MIXTURE OF SELF-IMPROVING BRANCHES FOR AGENT HARNESS OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Harness optimization provides a practical setting for recursive self-improvement (RSI), where agent-generated modifications inform subsequent changes through execution feedback. Recent work such as Meta-Harness implements this process through iterative code generation and evaluation, but retains a fixed development set and proposal policy. These constraints channel evolution along a single search trajectory, increasing the risk of converging to a local optimum. We make the improvement process itself adaptive by organizing search into branches with evolving development subsets and proposal policies. Each branch retains development cases solved by more of its leading harnesses than by those of other branches, drops cases solved by every leading harness across all branches, and revises its proposal policy using its own search history. To deploy the resulting complementary harnesses, we propose a router to select one development-selected branch head for each new input before execution. Across mathematical reasoning and agentic coding benchmarks, our system achieves relative improvements over Meta-Harness of 34.8% on Olympiad-level mathematical reasoning, 11.6% on Terminal-Bench 2.0, and 3.8% on SWE-bench Lite, with harness selection and router configuration based solely on development data. These results show that evolving branch objectives and proposal policies can yield complementary harnesses whose strengths a router combines without access to test outcomes.

1 INTRODUCTION

An LLM agent's capability depends not only on the underlying model but also on its surrounding *harness*. Through retrieval, tool interfaces, and control flow, the harness governs what information the model receives, which actions it can take, and when it verifies or revises its work (Lewis et al., 2020; Yao et al., 2023; Shinn et al., 2023). Prior work demonstrates that carefully designed reasoning and feedback mechanisms can improve agent performance on question answering, sequential decision-making, and coding tasks (Yao et al., 2023; Shinn et al., 2023). Building on these advances, recent work automates harness design by using evaluation feedback to refine prompts, code, and workflows (Khattab et al., 2023; Hu et al., 2024). For example, Meta-Harness uses an agentic proposer to generate harness implementations and evaluate them on a predefined development set, drawing on prior code, scores, and execution traces to guide subsequent proposals (Lee et al., 2026). This process provides a practical setting for recursive self-improvement (RSI), where experience from earlier implementations guides subsequent changes.

Although this feedback guides successive harness proposals, the development set and proposal instructions remain fixed throughout search (Hu et al., 2024; Lee et al., 2026). Evaluating candidates on the same development cases throughout search can overlook complementary strengths: a harness may solve cases missed by the leading ones yet rank lower overall, leaving those capabilities with limited opportunity for further development. Meanwhile, fixed proposal instructions do not explicitly adapt exploration priorities to the strengths and unresolved failures that emerge during search. These limitations motivate our central question: can adapting each branch's development subset and proposal guidance help discover complementary harnesses and improve deployment performance?

To answer this question, we propose organizing harness search into separate branches, each with a development subset that adapts to its emerging strengths (Figure 1). A branch is a separate search process with its own development subset, candidate history, and proposal guidance. For each devel-

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.

opment case, we compare how many leading harnesses in each branch solve it. When one branch has a clear advantage over the others, we retain the case in that branch's subset and remove it from the others. Cases solved by every leading harness across all branches are removed from search. By changing which cases contribute to each branch's score, these updates can promote previously overlooked harnesses and redirect subsequent proposals. To adapt proposal generation to the cases retained by each branch, we also revise each branch's guidance using its local search experience (Yang et al., 2026). The guidance records which modifications improved performance, which repeatedly failed, and which problems remain unresolved. These lessons guide subsequent proposals by identifying promising mechanisms, recurring failure modes, and directions for further exploration. As branches accumulate different evidence, their guidance develops distinct priorities that complement the specialization induced by their development subsets.

Branch-local search makes selecting a single harness for deployment nontrivial. Selecting by test accuracy requires held-out outcomes that are unavailable when deploying on new problems, while development scores are not directly comparable across branches with different subsets. To address this challenge, we retain the development-best harness from each branch and use a router to choose among them for each new problem, inspired by mixture-of-experts and model-routing approaches (Jacobs et al., 1991; Ong et al., 2024).

We evaluate our approach on Olympiad-level mathematical reasoning (Lee et al., 2026), Terminal-Bench 2.0 (Merrill et al., 2026), and SWE-bench Lite (Jimenez et al., 2024). With harnesses selected exclusively on development data, the routed system improves over Meta-Harness across all four benchmark–model settings. On mathematical reasoning, accuracy rises from 46.0% to 62.0% with Gemini 3 Flash and from 29.0% to 30.5% with Claude Sonnet 4.5, corresponding to relative gains of 34.8% and 5.2%. The corresponding relative gains on Terminal-Bench 2.0 and SWE-bench Lite are 11.6% and 3.8%, reaching 50.0% task completion and 66.0% issue resolution, respectively. The router achieves performance comparable to the branch expert with the highest test accuracy and surpasses it in two settings, without using test outcomes for selection. Trajectory analysis illustrates how changing development subsets redirects search, while ablations show the strongest performance when development updates and proposal adaptation are combined.

Our contributions are fourfold:

- **Search with dynamic development subsets.** Multiple search branches adapt their development subsets based on differences in task performance, encouraging complementary harnesses.
- **Branch-specific proposal adaptation.** Proposal guidance evolves from each branch's local search history, preserving distinct lessons and priorities for future modifications.
- **Deployable expert routing.** A router combines complementary development-selected branch experts, approaching or surpassing the test accuracy of the best individual expert.
- **Improved benchmark performance.** Development-selected systems outperform fixed baselines and Meta-Harness across four settings spanning mathematical reasoning and agentic coding.

2 Related Work

Self-evolving agents and harnesses. Automated agent design searches over executable programs and workflows, making agent implementations an object of optimization (Hu et al., 2024; Zhang et al., 2024). Meta-Harness extends this direction to harness code, using an agentic proposer that inspects prior implementations, scores, and execution traces through a persistent filesystem (Lee et al., 2026). It provides the search framework and primary baseline for our work. Self-Harness identifies failure patterns from execution traces, proposes targeted harness modifications, and validates them through regression testing (Zhang et al., 2026). Other work explores automated harness optimization, online adaptation, and joint evolution of system components (Sengupta & Wang, 2026; Karten et al., 2026; Chen et al., 2026b;a; Luo et al., 2026; Hao et al., 2026). These approaches establish harness evolution as a practical form of agent self-improvement. Our contribution concerns how that evolution is directed: multiple branches adapt their development subsets through comparative frontier coverage, changing which candidate harnesses are favored and extended.

Diversity through adaptive search inputs. LLM-driven evolutionary systems encourage novelty through diverse candidate archives, drawing on novelty and quality-diversity search (Lehman & Stanley, 2011; Mouret & Clune, 2015). AlphaEvolve (Novikov et al., 2025) preserves alternative

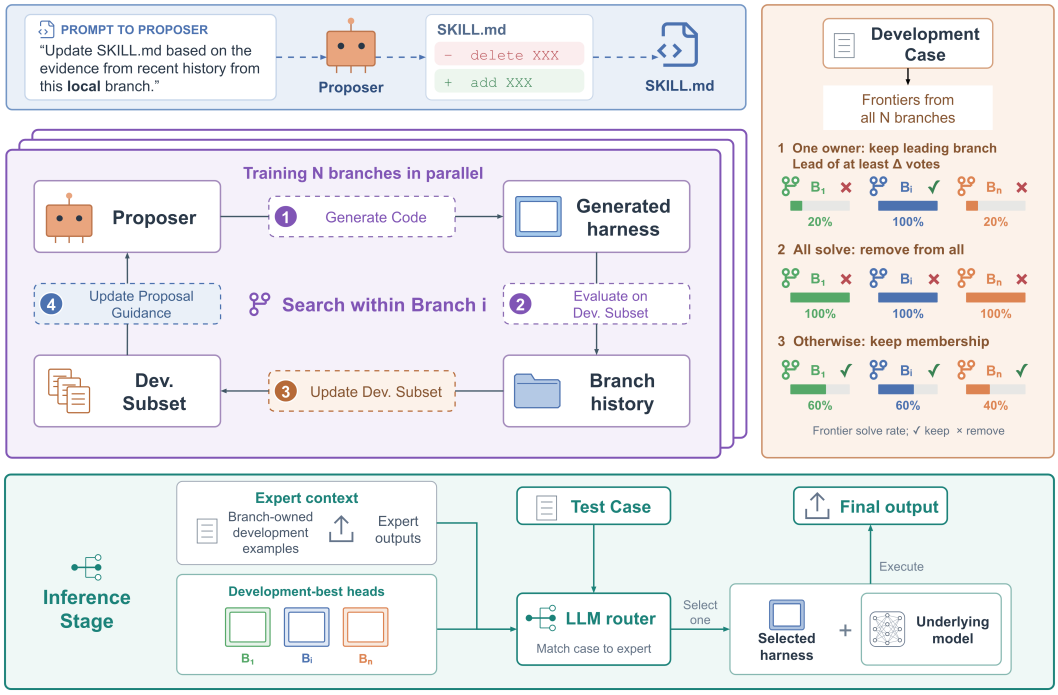


Figure 1: **Pipeline overview.** Left: each branch proposes and evaluates harnesses using its current development subset, branch history, and proposal guidance. Periodic guidance updates (upper inset) translate local search experience into priorities for subsequent proposals. Right: development subsets are updated by comparing how many leading harnesses in each branch solve each case. Bottom: a router selects one development-best branch harness for each new input before execution.

programs in a MAP-Elites-inspired database. AgenticGEO (Yuan et al., 2026) retains diverse rewriting strategies in behavioral cells. These mechanisms promote diversity through candidate scoring and retention. We instead adapt each branch's development set through comparative frontier coverage, changing the cases that guide selection and modification. This encourages complementary harnesses through different search objectives while keeping the task-level scoring rule fixed.

Evolving guidance from search experience. Another direction for improving self-evolution is to use accumulated experience to refine the guidance for future changes. Prompt and program optimizers revise instructions and programs using execution feedback (Pryzant et al., 2023; Yang et al., 2024a; Yuksekgonul et al., 2024; Khattab et al., 2023). DREvo (Guo et al., 2026) distills historical evidence into harness-search directions, while SkillOpt (Yang et al., 2026) updates external skill guidance from scored rollouts. We use branch-specific guidance to turn local successes and unresolved failures into distinct proposal priorities as objectives diverge.

Deploying evolved harnesses. Recent harness-evolution studies often use overlapping tasks for search and final evaluation (Lee et al., 2026; Wang et al., 2026). This practice can overstate generalizable improvements, with reported gains on search tasks largely failing to transfer to held-out tasks (Wang et al., 2026). We therefore evaluate deployable performance, selecting harnesses on development data alone. Selection is less straightforward in our setting because scores on different branch-specific development subsets are not directly comparable. To solve this, we route each input to a development-best branch harness, following input-level model selection (Chen et al., 2023; Ong et al., 2024). The broader principle of expert selection (Jacobs et al., 1991) also appears in model-internal routing (Zeng et al., 2025; Feng et al., 2026) and token-level collaboration (Xiong et al., 2026). Routing approaches or exceeds the stronger branch harness without test-based selection.

3 Branching Search with Evolving Objectives and Policies

Overview. We organize harness evolution into multiple search branches. Following Lee et al. (2026), we use Claude Opus 4.6 (Anthropic, 2026) as the coding proposer. During search, we adapt

each branch's development subset by comparing task performance across branches and periodically revise its proposal guidance using local search experience. At deployment, a router selects one branch expert for each new problem.

Branch-local search and proposal generation. A branch is a separate search process that evolves harnesses using its own development subset, candidate history, and proposal guidance. Let H denote a candidate harness, M the action language model, and P the coding proposer shared across branches. At iteration t, let $\mathcal{X}_b^t \subseteq \mathcal{X}$ denote branch b's current subset of the full development set $\mathcal{X}$. The candidate population $\{H\}_b^t$ contains the branch's harness implementations, whose source code, evaluation scores, and execution traces are stored in its local filesystem $\mathcal{D}_b^t$. The branch's proposal guidance S_b^t is stored in `SKILL.md`. All branches are initialized with the same seed harnesses $\{H\}^0$, full development set $\mathcal{X}$, and proposal guidance S^0.

Executing H with M on instance x produces a potentially stochastic trajectory $f_M(H, x)$ with task reward $r(f_M(H, x), x)$. Branch b seeks to maximize the expected reward on $\mathcal{X}_b^t$:

$$J_b^t(H) = \frac{1}{|\mathcal{X}_b^t|} \sum_{x \in \mathcal{X}_b^t} \mathbb{E}[r(f_M(H, x), x)] \,, \tag{1}$$

where the expectation is over execution randomness. We estimate this objective using the mean observed reward and select the highest-scoring candidate in $\{H\}_b^t$ as the branch head $\widehat{H}_b^t$.

At each iteration, P first reads the harness–score pairs $(H, J_b^t(H))$ for existing candidates in $\{H\}_b^{t-1}$. Within its own branch's filesystem $\mathcal{D}_b^t$, P is free to explore execution traces and other artifacts, deciding which evidence to inspect and which failures to investigate under guidance S_b^t. Using this evidence, P proposes a new harness H_b^t and records its design rationale. We evaluate the candidate on $\mathcal{X}_b^t$ and add its implementation and evaluation records to $\mathcal{D}_b^t$ for subsequent proposals.

Development-driven trajectory evolution. The development subset $\mathcal{X}_b^t$ determines which harnesses are favored within branch b, allowing us to redirect search by changing the problems on which candidates are evaluated. We update this subset by comparing performance on individual cases across branch frontiers $\mathcal{F}_b^t$ to determine which cases each branch retains. The frontier $\mathcal{F}_b^t$ is defined as the top-q harnesses in branch b ranked by mean reward on $\mathcal{X}_b^t$. Note that because $\mathcal{X}_b^t$ changes during search, candidate rankings and frontier membership can change even when the harness implementations remain unchanged.

Concretely, for each development case x, we first compute the coverage vote $c_b^t(x)$ on each branch's frontier $\mathcal{F}_b^t$, counting how many harnesses solve the case:

$$c_b^t(x) = \sum_{H \in \mathcal{F}_b^t} \mathbb{1}[H \text{ solves } x] \,. \tag{2}$$

If $c_b^t(x) = |\mathcal{F}_b^t|$ for every branch b, i.e., every harness in every branch's frontier solves x, we remove x from all subsets. This removes cases that offer little signal for further improvement. For the remaining cases, branch b gains ownership of x when $c_b^t(x) - \max_{b' \neq b} c_{b'}^t(x) \geq \Delta$, where Δ is the ownership margin. We remove these cases from all other branches' subsets, reserving them for the branch with a relative advantage. Otherwise, subset memberships remain unchanged. Retaining cases where a branch has a relative advantage gives subsequent search a distinct focus, allowing each branch to build on its emerging strengths without predefined task categories.

Branch-specific proposal adaptation. As development subsets $\mathcal{X}_b^t$ diverge, the shared initial guidance S^0 may no longer address the challenges specific to each branch. Inspired by SkillOpt (Yang et al., 2026), we make S_b^t adaptable, allowing each branch to refine its proposal strategy as its development objective evolves. Specifically, the same proposer P updates S_b^t from the current guidance and recent search records using the prompts reproduced in Appendix C.1. The update instructions direct the proposer to compare candidates within the same development stage and distinguish repeated evidence across implementations from isolated successes or failures. Development updates change which capabilities are rewarded, while guidance updates translate local successes and failures into priorities for subsequent proposals. A mechanism that helps solve one branch's retained cases can receive continued attention there, while another branch targets different failures.

Algorithm 1 Harness search with evolving objectives and proposal guidance

Input: tasks $\mathcal{X}$, model M, proposer P, branches B, iterations N, guidance S^0
Select and evaluate seed harnesses $\{H\}^0$ on $\mathcal{X}$, storing records in $\mathcal{D}^0$
for $b = 1, \ldots, B$ **do**
    $\{H\}_b^0 \leftarrow \{H\}^0$, $\mathcal{D}_b^0 \leftarrow \mathrm{Copy}(\mathcal{D}^0)$, $\mathcal{X}_b^0 \leftarrow \mathcal{X}$, $S_b^0 \leftarrow S^0$
for $t = 1, \ldots, N$ **do**
    for all branches b **in parallel do**
        $H_b^t \leftarrow P\big(\{(H, J_b^t(H)) : H \in \{H\}_b^{t-1}\}, \mathcal{D}_b^t, \mathcal{X}_b^t, S_b^t\big)$
        Evaluate H_b^t on $\mathcal{X}_b^t$ and store $(H_b^t, J_b^t(H_b^t), \text{traces})$ in $\mathcal{D}_b^t$
        $\{H\}_b^t \leftarrow \{H\}_b^{t-1} \cup \{H_b^t\}$
    if $t \bmod C_p = 0$ and $\min_b |\{H\}_b^t| \geq q$ **then**
        $\mathcal{F}_b^t \leftarrow \textsc{TopQ}(\{H\}_b^t, J_b^t, q)$ for each branch b
        $\{\mathcal{X}_b^t\}_{b=1}^B \leftarrow \textsc{PruneDevelopment}(\{\mathcal{X}_b^t, \mathcal{F}_b^t\}_{b=1}^B, \Delta)$
    if $t \bmod C_s = 0$ **then**
        $S_b^t \leftarrow \textsc{UpdateGuidance}(P, S_b^t, \mathrm{Recent}_{C_s}(\mathcal{D}_b^t))$ for each branch b
Select $\widehat{H}_b^N \in \arg\max_{H \in \{H\}_b^N} J_b^N(H)$ for each branch b
Build router R from heads $\widehat{H}_b^N$, labeled development cases, and expert outputs
return $(R, \{\widehat{H}_b^N\}_{b=1}^B)$ ▷ route each new input before execution

Development-informed routing. The resulting branches may develop complementary harnesses, but their scores on different development subsets are not directly comparable. At the end of search, we select the head $\widehat{H}_b^N$ with the highest mean reward on each branch's final development subset $\mathcal{X}_b^N$, retaining one expert per branch.

To choose among these experts, we use a router R, following mixture-of-experts and model-routing approaches (Jacobs et al., 1991; Chen et al., 2023; Ong et al., 2024). We construct labeled development examples from cases solved by exactly one final branch head, assigning the successful head as the routing label. R uses the action model M with expert source code, these head-exclusive cases, and both heads' outputs (Appendix D.1). We optimize its instruction using GEPA (Agrawal et al., 2026), with head-exclusive development cases as training data. For each new problem, R selects a head before execution without expert outputs or correctness feedback.

Full search and deployment pipeline. Our full pipeline is described in Algorithm 1. Branches evolve in parallel at each iteration. Pruning begins only after every branch has at least q candidate harnesses, after which we compare frontiers and update the development subsets every C_p iterations. Every C_s iterations, P also revises S_b^t. When both updates are due, we update the development subsets before revising proposal guidance and use both updated states in the next iteration. After N iterations, we construct router R from the final branch heads and their development evidence.

4 Experiments

We first evaluate deployable performance using development-selected harnesses, then examine how the branches develop distinct mechanisms and how routing combines their capabilities. We then assess the two adaptive components through search ablations and report retrospective test-best results as a reference for the quality of discovered candidates.

4.1 Experimental Setup

We evaluate on mathematical reasoning, terminal-based problem solving, and repository-level software engineering to assess whether the search procedure produces deployable gains across different tasks and action models.

Mathematical reasoning. We follow the retrieval-augmented mathematical-reasoning setting of Meta-Harness (Lee et al., 2026), in which a harness can retrieve worked solutions from a corpus of more than 500,000 problems to solve Olympiad-level questions. We use 250 problems for develop-

Table 1: Held-out performance (%) of fixed baselines and development-selected systems. Bold indicates the best score in each setting.

Method	Math		Terminal-Bench 2.0	SWE-bench Lite
	Gemini 3 Flash	Claude Sonnet 4.5	Claude Sonnet 4.5	Claude Sonnet 4.5
No-memory	34.0	24.5	—	—
Few-shot	40.0	23.0	—	—
BM25-all	40.5	25.0	—	—
BM25-geometry	46.5	27.0	—	—
Terminus 2	—	—	34.5	—
Terminus-KIRA	—	—	43.1	—
mini-swe-agent	—	—	—	62.8
Meta-Harness	46.0	29.0	44.8	63.6
Ours	**62.0**	**30.5**	**50.0**	**66.0**

ment and 200 held-out IMO-level problems for testing. We evaluate Gemini 3 Flash (Doshi, 2025) and Claude Sonnet 4.5 (Anthropic, 2025) separately as action models and report held-out accuracy.

Terminal-Bench 2.0. Terminal-Bench 2.0 evaluates agents on long-horizon tasks in realistic software environments (Merrill et al., 2026). We randomly split the 88 tasks into 30 development tasks and 58 held-out test tasks. All harnesses use Claude Sonnet 4.5 as the action model. We evaluate task completion using the benchmark's task-specific checks and report the held-out pass rate.

SWE-bench Lite. SWE-bench Lite pairs GitHub issues with repository snapshots and executable tests (Jimenez et al., 2024). We randomly split its 300 tasks into 50 development tasks and 250 held-out test tasks. Claude Sonnet 4.5 is used as the action model. We evaluate generated patches using the benchmark tests and report the held-out issue-resolution rate (%).

Baselines. We compare against fixed harnesses and automated harness search. Fixed baselines comprise no-memory and few-shot prompting, full-corpus and geometry-filtered BM25 retrieval for math (Robertson & Zaragoza, 2009), Terminus 2 (Merrill et al., 2026) and Terminus-KIRA (KRAFTON AI & Ludo Robotics, 2026) for Terminal-Bench 2.0, and unmodified mini-swe-agent[1] (Yang et al., 2024b) for SWE-bench Lite. Meta-Harness (Lee et al., 2026) is our search-based baseline, evolving harnesses with a fixed development set and proposal guidance.

Implementation details. Unless otherwise stated, we select harnesses using development performance and report their held-out test performance. Meta-Harness selects one harness on the full development set, while ours routes among heads selected on each branch's final development subset. All settings generate one candidate harness per branch per iteration. Throughout the experiments, we use $B = 2$ branches, $N = 20$ search iterations, frontier size $q = 5$, and ownership margin $\Delta = 2$. The default update intervals are $C_p = 1$ for development subsets and $C_s = 5$ for proposal guidance, with ablations varying the update schedule or disabling individual components. Appendix A reports task-solving and harness-authoring token usage across full search runs.

4.2 Performance of Development-Selected Systems

Table 1 compares our routed system with fixed harnesses and Meta-Harness under development-based selection. Our system surpasses the strongest fixed harness in every setting, with relative gains of 33.3% and 13.0% over BM25-geometry for the two math models, 16.0% over Terminus-KIRA on Terminal-Bench 2.0, and 5.1% over mini-swe-agent on SWE-bench Lite. These improvements support the usefulness of harness evolution as a practical form of recursive self-improvement. Compared with Meta-Harness, our system achieves further relative gains of 34.8% on Math–Gemini 3 Flash, 5.2% on Math–Claude Sonnet 4.5, 11.6% on Terminal-Bench 2.0, and 3.8% on SWE-bench Lite. The gains over Meta-Harness are largest on Math–Gemini and Terminal-Bench 2.0, but ex-

[1] https://github.com/SWE-agent/mini-swe-agent

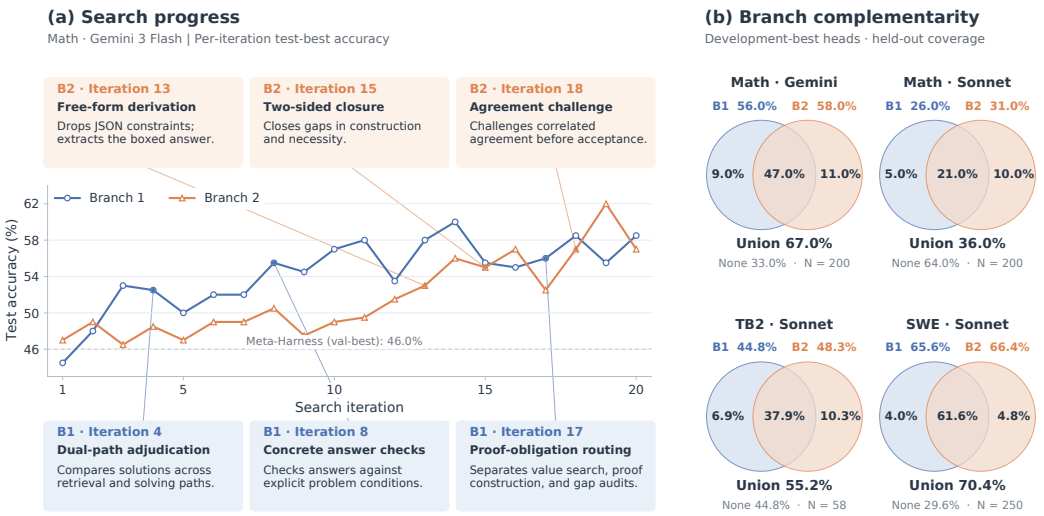


Figure 2: Left: per-iteration test-best accuracy and discovered mechanisms on Math–Gemini over 20 iterations. Right: shared and branch-exclusive coverage of development-selected heads in four settings. All rates (%) are computed over the full test sets. Venn areas are schematic.

tend across both mathematical reasoning and agentic coding, supporting applicability across the evaluated tasks and action models.

4.3 How Branches Develop Different Search Trajectories

We use Math–Gemini to examine how development-subset and proposal-guidance updates shape distinct search trajectories, then assess whether the resulting heads exhibit complementary capabilities across all four settings. The left panel of Figure 2 highlights the mechanisms discovered by each branch on Math–Gemini. Branch 1 develops concrete answer checks and proof-obligation routing, focusing on verifying proposed answers and identifying the type of argument a problem requires. Branch 2 explores free-form derivation, two-sided closure, and agreement challenges, focusing on constructing solutions and checking for shared errors when solvers agree. The trajectories thus develop different approaches to answer verification and solution construction.

We further examine how subset changes reshape the search trajectory, with subset histories in Appendix B.1. For example, at iteration 13 on Math–Gemini, Branch 2 removes four cases, reducing its subset from 44 to 40 problems. In the same iteration, a free-form derivation harness replaces the geometry-aware head, with an expanded analysis in Appendix B.2. Subsequent harnesses extend the new head with progressive derivation and two-sided closure.

Evolving proposal policies also shape the search trajectories. For example, we compare the branches' proposal guidance after iteration 15, S_1^{15} and S_2^{15}. Branch 1 prioritizes extracting concrete information from retrieved solutions, while Branch 2 emphasizes restructuring derivations after free-form derivation, progressive refinement, and two-sided closure become successive heads. These priorities direct further exploration toward improving the use of external evidence in one branch and the construction of solutions in the other. Appendix C.2 gives concrete examples of how proposer P updates branch-specific guidance S_b^t.

A Math–Gemini development case illustrates how Branch 1's targeted answer checks help it solve a case missed by Branch 2. The case asks for all real-coefficient polynomials that map positive integers consisting entirely of ones to integers of the same form. Finding a valid family is insufficient because the answer must include every such polynomial. Within Branch 1's harness, two candidate answers agree on the polynomial form but disagree on whether a parameter can be negative. Rather than accepting the narrower range, the harness tests an excluded example, $f(x) = (9x^2 + 2x - 1)/10$, which maps a repunit with k digits to one with $2k - 1$ digits. This counterexample exposes the

Table 2: Held-out performance (%) of development-selected branch heads and the routed system.

	Math		Terminal-Bench 2.0	SWE-bench Lite
	Gemini 3 Flash	Claude Sonnet 4.5	Claude Sonnet 4.5	Claude Sonnet 4.5
Head 1 only	56.0	26.0	44.8	65.6
Head 2 only	58.0	**31.0**	48.3	**66.4**
Router (ours)	**62.0**	30.5	**50.0**	66.0

Table 3: Math–Gemini held-out accuracy (%) with default routing: (a) development-update frequency and (b) development pruning and optimizer updates.

(a) Development-update frequency

Update interval	Accuracy
Every 5 iterations	53.0
Every 3 iterations	56.0
Every iteration (ours)	**62.0**

(b) Component settings

Optimizer updates → / Dev prune ↓	Off	On
Off	51.0	50.0
On	54.0	**62.0**

unnecessary restriction and leads the harness to retain the broader, correct family. This recovery illustrates how Branch 1's concrete answer checks, shown in the left panel of Figure 2, detect valid solutions omitted by a candidate answer.

Beyond this Math–Gemini case study, the right panel of Figure 2 shows that the development-selected heads solve complementary held-out cases across all four settings. Each branch solves cases missed by the other, raising combined coverage by 4.0–9.0% in absolute terms over the stronger individual head. This oracle upper bound can only be reached if the router selects a successful expert whenever one is available.

4.4 Router Analysis

Table 2 compares routing with the stronger of the two development-selected branch heads. The router exceeds this reference by 4.0% on Math–Gemini and 1.7% on Terminal-Bench 2.0 in absolute terms, while falling short by one test case on Math–Sonnet and SWE-bench Lite. The router thus approaches or exceeds the stronger head without test feedback. Routing nevertheless remains 4.4–5.5% below combined coverage in absolute terms, leaving some complementary capabilities unrealized. For example, the Math–Gemini router succeeds on 12/18 (66.7%) Branch 1-only cases and 18/22 (81.8%) Branch 2-only cases, recovering 75.0% of their exclusive successes. The remaining 10 missed cases explain its 5.0% gap to combined coverage.

To assess the contributions of router context and prompt optimization, we compare three input configurations: expert source code alone, code with head-exclusive development examples, and code with both examples and expert outputs. Each configuration is evaluated with and without GEPA. Without GEPA, adding expert outputs to these examples raises Math–Gemini accuracy from 57.0% to 59.5%, but lowers Terminal-Bench 2.0 accuracy from 48.3% to 46.6%. Additional context therefore offers setting-dependent benefits rather than a consistent improvement. GEPA provides modest average gains across the three input configurations, with the largest benefit from combining examples and expert outputs. Appendix D.2 reports the full ablation results.

4.5 Method Ablation Studies

More frequent development updates improve performance. Table 3a tests how quickly branch objectives should respond to emerging coverage differences by varying the development-update interval. We update the subsets every 1, 3, or 5 iterations, obtaining 62.0%, 56.0%, and 53.0% accuracy on Math–Gemini, respectively. Between development updates, each branch continues proposing harnesses against its unchanged subset. Longer intervals therefore allow more proposals before newly observed differences between branches affect the search objective, potentially prolonging

Table 4: Test-best performance (%) among ten harnesses per method: the top five by development score per branch for ours and the top ten for Meta-Harness.

Method	Math		Terminal-Bench 2.0	SWE-bench Lite
	Gemini 3 Flash	Claude Sonnet 4.5	Claude Sonnet 4.5	Claude Sonnet 4.5
Meta-Harness	49.0	**32.5**	46.6	63.6
Ours	**58.0**	**32.5**	**50.0**	**66.4**

exploration of problems that an earlier update would remove. The trend is consistent with timely specialization benefiting the routed system, although these aggregate scores do not establish whether update frequency changes the diversity of discovered mechanisms.

Combining development updates and proposal adaptation performs best. Table 3b compares all four combinations of development pruning and optimizer updates. Here, optimizer updates revise branch-local proposal guidance, rather than model weights or optimizer code. Disabling them keeps this guidance constant throughout search. Without development pruning, both branches retain the same fixed development set, yielding two independent Meta-Harness searches with either fixed or evolving proposal policies. All four conditions deploy the default router over development-selected branch heads. Enabling both components achieves 62.0% accuracy, compared with 51.0% with neither, 50.0% with optimizer updates alone, and 54.0% with pruning alone. Proposal adaptation alone does not improve on independent search, but increases accuracy when development pruning is enabled. These results suggest that different development subsets make proposal adaptation more useful by giving each branch distinct search challenges.

4.6 Test Performance Analysis

To assess the development–test selection gap, we evaluate 10 harnesses per method: the top 5 from each of our branches ranked on its final development subset and Meta-Harness's top 10 on the full development set. Table 4 reports the highest test accuracy within each pool. For Meta-Harness, test-based selection increases Math–Gemini accuracy from 46.0% to 49.0% and Math–Sonnet accuracy from 29.0% to 32.5%. These gaps reflect imperfect development rankings and optimistic test-based selection, complementing concerns about evaluation on search tasks (Wang et al., 2026).

Our test-best harness outperforms Meta-Harness's test-best harness in three settings and matches it on Math–Sonnet. These results support branching as a more effective search strategy for discovering high-performing harnesses under the evaluated configurations. Moreover, on Math–Gemini, our router reaches 62.0% accuracy against 58.0% for the test-best individual harness in the pool, despite relying solely on development data for selection. This illustrates why the best individual harness's test score need not bound the performance achievable by routing among complementary experts. Appendix E details the test-best harness from each setting.

5 Conclusion

We introduced branching search in which development-case assignments evolve alongside branch-local proposer guidance. Comparative frontier coverage changes the objective used to select each branch's parent, and separate skill documents preserve local priorities for subsequent proposals. A recorded head replacement demonstrates how this mechanism redirects search toward a different family of harnesses. Across four settings, development-selected heads with routing outperform development-selected Meta-Harness without test-based harness selection.

The evaluation establishes gains under the configured search procedures, while unequal total token usage and limited repeated evaluations constrain conclusions about efficiency and reliability. Routing also falls slightly below the stronger individual head in two settings. The component ablations support the combined design, but do not establish that each observed trajectory change caused its subsequent test gain. The results support adapting search objectives and branch-local proposal guidance together, then using development-informed routing to deploy the complementary harnesses.

AI USE STATEMENT

In this work, we used generative AI tools to assist in implementing the experimental system, providing feedback on experimental design, analyzing search trajectories and discovered harness mechanisms, and interpreting results. We did not use generative AI tools to develop the core methodology or propose research hypotheses. Additionally, we used these tools to draft and revise manuscript text, organize the paper, identify and summarize relevant literature, and create or modify scientific figures and supporting code. The LLM-based generation of harnesses, adaptation of proposal guidance, and expert routing are components of the research method described in the paper. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

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

Anthropic. Claude Sonnet 4.5 System Card. Technical report, Anthropic, September 2025. URL https://assets.anthropic.com/m/12f214efcc2f457a/original/Claude-Sonnet-4-5-System-Card.pdf.

Anthropic. Claude Opus 4.6 System Card. Technical report, Anthropic, February 2026. URL https://www-cdn.anthropic.com/0dd865075ad3132672ee0ab40b05a53f14cf5288.pdf.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Mingju Chen, Can Lv, Guibin Zhang, Heng Chang, and Shiji Zhou. Harnessforge: Joint harness and policy evolution for adaptive agent systems. *arXiv preprint arXiv:2606.01779*, 2026a.

Zhengyu Chen, Teng Xiao, Huaisheng Zhu, Yige Yuan, Luan Zhang, and Jingang Wang. Co-harness: Co-evolving harnesses and model weights for llm agents. *arXiv preprint arXiv:2607.22688*, 2026b.

Tulsee Doshi. Gemini 3 Flash: Frontier intelligence built for speed. Google Blog, December 2025. URL https://blog.google/products-and-platforms/products/gemini/gemini-3-flash/.

Jiarui Feng, Hanqing Zeng, Karish Grover, Ruizhong Qiu, Yinglong Xia, Qiang Zhang, Qifan Wang, Ren Chen, Dongqi Fu, Jiayi Liu, et al. Dag-moe: From simple mixture to structural aggregation in mixture-of-experts. *arXiv preprint arXiv:2606.01062*, 2026.

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.

Guangya Hao, Yunbo Long, and Zhuokai Zhao. Self-evolving multi-agent systems via decentralized memory. *arXiv preprint arXiv:2605.22721*, 2026.

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

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

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

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

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

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`.

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.

Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, 2020.

Yaxin Luo, Haobin Jiang, Jialv Zou, Xu Huang, Wenhao Yan, Haodong Li, Zhengrong Yue, Jing Li, Xiaofu Chen, Xiaohan Zhao, Jiacheng Liu, Jiacheng Cui, Zhiqiang Shen, and Xiaotong Li. Autodesign: Meta-harness optimization for long-horizon agentic design. *arXiv preprint arXiv:2608.13560*, 2026.

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

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi: 10.1561/1500000019.

Biswa Sengupta and Jinhua Wang. HARBOR: Automated harness optimization. *arXiv preprint arXiv:2604.20938*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

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. *arXiv preprint arXiv:2607.12227*, 2026.

Nuoya Xiong, Yuhang Zhou, Hanqing Zeng, Zhaorun Chen, Furong Huang, Shuchao Bi, Lizhu Zhang, and Zhuokai Zhao. Token-level llm collaboration via fusionroute. *arXiv preprint arXiv:2601.05106*, 2026.

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

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

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

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

Jiaqi Yuan, Jialu Wang, Zihan Wang, Qingyun Sun, Ruijie Wang, and Jianxin Li. AgenticGEO: A self-evolving agentic system for generative engine optimization. *arXiv preprint arXiv:2603.20213*, 2026.

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.

Hanqing Zeng, Yinglong Xia, Zhuokai Zhao, Chuan Jiang, Qiang Zhang, Jiayi Liu, Qunshu Zhang, Lizhu Zhang, Xiangjun Fan, and Benyu Zhang. S'MoRE: Structural mixture of residual experts for parameter-efficient LLM fine-tuning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 81441–81478, 2025.

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

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.

A Token Accounting

We account for full-run token usage by separating task solving from harness authoring. Across the three benchmark settings, task solving uses Claude Sonnet 4.5 as the action model M, while harness authoring uses Claude Opus 4.6 as the proposer P. Task-solving usage sums input and output tokens across recorded evaluation calls, including cached input once. Harness-authoring usage sums uncached input, cache-read, cache-creation, and output tokens across proposer sessions. Total usage combines both components. Table 5 reports these totals for each benchmark. The initial development pools contain 250 cases for math, 30 for Terminal-Bench 2.0, and 50 for SWE-bench Lite. Despite using $B = 2$ branches, ours consumes $1.21\times$, $1.71\times$, and $1.50\times$ as many task-solving tokens as Meta-Harness on these benchmarks, respectively. Development-set pruning helps limit this overhead by reducing the number of cases evaluated for subsequent candidates.

Table 5: Full-run token usage (millions) for Meta-Harness and ours across three benchmarks with Claude Sonnet 4.5 as the action model and Claude Opus 4.6 as the proposer. Task-solving and harness-authoring tokens sum to the total, with cached tokens counted once.

	Meta-Harness			**Ours**		
Benchmark	Task solving	Harness authoring	Total	Task solving	Harness authoring	Total
Math	11.10	68.82	79.92	13.40	159.50	172.90
Terminal-Bench 2.0	2,086.10	280.70	2,366.80	3,564.60	436.79	4,001.39
SWE-bench Lite	1,814.60	92.57	1,907.17	2,726.70	224.78	2,951.48

B Development-Set Changes and Head Selection

B.1 Subset Histories Across Settings

Figure 3 tracks how development subsets shrink and diverge during search across the four settings. Each branch's active subset comprises the shared cases and those retained exclusively by that branch. Updates require $q = 5$ candidate harnesses per branch and normally begin at iteration 5, while Math–Sonnet starts at iteration 6 because iteration 3 produced no candidate.

Across all four settings, the shared pool contracts while the number of branch-exclusive cases increases, giving the branches increasingly distinct evaluation sets. On SWE-bench Lite, the number of cases removed from both branches remains at 19 between iterations 5 and 20, while shared cases decrease from 30 to 24 and branch-exclusive cases increase from 1 to 7. Subset updates therefore promote specialization even when the total pool of retained cases stays unchanged.

B.2 How Subset Changes Redirect Head Selection

We expand the Math–Gemini Branch 2 transition described in Section 4.3. Between iterations 8 and 12, its development subset contracts from 58 to 44 cases, while the geometry-aware dual-solver harness remains the head. At iteration 13, the subset loses four more cases and the newly proposed `freeform_derivation_dual` becomes the head. This transition changes the design that subsequent proposals use as their starting point.

The new harness preserves the previous head's retrieval paths, geometry-specific corpus selection, and conditional adjudication, but changes how the solvers present their reasoning. Instead of writing derivations inside a JSON field, they produce ordinary mathematical prose and LaTeX, with the final answer extracted separately. Its design rationale connects this change to the remaining development failures, noting that earlier attempts to impose additional solution procedures had performed poorly. The proposer consequently explores a less constrained derivation format while retaining the established retrieval structure.

At iteration 14, the proposal record identifies the free-form harness as the head, solving 8 of the 40 retained cases, followed by an earlier progressive-refinement harness solving 7. The resulting `freeform_progressive` explicitly combines the new head's free-form reasoning with the earlier harness's targeted retrieval. It identifies a weak step in the initial derivation, retrieves relevant examples, and derives the solution again without the JSON constraint. At iteration 15, this combined harness is recorded as the head and the starting point for `two_sided_closure`. Its rationale identifies incomplete arguments in the head's traces, motivating separate derivations for constructing an attainable answer and establishing the constraints that every answer must satisfy. These proposals preserve free-form reasoning while developing new ways to construct and check solutions.

Later proposals extend this sequence through successive heads. At iteration 17, the rehearsal harness preserves the two-sided structure and adds a preliminary attempt at a retrieved problem, using feedback from its reference solution to guide the target derivation. At iteration 18, its successor adds a concrete challenge when the construction and necessity arguments agree, targeting errors shared by both paths. At iteration 19, `gap_directed_construction` extends that head's disagreement path by attempting the missing construction before resolving the competing answers. Its rationale explicitly preserves the retrieval, rehearsal, two-sided derivation, and agreement checks inherited

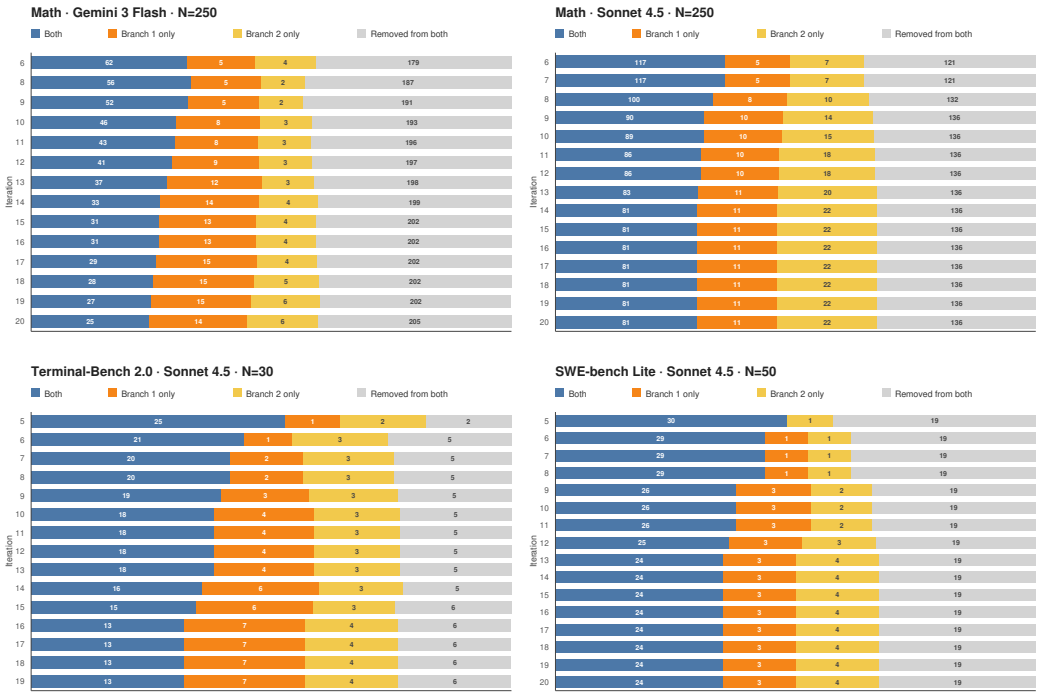


Figure 3: Development-subset changes across iterations in four settings. Filled segments show exact shared, branch-exclusive, and cumulatively removed counts, with widths scaled by $\log(1+n)$.

from the head. These recorded dependencies show how the transition at iteration 13 is followed by a sequence of harnesses that retain and extend mechanisms developed by subsequent heads.

C Proposal-Guidance Updates

C.1 Full Update Prompts

We use the same proposer P, Claude Opus 4.6, to optimize branch-specific proposal guidance S_b^t, stored in `SKILL.md`. The following prompts ask P to update this file using local search history.

Mathematical reasoning Update instructions.

```
---
name: branch-skill-gradient
description: Update one Math branch-local proposer skill from a completed-iteration window.
---

# Periodic Skill Gradient

Update the selected Math branch's local proposer skill using exactly the
iteration summaries supplied in the prompt.

## Constraints

- Read every supplied summary and do not seek older evolution history.
- Use validation accuracy as evidence. Compare within the same validation stage;
  cross-stage scores are reference only.
- Edit only the supplied branch-local `meta_skill/SKILL.md` and
  `meta_skill/gradient.json`.
- Do not edit agents, validation data, benchmark output, or source files.
- Preserve the proposer contract of exactly one candidate per iteration.
- Do not add problem text, answers, case IDs, or validation-specific rules.
- Treat each score as evidence about one implementation, not proof about a whole
  retrieval or reasoning approach.
- Use strong claims only when distinct implementations agree across at least two
```

```
  supplied iterations. Missing evaluations are missing data.
- Keep the skill open to evidence-informed exploration, not just exploitation.

## Method

1. Compare all candidate outcomes in the supplied iterations.
2. Separate candidate-specific facts from tentative broader hypotheses.
3. Give the latest validation stage the most weight.
4. State one objective for the next interval without collapsing exploration.
5. Write `gradient.json` per `gradient.schema.json` (validated fields:
   `completed_iteration`, `branch_id`, `objective`, non-empty `evidence`,
   non-empty `skill_changes`), then apply those calibrated changes to `SKILL.md`.
```

Terminal-Bench 2.0 Update instructions.

```
---
name: branch-skill-gradient
description: Update one branch-local TB2 proposer skill from a completed-iteration window.
---

# Periodic Skill Gradient

Update the selected TB2 branch's local proposer skill using exactly the
iteration summaries supplied in the prompt.

## Constraints

- Read every supplied summary and do not seek older evolution history.
- Use validation pass rate as evidence. Compare within the same validation stage;
  cross-stage scores are reference only.
- Edit only the supplied branch-local `meta_skill/SKILL.md` and
  `meta_skill/gradient.json`.
- Do not edit agents, validation task lists, benchmark output, or source files.
- Preserve the proposer contract of exactly one candidate per iteration.
- Do not encode task instructions, expected outputs, or task-specific solution steps.
- Treat each agent result as evidence about one implementation, not proof about a
  whole scaffold approach.
- Use strong claims only when distinct implementations agree across at least two
  supplied iterations. Missing evaluations are missing data.
- Keep the skill open to evidence-informed exploration, not just exploitation.

## Method

1. Compare all candidate outcomes in the supplied iterations.
2. Separate candidate-specific facts from tentative broader hypotheses.
3. Give the latest validation stage the most weight.
4. State one objective for the next interval without collapsing exploration.
5. Write `gradient.json` per `gradient.schema.json` (validated fields:
   `completed_iteration`, `branch_id`, `objective`, non-empty `evidence`,
   non-empty `skill_changes`), then apply those calibrated changes to `SKILL.md`.
```

C.2 Concrete Gradient-Update Examples

Using the prompts in Appendix C.1, P revises S_b^t from local search evidence. We present these updates as diffs of `SKILL.md`, with gray for context, red for deletions, and green for additions. Figures 4–7 illustrate branch-specific guidance updates across Math–Gemini, Math–Sonnet, Terminal-Bench 2.0, and SWE-bench Lite.

D Router Analysis

D.1 Router Instructions

The full router input combines the routing instruction optimized by GEPA, expert source code, and head-exclusive development cases with both experts' outputs. The template below repeats the example block for each supplied development case, using solutions for math and final actions for Terminal-Bench 2.0 and SWE-bench Lite.

```
{routing_instruction}

OUTPUT CONTRACT (follow exactly): return one JSON object with exactly two fields:
{"choice": "expert_1", "reason": "short evidence-based reason"}
or
```

Math (Gemini 3 Flash), Branch 1: iteration-15 guidance diff (reconstructed)

```
  ## Branch-local standings
- Prioritize head-derived verification, task-type routing, and answer-spec auditing.
+ Prioritize extracting and transferring concrete material from retrieved solutions.
+ Pattern extraction, grounded facts, and answer-key transfer produced all positive
+ coverage deltas in this window and both head changes, at iterations 13 and 15.
  ## Calibrated caution
+ Four distinct designs that adjudicated self-generated alternatives ranked lowest
+ within their respective iterations. Revisit this family with a changed mechanism,
+ such as adjudicating against retrieved evidence rather than generated alternatives.
  ## Read scores correctly
+ Compare candidate coverage as well as average scores. Identical union scores can
+ signal redundancy even when average scores differ.
+ Refreshed-stage identifiers without refreshed scores do not support comparisons
+ of accuracy across development stages.
  ## Calibrated caution
- Always-on justification and bookkeeping overlays regressed in one tested iteration.
+ A second group of reformatting and summarization overlays also lost coverage.
+ If revisited, use gated or targeted interventions rather than an always-on overlay.
```

Math (Gemini 3 Flash), Branch 2: iteration-15 guidance diff (reconstructed)

```
  ## Branch evidence carried forward
- The best score is flat, the head has not moved, and coverage gains are absent.
+ The head advanced at iterations 13, 14, and 15, with positive coverage deltas
+ observed at iterations 13 and 14.
+ Free-form derivation, progressive derivation, and two-sided closure each led
+ their iteration. Continue exploring distinct derivation-control mechanisms.
  ## Diversify proposal starting points
- Move proposals away from the standing head after repeated failures to improve it.
+ Head-based proposals produced three consecutive head changes in this window.
+ Keep this option while also considering strong prior candidates or combinations
+ of mechanisms supported by earlier results.
  ## Retrieval-side proposals
+ Weak selection and lookup variants do not rule out retrieval as a whole.
+ A renewed attempt should change the retrieved material or corpus query and explain
+ what the earlier implementation likely got wrong.
  ## Prototype checks
+ Before finalizing a harness, check that it returns a parsed, nonempty answer on
+ a logged development case.
```

unchanged deleted example gradient change

Figure 4: Branch-specific `SKILL.md` changes on Math–Gemini from iteration-15 gradients.

Math (Sonnet 4.5), Branch 1 SKILL.md diff at iteration 5

```
  Accuracy is the primary objective.
  Implement three mechanism-level candidates.
  Use branch-local scores and traces as evidence.
  ## Retrieval Guidance
+ Geometry-only BM25 retrieval is the established foundation for this branch. Mixing indexes
+ has been tried twice without benefit.
  ## Reasoning Structure Guidance
+ Dual-path independent solving with an agreement shortcut and conditional adjudication is the
+ strongest validated structure. Sequential verification risks overcorrection.
  ## Next Exploration Directions
+ Explore improvements to the adjudicator, diversity in reasoning structure between paths, and
+ retrieval reranking or query reformulation.
```

Math (Sonnet 4.5), Branch 2 SKILL.md diff at iteration 5

```
  Accuracy is the primary objective.
  Implement three mechanism-level candidates.
  Use branch-local scores and traces as evidence.
  ## Observed Evidence
+ The skill records validated and refuted mechanisms from iterations 1-5.
+ Critique-based adjudication over full reasoning chains is the strongest validated mechanism.
+ Preserve or improve it rather than replacing it with voting or blind re-solving.
+ Dual-index retrieval over the geometry and main corpora, with deduplication, outperforms
+ single-index or bare paths and remains the default second-path strategy.
+ Truncating retrieved solutions to technique hints regressed. Preserve complete retrieved
+ examples in solver prompts.
+ Improve adjudicator structure or reasoning within each solve path rather than sweeping
+ retrieval parameters or adding solve paths.
```

unchanged deleted example gradient change

Figure 5: Branch-specific `SKILL.md` changes on Math–Sonnet from iteration-5 gradients.

Terminal-Bench 2.0 (Sonnet 4.5), Branch 1 SKILL.md diff at iteration 5

```
  Change mechanisms rather than tuning constants.
  Use branch-local scores and traces as evidence.
  Implement the required candidate set.
  ## Proven Mechanisms
+ The anti-deletion prompt fix and step-budget awareness are validated baseline components.
+ The anti-deletion fix is supported across four iterations. Step-budget awareness produced
+ the best score at 46.15%.
  ## Prompt Engineering Guidance
- Comprehensive verification and process checklists are preferred.
+ Prefer minimal targeted changes. Heavy prompt additions for verification and long-running
+ process management are neutral or harmful.
  ## Duration Cap
- Use a 300-second duration cap to address polling waste.
+ A 300-second duration cap is harmful, scoring 26.92% at iteration 4. Keep the default at 60
+ seconds. Use step-budget awareness rather than longer waits.
  ## Next Exploration Direction
+ Move from prompt-only changes to structural modifications involving tools, execution logic,
+ or context and observation handling. Twelve of 26 tasks never passed across five
+ prompt-focused iterations.
```

Terminal-Bench 2.0 (Sonnet 4.5), Branch 2 SKILL.md diff at iteration 5

```
  Change mechanisms rather than tuning constants.
  Use branch-local scores and traces as evidence.
  Implement the required candidate set.
  ## Design Principles
+ Verification before completion is a validated mechanism. Aggressive intervention is harmful,
+ and completion behavior is sensitive to prompt wording.
+ Explore orthogonal mechanisms for hard tasks rather than continuing broad verification-cycle
+ changes.
```

unchanged deleted example gradient change

Figure 6: Branch-specific `SKILL.md` changes on Terminal-Bench 2.0 from iteration-5 gradients.

SWE-bench Lite (Sonnet 4.5), Branch 1 SKILL.md diff at iteration 5

```
  Change mechanisms rather than tuning constants.
  Use branch-local scores and traces as evidence.
  Implement the required candidate set.
- Best-of-N is the highest-value direction.
+ Best-of-N is cautionary. Iteration 3 regressed to 2/30 because dual-attempt and
+ test-selection overhead did not pay off at this task difficulty.
+ Context-window trimming or summarization is harmful. Iteration 5 regressed to 1/30 after
+ destroying reasoning continuity.
+ Test-gated submission is the strongest observed mechanism at 5/30. Use basename and
+ package-level test discovery with at most two retry cycles.
+ Test-gating and regression-gating solve complementary task subsets. A careful
+ delta-comparison addition to test-gating is promising.
+ Keep modifications minimal and close to the baseline. The three closest candidates
+ outperformed the two with deeper structural changes.
```

SWE-bench Lite (Sonnet 4.5), Branch 2 SKILL.md diff at iteration 5

```
  Change mechanisms rather than tuning constants.
  Use branch-local scores and traces as evidence.
  Implement the required candidate set.
+ Five single-mechanism iterations covering best-of-two, test-gated refinement,
+ str_replace_editor, reasoning_high, and retry-on-self-test all scored 4-5/31. Treat this
+ evidence as noise-dominated rather than proof of failure.
+ Twenty-one of 31 tasks remain unsolved across all iterations and require qualitatively
+ different localization or retrieval capabilities.
  ## Capability priorities
- Continue prioritizing edit-loop refinements.
+ Localization and retrieval are the highest-priority untested lever. Combining structured
+ editing from iteration 3 with self-test retry from iteration 5 is a secondary hypothesis.
+ Interpret each lever using its iteration evidence.
+ The tied-lowest reasoning_high result and the lack of benefit from budget splitting are
+ tentative single-observation findings.
```

unchanged deleted example gradient change

Figure 7: Branch-specific `SKILL.md` changes on SWE-bench Lite from iteration-5 gradients.

```
{"choice": "expert_2", "reason": "short evidence-based reason"}
The choice must be the literal string "expert_1" or "expert_2".
Return no text outside the JSON object.

EXPERT 1 ({expert_1_name}) IMPLEMENTATION:
{expert_1_source_code}

EXPERT 2 ({expert_2_name}) IMPLEMENTATION:
{expert_2_source_code}

DECISION EXAMPLES (owned development cases):
EXAMPLE {i}:
PROBLEM:
{development_case}
EXPERT 1 {SOLUTION or FINAL ACTIONS}:
{expert_1_output_on_development_case}
EXPERT 2 {SOLUTION or FINAL ACTIONS}:
{expert_2_output_on_development_case}
WINNER: {EXPERT 1 or EXPERT 2} solved this; the other did not.

PROBLEM:
{new_problem}

Return JSON only: {"choice":"expert_1"|"expert_2","reason":"..."}
```

D.2 Router Context and Prompt Ablations

Table 6 compares three input configurations, each with and without GEPA-based prompt optimization, across all four settings. All configurations receive expert source code and select an expert before execution on the new problem. Among configurations without GEPA, adding head-exclusive development cases changes accuracy from 60.5% to 57.0% on Math–Gemini, from 30.5% to 30.0% on Math–Sonnet, and from 46.6% to 48.3% on Terminal-Bench 2.0. Adding expert responses then raises Math–Gemini accuracy to 59.5% but lowers accuracy on Math–Sonnet and Terminal-Bench 2.0 to 29.0% and 46.6%, respectively. All three configurations achieve 66.4% on SWE-bench Lite. Additional context therefore helps in some comparisons, but its benefits vary across settings.

GEPA (Agrawal et al., 2026) optimizes the routing instruction using head-exclusive development cases as training data, labeled by the successful final head. The first two rows of Table 6 compare its effect when both development examples and expert outputs are available. GEPA increases accuracy from 59.5% to 62.0% on Math–Gemini, from 29.0% to 30.5% on Math–Sonnet, and from 46.6% to 50.0% on Terminal-Bench 2.0, while reducing SWE-bench Lite accuracy from 66.4% to 66.0%. Averaged across benchmarks, GEPA yields small gains in all three input configurations, with the largest benefit from full context. Table 1 reports this full-context configuration with GEPA.

E Discovered Harness Mechanisms

This appendix describes the harnesses underlying our test-best results in Table 4, selected from the ten-candidate pools defined in Section 4.6. Each description follows the execution sequence and identifies the main change from the preceding design.

E.1 Math–Gemini: Construction Before Adjudication

Overview. The `gap_directed_construction` harness attains the 58.0% test-best score in the development-ranked pool. It separates finding an attainable answer from establishing why no other answer is possible, then uses the relationship between these two arguments to determine the next computation. The execution proceeds in three stages:

1. **Retrieve and rehearse.** The harness generates a plan and uses it with the problem statement to retrieve worked examples through BM25. When a suitable reference is available, it attempts that example and compares its attempt with the reference solution to produce a calibration note for the target problem.
2. **Construct and constrain.** Two solver calls receive the plan, examples, and calibration note. One constructs and checks an attainable answer, while the other derives necessary conditions on the answer.

Table 6: Router configurations and held-out performance (%). All variants receive expert source code. Checkmarks indicate included context and whether GEPA optimization is applied.

Router components			**Math**		**Terminal-Bench 2.0**	**SWE-bench Lite**
Exclusive Cases	Expert Outputs	GEPA	Gemini 3 Flash	Claude Sonnet 4.5	Claude Sonnet 4.5	Claude Sonnet 4.5
✓	✓	✓	**62.0**	30.5	**50.0**	66.0
✓	✓	—	59.5	29.0	46.6	**66.4**
✓	—	✓	57.0	**31.0**	48.3	**66.4**
✓	—	—	57.0	30.0	48.3	**66.4**
—	—	✓	59.5	30.5	48.3	66.0
—	—	—	60.5	30.5	46.6	**66.4**

3. **Resolve agreement or disagreement.** If the normalized answers agree, a further call challenges the shared answer. If they disagree, an additional call attempts an explicit construction or concrete check that addresses the gap before a final call resolves the problem.

The construction step gives the adjudicator concrete evidence for resolving disagreements.

E.2 MATH–SONNET: CLEANED RETRIEVAL WITH ANSWER GROUNDING

Overview. The `clean_retrieval_answer_grounded` harness achieves 32.5% test accuracy using two retrieval-conditioned solution paths and conditional critique. Its main change is how retrieved solutions are prepared before they enter the solver context.

1. **Retrieve complementary references.** One path retrieves geometry examples, while the other retrieves distinct examples from the general corpus.
2. **Clean and ground the examples.** The harness removes reasoning tags, trims tagged reasoning at a sentence boundary when one is available, caps each solution at 4,000 characters, and appends the reference answer stored in the corpus.
3. **Solve and critique.** Both paths follow structured planning, solving, and verification instructions. Matching normalized answers are returned directly. On disagreement, a third call critiques both reasoning traces and produces the final answer.

It preserves the two- or three-call solver structure while changing the retrieved context.

E.3 TERMINAL-BENCH 2.0: VERIFICATION ONCE BEFORE COMPLETION

Overview. The `verification_once_gate` harness achieves 50.0% task completion. It changes the completion checks of a verification-enhanced Terminus-KIRA agent.

1. **Execute the task.** The agent interacts with the terminal through the inherited tool-use loop.
2. **Request verification once.** When the agent first signals completion, the harness issues a verification prompt and records that the prompt has been shown.
3. **Allow subsequent completion.** A later completion signal bypasses the same gate, allowing execution to finish instead of repeatedly requesting verification.

The flag remains set during subsequent tool use, preventing the verification gate from reopening after the agent resumes work.

E.4 SWE-BENCH LITE: ISSUE-GUIDED LOCALIZATION BEFORE EDITING

Overview. The `test_aware_edit` harness achieves 66.4% issue resolution. It extends a structured-editing agent with a localization stage before the first model-driven action.

1. **Extract issue identifiers.** The harness collects class names, function names, Python file paths, and exception names from the issue description, retaining up to eight identifiers.

2. **Assemble repository context.** It searches Python files for these identifiers, separates source and test files, and ranks them by identifier matches. It reads the top two source files and the top test file, caps their contents at 4,000 characters per source file and 3,000 for the test file, and inserts the context into the initial user message. If no useful matches are found, the localization stage supplies no extra context.
3. **Run the editing loop.** The agent continues with its existing tools, including a string-replacement operation that requires a unique match and returns the resulting diff.

The retrieved repository tests provide examples of expected behavior before the agent begins editing.