselines, Terminus 2 (Merrill et al., 2026) and Terminus-KIRA (KRAFTON AI & Ludo Robotics, 2026). We use the same evolutionary comparison methods as in the online text classification experiments. The 89 tasks form a single pool in which most tasks are either always or

Table 2: **Terminal-Bench 2.1.** Full-89 task pass rate (%) of the harness selected by each protocol. "ρ": evaluation budget as a fraction of the 89-task pool per iteration. Task-CoEvolve at ρ=20% nearly matches full-set search on both models while using 5× fewer evaluations, and beats both subset baselines.

Method	ρ	GPT-5.6 Luna	Qwen3.6-35B-A3B	Avg
Starting harnesses (hand-designed)				
Terminus-KIRA	–	49.4	32.6	41.0
Terminus 2	–	52.8	34.8	43.8
Evolved harnesses (harness search)				
Meta-Harness (Full search)	100%	62.9	42.7	52.8
Naive	20%	55.1	39.3	47.2
Random-Resample	20%	59.6	37.1	48.4
Task-CoEvolve (ours)	20%	**61.8**	**41.6**	**51.7**

Table 3: **Cost of the search phase on Terminal-Bench 2.1.** "vs. Full" is the reduction in input tokens relative to full search. Qwen3.6 was served locally, so no API cost applies. Running time is measured including candidate generation, with 10 trials executed in parallel.

Method	Trials	Input (M)	Output (M)	vs. Full	Cost (USD)	Time (h)
GPT-5.6 Luna						
Full search	890	2,888	17.0	–	117	22.2
Naive	180	223	2.3	−92%	12	11.9
Random-Resample	180	124	2.1	−96%	8	9.6
Task-CoEvolve (ours)	180	579	5.7	−80%	30	11.5
Qwen3.6-35B-A3B (self-hosted)						
Full search	890	741	24.6	–	–	38.0
Naive	180	178	6.7	−76%	–	15.1
Random-Resample	180	215	4.5	−71%	–	14.4
Task-CoEvolve (ours)	180	246	4.8	−67%	–	20.5

never solved by the two starting harnesses (Table 5), while their mean success rate on the same 89 tasks is near 0.5, so a rarely sampled task carries an outcome far from the pool mean. Following the rule in Section 3.3, Task-CoEvolve therefore uses the anchored difference estimator (Equation (4)) here.

4.4 Results on Terminal-Bench 2.1

We show the results on Terminal-Bench 2.1 in Table 2. The main findings are as follows.

Task-CoEvolve outperforms the comparison methods and is comparable to Full Search. As shown in Table 2, Task-CoEvolve outperforms both Naive and Random-Resample with GPT-5.6 Luna and Qwen3.6-35B-A3B. Compared with Meta-Harness (Full Search), Task-CoEvolve is 1.1% percentage points lower in both settings, which corresponds to just one task out of 89. Following prior work, Terminal-Bench uses the validation accuracy itself as the final evaluation result. Therefore, Meta-Harness (Full Search) has an inherent advantage because it can evaluate all tasks at every iteration. Considering this advantage, the small performance gap suggests that Task-CoEvolve achieves performance comparable to Full Search.

4.5 Search Cost and Time on Terminal-Bench 2.1

Terminal-Bench 2.1 requires tens of minutes of sandbox execution per task, making the search process itself a practical bottleneck. We therefore show the actual search cost reduction achieved by our method in Table 3.

Task-CoEvolve reduces search-phase input-token usage by 67–80% and wall-clock time by approximately 46–48%. For GPT-5.6 Luna, Full Search consumes 2,888M input tokens (22.2 hours), while Task-CoEvolve requires only 579M tokens (11.5 hours). For Qwen3.6, the input

Table 4: Ablation on the contribution of each Task-CoEvolve component. We set ρ to 20%. Each component is added on top of the row above. Mean±SD of the held-out test accuracy over n runs.

Configuration	Rot.	Est.	VWS	n	Mean
Naive (fixed subset)				3	47.2±0.6
+ Random-Resample (uniform, raw val)	✓			3	48.2±0.5
+ Full estimation + $\hat{S}$-max	✓	✓		3	48.8±1.4
+ Variance-weighted selection (Task-CoEvolve)	✓	✓	✓	3	**49.3**±**0.8**

token usage decreases from 741M to 246M, and the search time decreases from 38.0 to 20.5 hours. Despite these substantial reductions, the final performance is only 1.1 points lower than Full Search (Table 2), showing that Task-CoEvolve achieves comparable search results using $20 - -33\%$ as many input tokens.

The same 20% evaluation budget can result in very different costs. Although all 20% protocols evaluate the same number of tasks, their token consumption differs substantially. The average input tokens per trial are 0.7M for Random-Resample and 1.2M for Naive, while Task-CoEvolve uses 3.2M, comparable to Full Search (3.2M). This is because variance-weighted selection concentrates the evaluation budget on long-running, multi-turn tasks where candidate harnesses show different success and failure outcomes. Therefore, the large cost reduction of Random-Resample (96%) should not necessarily be interpreted as higher efficiency; rather, uniform sampling also spends its budget on easier tasks that terminate quickly. Indeed, under the same 20% evaluation budget, Random-Resample achieves a final performance 3.3 points lower than Task-CoEvolve.

Search time is reduced by about half. The reduction in search time ($1.9\times$) is smaller than the reduction in token usage ($5.0\times$) for two reasons. First, trials are always executed with 10-way parallelism, so the search time is mainly determined by the longest-running tasks rather than the number of evaluated tasks. Second, candidate proposal by the meta-agent takes about 2.4–3.3 hours regardless of the evaluation budget. This is a fixed cost and therefore becomes more significant as the evaluation cost decreases. Even with this fixed cost, Task-CoEvolve reduces the total search time by about half: from 22.2 to 11.5 hours for GPT-5.6 Luna and from 38.0 to 20.5 hours for Qwen3.6, while maintaining performance close to Full Search.

5 Analysis

5.1 Ablation on Each Component

We show the contributions of the three components of Task-CoEvolve: subset resampling, design-based estimation with $\hat{S}$-max selection, and variance-weighted adaptive selection in Table 4, at ρ=20%.

Random-Resample gives the largest single improvement, but scores are still not comparable. Resampling the subset uniformly at random at every iteration improves the average performance from 47.2 to 48.2, which is the largest gain among the three components. This suggests that repeatedly using the same subset is a major reason for the poor performance of Naive. However, the raw subset scores are still affected by the difficulty of each sampled subset, making comparisons across iterations unreliable.

Estimation and variance-weighted selection further improve performance. Adding full-set estimation and $\hat{S}$-max selection (Est.) improves the average performance from 48.2 to 48.8, showing the benefit of comparing candidates on a common scale. Adding variance-weighted selection (VWS) further improves the performance to 49.3, achieving the best result.

5.2 Analysis of Sample Discriminability

Table 5 divides samples into four groups based on historical accuracy $\bar{p}$: never solved ($\bar{p} = 0$), sometimes solved ($0 < \bar{p} < 1/3$), mixed ($1/3 \leq \bar{p} \leq 2/3$), and mostly solved ($\bar{p} > 2/3$).

Table 5: Distribution of task solvability across iterations. We show how the number of validation tasks solved by the harness changes over the optimization iterations.

Online text classification (7%)	**iter 0**	**iter 4**	**iter 10**	**iter 20**
Nobody solves it	74	45	45	45
A few systems solve it	0	23	20	14
Roughly half solve it (most discriminative)	22	14	10	13
Almost everybody solves it	34	48	55	58
Terminal-Bench 2.1 (GPT-5.6 Luna, 20%)	**iter 0**	**iter 4**	**iter 7**	**iter 10**
Nobody solves it	32	26	23	21
A few systems solve it	0	3	3	4
Roughly half solve it (most discriminative)	23	22	21	21
Almost everybody solves it	34	38	42	43

Table 6: **Accuracy of $\hat{S}$ vs. full-val ground truth.** Spearman: rank correlation between $\hat{S}$ and true scores. Bottom rows: the $\hat{S}$-max winner's rank and full-val score when all candidates are re-ranked by true score, next to the true best candidate's score.

	ρ=7%	ρ=20%
Spearman ($\hat{S}$ vs. true)	0.13	0.62
Winner rank by true score	10 / 60	12 / 60
True score: winner / best	46.4 / 47.8	51.3 / 54.7

Most of the evaluation budget is spent on samples that cannot distinguish between candidates. In both settings, the two extremes ($\bar{p} = 0$ and $\bar{p} > 2/3$), which provide little information for ranking candidates, consistently account for more than 70% of the sample pool. In contrast, samples where candidate outcomes are most divided ($1/3 \leq \bar{p} \leq 2/3$) remain a relatively small fraction of the task pool throughout the search. Uniform subset sampling naturally preserves this imbalance.

Sample difficulty changes as the harness evolves. The distribution is not static. In text classification, the number of samples solved by almost all candidates increases from 34 to 58, meaning that previously useful samples become too easy and therefore less useful for distinguishing candidates. In Terminal-Bench 2.1, the number of tasks that no candidate can solve decreases from 32 to 21, as previously unsolved tasks become solvable. These changes support our motivation for adapting the evaluation samples throughout the search.

5.3 Accuracy of the Estimated Scores

To evaluate the accuracy of $\hat{S}$, we re-evaluate all 60 candidates generated during a search run on the full validation set (130 samples) and compare their estimated scores $\hat{S}$ with the true scores (Table 6).

At $\rho = 20\%$, $\hat{S}$ provides a useful ranking signal. The rank correlation between $\hat{S}$ and the true score is 0.62. The candidate with the highest $\hat{S}$ turns out to be the 12th best of the 60, with a true score of 51.3% against 54.7% for the best candidate. So $\hat{S}$ does not find the very best candidate, but it keeps the choice within the top fifth of the pool.

At $\rho = 7\%$, ranking is difficult, but $\hat{S}$ still prevents poor candidate selection. The rank correlation drops to 0.13, since two or three samples per dataset carry little information about a candidate; the limitation comes from the sample size rather than from the estimator. The selected candidate still ranks 10th of 60, at 46.4% against 47.8% for the true best.

6 Conclusion, Limitations and Future Work

We introduce a new problem setting for automated harness optimization, where we optimize not only the harness itself but also which tasks are used to evaluate candidate harnesses. We propose Task-CoEvolve, which combines adaptive task selection based on discriminability with sampling-aware estimation of full-set performance. On text classification, Task-CoEvolve achieves performance close to Full Search using only a 7% evaluation budget and even outperforms it at 20%. On Terminal-Bench 2.1, it reduces the search cost by 67–80% while achieving performance similar to Full Search.

A limitation of Task-CoEvolve is that the number of tasks evaluated for each candidate is fixed in advance. The method cannot stop evaluating a candidate once it is clearly worse than the current best, or sample additional tasks when the comparison remains uncertain. Future work could determine the number of evaluated tasks dynamically based on the observed outcomes.

THE USE OF LARGE LANGUAGE MODELS

Research Ideation. The main research idea, including adaptive validation-task selection during harness optimization, was developed by the authors without LLM assistance.

Method Design. We used LLMs to discuss and compare several design choices, including the choice of estimators. The authors made the final methodological decisions and independently verified the properties and claims reported in the paper.

Implementation. We used AI coding assistants to write parts of the experimental code. The authors reviewed and tested the generated code and verified the experimental results reported in the paper.

Writing. We used LLMs to improve the wording and clarity of the manuscript. The scientific content and claims were determined by the authors, who reviewed all LLM-assisted text.

The authors reviewed all AI-assisted work and take full responsibility for the final paper.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Anthropic. Introducing claude opus 4.6, 2026. URL https://www.anthropic.com/news/claude-opus-4-6. Accessed: 2026-08-01.

Lirong Che, Yuzhe Yang, Peiwen Lin, Chuang Wang, Xueqian Wang, and Jian Su. DemoEvolve: Overcoming sparse feedback in agentic harness evolution with demonstrations. *arXiv preprint arXiv:2605.24539*, 2026.

Zhiwei Fei, Xiaoyu Shen, Dawei Zhu, Fengzhe Zhou, Zhuo Han, Alan Huang, Songyang Zhang, Kai Chen, Zhixin Yin, Zongwen Shen, Jidong Ge, and Vincent Ng. LawBench: Benchmarking legal knowledge of large language models. In *EMNLP*, pp. 7933–7962, 2024.

Gretel AI. Symptom to diagnosis dataset. https://huggingface.co/datasets/gretelai/symptom_to_diagnosis, 2023. Accessed: 2026-07-22.

Hanghui Guo, Weijie Shi, Zhangze Chen, Shengxiang Xu, Yishu Wang, Yimei Zhang, Wangze Ni, Jia Zhu, and Shimin Di. DREvo: Distilling recalibrated historical experience for harness self-evolution. *arXiv preprint arXiv:2607.26722*, 2026.

Jaroslav Hájek. Asymptotic theory of rejective sampling with varying probabilities from a finite population. *The Annals of Mathematical Statistics*, 35(4):1491–1523, 1964.

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.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. In *EMNLP*, pp. 1051–1068, 2023.

Yuheng Huang, Jiayang Song, Qiang Hu, Felix Juefei-Xu, and Lei Ma. AcTracer: Active testing of large language model via multi-stage sampling. *ACM Trans. Softw. Eng. Methodol.*, 35(3), 2026.

Jannik Kossen, Sebastian Farquhar, Yarin Gal, and Tom Rainforth. Active Testing: Sample-efficient model evaluation. In *ICML*, pp. 5753–5763, 2021.

KRAFTON AI and Ludo Robotics. Terminus-KIRA: Boosting frontier model performance on terminal-bench with minimal harness, 2026. URL https://github.com/krafton-ai/kira. Accessed: 2026-08-04.

Robert Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *ICLR*, pp. 74026–74078, 2026.

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

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinyBenchmarks: evaluating LLMs with fewer examples. In *ICML*, pp. 34303–34326, 2024.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

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

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition. https://openai.com/index/gpt-5-6/, 2026. Accessed: 2026-08-04.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Nadine Schneider, Nikolaus Stiefl, and Gregory A. Landrum. What's What: The (nearly) definitive guide to reaction role assignment. *Journal of Chemical Information and Modeling*, 56(12):2336–2346, 2016.

Muxin Tian, Zhe Wang, Zhenwei Tang, Blair Yang, Kunlun Zhu, Honghua Dong, Hanchen Li, Xinni Xie, Guangjing Wang, and Jiaxuan You. Swe-bench mobile: Can large language model agents develop industry-level mobile applications? In *KDD*, pp. 8077–8087, 2026.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*, 2026.

Lilian Weng. Harness engineering for self-improvement. *lilianweng.github.io*, July 2026. URL https://lilianweng.github.io/posts/2026-07-04-harness/. Accessed: 2026-08-04.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Yang Yang, Zining Zhong, Jindong Li, Jiemin Wu, Kaishen Yuan, Wenshuo Chen, Menglin Yang, and Yutao Yue. TurboEvolve: Towards fast and robust llm-driven program evolution. *arXiv preprint arXiv:2604.18607*, 2026.

Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. Meta context engineering via agentic skill evolution. In *ICML*, 2026.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. Self-rewarding language models. In *ICML*, pp. 57905–57923, 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. In *NeurIPS*, pp. 15476–15488, 2022.

Jiayi Zhang, Yongfeng Gu, Jianhao Ruan, Maojia Song, Yiran Peng, Zhiguang Han, Jinyu Xiang, Zhitao Wang, Caiyin Yang, Yixi Ouyang, Bang Liu, Chenglin Wu, and Yuyu Luo. Harnessing agentic evolution. *arXiv preprint arXiv:2605.13821*, 2026a.

Luan Zhang, Ruochen Zhou, Dandan Song, Zhengyu Chen, Yuhang Tian, Jun Yang, Huipeng Ma, Chenhao Li, Guangyuan Feng, Xudong Li, Yizhou Jin, and Yan Xu. HarnessCompass: Guiding automatic harness evolution toward generalizable and effective agent harnesses. *arXiv preprint arXiv:2608.01918*, 2026b.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y Zou, and Kunle Olukotun. Agentic Context Engineering: Evolving contexts for self-improving language models. In *ICLR*, pp. 86069–86100, 2026c.

Adam Zweiger, Jyo Pari, Han Guo, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. In *NeurIPS*, pp. 74084–74115, 2025.

APPENDIX

A EFFECT OF THE ESTIMATOR CHOICE

In this section, we swap the full-set estimator used for each benchmark while keeping all other components fixed: the task selection method, evaluation budget, $\hat{S}$-max selection rule, and meta-agent. Thus, the only difference is how the sampled subset is used to estimate the full-set score $\hat{S}$.

A.1 DIFFERENCE ESTIMATION ON ONLINE TEXT CLASSIFICATION

Table A shows that replacing the estimator with difference estimation degrades performance on online text classification. Mean test accuracy drops by 3.3 points at ρ=20% and 3.6 points at ρ=7%. In text classification, each dataset contains only two to ten sampled examples per iteration. Subtracting the historical anchor therefore reduces the differences between candidates, which makes candidate comparison less reliable.

A.2 HÁJEK ESTIMATION ON TERMINAL-BENCH 2.1

On Terminal-Bench 2.1, the Hájek estimator becomes unstable. We re-estimate a GPT-5.6-Luna run with ρ=20% using the same recorded rollouts. For the 10 candidates, the average error from the full scores on all 89 tasks is 14.6 points, while it is 9.5 points for Equation (4). In the worst case, a candidate scores 52.8% on all 89 tasks, but the Hájek estimate is $\hat{S} = 82.8\%$. This is because two tasks sampled with small inclusion probabilities of $\pi_t \approx 0.03$ account for 59% of the total weight $\sum_{t \in \mathcal{S}_k} 1/\pi_t$, and the candidate solves both tasks. Equation (4) avoids this problem by using the historical mean as an anchor. For example, if the starting harnesses always solve a task, then $\bar{p}_t = 1$. If a new candidate also solves this task, its residual becomes zero. Therefore, even if the task has a small inclusion probability and a large weight, it does not affect the estimate.

B SENSITIVITY TO THE TIE-BREAK RULE

Throughout the paper, when multiple candidates have the same selection score, we select the candidate from the earliest iteration, following Section 3.3. An alternative is to select the latest candidate, since it is generated with more feedback from previous iterations. Table B compares these two strategies.

Task-CoEvolve is not affected by the tie-breaking rule because $\hat{S}$ is continuous, and no candidates share the highest value in our runs. The rule only affects the baselines, which select candidates using raw subset scores. When the subset is small, many candidates can have the same score. For example, at ρ=7%, 13–16 out of 60 candidates in a Naive run share the highest score.

The choice of tie-breaking rule does not change the ordering of the methods. In fact, selecting the latest candidate makes the baselines worse: at ρ=7%, Naive decreases from 45.2 to 41.8, and Random-Resample from 47.0 to 45.6. We therefore use the earliest candidate, which is more favorable to the baselines and also follows the implementation of Lee et al. (2026), where the first candidate reaching the best score is returned.

Table A: **Estimator swap on online text classification.** Both settings use the same search procedure and $\hat{S}$-max selection rule, differing only in the estimator. Using the difference estimator lowers the mean test accuracy.

ρ	Estimator	Test accuracy
20%	Hájek (Eq. 3)	**49.3**±**0.8**
	Difference (Eq. 4)	46.0±2.6
7%	Hájek (Eq. 3)	**47.6**±**0.9**
	Difference (Eq. 4)	44.0±3.0

Table B: **Sensitivity to the tie-break rule.** Held-out test accuracy when exact ties in the selection score are broken towards the earliest iteration (the rule used throughout the paper) and towards the latest one. "Ties" is the number of candidates sharing the top score, over the three runs of each cell.

ρ	Method	Ties	Earliest	Latest
7%	Naive	13–16	45.2±3.2	41.8±1.0
	Random-Resample	2–5	47.0±2.2	45.6±3.7
	Task-CoEvolve	1	**47.6**±**0.9**	**47.6**±**0.9**
20%	Naive	4–12	47.2±0.6	47.1±1.2
	Random-Resample	1–2	48.2±0.5	47.9±1.0
	Task-CoEvolve	1	**49.3**±**0.8**	**49.3**±**0.8**

Table C: **Phase-0 validation pool (online text classification).** Samples at $\bar{p}$=0 or $\bar{p}$=1 receive a tiny inclusion probability, and in every dataset the majority of them lies on the side nearer the dataset mean, so drawing one of them does not pull the estimate away from that mean.

Dataset	**Zero-shot**	**Few-shot**	**Mean $\bar{p}$**	$\bar{p} = 0$	$\bar{p} = 1$
USPTO (n=30)	3%	13%	0.08	26	1
S2D (n=50)	62%	80%	0.71	9	30
LawBench (n=50)	6%	22%	0.14	39	3

C A Stronger Model on Terminal-Bench 2.1

The two models in Table 2 leave room for harness optimization. To examine whether this also holds for a stronger model, we additionally run Meta-Harness with DeepSeek-V4-Flash (Xu et al., 2026). Following the same setup as our main experiments, Meta-Harness evaluates all 89 tasks at each iteration for ten iterations, generating one candidate per iteration.

Table E shows the result. We found that the search in DeepSeek-V4-Flash yielded no improvement. Terminus 2 already solves 70.8% of the tasks, and the best candidate also reaches 70.8%. The other nine score between 64.0% and 69.7%. Thus, none of the candidates improves on the starting harness.

This setting provides little measurable headroom for comparing harness-search methods. For stronger models, the starting harness already achieves high performance, leaving less room for improvement through harness optimization. In this setting, even Meta-Harness does not improve over the starting harness, making it difficult to meaningfully compare different harness optimization methods. This is not a limitation specific to Task-CoEvolve, but a general limitation of harness optimization. When the starting harness already achieves high performance, there is little room for further improvement by any optimization method. We therefore use GPT-5.6 Luna and Qwen3.6-35B-A3B in the main experiments, where harness optimization still provides clear performance gains.

D Examples of Discovered Harnesses

We describe the harness selected by Task-CoEvolve in each setting.

D.1 Online Text Classification

The selected harness on this benchmark, at ρ=20%, retrieves few-shot examples with two tokenizers instead of one. It builds two TF-IDF indices over the stored examples, one on word bigrams and one on character n-grams, ranks the stored examples separately under each, and merges the two rankings by reciprocal rank fusion:

$$\text{score}(i) = \frac{1}{k + \text{rank}_{\text{bigram}}(i)} + \frac{1}{k + \text{rank}_{\text{char}}(i)}. \tag{5}$$

Table D: **Hyperparameters of variance-weighted task selection.** The same values are used for online text classification and Terminal-Bench 2.1.

Symbol	Description	Value
ℓ	Weight floor for tasks never solved so far	0.125
λ	Uncertainty bonus coefficient in $\lambda/\sqrt{n_t}$	0.025
–	Monte Carlo repetitions for estimating π_t	4,000
–	History window for computing $\bar{p}_t$	all past iterations
ρ	Sampling rate (evaluation budget)	7%, 20%

Table E: Full-set search with three models. "Terminus 2" is the starting harness and "Full Search" is the best candidate found in ten iterations with Meta-Harness. The strongest model shows no gain from harness search under this protocol.

Model	Terminus 2	Full Search	Gain
Qwen3.6-35B-A3B	34.8	42.7	+7.9
GPT-5.6 Luna	52.8	62.9	+10.1
DeepSeek-V4-Flash	70.8	70.8	+0.0

It then fills the prompt greedily by fused score, dividing the score of an example by a factor that grows with how many examples of the same label are already in the prompt, so that a single label cannot occupy the whole context. Figure A gives the retrieval step in pseudocode.

```
# query q, stored examples E, character budget B, constants k = 60, alpha
I_word <- tfidf_index(E, tokenizer = word bigrams)
I_char <- tfidf_index(E, tokenizer = character n-grams)

r_word <- rank of each e in E by cosine(q, e) under I_word
r_char <- rank of each e in E by cosine(q, e) under I_char

for e in E:                                          # reciprocal rank fusion
    s[e] <- 1 / (k + r_word[e]) + 1 / (k + r_char[e])

P <- []                                              # greedy fill, diversity penalty
while E is not empty and chars(P) < B:
    e* <- argmax over e in E of  s[e] / (1 + alpha * count(label(e) in P))
    if chars(P) + chars(e*) > B: break
    move e* from E to P
return P as the few-shot examples of the prompt
```

Figure A: Retrieval in the harness selected for online text classification. Two rankings are fused, then the prompt is filled greedily while down-weighting labels that are already present.

This helps because the three datasets need different kinds of similarity. Word bigrams work well on legal case descriptions, where a phrase often indicates the charge. Character n-grams work well on molecule strings and on symptom text, where the useful unit is a substring rather than a word. A single tokenizer must be a compromise between the two. By fusing the two rankings, the harness can use the stronger signal for each query, without knowing which dataset the query comes from. The meta-agent wrote this as its hypothesis before the candidate was evaluated, and $\hat{S}$ later selected this harness in its run.

D.2 Terminal-Bench 2.1

The selected harness on this benchmark, with GPT-5.6 Luna, changes when the agent stops waiting after it types a command. The stock agent always sleeps for a fixed time after it sends keystrokes. The evolved harness keeps this wait for keystrokes that do not run a command, such as a lone `C-c`. For keystrokes that do run a command, it checks the terminal and returns as soon as the shell prompt is back. Figure B gives this in pseudocode.

```
def wait_after(keystrokes, duration):
    if not runs_a_command(keystrokes):          # C-c, a pure wait turn, ...
        send(keystrokes, wait = duration)       # stock behaviour, unchanged
        return

    before <- capture_pane()
    send(keystrokes, wait = min(duration, 0.5))
    prev <- capture_pane()
    delay, elapsed <- 0.5, 0.5

    while elapsed < duration:                   # poll with backoff
        sleep(delay); elapsed <- elapsed + delay
        delay <- min(1.5 * delay, 3.0)
        pane <- capture_pane()
        if pane != before                       # the command has echoed
           and pane == prev                     # the terminal is quiet
           and last_nonblank_line(pane) ends in "#" or "$":
            return                              # prompt is back: stop waiting
        prev <- pane
    return                                      # fell back to the full wait

def runs_a_command(k):
    return k is a literal Enter key, or k ends in a newline
```

Figure B: The wait after a keystroke batch in the harness selected for Terminal-Bench 2.1. All three conditions must hold before the wait is cut short, so pagers, REPLs and silent long compiles keep the full duration.

This helps because each task has a time budget, and the agent fails if it runs out of turns. Before proposing the change, the meta-agent measured the waste in the search log: the fixed sleeps used a quarter to a third of the budget, and about one fifth of the trials ended in a timeout instead of a wrong answer.

E Implementation Details

E.1 Phase-0 Validation Pool for Online Text Classification

Before any candidate is generated, we evaluate the two starting harnesses on the full validation set in Phase 0. We use their per-sample success and failure outcomes to determine the estimator, following the rule in Section 3.3. Table C summarizes the resulting validation-pool statistics. Since there are only two starting harnesses, each sample has $\bar{p}_t \in 0, 0.5, 1$. For every dataset, the mean of $\bar{p}_t$ is close to either 0 or 1. Samples with very small inclusion probabilities also tend to have $\bar{p}_t = 0$ or 1, with outcomes close to the dataset mean. Giving these samples large inverse-probability weights therefore does not strongly distort the estimate. This makes the Hájek estimator suitable for this setting, unlike on Terminal-Bench 2.1.

E.2 Hyperparameters of Experiments

Table D lists the hyperparameters. Further implementation details are provided in the supplementary code.