VIBETRAINING-BENCH: DATA–TRAINER–HARNESS CO-DESIGN FOR ITERATIVE SYSTEM IMPROVEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Recursive self-improvement requires research agents that can iteratively improve a designated target agent system as a whole, rather than one component in isolation. A target agent system's performance depends jointly on its training data, trainer, and inference harness, and because these components interact, the most useful intervention can shift as the system evolves. Existing benchmarks either fix a single component in advance or open only training-side components. They therefore do not evaluate whether research agents can select, coordinate, and revise interventions across training and inference, a capability we call *cross-component co-design*. We introduce VibeTraining-Bench, a benchmark of five tasks spanning terminal interaction, software engineering, multi-turn tool use, and mathematical reasoning. Within each task, research agents start from the same initial target agent system and training setup, then use validation scores and execution trajectories to guide iterative changes to the permitted components under a fixed budget. To limit reward hacking, an isolated grader evaluates the selected target agent system once on hidden test tasks that the research agent cannot access. Evaluating six research-agent configurations, we find large hidden-test gains (32.9% to 53.7% on SWE-bench Verified; 20.3% to 52.1% on τ^3-Bench), yet improvement is not guaranteed and component exploration remains uneven: one unrestricted SWE run does not edit the harness, although a separate harness-only run improves the initial target agent system. Validation search is also non-monotonic, and validation leaders do not always lead on the hidden test. These results expose a gap between finding effective system changes and reliably exploring and selecting them.

1 INTRODUCTION

Recursive self-improvement (RSI) refers to AI systems that improve themselves and use those gains to drive further improvement (Liu et al., 2026; Yang et al., 2026). A key capability for RSI is a research agent's ability to iteratively improve a designated target agent system. A target agent system comprises a model checkpoint and an inference harness: training data and the trainer shape the checkpoint, while the harness governs how the model acts at inference time (Ursekar et al., 2026). These design choices can interact: a harness that works well with one checkpoint may become less effective after training changes the model's behavior (Chen et al., 2026a;d). Improving the system thus requires the research agent to select and coordinate interventions across components, revisiting those choices as the system evolves. We refer to this capability as *cross-component co-design*.

Existing benchmarks do not directly evaluate cross-component co-design of a designated target agent system Frameworks such as MLE-Bench (Chan et al., 2025) and RE-Bench (Wijk et al., 2024) evaluate research agents on diverse machine learning engineering and research tasks. They do not specifically assess the repeated improvement of the same designated target agent system. More directly related RSI benchmarks often evaluate single-component improvement. RSIBench-Data (Meng et al., 2026), AI4AI-Bench (Chi et al., 2026), and HarnessOpt-Bench (Ursekar et al., 2026) focus on data strategies, training algorithms, and inference harnesses, respectively; each MLS-Bench task specifies one ML component to improve (Lyu et al., 2026). While these single-component settings simplify attribution, they fix the scope of improvement in advance and do not test how a research agent allocates effort across components. PostTrainBench (Rank et al., 2026) and Agent^2 RL-Bench (Chen et al., 2026c) broaden this scope to data construction and post-training procedures. Simply broadening the editable scope also raises evaluation risks: PostTrainBench re-

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.

ports reward hacking, including research agents training on test data (Rank et al., 2026). Even these multi-component benchmarks focus on training-side changes; they do not evaluate how a research agent allocates, coordinates, and revises interventions across training- and inference-time components of the same evolving target agent system.

To evaluate cross-component co-design, we introduce VIBETRAINING-BENCH, a benchmark of cross-component co-design in target agent systems. Rather than fixing a single component for optimization, the benchmark asks research agents to choose and coordinate changes across editable components. It comprises five environments spanning terminal interaction, software engineering, multi-turn tool use, and mathematical reasoning. Within each task, research agents start from the same base model and initial training data, and receive the same task interface and research budget. Research agents may modify training data and the trainer, including its optimization algorithms and configurations; SWE-bench Verified and Terminal-Bench 2.1 additionally allow changes to the inference harness. They run experiments, use validation feedback to guide subsequent revisions, and submit a final model checkpoint and, where editable, a harness. To avoid reward hacking, VIBETRAINING-BENCH enforces strict architectural decoupling between the optimization workspace and the evaluation infrastructure. Research agents work in a bounded sandbox where they can modify only permitted components; hidden-test examples and grader code remain inaccessible. An isolated grader then evaluates the selected target agent system on the hidden test.

Our evaluation shows that research agents can substantially improve target agent systems, yet their exploration of editable components remains uneven. On SWE-bench Verified, the GPT-6-Astra run explores changes to training data, the trainer, and the inference harness; its selected target agent system raises hidden-test performance from 32.9% to 53.7%. With model weights held fixed, one harness revision increases solved cases from 45 to 55 on validation tasks. In contrast, the Kimi-K3 unrestricted run does not edit the SWE harness, although a separate harness-only run improves the initial target agent system from 32.9% to 38.0%. On Terminal-Bench 2.1, the Kimi-K3 harness-only run scores 8.9%, matching the initial target agent system, while its data-and-trainer-only and unrestricted runs score 13.0% and 22.4%, respectively. These results suggest that productive component choices differ across tasks. Beyond component selection, validation trajectories show that attempts can regress and that leading validation candidates do not always lead on the hidden test.

In summary, we make the following contributions:

- **A benchmark for cross-component co-design.** We introduce VIBETRAINING-BENCH, a five-task benchmark for cross-component co-design of target agent systems. Its task-specific edit scopes cover training data, the trainer, and the inference harness.
- **A protected evaluation protocol.** To limit reward hacking under broad edit access, we separate the research workspace from the final evaluator. Agents can experiment with permitted components using validation feedback, while hidden-test examples and grader code remain inaccessible.
- **Limits in cross-component research.** We evaluate six research-agent configurations using final scores, design records, and component ablations. Kimi-K3 leaves the SWE-bench Verified harness unchanged in its unrestricted run, although harness-only editing- improves the initial target agent system. Leading validation candidates also do not consistently lead on the hidden test.

2 RELATED WORK

Self-evolving language models and agents. Self-improving Agent systems expose several non-exclusive intervention surfaces: training data, trainer, and inference-time harnesses. On the data front, prior work improves models through automated experience generation, student-feedback loops, and self-synthesized trajectories (Zweiger et al., 2026; Huang et al., 2026a; Zhao et al., 2026). On the trainer front, recent methods leverage LLMs to discover novel optimization objectives and adaptive reinforcement learning strategies Lu et al. (2024); Fang et al. (2026). On the inference front, self-evolving agent frameworks optimize inference harnesses by tuning prompts, tool interfaces, and runtime control flow, thereby enhancing performance under frozen model weights(Yin et al., 2025; Zhang et al., 2026). More recently, work has begun to explore multi-component coupling. For example, Chen et al. (2026b) reuse inference rollouts for training, while Chen et al. (2026a) co-evolve harness and model weights; nevertheless, these interactions follow static, predefined pathways. Taken together, these efforts reflect an emerging shift toward coupled system-level

Table 1: Protocol comparison of benchmarks for recursive self-improvement. *Legend:* ✓ = enforced; △ = partial or insufficiently documented; ✗ = unsupported; – = not applicable. Trajectory feedback requires task-level traces or diagnostics, and iterative revision requires repeated propose–evaluate–revise cycles. Joint requires coordinated changes to training data, the trainer, and the inference harness within one task. Dev/test isolation keeps the final test inaccessible during development; train/eval isolation separates training, serving, and evaluation services.

Benchmark	**Research Loop**		**Editable Training**		**System Co-Design**		**Controlled Infrastructure**	
	Trajectory Feedback	Iterative Revision	Data	Trainer	Harness	Joint	Dev/Test Isolation	Train/Eval Isolation
MLS-Bench	△	✓	✗	✓	✗	✗	△	✗
PostTrainBench	✓	✗	✓	✓	✗	✗	✗	✗
Curation-Bench	✓	✓	✓	✗	✗	✗	✗	△
RSIBench-Data	✓	✓	✓	✗	✗	✗	✗	✓
Agent^2 RL-Bench	✗	✓	✓	✓	✗	✗	✗	✓
AI4AI-Bench	△	✓	✗	✓	✗	✗	✓	✓
HarnessOpt-Bench	✓	✓	✗	✗	✓	✗	✓	–
Evo-Bench	✓	✓	✗	✗	✓	✗	✓	–
Vibe Training Bench	✓	✓	✓	✓	✓	✓	✓	✓

optimization. VIBETRAINING-BENCH advances this paradigm to an open-ended setting, evaluating whether research agents can autonomously decide where and how to coordinate improvements across data, trainer, and inference harnesses without a prescribed recipe.

Benchmarks for autonomous ML research and recursive self-improvement. Existing benchmarks evaluate autonomous ML research across diverse axes, including engineering, experimentation, replication, and algorithmic discovery (Huang et al., 2023; Wijk et al., 2024; Chan et al., 2025; Starace et al., 2025; Lyu et al., 2026). Within the context of recursive self-improvement (RSI), existing benchmarks differ primarily in the scope of system components exposed to the agent. To ensure attribution clarity, most prior works isolate a single component: Curation-Bench (Kang et al., 2026) and RSIBench-Data (Meng et al., 2026) focus on data curation, AI4AI-Bench (Chi et al., 2026) targets training algorithms, while HarnessOpt-Bench (Ursekar et al., 2026) and Evo-Bench (Huang et al., 2026b) optimize inference scaffolding around fixed model weights. PostTrainBench and Agent^2 RL-Benchbroaden this scope to data construction and training-procedure design, yet keep the deployed inference harness outside the optimization loop (Rank et al., 2026; Chen et al., 2026c). Table 1 summarizes these protocol differences across the research loop, editable components, and evaluation isolation. VIBETRAINING-BENCH bridges this gap by enabling cross-component co-design, placing training data, the trainer, and the inference harness jointly under research agent control. Crucially, because expanding the editable space naturally heightens the risk of reward hacking (Rank et al., 2026), our protocol strictly isolates the research agent's workspace from the grading service, preventing test-set leakage during iterative improvement.

3 VIBE TRAINING BENCH

3.1 OVERVIEW AND TASK FORMULATION

VibeTraining-Bench evaluates whether a research research agent can improve a target agent system by iteratively selecting and coordinating changes to its training data, trainer, and, where editable, inference harness. Each task defines the permitted components and a fixed research budget. Within this space, the agent uses validation feedback to decide what to modify next and which candidate target agent systemto retain. Its objective is to selected target agent system that improves on the target agent system when evaluated on hidden test problems outside the research loop.

We formalize this process as follows: Each task supplies a base model M_0, initial training data and trainer, an inference harness H_0, and a research budget $\mathcal{B}$. The initial target agent system is $P_0 = (M_0, H_0)$, and the research agent receives its validation score and execution trajectories be-

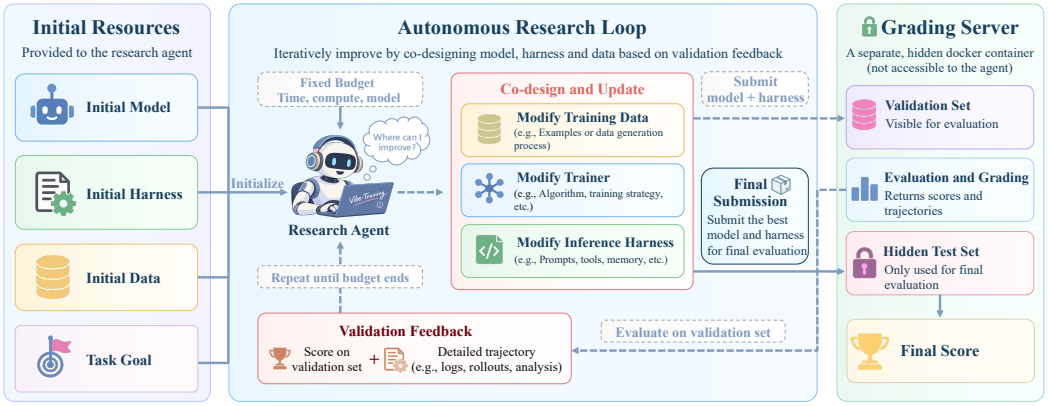

Figure 1: **Overview of VIBETRAINING-BENCH.** The benchmark assesses a research agent's capacity for systemic co-design across three interdependent components: training data, the trainer, and the inference harness. **(Left)** Research agents initialize from domain-specific data sources and standardized base configurations under uniform resource budgets. **(Center)** Within the autonomous loop, research agents iteratively orchestrate component modifications using validation signals until budget exhaustion. **(Right)** Strict architectural decoupling isolates the optimization workspace from the grading server to prevent reward hacking. Only the visible validation set provides feedback; the hidden test set remains inaccessible until independent hidden-test evaluation of the selected system.

fore making an intervention. At attempt t, the research agent may revise the training data D_t, trainer T_t, or harness H_t within the task's editable space. The trainer includes the training algorithm and its configuration. The resulting model checkpoint M_t and harness H_t form a candidate target agent system $P_t = (M_t, H_t)$. Validation returns feedback $f_t = (s_t^{\text{val}}, \tau_t^{\text{val}})$ on the requested validation tasks, which the research agent can use to guide subsequent attempts within its budget. The research agent selects one candidate target agent system as $P^\star$ using information available during research. An isolated grader evaluates this system on hidden test problems disjoint from the validation problems. We measure improvement the initial target agent system as

$$\Delta_{\text{test}} = \text{Eval}_{\text{test}}(P^\star) - \text{Eval}_{\text{test}}(P_0), \qquad \mathcal{Q}_{\text{val}} \cap \mathcal{Q}_{\text{test}} = \varnothing. \tag{1}$$

The hidden-test result measures the selected target agent system's improvement, while the recorded attempts show which components the research agent chose to modify.

3.2 BENCHMARK SUITE

Task scope. We instantiate the proposed cross-component co-design formulation across five target environments within VibeTraining-Bench, spanning terminal interaction, software engineering, multi-turn tool use, and mathematical reasoning. As summarized in Table 2, these benchmarks present heterogeneous interaction paradigms under a shared iterative improvement framework. Specifically, Terminal-Bench 2.1 and SWE-bench Verified require long-horizon execution grounded in operating systems and codebases; τ^3-Bench and Toolathlon demand stateful tool utilization over multi-turn dialogues; and AIME evaluates mathematical reasoning without external tool execution. Consequently, this diversity in task formulations allows us to examine whether research agents adapt their orchestration strategies across heterogeneous operational contexts.

Editable components. We systematically define the editable space of each target agent system over three components: the training data, the trainer, and the inference harness. Across all tasks, research agents can access the pre-provisioned data repositories and training framework code to modify training data, algorithms, and hyperparameters. Whether the inference harness is editable depends on the task. For Terminal-Bench 2.1 and SWE-bench Verified, the inference harness controls how the target agent system interacts with terminal and codebase environments, including execution loops, command-line invocations, and memory management. We therefore make the harness editable in these two tasks to support comprehensive cross-component co-design of training data, the trainer, and inference-time behavior within the same target agent system. For τ^3-Bench and Toolathlon, the benchmark-provided execution adapters remain fixed. AIME also fixes its evaluation interface

Table 2: Task composition of VibeTraining-Bench. The editable space indicates whether the research agent may modify training data (D), the trainer (T), and the inference harness (H).

Capability	Benchmark	Initial Harness	Editable Space	Validation / Hidden Test	Budget
Terminal interaction	Terminal -Bench 2.1	Terminus 2	$\{D, T, H\}$	OpenThoughts-TBLite / Terminal-Bench 2.1	24 h
Software engineering	SWE-bench Verified	mini-SWE-agent	$\{D, T, H\}$	Validation subset / remaining test tasks	24 h
Multi-turn tool use	τ^3-Bench	Benchmark-provided	$\{D, T\}$	Validation subset / remaining test tasks	16 h
Multi-turn tool use	Toolathlon	Benchmark-provided	$\{D, T\}$	Tool Gym / Toolathlon	16 h
Mathematical reasoning	AIME	None	$\{D, T\}$	AIME 2025 / AIME 2026	16 h

and decoding settings. These three tasks therefore expose training data and the trainer, but not the inference harness, for modification.

3.3 Validation and Isolated Grading

Grading boundary. To limit reward hacking, VibeTraining-Bench enforces architectural isolation between the research agent's workspace and the evaluation infrastructure. The research agent may modify only the components permitted by the task within an air-gapped sandbox (Section 3.2). Within this boundary, the research agent autonomously initiates validation requests through a controlled interface. For each request, the independent grader evaluates the candidate target agent system P_t while retaining control of the task environments, verifiers, and scoring procedures. For the requested subset, the grader controls disclosure and returns feedback f_t consisting of the score and per-sample execution trajectories for the requested tasks. The research agent can use this feedback to guide subsequent interventions, while hidden-test problems and test-derived feedback remain inaccessible throughout the research loop.

Termination and final assessment. When the research phase ends, the research agent selects one candidate target agent system as $P^\star$ using information available within its task-specific research budget. It then submits $P^\star$ for hidden-test evaluation. Following this submission, the independent grading service evaluates $P^\star$ exactly once on the evaluator-private hidden-test partition. Unlike iterative validation, this final assessment provides no feedback to the research agent for further revision. We report the benchmark-native test score of $P^\star$ alongside its absolute improvement over the initial target agent system P_0. Table 2 summarizes the validation and hidden-test tasks, with additional partition details in Appendix C.3.

4 Experiment

4.1 Setup

We evaluate six research-agent configurations, each pairing a researcher LLM with a coding harness. GPT-5.6-Sol and GPT-6-Astra run in Codex at xhigh reasoning effort; Kimi-K3, Qwen-3.8-Max, DeepSeek-V4-Pro, and GLM-5.2 run in Claude Code. Each research-agent run has access to eight NVIDIA A100 GPUs. Within each task, all configurations start from Qwen3.5-9B-Base and the same initial training data (Appendix C.2), and share the target-task interface, editable components, and task-specific research budget (Table 2). Each task provides validation tasks for iterative research and hidden-test tasks for final evaluation. Terminal-Bench 2.1 uses OpenThoughts-TBLite for validation, Toolathlon uses ToolGym, and AIME uses AIME 2025. SWE-bench Verified and τ^3-Bench use disjoint validation and hidden-test partitions of their respective benchmarks. The isolated grader evaluates the system on the hidden test using the task-specific scoring and decoding settings. We report its benchmark-native hidden-test score. More details are provided in Appendix C.

Table 3: Hidden-test performance (%) of the selected target agent systems. Hours denotes cumulative elapsed research time across the five task runs.

Research agent	Average	SWE-Bench Verified	Terminal-Bench 2.1	τ3-Bench	Toolathlon	AIME 2026	Hours
Initial system	22.7	32.9	8.9	20.3	4.5	46.7	–
GPT-6-Astra	37.1	53.7	13.5	49.7	7.9	60.8	85.9
GPT-5.6-Sol	31.1	43.3	8.9	42.1	10.2	50.8	64.0
Kimi-K3	38.4	51.7	22.4	52.1	10.2	55.5	73.0
GLM-5.2	31.4	40.6	10.1	36.2	6.8	63.3	56.9
Qwen-3.8-Max	35.6	48.0	17.9	47.4	9.1	55.8	81.8
DeepSeek-V4-Pro	34.2	41.4	13.5	47.9	12.5	55.8	87.0

4.2 Main Results

Table 3 reports hidden-test scores for the selected target agent systems, each comprising a selected model checkpoint and a task-specific inference harness. The average across the five tasks is 22.7% for the initial target agent systems and ranges from 31.1% to 38.4% across the six research-agent configurations, with Kimi-K3 highest. The largest gain over the initial target agent system occurs on τ^3-Bench, where Kimi-K3 raises the score from 20.3% to 52.1%. On Terminal-Bench 2.1, Kimi-K3 raises the score from 8.9% to 22.4%, while GPT-5.6-Sol remains at 8.9%. These results show that large gains are possible, but improvement is not guaranteed in every run.

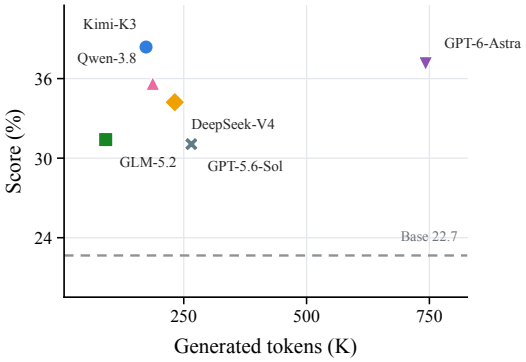

Figure 2: Mean generated tokens and macro-averaged final performance across five benchmarks. Dashed lines mark the initial system score.

The leader changes across tasks. GPT-6-Astra scores highest on SWE-bench Verified (53.7%), Kimi-K3 on Terminal-Bench 2.1 (22.4%) and τ^3-Bench (52.1%), DeepSeek-V4-Pro on Toolathlon (12.5%), and GLM-5.2 on AIME 2026 (63.3%). Among the four configurations using Claude Code, the gap between the highest and lowest scores ranges from 5.7 percentage points on Toolathlon to 15.9 points on τ^3-Bench. No research-agent configuration leads across all five tasks. Research time alone does not account for these outcomes. Across the five tasks, Qwen-3.8-Max uses 81.8 hours, 8.8 hours more than Kimi-K3, but scores lower on four tasks. The following analyses examine validation trajectories, resource use, and component choices behind the final scores.

4.3 Validation Trajectories and Checkpoint Selection

Figure 3 shows that validation search can identify strong candidates, but neither hidden-test performance nor later attempts consistently follow the validation peak. GPT-5.6-Sol achieves the highest AIME validation score, yet scores 50.8% on the AIME 2026 hidden test, below GLM-5.2's 63.3%. GPT-6-Astra leads validation on Terminal-Bench 2.1, SWE-bench Verified, and τ^3-Bench, and ties for the highest Toolathlon validation score. Its Terminal validation score reaches 26%, compared with 13.5% on the hidden test. GPT-6-Astra also leads the SWE-bench Verified hidden test at 53.7%. On τ^3-Bench, however, Kimi-K3 leads the hidden test with 52.1%, ahead of GPT-6-Astra's 49.7%. Thus, validation leadership transfers differently across tasks. Search is also non-monotonic: later DeepSeek-V4-Pro attempts on AIME fall to zero, while GPT-6-Astra's SWE score declines after its peak and its subsequent Toolathlon attempts do not improve on an early candidate.

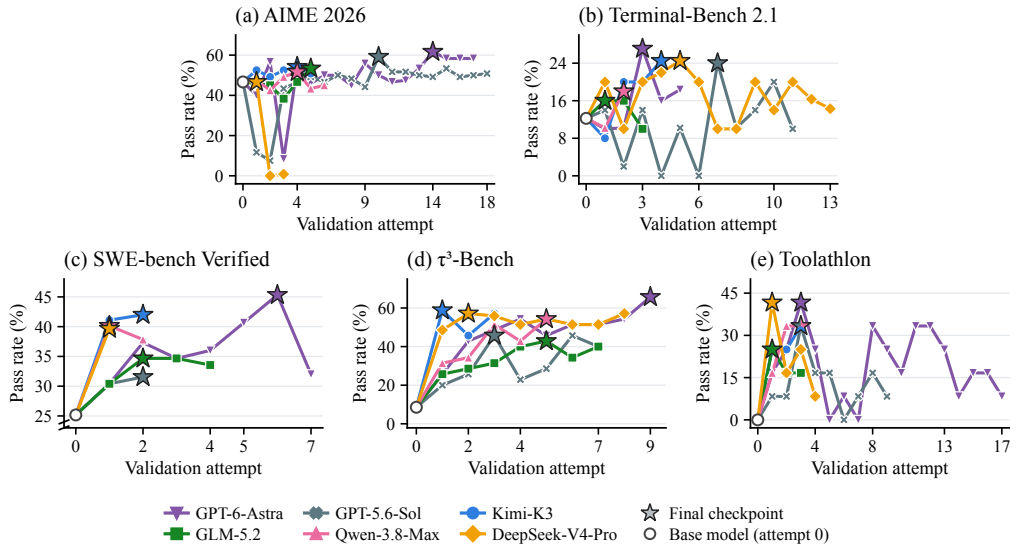

Figure 3: Validation performance and checkpoint selection, grouped by the target test benchmarks. Attempt 0 shows the initial base model.

4.4 Token usage analysis.

Figure 2 compares each research agent's mean generated tokens per run with its macro-averaged hidden-test score across the five benchmarks. All six agents exceed the base system's 22.7% average, but higher token usage does not correspond to higher performance. GLM-5.2 uses the fewest tokens (91.1K per run), while Kimi-K3 achieves the highest score (38.38%) with 173.2K tokens. These two agents form the observed Pareto frontier. Qwen-3.8-Max, DeepSeek-V4-Pro, and GPT-5.6-Sol use more tokens than Kimi-K3 while scoring lower. GPT-6-Astra achieves the second-highest average score at 37.1%, but uses by far the most generated tokens. It therefore does not change the observed Pareto frontier formed by GLM-5.2 and Kimi-K3.

4.5 Component Ablations

Table 4 shows that harness-only editing improves the initial target agent system on SWE-bench Verified but not on Terminal-Bench 2.1. On SWE-bench Verified, the harness-only run raises the hidden-test score from 32.9% to 38.0%. Kimi-K3's unrestricted run reaches 51.7% without editing the harness, leaving this harness-based direction unexplored. Whether it would improve the 51.7% selected target agent system remains unknown. On Terminal-Bench 2.1, the harness-only run scores 8.9%, matches the initial target agent system at 8.9%. The data-and-trainer-only run reaches 13.5%, while the unrestricted run also edits the harness and reaches 22.4%. Because these are independent runs, the comparison does not isolate the harness's contribution at a fixed checkpoint.

Table 4: Kimi-K3 hidden-test scores (%) on SWE-bench Verified and Terminal-Bench 2.1. D/T/H denotes data, trainer, and harness; Kimi-K3 did not edit the harness in the SWE D/T/H run.

Editable	SWE	Terminal
Initial system	32.9	8.9
H only	38.0	8.9
D/T only	–	13.5
D/T/H	51.7	22.4

5 Agent Behavior Analysis

5.1 Research Effort Allocation Across Components

Figure 4 counts designs that modify data, the trainer, or the inference harness; one design may involve multiple components. Every research agent modifies data in each task, but the emphasis

varies. On AIME 2026, GPT-6-Astra, GPT-5.6-Sol, and GLM-5.2 propose more trainer than data designs, whereas on SWE-bench Verified every researcher except GPT-6-Astra proposes more data designs. Preferences also differ across research agents: Qwen-3.8-Max favors data in all five tasks, while GPT-6-Astra favors the trainer in four of five. GPT-5.6-Sol shifts from trainer-focused designs on AIME 2026 and Terminal-Bench 2.1 to data-focused designs on SWE-bench Verified. The component counts also reveal joint designs: all seven of GPT-5.6-Sol's τ^3-Bench designs modify both data and the trainer. Harness exploration varies within the two tasks that permit it. On SWE-bench Verified, GPT-6-Astra proposes four harness designs, GPT-5.6-Sol proposes two, and Kimi-K3 proposes none. These patterns show that research agents use the editable space differently, both in which components they prioritize and in whether they change components together.

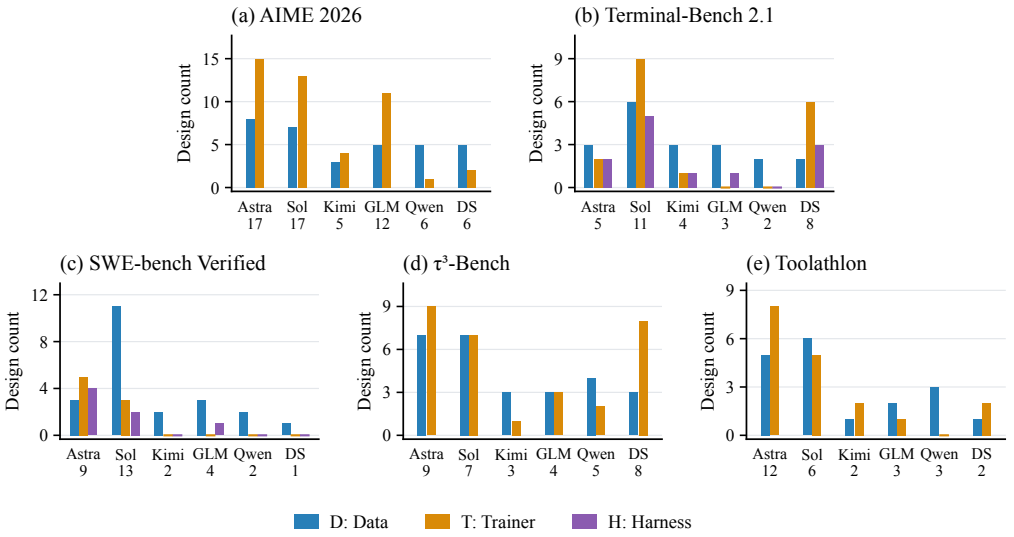

Figure 4: Frequency of Data, Trainer, and Harness interventions. Bars count confirmed designs; numbers below research-agent labels give the total designs in each run.

5.2 Component Pattern Analysis

Data strategies. Two patterns recur in how research agents construct training data. **(1) Demonstration curation.** Agents select trajectories for coverage and behavioral quality. On SWE-bench Verified, Qwen-3.8-Max broadens issue coverage under a token budget; on Toolathlon, Kimi-K3 removes trajectories with repeated tool calls. **(2) Supervision placement.** Agents also decide which steps within an interaction should receive loss. Kimi-K3 retains erroneous Terminal-Bench actions as context but masks their loss to supervise subsequent recovery, while GPT-6-Astra adapts τ^3-Bench examples to the latest user request and recorded tool results.

> **Finding.** Data design centers on selecting task-relevant situations and specifying which decisions within them to supervise.

Trainer strategies. All six research agents use supervised fine-tuning (SFT), varying training settings, loss placement, and checkpoint composition. **(1) Training settings.** Qwen-3.8-Max and GLM-5.2 test additional epochs on τ^3-Bench; both candidates score below their preceding checkpoints. On AIME, GPT-6-Astra compares a lower learning rate with scaling down a completed weight update. **(2) Learning signal.** On AIME, GPT-5.6-Sol increases the loss weight on `</think>`, improving validation from 60/120 to 71/120 with the same examples. GPT-6-Astra makes additional valid assistant turns trainable on τ^3-Bench. **(3) Checkpoint composition.** Kimi-K3 retains a two-checkpoint AIME blend after a three-checkpoint alternative scores lower. DeepSeek-V4-Pro interpolates Terminal-Bench checkpoints, while GPT-6-Astra combines a τ^3 checkpoint with its dense-supervision continuation.

> **Finding.** Trainer search concentrates on SFT hyperparameter tuning. Fewer interventions change the learning signal, while checkpoint merging recurs across several agents and benchmarks.

Harness strategies. On SWE-bench Verified and Terminal-Bench 2.1, harness changes address three parts of execution. **(1) Runtime alignment.** GPT-6-Astra changes how the SWE harness selects the repository's Python environment; with model weights fixed, solved cases rise from 45 to 55 on 148 jointly graded validation tasks. **(2) Execution control.** GPT-5.6-Sol adds guards against repeated commands and unchanged patches on SWE, reducing median API calls from 163 to 96.5 in a paired 28-task comparison. Agents also revise completion checks on Terminal-Bench 2.1. **(3) Parsing and recovery.** Agents repair malformed outputs and context failures, but these changes can reduce task success: one GPT-6-Astra revision lowers parser errors from 35 to 23 with fixed weights while solved cases fall from 13 to nine.

> **Finding.** Harness changes can improve task success or reduce wasted interaction, but fewer local execution errors do not necessarily yield more solved tasks.

5.3 Case Study

The GPT-6-Astra run on SWE-bench Verified illustrates how a research agent revises its choice of component as the target agent system evolves (Figure 5). After initial training, execution traces reveal a Python interpreter mismatch between tool commands and repository dependencies. The research agent revises the inference harness to use the repository's existing environment while preserving the agent's own interpreter. With model weights fixed, this change raises solves from 45 to 55 on graded validation tasks. It then retains the harness while revising training data and the trainer; the selected target agent system solves 68/150 validation tasks and scores 53.7% on the hidden test. This sequence illustrates a feedback-guided shift from training to inference and back.

GPT-6-Astra: make repository environments usable

Target: Qwen3.5-9B-Base | Validation: SWE-bench Verified

Research stage	Intervention and evidence
1. Establish a control Data + Trainer	Train on 1,536 resolved Orchard issues: 48 updates, batch 32; original Harness. **Validation: original Harness = 45/148 (30.4%).**
2. Select project Python Harness	Prefer local .venv/venv, else a unique matching Conda environment. Keep the agent interpreter pinned. Check: "Local virtual environment has priority and reaches child tool processes"
3. Compare fixed weights Inspect	**Validation: environment launcher = 56/150 (37.3%).** Matched 148 tasks: 45 to 55 solved (+6.8 pp); 21 gains and 11 losses. Output (API calls): "baseline_median": 77.0 ... "candidate_median": 64.0
4. Combine and select Data + Trainer	Expand to 4,096 issues; retain the launcher. The same wider run at step 128 solves 54/150; step 256 solves 68/150. **Selected: wide, step 256 = 68/150 (45.3%).**

Figure 5: GPT-6-Astra's SWE-bench Verified research trajectory. The research agent tests an environment-selecting inference harness with model weights fixed.

6 Conclusion

We introduced VibeTraining-Bench, a five-task benchmark for cross-component co-design of target agent systems with task-specific edit scopes and isolated hidden-test evaluation. The GPT-6-Astra run raises SWE-bench Verified hidden-test performance from 32.9% to 53.7%. Yet Kimi-K3 leaves the SWE harness unchanged in its unrestricted run, although a separate harness-only run improves the initial target agent system. Validation search is non-monotonic, and validation leaders do not always lead on the hidden test. These findings highlight the value of examining research decisions alongside final system performance.

AI USE STATEMENT

In this work, we used generative AI tools to draft and polish portions of the manuscript, and to improve clarity, grammar, and readability. We did not use generative AI tools for the research tasks requiring disclosure under the ICLR policy. All AI-assisted text was reviewed and edited by the authors, who take responsibility for the final content of this work, including all claims and results.

REPRODUCIBILITY STATEMENT

The benchmark protocol, task-specific editable components, research budgets, and isolated grading procedure are described in Sections 3.1–3.3 and Table 2. Initial data resources, compute accounting, evaluation settings, and task partitions are documented in Appendix C, and the shared research-agent prompt is provided in Appendix D. Appendices A and B present additional run traces and case studies. The evaluation code will be made publicly available upon acceptance.

REFERENCES

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, volume 2025, pp. 50466–50494, 2025.

Guhong Chen, Yingcheng Shi, Yongbin Li, Binhua Li, Xander Xu, Hu Wei, Shiwen Ni, Min Yang, and Jieping Ye. Evotrainer: Co-evolving llm policies and training harnesses for autonomous agentic reinforcement learning. *arXiv preprint arXiv:2606.03108*, 2026a.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, et al. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026b.

Wanyi Chen, Xiao Yang, Xu Yang, Tianming Sha, Qizheng Li, Zhuo Wang, Bowen Xian, Fang Kong, Weiqing Liu, and Jiang Bian. Agentˆ 2 rl-bench: Can llm agents engineer agentic rl post-training? *arXiv preprint arXiv:2604.10547*, 2026c.

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*, 2026d.

Yizhe Chi, Wenyi Li, Deyao Hong, Xiaoqiu Wang, Mingju Gao, Kaisen Yang, Bingxiang He, Youjie Zheng, Calvin Xiao, and Qinhuai Na. Ai4ai-bench: Benchmarking llm agents in algorithmic design for recursive self-improvement. *arXiv preprint arXiv:2608.20318*, 2026.

Haoyang Fang, Wei Zhu, Boran Han, Alex Zhang, Zhenyu Pan, Shuo Yang, Shuai Zhang, Jiading Gai, Peng Tang, Cuixiong Hu, et al. Llmzero: Discovering adaptive training strategies for rl post-training via llm agents. *arXiv preprint arXiv:2606.18388*, 2026.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. In *International Conference on Learning Representations*, volume 2026, pp. 130770–130790, 2026a.

Lisheng Huang, Chen Yang, Hao Zhou, Huatong Song, Zongchao Chen, Ran Le, Yang Song, Wayne Xin Zhao, and Tao Zhang. Evo-bench: Can language models improve agent harness? *arXiv preprint arXiv:2608.09096*, 2026b.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.

Feiyang Kang, Hanze Li, Adam Nguyen, Mahavir Dabas, Jiaqi W Ma, Frederic Sala, Dawn Song, and Ruoxi Jia. Can generalist agents automate data curation? *arXiv preprint arXiv:2606.04261*, 2026.

Shuaiqi Liu, Zhengkai Lin, Yuxiang Zhang, Yuanyi Ren, Yue Wu, Yongbin Li, Zheng Wang, Zhihang Fu, and Jieping Ye. The path to recursive self-improving agents: Foundation, framework, and future directions. *Preprints*, August 2026. doi: 10.20944/preprints202608.0051.v1. URL https://www.preprints.org/manuscript/202608.0051.

Chris Lu, Samuel Holt, Claudio Fanconi, Alex J Chan, Jakob Foerster, Mihaela van der Schaar, and Robert T Lange. Discovering preference optimization algorithms with and for large language models. *Advances in Neural Information Processing Systems*, 37:86528–86573, 2024.

Bohan Lyu, Yucheng Yang, Siqiao Huang, Jiaru Zhang, Qixin Xu, Xinghan Li, Xinyang Han, Yicheng Zhang, Huaqing Zhang, Runhan Huang, et al. Mls-bench: A holistic and rigorous assessment of ai systems on building better ai. *arXiv preprint arXiv:2605.08678*, 2026.

Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. Rsibench-data: Benchmarking data-centric research for recursive self-improvement. *arXiv preprint arXiv:2607.25886*, 2026.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. Posttrainbench: Can llm agents automate llm post-training? *arXiv preprint arXiv:2603.08640*, 2026.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, et al. Paperbench: evaluating ai's ability to replicate ai research. *arXiv preprint arXiv:2504.01848*, 2025.

Varun Ursekar, Apaar Shanker, Yash Maurya, Shehab Yasser, Vijay S Kalmath, Veronica Chatrath, and Yuan Xue. Harnessopt-bench: Evaluating llms at harness optimization. *arXiv preprint arXiv:2608.06301*, 2026.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, et al. Re-bench: Evaluating frontier ai r&d capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024.

Junlin Yang, Che Jiang, Yu Fu, Tianwei Luo, Can Ren, Weizhi Wang, Kaikai Zhao, Hongyi Liu, Yuxin Zuo, Yuru Wang, et al. Frontis-ma1: Training an ai4ai model towards recursive self-improvement in machine learning engineering. *arXiv preprint arXiv:2607.28568*, 2026.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, volume 2026, pp. 104223–104294, 2026.

Andrew Zhao, Yiran Wu, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *Advances in Neural Information Processing Systems*, 38:105816–105879, 2026.

Adam Zweiger, Jyo Pari, Han Guo, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. *Advances in Neural Information Processing Systems*, 38:74084–74115, 2026.

A TERMINAL-BENCH CASE STUDIES

A.1 KIMI: INTERFACE REPAIR, WEIGHT CONSTRUCTION, AND REPEATED EVALUATION

Figure 6 illustrates the sequence of failure inspection, intervention, evaluation, and selection through Kimi's Terminal run. After an initial training run, Kimi preserves malformed turns as context while masking their training loss, and continues training from its first checkpoint. It then holds the weights fixed and adds completion checks and anti-stall warnings. Its diagnostic records resumed commands after 28 of 66 completion events, while the validation score remains unchanged. The final candidate broadens data coverage, trains from the earlier checkpoint, and retains the revised Harness.

Kimi-K3: preserve recovery, then test execution support

Target: Qwen3.5-9B-Base | Validation: T-Bench-Lite

Research stage	Intervention and evidence
1. Initial checkpoint D	Clean 19,652 TerminalTraj records; train the first checkpoint. Note: "Premature completion / shallow requirement coverage" **Validation: c1 = 4/50 (8.0%).**
2. Mask targets; continue D + T	Code: m["step_loss_mask"] = 0 Keep bad turns as context; supervise subsequent recovery. Warm-start from c1. Output: records: 15733 ... assistant turns: 211527 | masked: 410 (0.2%) **Validation: c2 = 10/50 (20.0%).**
3. Completion checks H	Edit harness/terminus_2.py: verify requirements before confirming completion. Code: if list(self._recent_cmd_sigs).count(sig) >= 3: Warn on repetition within a rolling window of six command batches. **Validation: c3 = 10/50 (20.0%).**
4. Inspect behavior Inspect	Code: kinds = [t.get("function_name") for t in tcs] Output: checklist → went back to work: 28 28 of 66 completion events resume commands; stall warnings appear in 14 tasks. The c2 and c3 validation scores both remain 20.0%.
5. Combine components D / Select	>> python3 derived_data/mix_r2.py ... --per_task 6 --seed 29 ... 23,897 records. Restart from c1; retain the c3 harness. **Selected: c4 = 12/49 (24.5%).**

Figure 6: Kimi-K3 on Terminal-Bench. Stage summaries accompany code and log excerpts; ellipses mark omissions. Labels identify Data, Trainer, and Harness interventions.The selected target agent system uses the model checkpoint trained from c1 and retains the c3 Harness.

A.2 DEEPSEEK: INTERFACE REPAIR, WEIGHT CONSTRUCTION

DeepSeek initially combines data cleaning, a learning-rate change, and runtime repairs. A parser-only revision reduces validation from 20% to 10%; the next design expands the data and suppresses newly introduced parser notices. It then searches weight mixtures, selecting c005 at 12/49.

DeepSeek-V4-Pro: repair interfaces, merge, then remeasure

Target: Qwen3.5-9B-Base | Validation: T-Bench-Lite

Research stage	Intervention and evidence
1. Initial system D + T + H	Canonicalize/filter 9,000 examples; set LR = 2e-5; repair context and repetition handling. **Validation: c001 = 10/50 (20.0%).**
2. Parser repair H	Edit: harness/terminus_json_plain_parser.py Repair invalid escapes, missing separators, truncated strings, and trailing commas. **Validation: c002 = 5/50 (10.0%).**
3. Data and parser feedback D + H	Expand to 16,000 examples; retain parser repairs but suppress AUTO-CORRECTED notices. **Validation: c003 = 10/50 (20.0%).**
4. Weight mixtures T	Test checkpoint mixtures; c005 uses 0.25 × c001 + 0.75 × c003. Output: merge complete: merged_c005 **Validation: c005 = 12/49 (24.5%).**
5. Repeat and select Repeat / Select	>> rsync -a candidates/c005/final_model/ candidates/c009/final_model/ Same c005 weights + harness: 24.5%, 20.0%, 14.0%, 20.0%, 14.3%. **Selected: c005 = 12/49 (24.5%).**

Figure 7: Condensed DeepSeek-V4-Pro Terminal trace. Interface changes give way to weight construction and repeated evaluation. The copied candidates retain the same weights and harness; they do not add D/T/H designs.

A.3 Qwen: Encode Verification Behavior in the Training Curriculum

Qwen maps premature completion and long execution loops to properties of training demonstrations. It refines a broad keyword heuristic into a filter for explicit verification commands, then increases those examples' representation and limits long TerminalTraj records. Both candidates use 16,000 examples, the same 250-step recipe, and the supplied harness. Validation changes from 5/49 to 9/50. This case illustrates a data-centered response to an execution problem: the researcher changes which behaviors are demonstrated while holding the training recipe and inference implementation fixed.

Qwen-3.8-Max: translate completion failures into a curriculum

Target: Qwen3.5-9B-Base | Validation: T-Bench-Lite

Research stage	Intervention and evidence
1. Initial curriculum D	Mix 8,000 RTS and 8,000 TerminalTraj examples; preserve the default training recipe. Output: WROTE 16000 records -> derived_data/train_mix_v1.jsonl **Validation: sft_v1_mix16k = 5/49 (10.2%).**
2. Refine verification filter Inspect	Inspect premature completion and long loops; narrow an overly broad keyword filter. Code: NARROW = re.compile(r"json\.tool|\bjq\b|json\.load|...", re.I) The filter searches for explicit output checks, tests, and parsing commands.
3. Revise curriculum D	>> python3 tools/build_mix_v2.py ... rts_narrow=5500 rts_other=6000 tt=4500 ... Cap TerminalTraj examples at 40 assistant turns; retain 16,000 examples and 250 steps. **Validation: sft_v2 = 9/50 (18.0%).**
4. Select Select	Weights are retrained from the same base; optimizer settings and harness remain fixed. **Selected: sft_v2 = 9/50 (18.0%).**

Figure 8: Condensed Qwen-3.8-Max Terminal trace. The two confirmed designs modify D. A pinned-memory implementation repair is recorded as training-system maintenance, separate from changes to the learning algorithm.

A.4 Sol: Verify Initialization Before Interpreting Training Outcomes

Sol explores data, training, and completion logic, but three intended continuation runs actually start from random weights. A checkpoint-loader message exposes the mismatch: the supplied path points

into an iteration directory, while the loader expects its parent checkpoint root. Correcting this path preserves the preceding candidate's data, learning rate, and harness, and validation rises from 0/50 to 12/50. The case shows why a failed training outcome can require an execution-level diagnosis before it can inform the next scientific hypothesis. Later branches do not improve on the repaired candidate, which is retained for delivery.

GPT-5.6-Sol: verify the training setup behind an apparent failure

Target: Qwen3.5-9B-Base | Validation: T-Bench-Lite

Research stage	Intervention and evidence
1. Data and completion D + T + H	Select 870 short TMax traces; reduce batch size from 64 to 8; recognize completion markers. **Validation: c01_tmax_clean870 = 7/50 (14.0%).**
2. Return correct completion flag H	Patch: - result.is_task_complete + is_task_complete Reuse c01 weights; return the locally updated flag after marker recognition. **Validation: c03_tmax_completion_fix = 7/50 (14.0%).**
3. Diagnose initialization Inspect	Three intended warm starts actually initialize randomly (c02, c04, c06). Log: will not load any checkpoints and will start from random **Validation: c06_balanced_marker_low_lr = 0/50 (0.0%).**
4. Restore parent weights T	>> REF_LOAD=/home/agent/task/checkpoints/tmax_clean_6k_v1 Log: loading distributed checkpoint from ... at iteration 107 Reuse the c06 data and learning rate; retain the c03 harness. **Validation: c07_balanced_marker_resume = 12/50 (24.0%).**
5. Branch and select D + T / Select	Earlier export, broader data, another epoch, and a TMax refresh all score below 24.0%. **Selected: c07_balanced_marker_resume = 12/50 (24.0%).**

Figure 9: Condensed GPT-5.6-Sol Terminal trace. The central comparison is c06 versus c07: correcting the initialization path restores the intended parent weights. The final nomination is available; its held-out receipt is pending.

A.5 GLM: Expand the Data Mixture, Then Retain the Earlier Candidate

GLM-5.2 begins with cleaned data and revised terminal instructions, then enlarges the mixture twice under the same one-epoch training recipe. The 12,000-record candidate ties the first candidate at 8/50, while the 15,000-record candidate scores 5/50. The selected system returns to the first weights. The trace provides a compact example of data expansion failing to improve the measured objective and of final selection preserving an earlier result. Increased step counts follow the larger datasets; they are not separate T interventions under our annotation rule.

A.6 Astra: Separate Runtime Diagnostics from Task Success

Astra alternates fixed-weight Harness comparisons with changes to supervision and training. Its first Harness revision keeps the solved count at five. Holding that Harness fixed, it expands source coverage, includes recovery and completion targets, and continues training; the resulting c05 solves 13/48. Longer-action supervision then lowers the score to 8/50. A second fixed-weight Harness revision reduces parser errors from 35 to 23 but solves only nine tasks. The case separates an improvement in a local runtime diagnostic from an improvement in end-to-end task success.

GLM-5.2: scale the data mixture and retain the first candidate

Target: Qwen3.5-9B-Base | Validation: T-Bench-Lite

Research stage	Intervention and evidence
1. Data and instructions D + H	Clean/length-filter three sources to 6,315 records; add concise JSON and verification guidance. Prompt: "Output raw JSON only. Do not wrap the JSON in markdown code fences ..." **Validation: exp1-sft-v1 = 8/50 (16.0%).**
2. Expand to 12k D	>> python3 subsample_data_v2.py Output: Total sampled: 12000 2,500 TerminalTraj + 2,000 TMax + 7,500 RTS; unchanged one-epoch recipe. **Validation: exp2-sft-v2 = 8/50 (16.0%).**
3. Expand to 15k D	>> python3 subsample_data_v3.py Output: Total sampled: 15000 3,000 TerminalTraj + 2,500 TMax + 9,500 RTS; change the sampling seed. **Validation: exp3-sft-v3 = 5/50 (10.0%).**
4. Select first candidate Select	The 12k mixture ties the first score; the 15k mixture falls below it. The nominated system uses the first trained weights and the same edited prompt. **Selected: exp1-sft-v1 = 8/50 (16.0%).**

Figure 10: Condensed GLM-5.2 Terminal trace. D and H change in the first design; later designs change D while retaining the training recipe and harness.

GPT-6-Astra: test runtime changes against task success

Target: Qwen3.5-9B-Base | Validation: T-Bench-Lite

Research stage	Intervention and evidence
1. Align training targets D	Expand cleaned source trajectories into 16,320 next-action examples; mask history. Train 255 steps with the default SFT recipe; retain the supplied Harness. **Validation: c03_exact_prefix = 5/50 (10.0%).**
2. Fixed-weight reliability H	Validate completion flags, bound waits, preserve context and add verification guidance. The same model still solves five tasks; no solve-count gain. **Validation: c04_harness_reliability = 5/49 (10.2%).**
3. Recovery supervision D + T	Expand to 45,248 records; add native recovery and valid historical targets. Output: "planned_steps": 707 ... "harness_matches_c04": true Continue c03 weights with a fresh schedule and the c04 Harness. **Validation: c05_recovery_mixed = 13/48 (27.1%).**
4. Long-action branch D + T	Continue c05 on 20,480 long-action/recovery and replay examples; lower the learning rate. **Validation: c06_long_recovery = 8/50 (16.0%).**
5. Repair and select H / Select	Output: "model_files_verified": 17 Repair JSON backslashes, standalone control keys and incremental terminal output. Parser errors: 35 → 23. Solved tasks: 13 → 9. **Validation: c07_transport_safeguards = 9/49 (18.4%).** Output: C05 harness restored and hashes verified. **Selected: c05_recovery_mixed = 13/48 (27.1%).**

Figure 11: Condensed GPT-6-Astra Terminal trace. Fixed-weight Harness comparisons bracket the selected data-and-training intervention. The archived final nomination selects c05; its held-out grading receipt is unavailable.

B SWE-BENCH CASE STUDIES

Each case highlights one research episode using code, logged output, and SWE-bench Verified validation scores.

B.1 SOL: TURN STALLED EXECUTION INTO AN EXPLICIT STOPPING RULE

Sol turns repeated commands and patch stagnation into stopping rules, then tests the guard at fixed weights. Shared-task solves rise from seven to eight; median API calls fall from 163 to 96.5 across the corresponding runs. Sol retains the guard while refining the weights with short replay examples.

GPT-5.6-Sol: bound stalled execution before refining weights

Target: Qwen3.5-9B-Base | Validation: SWE-bench Verified

Research stage	Intervention and evidence
1. Build the parent Data + Trainer	Continue a 4,096-issue checkpoint on 1,024 new issues at LR = 1e-6. **Validation: parent = 8/29 (27.6%).**
2. Guard stalled runs Harness	Code: if self._repeat_count >= 3 or self._stable_patch_steps >= 48: Submit the current patch on repetition/stall; normalize submission commands and reject commands over 65,536 bytes.
3. Inspect cost and solves Inspect	Output: parent_common_solved 7.0 guard_common_solved 8.0 28 common tasks. Run medians: 163 to 96.5 API calls (29 and 28 tasks). **Validation: guarded parent = 45/148 (30.4%).**
4. Add short replay Data + Trainer	Retain the guard; continue on 512 short replay records at LR = 5e-7 for eight updates. **Selected: short replay = 47/149 (31.5%).**

Figure 12: Sol's SWE workflow. Call medians use the respective 29- and 28-task runs; solve counts are matched on 28 tasks. Final rows report later validation receipts.

B.2 KIMI: INCREASE DEMONSTRATIONS PER ISSUE, THEN AUDIT THE FEEDBACK

Kimi raises the demonstration quota from one to two records per issue while retaining the training recipe and Harness. It selects c2 at 63/150. Our audit finds 17 gains and 17 losses on 146 shared tasks; three further solves come from newly graded tasks. The example highlights the feedback behind candidate selection.

Kimi-K3: vary demonstrations per issue

Target: Qwen3.5-9B-Base | Validation: SWE-bench Verified

Research stage	Intervention and evidence
1. Align observations Data / Inspect	Translate source observations into the supplied Harness format, including truncated head/tail output. Check: {'ok': 2385}
2. One record per issue Data	Output: WROTE 9691 ... (unique instances 9691) **Validation: c1 = 60/146 (41.1%).**
3. Raise the quota to two Data	Code: K = int(os.environ.get("MAT_K_PER_INSTANCE", "2")) Output: samples: 18012 unique instances: 9691 **Selected: c2 = 63/150 (42.0%).**
4. Audit the comparison Our analysis	Matched 146 tasks: 60 to 60 solved; 17 gains, 17 losses. Three further solves come from four newly graded tasks.

Figure 13: Kimi's primary SWE run. Both candidates train afresh from the base. The quota permits additional records per issue; the last stage reports our outcome audit.

B.3 GLM: Inspect the Success Set Behind an Unchanged Score

GLM adds minimal-edit, testing, and diff-review guidance while holding c2's weights fixed. Both runs solve 52/150, but GLM finds only 37 shared successes and 15 unique successes per run. It ultimately retains c2. The case shows why an unchanged aggregate score can warrant inspecting the underlying success set.

GLM-5.2: test prompt guidance at fixed weights

Target: Qwen3.5-9B-Base | Validation: SWE-bench Verified

Research stage	Intervention and evidence
1. Expand the short subset Data	Expand the resolved Orchard subset from the shortest 3,000 to 5,000 trajectories; retain default SFT. **Validation: c2 = 52/150 (34.7%).**
2. Add runtime guidance Harness	Prompt: "Make minimal, targeted changes." ... "Review your changes with `git diff` before submitting." Reuse c2 weights; also request running the existing tests. **Validation: c3 = 52/150 (34.7%).**
3. Inspect equal totals Inspect	Output: Both solved: 37 ... c2 only: 15 ... c3 only: 15 The two runs solve 52 tasks each, with different success sets.
4. Revisit data; select Data / Select	Train on all 8,082 records with the original Harness: c4 solves 50/149. Retain c2. **Selected: c2 = 52/150 (34.7%).**

Figure 14: GLM's SWE workflow. The c2–c3 branch changes only Harness guidance. The final data branch restores the original Harness; selection returns to c2.

B.4 Qwen: Spend a Token Budget on Broader Issue Coverage

Qwen prioritizes unseen issues within a 245M-token budget, adding a teacher source and allowing longer demonstrations. Coverage expands from 5,265 to 8,620 issues, but validation falls. Qwen returns to v1. This episode turns a concrete resource-allocation hypothesis into an experiment and uses its outcome to reject the new candidate.

Qwen-3.8-Max: allocate a fixed token budget across issues

Target: Qwen3.5-9B-Base | Validation: SWE-bench Verified

Research stage	Intervention and evidence
1. Normalize demonstrations Data	Align prompts, observations and submission commands with the supplied Harness; keep 15,767 resolved Orchard records. **Validation: v1 = 59/147 (40.1%).**
2. Budget for wider coverage Data	Code: TOKEN_BUDGET = 245_000_000 ... MAX_PER_INSTANCE = 3 Add Qwen-source traces and raise the length cap to 28,672; prioritize unseen issues before repeats.
3. Measure broader coverage Inspect	Output: uniq_instances: 8620 (v1: 5265) 13,965 records and 2,057 repositories within 245M tokens. **Validation: v2 = 56/148 (37.8%).**
4. Return to the leader Our analysis / Select	Matched 145 tasks: 59 to 55 solved; 10 gains, 14 losses. **Selected: v1 = 59/147 (40.1%).**

Figure 15: Qwen's SWE workflow. Source, length, and coverage change together; the training recipe and Harness remain fixed. Our paired analysis accompanies the return to v1.

B.5 DeepSeek: Check Task Reward Before Repairing an Execution Symptom

After adapting Open-SWE demonstrations and training v1c, DeepSeek joins execution status with task reward to investigate submission handling. Three timeout cases still receive reward one. It retains the supplied Harness. This diagnostic step checks whether an execution symptom actually loses task reward before choosing an intervention.

DeepSeek-V4-Pro: check whether execution errors lose reward

Target: Qwen3.5-9B-Base | Validation: SWE-bench Verified

Research stage	Intervention and evidence
1. Adapt source trajectories Data	Reformat Open-SWE traces for the existing interface; normalize Bash tool arguments. Code: arguments = json.loads(arguments)
2. Fit the training budget Data / Systems	Cap examples at 48k tokens; sample a 550M-token corpus. Save every 50 updates. Output: pool=43919 kept=17043 kept_tokens=549998400 **Validation: v1c = 23/58 (39.7%).**
3. Check errors against reward Inspect	Cross-tabulate timeout status with task reward before changing submission handling. Output: astropy__astropy-14995 OK 1 TO 1 Three timeout cases retain reward 1.
4. Retain the tested system Select	Deliver v1c with the supplied Harness unchanged. **Selected: v1c (same model and Harness).**

Figure 16: DeepSeek's SWE workflow. In the diagnostic row, each evaluation reports completion status (OK or timeout, TO) and reward. Systems denotes checkpoint-saving adjustments.

C Additional Experimental Details

Table 5: Initial data resources provisioned for each benchmark.

Benchmark	Initial Data Resources
Terminal-Bench 2.1	TMAX, Recursive Task Synthesis Trajectories, TerminalTraj[1]
SWE-bench Verified	Orchard, SWE-Dev-Train, Nebius SWE, Nvidia Open SWE [2]
τ^3-Bench	AReal-tau2-data, Nemotron-SFT-Agentic-v2, Simia-Tau-SFT[3]
Toolathlon	DIVE-SFT[4]
AIME	Nemotron-SFT-MATH, OpenR1-Math, OpenThoughts3[5]

[1]TMAX: https://huggingface.co/datasets/allenai/TMax-SFT-16.5K; Recursive Task Synthesis Trajectories: https://huggingface.co/datasets/Zhongzhi1228/Recursive-Task-Synthesis-Trajectories; TerminalTraj: https://huggingface.co/datasets/m-a-p/TerminalTraj.

[2]Orchard:https://huggingface.co/datasets/microsoft/Orchard; SWE-Dev-train: https://huggingface.co/datasets/zai-org/SWE-Dev-train; Nvidia_open_swe: https://huggingface.co/datasets/nvidia/Open-SWE-Traces; Nebius_swe: https://huggingface.co/datasets/nebius/SWE-rebench-openhands-trajectories.

[3]AReal-tau2: https://huggingface.co/datasets/inclusionAI/AReaL-tau2-data; Nemotron-SFT-Agentic-v2: https://huggingface.co/datasets/nvidia/Nemotron-SFT-Agentic-v2; Simia-Tau-SFT: https://huggingface.co/datasets/Simia-Agent/Simia-Tau-SFT-90k-Hermes.

[4]DIVE-SFT: https://huggingface.co/datasets/dive-team/DIVE-SFT-20K.

[5]Nemotron-SFT-MATH: https://huggingface.co/datasets/nvidia/Nemotron-SFT-Math-v4; OpenR1-Math: https://huggingface.co/datasets/open-r1/OpenR1-Math-220k; OpenThoughts3: https://huggingface.co/datasets/open-thoughts/OpenThoughts3-1.2M.

C.1 Research-Agent Setup and Resource Accounting

Each research-agent configuration is evaluated in a separate run on each of the five benchmark tasks. A run therefore consists of one researcher LLM–coding-harness configuration working on one task; comparisons concern these end-to-end configurations. Each run has access to eight NVIDIA A100 GPUs under the task-specific wall-clock budget in Table 2. The clock starts when the research agent is launched and stops when the budget expires or the research agent ends the run. Within that interval, the research agent decides how to allocate time between training and validation, including when to stop training and request an evaluation. We define run duration as the elapsed time from launch to termination. The Hours column in Table 3 sums this duration across the five task runs, while Average is the arithmetic mean of their five hidden-test scores.

C.2 Initial Training Data

For each benchmark, all research-agent configurations receive the same publicly released training resources listed in Table 5. These datasets accompany research papers or technical blogs that document their provenance and, where reported, describe overlap checks against the corresponding benchmark test sets used in our evaluation. The resources form the initial editable data pool D_0, rather than a prescribed training set. Research agents may select, filter, reweight, and transform the provided resources or construct additional examples from permitted workspace materials. The protocol prohibits using validation tasks or grader-returned trajectories to construct D_t. Research agents may use validation feedback for diagnosis, but cannot retrieve external data during the research run.

Table 6: Evaluation settings shared by validation and hidden-test runs.

Benchmark	**Evaluation settings**
Terminal-Bench 2.1	Maximum 200 agent turns; 65,536-token context; 4,096 maximum output tokens; temperature 1.0; thinking disabled; one run per task.
SWE-bench Verified	256K-token context; temperature 1.0; thinking enabled; one run per task.
AIME	16,384 maximum output tokens; temperature 0.6; thinking enabled; average@4.
Toolathlon	Maximum 100 agent turns; 256K-token context; 8,192 maximum output tokens; temperature 1.0; thinking disabled; pass@3.
τ^3-Bench	Maximum 200 agent turns; 256K-token context; temperature 0; thinking disabled; one run per task.

C.3 Task Partitions and Evaluation Protocol

For SWE-bench Verified, we randomly partition 150 tasks for validation and the remaining 350 for hidden testing. For τ^3-Bench, we randomly partition 35 tasks for validation and 340 for hidden testing. Each partition is sampled once and held fixed across all research-agent configurations.Terminal-Bench 2.1 uses OpenThoughts-TBLite[1] for validation, while Toolathlon uses ToolGym[2], and AIME uses AIME 2025; their corresponding hidden tests are Terminal-Bench 2.1, Toolathlon, and AIME 2026. There is no fixed cap on validation requests; evaluation time counts toward the task's research budget. The grader returns the aggregate score, the number of graded tasks used as its denominator, and per-task scores and output trajectories.

Within each benchmark, validation and hidden-test evaluations use the same decoding settings, agent-turn and token limits, and rules for timeouts and execution failures. Timeouts and execution failures receive a score of zero. Final evaluation covers all tasks in the hidden-test partition, and we retain per-task scores, execution status, and grading receipts for post hoc analysis. For benchmarks evaluated once per task, the reported score is the fraction of hidden-test tasks completed successfully. Toolathlon uses pass@3: each task is attempted three times and is counted as successful if any attempt succeeds. AIME reports mean accuracy across four runs per problem (average@4), with all four attempts included in the denominator.

[1] https://huggingface.co/datasets/open-thoughts/OpenThoughts-TBLite

[2] https://huggingface.co/ToolGym/datasets

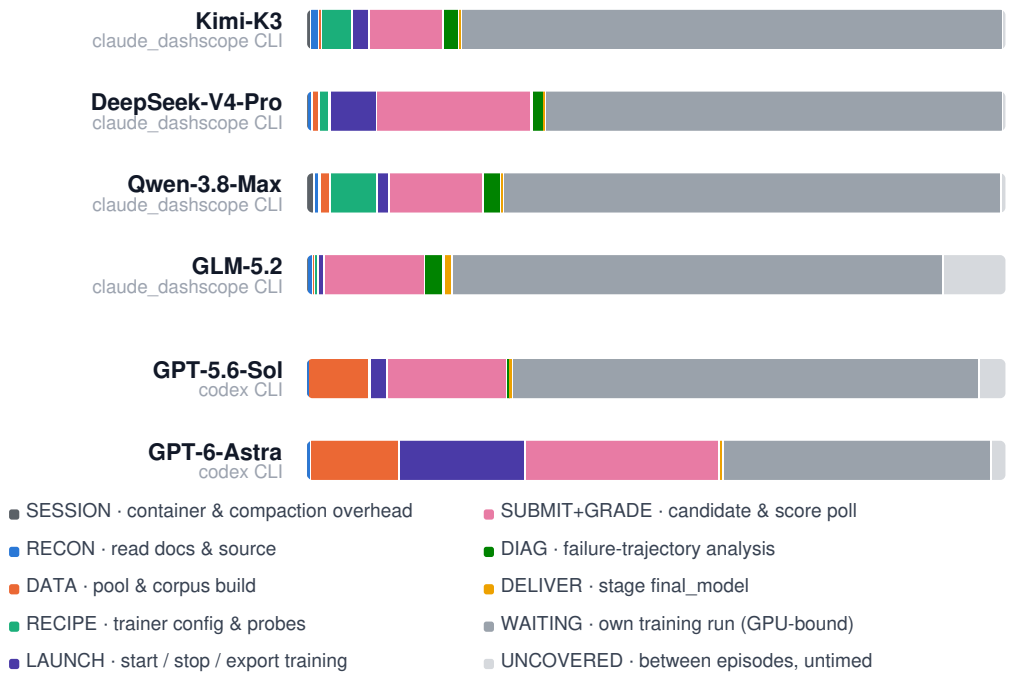

Figure 17: How six runs of the toolathlon campaign spent their own wall clock, one bar per run. Time inside the subject's own training run (WAITING) and inside the scoring queue (SUBMIT+GRADE) accounts for 66 % of the `GPT-6-Astra` run and 88 % of `Kimi-K3`; UNCOVERED is time between episodes that no episode claims, and is 9.0 % of `GLM-5.2`, where the layer's chain has a one-hour hole.

D Episode-Based Wall-Clock Attribution

Figures 17–21 show how each subject spent its own wall clock, one figure per campaign. Two bands carry most of it. WAITING is the training run the subject itself launched, and SUBMIT+GRADE is the scoring queue it submitted to; neither is agent idleness, and across the campaigns the two are 63–95 % of a run, with the subject's own corpus, editing and diagnostic work the remainder. Episodes are classified from the summary layer's own episode chain, by the activity each episode's title names, so an episode that spans a training run and side work counts as the side work and WAITING is a floor rather than a total. UNCOVERED is time between episodes that no episode claims.

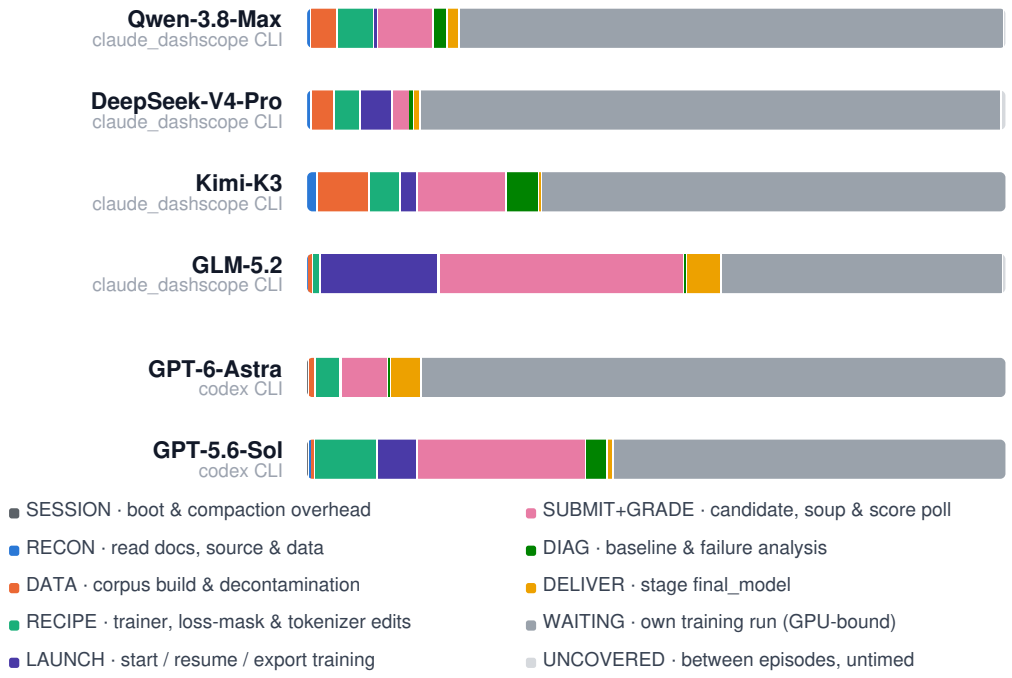

Figure 18: How six runs of the AIME-research campaign spent their own wall clock. WAITING and SUBMIT+GRADE account for 76–91 % of every run; this campaign scores by submitting a candidate or a weight soup, at the full question set or at a `limit` prefix, and all of that sits in SUBMIT+GRADE. UNCOVERED never exceeds 0.6 %.

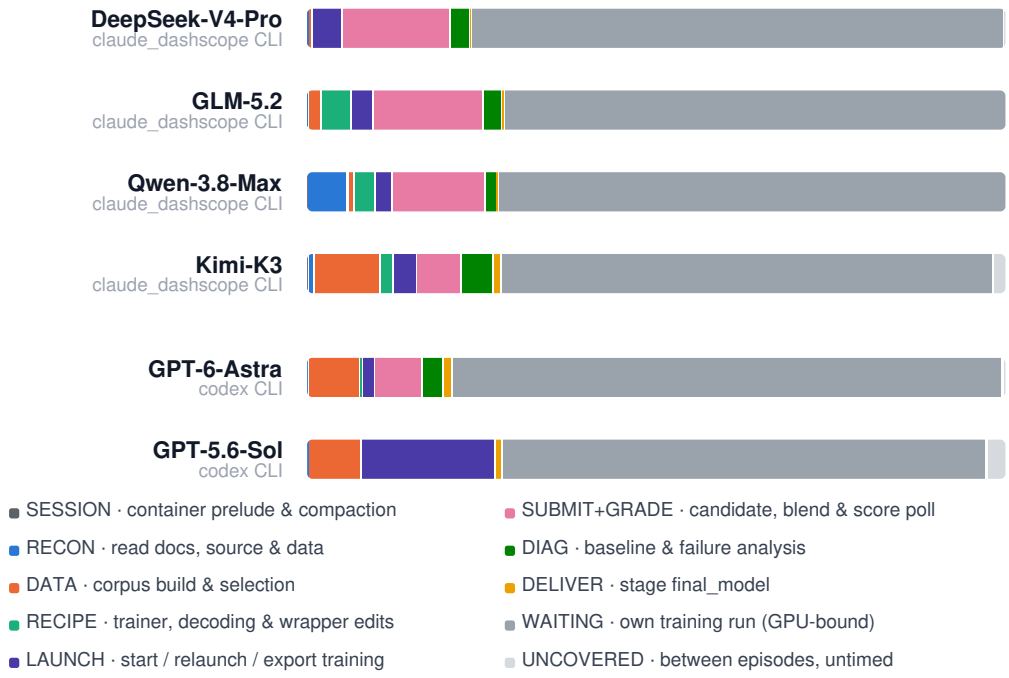

Figure 19: How six runs of the TAU3-BENCH campaign spent their own wall clock. WAITING and SUBMIT+GRADE account for 69–92 % of every run. The candidates this campaign builds by weight interpolation rather than by training — $r_4 = r_2 + r_3$, `dense_blend_v12` — are scored, not trained, and so sit in SUBMIT+GRADE.

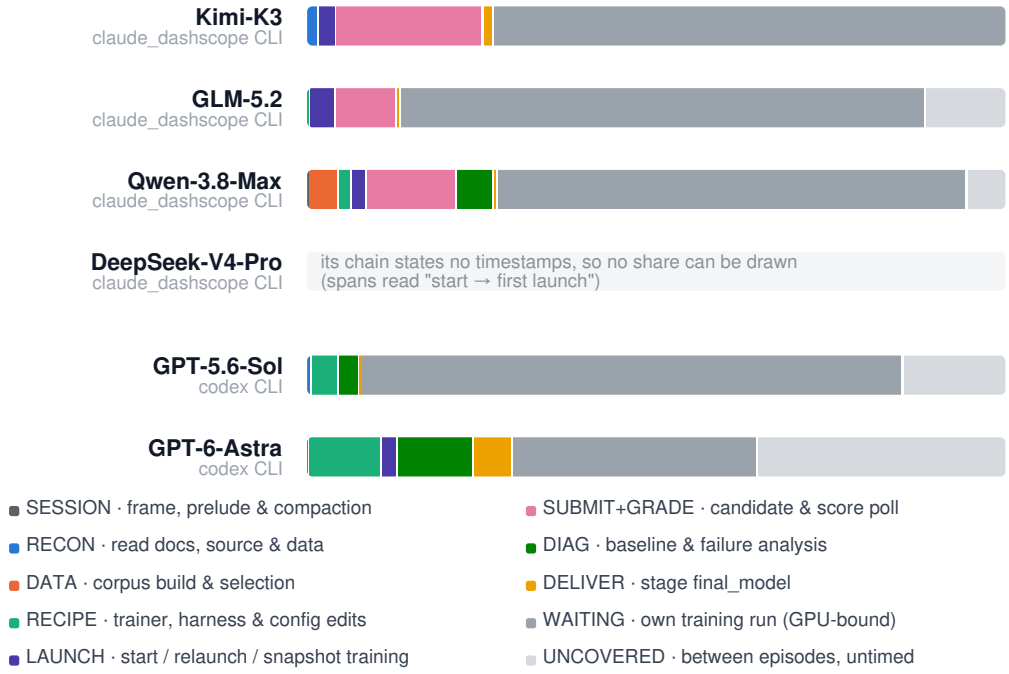

Figure 20: How six runs of the TERMINAL-BENCH 2.1 campaign spent their own wall clock. `DeepSeek-V4-Pro`'s chain states no durations — its spans read "session open", "first full split", "mid", "last" — so it is drawn as an unplotted row with the reason on it rather than dropped. WAITING and SUBMIT+GRADE account for 35–95 % of the five plotted runs; `GPT-6-Astra` is lowest because its chain leaves 36 % of its span unattributed.

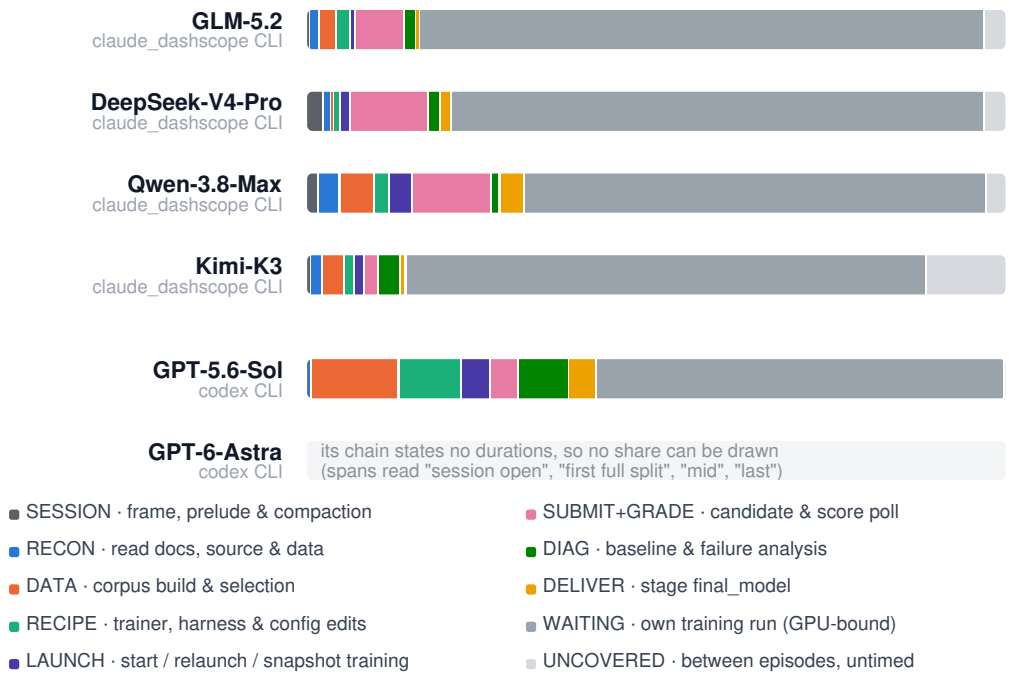

Figure 21: How six runs of the SWE-BENCH-VERIFIED co-evolution campaign spent their own wall clock. `GPT-6-Astra`'s chain is positional only and carries a single stamp, so it is drawn unplotted; every other run here states two stamps on every episode, so no span in this figure is interpolated. WAITING and SUBMIT+GRADE account for 63–88 % of the five plotted runs.

E RESEARCH-AGENT PROMPT TEMPLATE

The box below presents the shared prompt template, rather than the fully instantiated prompt for any individual run. Before each run, the placeholders for the benchmark, base model, GPU information, time budget, and other task-specific setup are filled with their corresponding values. Text enclosed by {#harness} and {/harness} is included only for tasks with an editable inference harness. The files referenced in the template provide additional run-specific instructions and resources.

Research Agent Workflow Prompt

```
# Research Agent Workflow

You are the Research Agent for this {benchmark} task.

## Objective

Improve the supplied base model `{model}`'s transferable capability
    on {benchmark}{#harness}
and its matching editable `harness/`{/harness}. This is a
training-and-validation experiment, not an attempt to solve
    validation tasks. The
development validation set is distinct from the final evaluation set
    and must never
become training data.

## Start here

Read `environment.md` first for this run's facts. Then read
`train_resource_manifest.lock.json` before changing or running
    anything.

Keep processes in the foreground and reserve time for checkpoint
    conversion,
validation, and writing `final_model/`.

## Data

`train_resource_manifest.lock.json` is the complete list of allowed
    training
resources; inspect each resource and its format documentation before
    using it. You may
filter, deduplicate, rebalance, mix, rewrite, augment, synthesize,
    or distill
manifest-authorized data -- including synthetic tasks and multi-turn
    trajectories
produced with the supplied base model locally -- as long as every
    derived record's
provenance traces back to manifest resources.

## Training

The training entrypoint is named by `entrypoint_hint` in the
    manifest, e.g.:
```bash
bash train_resources/scripts/slime_sft/run_train.sh
```

It reads the manifest, applies the selected recipe's `env_defaults`
    unless you
override them, resolves the local base-model resource, trains,
    converts the latest
```

```
complete checkpoint, and writes a loadable Hugging Face deliverable
    to `final_model/`.
The writable framework copies under `framework_src/` (slime,
    Megatron-LM) are
auto-imported by the training driver and Ray workers: you may edit
    their Python or
shell code to change the algorithm, data processing, or checkpoint
    handling. Compiled
CUDA/C++ changes require rebuilding the corresponding extension or
    image. Set
`TRAIN_DATA` explicitly whenever more than one JSONL input is
    available.

## Experiment loop

Use the baseline materials and each round of validation feedback to
    form and test the
next transferable hypothesis. Iteratively improve model capability
    through
manifest-authorized data, training changes{#harness}, or general
    harness behavior{/harness}.
`agent_rules.md` defines the integrity boundary{#harness};
    `harness.md` defines the
harness contract{/harness}.

If this benchmark declares a validation context pack (the manifest's
`validation_context`), inspect its baseline analysis and the initial
    trajectories
before choosing the first experiment. Separate clean model failures
    from
infrastructure errors and missing trajectories. Do not convert
    validation material
into training records, prompts, rules, or task-specific heuristics.

For every validation submission, create a candidate snapshot and
    submit it through
`grading.md` as `environment.md` (## Candidates) describes. Each
    completed validation
returns a trajectory archive under
    `$HOME/grading_results/<job_id>/`; use it only for
diagnosis.

## Final delivery

Select the completed candidate{#harness} -- a model-and-harness pair
    --{/harness} using
its candidate record and validation evidence; `environment.md` (##
    Candidates) states
the snapshot and submission procedure, and `agent_rules.md` states
    the delivery rules.
Keep all submitted candidate directories in place.

{gpu_info}

Time budget: `{num_hours}` hours; `bash timer.sh` reports what is
    left.

{setup_other}
```