RSIBENCH-DATA: BENCHMARKING DATA-CENTRIC RESEARCH FOR RECURSIVE SELF-IMPROVEMENT

Anonymous authors
Paper under double-blind review

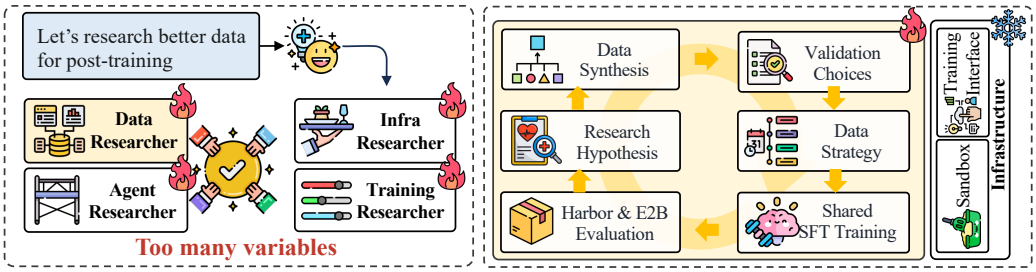
ABSTRACT

Recursive self-improvement requires systems that can turn evidence about model failures into better models. A central part of this process is data-centric post-training research: diagnosing a capability gap, designing and validating a training-data strategy, and learning from checkpoint feedback. Can LLM agents automate this data-centric improvement loop? Existing automated post-training benchmarks entangle research decisions with optimization, serving, evaluation, and system implementation, making it difficult to isolate the agent's research capability. To fill this gap, we introduce RSIBENCH-DATA, a controlled benchmark evaluating LLM agents as data-centric researchers while fixing the post-training stack. Specifically, agents are required to iteratively develop and revise training-data strategies for a fixed target model, while training and serving use shared TINKER-backed services, official evaluation runs through HARBOR-orchestrated E2B sandboxes, and resource budgets remain fixed across agents. We evaluate four frontier agents across six benchmarks spanning software engineering, terminal use, science question answering, and mathematics. LLM agents already demonstrate some of the core capabilities of data-centric researchers: in 58.33% of settings, they improve upon the first valid attempt by refining their training-data strategies from feedback. However, they do not improve consistently from feedback: among searches that continue after reaching their best observed score, 78.26% end with a lower-scoring final attempt, while the rest only recover the same peak. Thus, a strong candidate can appear early or midway through a run even as later feedback-driven revisions fail to improve it. Trajectory analysis identifies four patterns in stronger runs: accurate hypotheses, validation-grounded supervision, behavior-aligned data, and preservation of strong checkpoints. These findings suggest that current agents can make useful data-centric research discoveries but cannot yet translate feedback into improvements consistently. Overall, RSIBENCH-DATA provides a measurable and auditable testbed for studying the data-centric research capabilities required for recursive self-improvement. We opensource our code at [URL withheld for anonymous review].

1 INTRODUCTION

Recursive self-improvement requires an AI system to repeatedly turn evidence about model failures into effective model improvements. Recently, LLM agents are increasingly evaluated on tasks that require long-horizon interaction with software repositories, terminals, and external tools (Jimenez et al., 2024; Merrill et al., 2026; Yao et al., 2025). Unlike static input–output tasks, these environments expose the agent's intermediate plans, tool calls, observations, recovery attempts, and verifier outcomes (Guo et al., 2024; Chen et al., 2025; 2026a). The resulting trajectories make failures richly observable, but they remain evidence about model behavior rather than training experience that can directly improve it (Barke et al., 2026). To address this gap, recent work addresses different parts of this evidence-to-training pipeline (Gadre et al., 2023; Pan et al., 2025; Rank et al., 2026). SWE-Gym(Pan et al., 2025) and R2E-Gym (Jain et al., 2025) provide executable environments and training resources for software-engineering agents. DataComp (Gadre et al., 2023) and DataComp-LM (Li et al., 2024) study data selection under a controlled learning stack. PostTrainBench (Rank et al., 2026) and Agent^2 RL-Bench (Chen et al., 2026b) instead evaluate broader automated post-training systems.

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.



(a) Existing agentic post-taining benchmarks (b) RSIBench-Data

Figure 1: Existing evaluations entangle multiple post-training choices; RSIBENCH-DATA isolates the LLM agent's ability to conduct data-centric post-training research.

These paradigms address broader questions spanning data selection, environment construction, and end-to-end post-training, but they do not isolate a central research problem for recursive self-improvement: *can LLM agents reliably automate data-centric post-training research?* Specifically, to achieve this goal, the agent must formulate a hypothesis about a model failure, turn it into executable and verifiable training data, and revise the synthesis strategy from controlled training and evaluation feedback (Pan et al., 2025; Jain et al., 2025). We call an LLM agent that carries out this closed-loop process a *data-centric researcher agent*. However, existing benchmarks entangle research decisions with optimization, serving, evaluation, and system implementation, making it difficult to isolate the agent's research capability (Figure 1(a)).

To address this gap, we introduce RSIBENCH-DATA, which isolates the data-centric research component of recursive self-improvement under a fixed surrounding stack (Figure 1(b)). Specifically, we fix the base model and bounded training interface, expose training and serving through shared TINKER services, and isolate official evaluation behind HARBOR-orchestrated E2B sandboxes under a common budget. More importantly, RSIBENCH-DATA evaluates an iterative training-data research process, not imitation-based dataset generation. In each round, the agent decides what the target model should learn, develops and validates a training-data strategy, trains a checkpoint through shared LoRA SFT, and uses controlled evaluation feedback to choose the next strategy.

Our results reveal a **discovery–reliability gap**. Across four frontier agents and six benchmarks, agents improve on their first valid attempt in 58.33% of settings by refining training-data strategies from feedback. This result shows that agents already exhibit some core capabilities of data-centric researchers. However, these gains are task dependent and fragile. No agent consistently dominates: Codex `gpt-5.6-sol` leads the three non-SWE benchmarks, but the three SWE-style tasks are still won by three different agents. Moreover, agents do not improve consistently from feedback: among searches that continue after reaching their best observed score, 78.26% finish with a lower-scoring final attempt, while the remainder only recover the same peak. An early or intermediate attempt can therefore remain strongest even after the agent observes additional feedback and revises its strategy. The round-by-round records reveal recurring failures: agents misdiagnose target capabilities, generate misaligned supervision, search past strong checkpoints, or fail to turn new feedback into a stronger training-data strategy.

Overall, our contributions are threefold:

- **A new evaluation target: data-centric research for recursive self-improvement.** We formulate post-training data synthesis as an automatable research loop in which an LLM agent forms capability hypotheses, designs and validates training-data strategies, learns from checkpoint evidence, and manages candidate selection under a fixed budget.
- **A controlled benchmark for data-centric researcher agents.** We introduce RSIBENCH-DATA, with shared TINKER training and serving and official evaluation through HARBOR and E2B sandboxes. These boundaries make experiments auditable and separate infrastructure changes from research progress.
- **Empirical evidence: researcher capabilities emerge, but remain fragile.** Agents improve on the first valid attempt in 58.33% of settings, yet among searches that continue after reaching a peak, 78.26% finish with a lower-scoring attempt and the rest only recover that

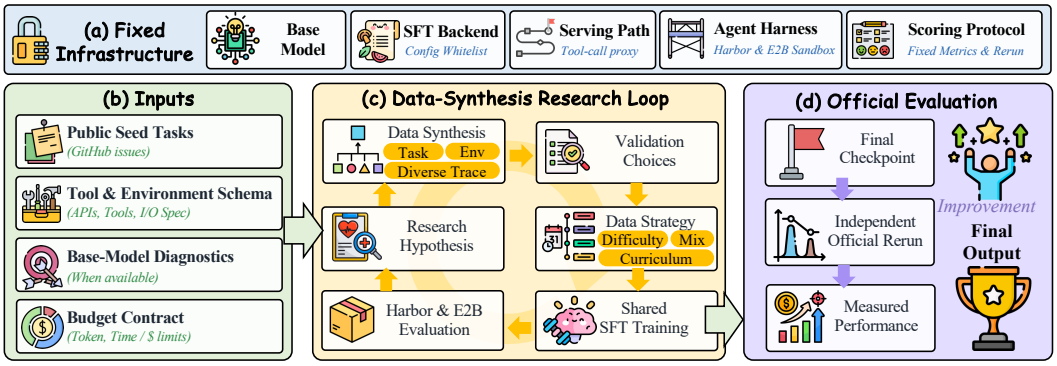


Figure 2: Overview of RSIBENCH-DATA, which evaluates how an LLM researcher agent evolves a training-data research policy under shared training, serving, evaluation, and budget controls.

peak. This non-monotonicity shows that agents do not consistently turn feedback into better training-data strategies. Stronger runs feature accurate hypotheses, validation-grounded supervision, behavior-aligned data, and checkpoint preservation.

2 RSIBENCH-DATA: EVALUATION TASK AND PROTOCOL

Conventional data-generation evaluations ask whether a system can produce a useful dataset or checkpoint. RSIBENCH-DATA asks a research question central to recursive self-improvement: can an LLM agent form a useful hypothesis about a fixed model, design a targeted training-data strategy, interpret checkpoint evidence, and preserve its best finding under a finite budget?

2.1 PROBLEM DEFINITION

We evaluate data-centric research capability through the final checkpoint and the experimental process that produces it. We study a bounded, cross-model setting in which a frontier researcher agent iteratively improves a separate target model through data-centric post-training. Official checkpoint performance remains the primary outcome; the recorded process supports attribution and diagnosis rather than a separate subjective score. *Training experience* includes examples, executable tasks and states, tool-use trajectories, and verifier-grounded supervision. Formally, given a fixed base model M_0, benchmark evidence S, and resource budget $\mathcal{C}$, a researcher policy π observes S and its attempt history $H_{<t}$, then proposes training data D_t and a whitelisted training config c_t:

$$(D_t, c_t) = \pi(S, H_{<t}), \qquad M_t = \mathrm{Train}(M_0, D_t; c_t). \tag{1}$$

Each M_t is evaluated by a fixed evaluation service, producing permitted selection feedback $h_t = \mathrm{Eval}_{\mathrm{sel}}(M_t)$, such as scores, trajectories, verifier outcomes, and execution diagnostics. Proposals and feedback form H_t, which the agent uses to revise or stop within budget $\mathcal{C}$.

After T realized attempts, the agent selects a checkpoint $M_{t^\star}$ using only H_T, before observing official outcomes. The evaluator then assesses it in fresh task environments without further training. We report its official performance and, secondarily, improvement over the fixed base model:

$$s_{\mathrm{off}}(\pi) = \mathrm{Eval}_{\mathrm{off}}(M_{t^\star}), \qquad \Delta_{\mathrm{off}}(\pi) = s_{\mathrm{off}}(\pi) - \mathrm{Eval}_{\mathrm{off}}(M_0). \tag{2}$$

Two separations support attribution: agents modify training experience through the bounded interface but not the surrounding stack, and use selection feedback, not official outcomes, for iteration and checkpoint selection. These separations are instantiated through fixed infrastructure, benchmark inputs, a closed-loop data-synthesis research process, and a separate official evaluation (Figure 2).

2.2 FIXED INFRASTRUCTURE

RSIBENCH-DATA fixes the infrastructure that turns a training-data strategy into an evaluated checkpoint, so that a researcher agent works inside a fixed boundary rather than over the whole post-training stack. Inside the boundary, the agent decides what training experience to construct and,

through a bounded configuration interface, how to expose it to training. Outside it, the base model, optimizer, serving path, evaluation sandbox, verifier, and scoring rule are fixed and shared by every agent: the agent submits training artifacts rather than training code, and may not replace any component of the surrounding stack.

Every attempt therefore follows the same path from submitted data to evaluated checkpoint. In each round, the agent submits message-format supervision together with a whitelisted training configuration. The runner validates these artifacts and invokes the same TINKER-backed LoRA SFT backend from the fixed base model, and the resulting checkpoint is served through the same sampler interface that the evaluator uses to query it. Execution is then delegated to HARBOR in E2B sandbox environments through a fixed service boundary that owns the benchmark environment, runner, verifier, and scoring rule, and returns controlled evidence to the agent. Optimization, serving, and evaluation are thus identical across agents, while the submitted data distribution and the configuration within the whitelist remain free to vary.

Because data submission, training, serving, and evaluation all pass through this boundary, each attempt leaves a structured record that links a training-data strategy to its checkpoint and evaluation behavior. This makes the loop auditable: we can analyze why a run improves or regresses, and attribute performance differences to the agent's training-data research policy rather than to changes in the post-training stack.

2.3 AGENT INPUTS AND DATA-USE CONSTRAINTS

Each agent receives the target benchmark description, success criteria, task-matched public seed repositories or seed examples, tool and environment schemas, available base-model diagnostics, and a resource budget. These inputs form S without providing a ready-made training corpus (Figure 2(b)). Seed repositories provide realistic task structure and a reference for the target distribution: without such grounding, synthesis can collapse to simple rule-based examples that are detached from real tasks and circumvent the intended research challenge.

Not all information available to the agent may be included in D_t. Each benchmark specifies which seeds and public sources may support data construction; evaluation-only tasks, labels, trajectories, and other protected benchmark materials may not be used as supervision. This separation allows the agent to diagnose failures without converting evaluation information into training data.

2.4 CLOSED-LOOP DATA-SYNTHESIS RESEARCH

RSIBENCH-DATA fixes only the outer sequence and service boundaries of the budgeted loop: the agent synthesizes and submits data, the benchmark trains a checkpoint through the shared SFT service, and the fixed sandboxed evaluator returns permitted signals (Figure 2(c)). Filtering, verification, curriculum design, data mixing, failure attribution, and the decision to continue searching are not prescribed benchmark stages; they are choices available to the researcher agent's data-synthesis policy. The budget constrains total resources, while the agent decides how training and evaluation feedback should change the next training-data strategy. Appendix B describes representative strategies rather than mandatory phases.

Data synthesis. Within each round, the agent may choose data sources, task and environment construction, trajectory representation, filtering rules, verification methods, difficulty curricula, data mixtures, and training exposure. The benchmark neither requires a fixed preliminary diagnosis nor prescribes an internal order for these operations; it requires only that the submitted artifacts satisfy the data contract and can be consumed by the shared SFT interface. After training and evaluation, the agent decides whether to continue and how to evolve its next training-data strategy.

Shared SFT training. The benchmark validates the submitted artifacts and configuration, then trains a LoRA checkpoint from M_0 through the shared SFT backend. All agents use the same training implementation and the same bounded configuration interface. This converts each proposed training distribution into a checkpoint without allowing the agent to replace the optimization backend or otherwise redesign the learning stack.

Harbor and E2B evaluation. The checkpoint is queried through the fixed serving path and evaluated by HARBOR in E2B sandbox environments with the official verifier and scoring protocol. The service returns the permitted selection score together with task trajectories, verifier outcomes, execution and infrastructure errors, token usage, training and sampling cost, and elapsed time. Because evaluation is fixed, these signals provide comparable evidence about the submitted training-data strategy.

2.5 FINAL CHECKPOINT AND OFFICIAL EVALUATION

After completing its attempts, the agent selects one checkpoint from those it has produced. We evaluate the selected checkpoint on the target benchmark's test set using the fixed evaluation protocol and report the resulting performance as the agent's final score.

3 EXPERIMENTAL SETUP

To isolate data-centric research capability, we instantiate a controlled 4×6 matrix: the researcher agent and target benchmark vary, while the target model, bounded training interface, evaluation services, and per-run budget remain fixed.

3.1 RESEARCHER AGENTS

We evaluate four frontier LLM researcher agents in the main benchmark matrix: Claude Code (Anthropic, 2025) with `Opus-4.8` and `Sonnet-5`, and Codex (OpenAI, 2025) with `gpt-5.6-sol` and `gpt-5.6-terra`. The Claude Code agents run at high reasoning effort, while the Codex agents run at max reasoning effort; all agents receive the same task interface, data-use constraints, and budget. Across all experiments, Claude Opus 4.8 is fixed as the external rollout model for generating reasoning traces, tool-use sequences, or full trajectories. Agents submit training data D_t together with a whitelisted configuration c_t; our comparisons therefore evaluate data-centric research through the shared bounded interface of Section 2, rather than data alone under a fully frozen training recipe. We separately include a high-versus-max reasoning-effort comparison for one Claude Code `Sonnet-5` setting in Section 4.3.

3.2 SHARED TRAINING PIPELINE

Each attempt trains a LoRA-adapted checkpoint of `Qwen/Qwen3.5-35B-A3B-Base` (Yang et al., 2025) through the shared TINKER SFT backend (Thinking Machines Lab, 2025)[1]. A HARBOR-orchestrated evaluation service executes each checkpoint in E2B sandboxes and returns the controlled evidence used for subsequent research decisions and checkpoint choice. The selected checkpoint is then evaluated without additional training through the same fixed service boundary[2]. Each main run has a nominal 16-hour wall-clock budget and a $500 TINKER budget.

3.3 BENCHMARKS AND EVALUATION

We select six benchmarks spanning distinct training-experience regimes. The three SWE-BENCH variants—Verified (Jimenez et al., 2024), Multilingual (Zan et al., 2025), and Pro (Deng et al., 2025)—require repository-grounded supervision for code modification and task completion. Terminal-Bench 2.0 (Merrill et al., 2026) emphasizes long-horizon tool use and verification, whereas GPQA Diamond (Rein et al., 2023) and AIME 2026[3] emphasize verifiable reasoning and difficulty control.

For each benchmark, all agents share the same evaluation subset, runner, verifier, and parameters. The SWE-BENCH variants use MINI-SWE-AGENT (Yang et al., 2024)[4], while the remaining benchmarks use TERMINUS-2 (Merrill et al., 2026). The three SWE-BENCH variants and

[1] `https://github.com/thinking-machines-lab/tinker`.
[2] `https://github.com/e2b-dev`.
[3] `https://huggingface.co/datasets/MathArena/aime_2026`.
[4] `https://github.com/SWE-agent/mini-swe-agent`.

GPQA Diamond use fixed 100-instance subsets; Terminal-Bench 2.0 and AIME 2026 use their full sets, with four decoding runs per AIME problem. Evaluation hyperparameters, including temperature, follow each benchmark's official settings. Appendix A reports the full evaluation suite and the public seed repositories available to the researcher agents.

The primary outcome is the selected checkpoint's official benchmark-native score. Secondary analyses track improvement over the fixed base model, within-run selection trajectories, the first and best candidates, training and sampling cost, and wall-clock time.

4 Results and Analysis

4.1 Main Results

RSIBench-Data exposes two sides of autonomous data-centric research. First, iteration can produce genuine discoveries: later candidates improve over the first valid candidate in 14 of 24 settings. Second, agents do not consistently convert subsequent feedback into further gains: among the 23 settings that continue after reaching an observed peak, 18 finish with a final attempted candidate below their historical best and 5 only return to the same peak. Historical-best checkpoint selection can preserve an earlier discovery despite these later regressions, but the trajectories still show that feedback-driven revisions often fail to refresh the candidate frontier. We therefore analyze the benchmark in three stages: task-dependent official performance, the consistency and efficiency of feedback-driven improvement, and the research decisions associated with stronger trajectories.

Performance analysis. Official scores and resource profiles for every agent–benchmark setting are summarized in Table 1. Each score comes from a pre-specified representative run. Performance remains weak on several executable tasks: the best agent reaches only 9.00% on SWE-bench Pro, 22.00% on SWE-bench Multilingual, and 20.22% on Terminal-Bench 2.0. These row-wise ceilings indicate a shared weakness across current researcher agents. Some trained checkpoints also fail to improve over the base model, especially on GPQA Diamond and SWE-bench Multilingual, showing that training-data strategies can be neutral or harmful as well as helpful. At the same time, no researcher agent dominates the suite. Claude Code `Opus-4.8` leads on SWE-bench Verified, Claude Code `Sonnet-5` on SWE-bench Multilingual, and Codex `gpt-5.6-sol` on SWE-bench Pro, GPQA Diamond, AIME 2026, and Terminal-Bench 2.0. The three SWE-style tasks are still won by three different agents, showing that broad task-family similarity does not induce a stable ranking. Among the four trained agents, the best-to-worst spread further confirms the strength of this interaction: it is 8.00 percentage points on SWE-bench Pro, 13.00 on GPQA Diamond and SWE-bench Verified, 14.60 on Terminal-Bench 2.0, 17.00 on SWE-bench Multilingual, and 20.00 on AIME 2026. Because every run uses the same target model and bounded interface, these gaps reflect the interaction between researcher-agent policy and benchmark structure, not variation in the target model or evaluation stack.

Table 1: Official performance and resource use for the main Agent–Benchmark matrix. The base-model row reports the unadapted `Qwen/Qwen3.5-35B-A3B-Base` score under the same evaluation split and sampling protocol. Time and TINKER cost report the resources used by each setting. Bold marks the best official score within each benchmark.

Benchmark	Agent	Official	Time (h)	TINKER cost ($)
SWE-BENCH Verified	Base model	12.00%	–	–
	Claude Code `Opus-4.8`	**46.00%**	10.21	195.90
	Claude Code `Sonnet-5`	35.00%	14.91	181.70
	Codex `gpt-5.6-sol`	33.00%	4.41	55.61
	Codex `gpt-5.6-terra`	42.00%	2.80	59.79
SWE-BENCH Multilingual	Base model	7.00%	–	–
	Claude Code `Opus-4.8`	5.00%	12.68	195.70
	Claude Code `Sonnet-5`	**22.00%**	14.20	363.77
	Codex `gpt-5.6-sol`	15.00%	5.99	78.64
	Codex `gpt-5.6-terra`	6.00%	5.18	56.87
SWE-BENCH Pro	Base model	0.00%	–	–
	Claude Code `Opus-4.8`	2.00%	2.19	17.54
	Claude Code `Sonnet-5`	4.00%	9.54	45.63
	Codex `gpt-5.6-sol`	**9.00%**	11.61	300.49
	Codex `gpt-5.6-terra`	1.00%	3.85	34.39
GPQA Diamond	Base model	61.00%	–	–
	Claude Code `Opus-4.8`	56.00%	5.98	27.92
	Claude Code `Sonnet-5`	52.00%	6.43	16.03
	Codex `gpt-5.6-sol`	**65.00%**	2.42	10.37
	Codex `gpt-5.6-terra`	64.00%	1.14	4.80
AIME 2026	Base model	30.00%	–	–
	Claude Code `Opus-4.8`	40.83%	4.89	61.36
	Claude Code `Sonnet-5`	49.17%	9.29	121.21
	Codex `gpt-5.6-sol`	**53.33%**	8.90	63.65
	Codex `gpt-5.6-terra`	33.33%	1.85	8.50
Terminal-Bench 2.0	Base model	1.12%	–	–
	Claude Code `Opus-4.8`	10.11%	8.99	175.63
	Claude Code `Sonnet-5`	5.62%	8.87	156.93
	Codex `gpt-5.6-sol`	**20.22%**	9.56	69.07
	Codex `gpt-5.6-terra`	12.36%	6.67	186.96

Resource analysis. The resource columns show that high official scores are not obtained at a uniform cost. The 24 main runs have a median total wall-clock time of 6.55 hours and a median valid-candidate TINKER cost of $62.51, but the spread is large: time ranges from 1.14 to 14.91 hours, and valid-candidate cost ranges from $4.80 to $363.77. Several high-scoring runs are comparatively inexpensive. Codex `gpt-5.6-sol` reaches 65.00% on GPQA Diamond in 2.42 hours with $10.37 of TINKER cost, and Codex `gpt-5.6-terra` reaches 64.00% on the same benchmark in 1.14 hours with $4.80. Other settings consume far more budget for modest or weak outcomes: Claude Code `Sonnet-5` on Terminal-Bench 2.0 uses 8.87 hours and $156.93 but scores 5.62%, while Codex `gpt-5.6-sol` on SWE-BENCH Pro spends $300.49 to reach 9.00%. These contrasts show that official score should be interpreted together with search time and training cost.

4.2 ITERATIVE EVOLUTION AND CANDIDATE SELECTION

Iteration improves the candidate frontier in most settings. A final official score evaluates only the selected checkpoint and therefore does not reveal whether iterative feedback improved upon the agent's first valid candidate. We therefore compare the first and best valid candidates across the 24 settings with valid official evaluations, excluding failed attempts and attempts without selection scores. We evaluate all candidates within a run using the same selection protocol. A later candidate outperforms the first in 14 of 24 settings, whereas in the other 10 settings the first candidate remains

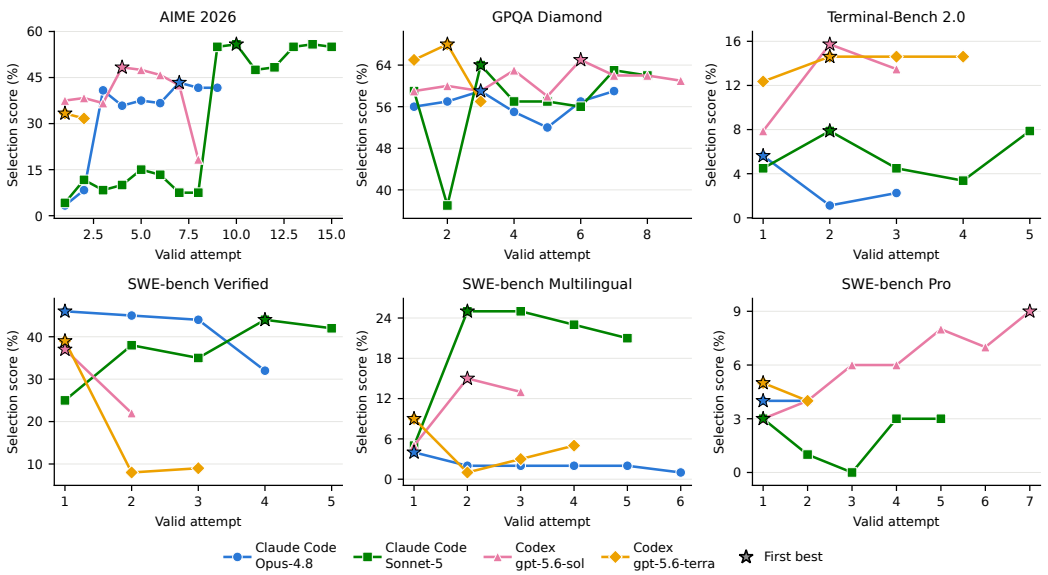


Figure 3: Selection-score trajectories across valid attempts in their original order. Stars mark the first candidate that reaches the historical-best selection score.

the best. Thus, iterative feedback often helps agents discover a stronger candidate, but not in every setting (Figure 3). Official evaluation is conducted separately on the selected checkpoint.

Feedback-driven improvement is rarely monotonic. Among the 23 settings that continue after first reaching their best selection score, 18 finish with a final attempted candidate below that peak and the remaining 5 only return to it. This statistic describes the sequence of attempted candidates, not the checkpoint ultimately submitted: historical-best selection can preserve an earlier peak even when later attempts score lower. The result instead shows that agents do not consistently turn additional feedback into better training-data strategies. In some settings the first candidate remains strongest; in others, an intermediate improvement is followed by weaker revisions. Historical-best selection mitigates the effect of such regressions on final submission, while effective stopping and strategy revision remain necessary for efficient search.

The value and speed of iteration are strongly benchmark dependent. To distinguish where iteration helps from how quickly it helps, we compare first-to-best gains and time to peak. AIME 2026 improves for three of four agents, with a median gain of 25.42 percentage points: Claude Code `Sonnet-5` stays below 15.00% for eight valid attempts before jumping to 55.00% and peaking at 55.83%, whereas Codex `gpt-5.6-terra`'s first candidate is already its best. GPQA Diamond begins from a stronger frontier (56.00%–65.00%) and gains only 4.00 points at the median, even though all four agents refresh their first-candidate score. Iteration value therefore reflects both available headroom and how efficiently an agent converts feedback into a new training-data strategy.

Broad task families do not determine a common evolution trajectory. To test whether task families induce common search dynamics, we compare the three SWE-style profiles and all four agents on Terminal-Bench 2.0. SWE-BENCH Verified improves for one agent, SWE-BENCH Multilingual for two, and SWE-BENCH Pro accumulates gains only for Codex `gpt-5.6-sol`. On Terminal-Bench 2.0, Claude Code `Sonnet-5` improves from 4.49% to 7.87%, Codex `gpt-5.6-sol` recovers from an initial failed attempt and rises from 7.87% to 15.73% before falling to 13.48%, Codex `gpt-5.6-terra` improves from 12.36% to a 14.61% plateau, and Claude Code `Opus-4.8` peaks immediately at 5.62%. These differences make the agent–benchmark interaction, rather than the family label, the relevant unit of analysis.

4.3 MECHANISM ANALYSIS

Mechanism analysis links score changes to concrete data-synthesis research decisions. To move beyond final-score comparisons, we code each attempt by its diagnosed capability gap, data source, validation signal, and supervision while the training and evaluation stack remains fixed. We compare these decisions with selection and official scores to identify the cross-case mechanisms in Table 4. Appendix B.6 provides the full trajectories and case-level analysis.

Higher reasoning effort improves this diagnostic setting by changing the training-data research policy. We compare the Claude Code `Sonnet-5` researcher agent on SWE-BENCH Verified under high and max reasoning effort, holding the rollout model, target model, benchmark subset, budgets, and training–evaluation pipeline fixed. Under this diagnostic (Figure 6), max effort raises the mean selection score over the first four comparable attempts from 35.5% to 44.0%, improves the first candidate from 25% to 36%, and moves the historical best from 44% to 49%. The official score also increases from 35% to 52%, although this single-run difference should not be interpreted as a fully isolated causal estimate.

The resource profile clarifies the mechanism. Compared with high effort, max effort completes one fewer valid attempt, but its selected dataset is substantially larger: 149 training records rather than 50, 2,681 trainable assistant turns rather than 595, and $\$401.70$ rather than $\$181.70$ in candidate-loop TINKER cost. The result is therefore consistent with a depth–breadth tradeoff: higher effort finds a stronger candidate earlier and constructs a more resource-intensive training-data program, while reducing the number of candidates that fit within the fixed budget.

Effective evolution combines sound data decisions with disciplined search. Across the main trajectories, improvements are associated with diagnosing the target capability, grounding data in validation signals, matching supervision to evaluated behavior, and preserving strong checkpoints. The reasoning-effort diagnostic further suggests that deeper data construction can trade search breadth for candidate quality. Together, these patterns support closed-loop training-data research rather than imitation-based generation.

Further cost and early RSI analyses appear in Appendices D and F, respectively.

5 DISCUSSION AND LIMITATIONS

Official evaluation does not yet establish adaptive generalization. Our evaluation prioritizes broad coverage under substantial sandbox and training costs, using one representative run per setting and fixed subsets for four benchmarks. Checkpoint selection and official evaluation use the same task subset, so the reported scores measure performance in fresh environments but not generalization to statistically held-out tasks. Repeated trials, evaluator-private splits, and refreshed tasks are needed to measure the stability and transferability of agentic training-data research strategies. Researcher identity also bundles the underlying LLM, agent scaffold, and configured reasoning effort, so the main matrix compares complete researcher systems rather than isolating any one component.

6 CONCLUSION

We introduced RSIBENCH-DATA, a controlled benchmark for evaluating the data-centric research capability required for recursive self-improvement under a fixed post-training stack. Across four frontier agents and six benchmarks, agents exhibit some core capabilities of data-centric researchers: iteration improves on the first valid attempt in 58.33% of settings. Yet agents do not improve consistently from feedback: among searches that continue after reaching their best observed score, 78.26% finish with a lower-scoring final attempt and the rest only recover the same peak. Historical-best selection can protect the submitted checkpoint from these regressions, but it does not make the underlying research process reliably progressive. Together, these results reveal a discovery–reliability gap in how agents translate feedback into better training-data strategies. We hope RSIBENCH-DATA can help develop more dependable data-centric researcher agents for recursive self-improvement by making their research process measurable and auditable.

AI USE STATEMENT

We used generative AI tools to polish the language of the manuscript and assist with writing code. OpenAI Codex also assisted with LaTeX formatting and compilation checks. The benchmark itself uses LLM-based researcher agents to develop and revise synthetic training-data strategies and a fixed LLM rollout model to produce training experience, as described in Sections 3 and 4. The authors take responsibility for the final manuscript, code, and reported results.

REFERENCES

Anthropic. Claude code. https://github.com/anthropics/claude-code, 2025. Agentic coding tool.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. SWE-rebench: An Automated Pipeline for Task Collection and Decontaminated Evaluation of Software Engineering Agents, 2025. URL https://arxiv.org/abs/2505.20411.

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. AgentRx: Diagnosing AI Agent Failures from Execution Trajectories. *arXiv preprint arXiv:2602.02475*, 2026. URL https://arxiv.org/abs/2602.02475.

Jonathan D. Chang, Andrew Drozdov, Shubham Toshniwal, Owen Oertell, Alexander Trott, Jacob Portes, Abhay Gupta, Pallavi Koppol, Ashutosh Baheti, Sean Kulinski, Ivan Zhou, Irene Dea, Krista Opsahl-Ong, Simon Favreau-Lessard, Sean Owen, Jose Javier Gonzalez Ortiz, Arnav Singhvi, Xabi Andrade, Cindy Wang, Kartik Sreenivasan, Sam Havens, Jialu Liu, Peyton DeNiro, Wen Sun, Michael Bendersky, and Jonathan Frankle. KARL: Knowledge Agents via Reinforcement Learning, 2026. URL https://arxiv.org/abs/2603.05218.

Qiguang Chen, Mingda Yang, Libo Qin, Jinhao Liu, Zheng Yan, Jiannan Guan, Dengyun Peng, Yiyan Ji, Hanjing Li, Mengkang Hu, Yimeng Zhang, Yihao Liang, Yuhang Zhou, Jiaqi Wang, Zhi Chen, and Wanxiang Che. AI4Research: A Survey of Artificial Intelligence for Scientific Research, 2025. URL https://arxiv.org/abs/2507.01903.

Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. Towards Reasoning Era: A Survey of Long Chain-of-Thought for Reasoning Large Language Models. *Science China Information Sciences*, 69 (6):161101, 2026a. doi: 10.1007/s11432-025-4665-8. URL https://doi.org/10.1007/s11432-025-4665-8.

Wanyi Chen, Xiao Yang, Xu Yang, Tianming Sha, Qizheng Li, Zhuo Wang, Bowen Xian, Fang Kong, Weiqing Liu, and Jiang Bian. Agent^2 RL-Bench: Can LLM Agents Engineer Agentic RL Post-Training?, 2026b. URL https://arxiv.org/abs/2604.10547.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks?, 2025. URL https://arxiv.org/abs/2509.16941.

Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, Eyal Orgad, Rahim Entezari, Giannis Daras, Sarah Pratt, Vivek Ramanujan, Yonatan Bitton, Kalyani Marathe, Stephen Mussmann, Richard Vencu, Mehdi Cherti, Ranjay Krishna, Pang Wei Koh, Olga Saukh, Alexander J Ratner, Shuran Song, Hannaneh Hajishirzi, Ali Farhadi, Romain Beaumont, Sewoong Oh, Alex Dimakis, Jenia Jitsev, Yair Carmon, Vaishaal Shankar, and Ludwig Schmidt. DataComp: In search of the next generation of multimodal datasets. In *Advances in Neural Information Processing Systems*, volume 36, pp. 27092–27112. Curran Associates, Inc., 2023. doi: 10.52202/075280-1179. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/56332d41d55ad7ad8024aac625881be7-Paper-Datasets_and_Benchmarks.pdf.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large Language Model based Multi-Agents: A Survey of Progress and Challenges, 2024. URL https://arxiv.org/abs/2402.01680.

Yihao Hu, Zhihao Wen, Xiujin Liu, Pan Wang, Xin Zhang, and Wei Wu. SEAL: Synergistic Co-Evolution of Agents and Learning Environments, 2026. URL https://arxiv.org/abs/2605.24426.

Benhao Huang, Yingzhuo Yu, Jin Huang, Xingjian Zhang, and Jiaqi Ma. DCA-Bench: A Benchmark for Dataset Curation Agents, 2024. URL https://arxiv.org/abs/2406.07275.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural Environments and Hybrid Verifiers for Scaling Open-Weights SWE Agents, 2025. URL https://arxiv.org/abs/2504.07164.

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?, 2024. URL https://arxiv.org/abs/2310.06770.

Feiyang Kang, Hanze Li, Adam Nguyen, Mahavir Dabas, Jiaqi W. Ma, Frederic Sala, Dawn Song, and Ruoxi Jia. Can Generalist Agents Automate Data Curation?, 2026. URL https://arxiv.org/abs/2606.04261.

Zaid Khan, Elias Stengel-Eskin, Jaemin Cho, and Mohit Bansal. DataEnvGym: Data Generation Agents in Teacher Environments with Student Feedback. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.06215.

Seungone Kim, Juyoung Suk, Xiang Yue, Vijay Viswanathan, Seongyun Lee, Yizhong Wang, Kiril Gashteovski, Carolin Lawrence, Sean Welleck, and Graham Neubig. Evaluating Language Models as Synthetic Data Generators, 2025. URL https://arxiv.org/abs/2412.03679.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. DataComp-LM: In search of the next generation of training sets for language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 14200–14282. Curran Associates, Inc., 2024. doi: 10.52202/079017-0455. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/19e4ea30dded58259665db375885e412-Paper-Datasets_and_Benchmarks_Track.pdf.

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

OpenAI. Codex CLI: A lightweight coding agent for the terminal. https://github.com/openai/codex, 2025. Terminal-based coding agent.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training Software Engineering Agents and Verifiers with SWE-Gym. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47717–47737. PMLR, 2025. URL https://proceedings.mlr.press/v267/pan25g.html.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, Jiacheng Zhu, Xuan Jiang, Sirui Li, Cathy Wu, Bryan Kian Hsiang Low, Jinhua Zhao, and Paul Pu Liang. CORAL: Towards Autonomous Multi-Agent Evolution for Open-Ended Discovery, 2026. URL https://arxiv.org/abs/2604.01658.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. PostTrainBench: Can LLM Agents Automate LLM Post-Training?, 2026. URL https://arxiv.org/abs/2603.08640.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Yuval Shalev, Zifeng Ding, and Mateja Jamnik. Training Language Agents to Learn from Experience, 2026. URL https://arxiv.org/abs/2605.20477.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. AgentBank: Towards Generalized LLM Agents via Fine-Tuning on 50000+ Interaction Trajectories, 2024. URL https://arxiv.org/abs/2410.07706.

Thinking Machines Lab. Tinker: A training API for fine-tuning language models. https://thinkingmachines.ai/tinker/, 2025. Managed LoRA fine-tuning API.

Chuan Xiao, Zhengbo Jiao, Shaobo Wang, Wei Wang, Bing Zhao, Hu Wei, Linfeng Zhang, and Lin Qu. Socratic-SWE: Self-Evolving Coding Agents via Trace-Derived Agent Skills, 2026. URL https://arxiv.org/abs/2606.07412.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. AgentTrek: Agent Trajectory Synthesis via Guiding Replay with Web Tutorials, 2024. URL https://arxiv.org/abs/2412.09605.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R. Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. doi: 10.52202/079017-1601. URL https://arxiv.org/abs/2405.15793.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b126cc38b8638e07bef37e7b2bb72bf-Abstract-Conference.html.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Kai Shen, and Liang Xiang. Multi-swe-bench: A multilingual benchmark for issue resolving, 2025. URL https://arxiv.org/abs/2504.02605.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. SWE-bench Goes Live!, 2025. URL https://arxiv.org/abs/2505.23419.

Yudi Zhang, Meng Fang, Zhenfang Chen, and Mykola Pechenizkiy. Self-evolving LLM agents with in-distribution Optimization, 2026. URL https://arxiv.org/abs/2606.07367.

Appendix

A Evaluation Suite and Seed Repositories

The complete evaluation protocol used in the main experiments is listed in Table 2. All agents within a benchmark use the same evaluation split, sampling protocol, runner, verifier, and decoding settings. The final column lists the public seed repositories or data sources available to the researcher agents for constructing training experience.

Table 2: Evaluation suite and seed repositories. The metric matches each benchmark's success criterion, and the same evaluation split and sampling protocol are used for every agent. AIME uses four decoding runs per problem.

Benchmark	Evaluation items	Original size	Coverage	Evaluation runner	Metric	Seed repositories
SWE-bench Verified	100	500	Fixed subset, 20.0%	mini-SWE-agent	Resolved rate	SWE-smith; SWE-Gym; swe-factory
SWE-bench Multilingual	100	300	Fixed subset, 33.3%	mini-SWE-agent	Resolved rate	SWE-smith; SWE-Gym; swe-factory
SWE-bench Pro	100	731	Fixed subset, 13.7%	mini-SWE-agent	Resolved rate	SWE-smith; SWE-Gym; swe-factory
Terminal-Bench 2.0	89	89	Full set	Terminus-2	Task success rate	endless-terminals; tmax
GPQA Diamond	100	198	Fixed subset, 50.5%	Terminus-2	Accuracy	synthetic-data-kit
AIME 2026	30 (4× each)	30	Full set	Terminus-2	Accuracy	synthetic-data-kit

B Representative Actions Within the Data-Synthesis Research Loop

The operations below elaborate the agent-controlled stages of the loop in Figure 2(c). They define a representative action space, not a required modular architecture: an agent may implement any subset, combine several operations in one procedure, or introduce alternatives that respect the benchmark's input, integrity, training, and evaluation contracts.

B.1 Diagnose the Next Training-Data Need

The agent can organize available failures by capability, error type, task family, environment, tool, interaction length, or stage of execution. It may distinguish missing knowledge from failures of reasoning, planning, tool use, state tracking, recovery, verification, or task completion, and use base-model or prior-attempt behavior to estimate where additional supervision is most likely to transfer. The resulting diagnosis specifies a concrete target for the next dataset rather than treating data volume as the solution.

Diagnosis can also compare the intended lesson of a previous dataset with the behavior actually changed by training. A failed attempt may indicate that the original capability hypothesis was wrong, that the generated examples did not express the intended behavior, that invalid examples survived filtering, or that the supervision format did not match the target evaluation protocol. These alternatives imply different revisions even when they produce the same aggregate score.

B.2 Synthesize Targeted Training Experience

Tasks and executable states. The agent may generate synthetic tasks that exercise capabilities implicated by its diagnosis. For interactive tasks, it may also construct the initial state required to execute them, including files and repositories, databases, terminal or filesystem states, tool configurations, or other environment state. Generated tasks should satisfy the target interaction contract and expose behavior that can be exercised by the fixed evaluation service.

Task construction can vary capability coverage and composition rather than only surface wording. An agent may target particular error types, repositories, tools, or interaction lengths; create controlled variants of existing task families; or combine permitted public sources with newly synthesized instances. These choices determine which regions of the target behavior receive supervision.

Successful, failed, and recovery trajectories. For long-horizon tasks, the agent may generate trajectories containing actions, observations, intermediate states, and a completion signal. Successful or expert trajectories demonstrate behavior that reaches a verified outcome. Failed trajectories expose invalid tool calls, incorrect plans, state-tracking errors, premature completion, or other recurrent failure modes. Recovery and repair trajectories demonstrate how to recognize a failed state, revise the plan, and complete the task after receiving new evidence.

The supervision need not reproduce every interaction turn. An agent may train on complete trajectories, selected decision points, final actions, critiques, repaired continuations, or mixtures of these representations. It may also decide which messages contribute to the training loss and which remain as context. The relevant design choice is whether the representation teaches behavior required by the target evaluation protocol without overwhelming training with irrelevant interaction history.

B.3 Validate and Organize Training Experience

Verification, filtering, and deduplication. Candidate tasks and trajectories may be checked through executable environments, agent-designed or permitted auxiliary verifiers, answer agreement, schema validation, consistency checks, or other decidable signals. The agent may discard examples that are invalid, unverifiable, contaminated, malformed, or inconsistent with their claimed outcome, and may deduplicate exact, near-duplicate, or structurally redundant examples. An auxiliary verifier or checker is therefore useful not as an isolated component, but as a mechanism for deciding which proposed experiences are trustworthy enough to influence the model.

Long-trajectory representation. Agentic trajectories often contain long observations, tool outputs, test logs, and repeated interaction history. The agent may truncate, compress, summarize, segment, or reconstruct these contexts while preserving the state and evidence needed for the supervised decision. Representation choices include the granularity of supervised actions, the placement of verifier observations, the treatment of failed turns, and the protocol used to express task completion.

Difficulty, curriculum, and mixture. The agent may estimate difficulty using task metadata, execution length, tool-use complexity, verifier outcomes, generator disagreement, base-model performance, or other available signals. It can use these estimates to balance easy, medium, and hard examples, order their training exposure, or focus on the region where the model is most likely to improve. Mixture decisions determine the proportions of task families, capability categories, trajectory types, supervision formats, and difficulty levels, including the balance among successful behavior, failure recognition, recovery, verification, and task completion.

B.4 Revise the Strategy and Manage Search

After each shared training and evaluation cycle, the agent may revise any preceding decision. It can replace a data source, narrow or expand task coverage, change the trajectory representation, strengthen an auxiliary data checker, alter filtering thresholds, adjust training exposure, or rebalance the curriculum and mixture. A useful revision follows from a diagnosis of the previous checkpoint rather than from an assumption that another attempt or a larger dataset will necessarily help.

These revisions compete for the remaining wall-clock and training-cost budgets. The agent must decide whether to exploit a promising recipe, explore a different hypothesis, return to an earlier strategy, preserve a historical-best checkpoint, or stop. Budget allocation, rollback, stopping, and final checkpoint selection are consequently part of the same iterative policy.

B.5 Instantiation Across Benchmark Families

For software-engineering benchmarks, the loop may connect repository state, issue interpretation, code localization, patch generation, test execution, review, verification, and submission behavior. For terminal tasks, it may connect initial filesystem or tool state, command sequences, state changes, recovery from unsuccessful commands, verification observations, and completion timing. For science question answering and mathematics, the emphasis may shift toward problem construction,

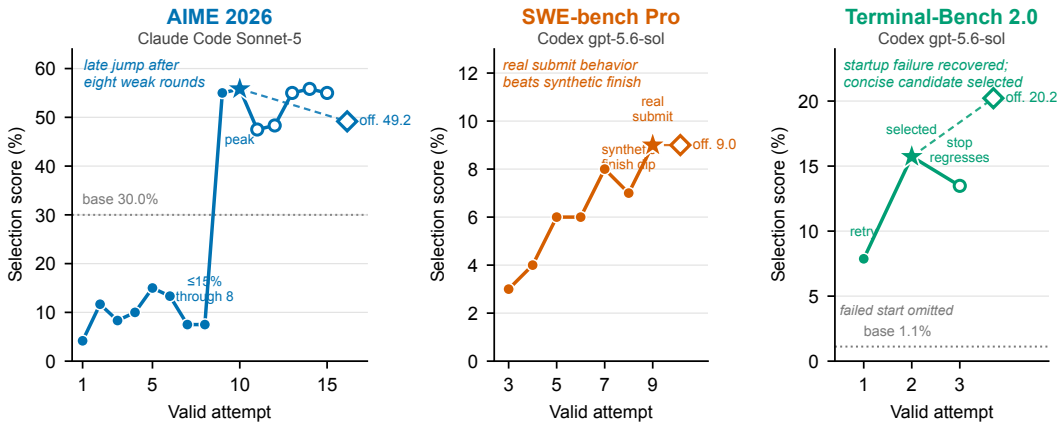


Figure 4: Representative evolution trajectories on AIME 2026, SWE-BENCH Pro, and Terminal-Bench 2.0. Circles show valid-attempt selection scores, stars mark selected historical-best candidates, and open diamonds show official evaluations.

reasoning traces, answer verification, generator agreement, difficulty control, and the selection of which reasoning steps receive supervision.

These examples illustrate why one generic synthetic-data recipe is unlikely to transfer uniformly across the suite. The appropriate training-data strategy is conditioned on the target benchmark and current model failures, while the surrounding SFT and evaluation machinery remains fixed.

B.6 REPRESENTATIVE EVOLUTION CASES

We select three runs with valid official evaluations, at least three valid candidates, and auditable data changes. We plot their quoted selection and official scores in Figure 4 and trace the strategy changes associated with each trajectory below.

Late strategy shifts can produce discrete AIME gains. Claude Code `Sonnet-5`'s AIME trajectory shows that improvement need not be gradual. Its first eight valid candidates remain weak, never exceeding a 15.00% selection score. The ninth candidate jumps to 55.00%, and the tenth reaches 55.83%; later candidates fluctuate between 47.50% and 55.83% without surpassing that peak. The selected tenth checkpoint obtains 49.17% officially, above the 30.00% base-model reference but below the development-stage peak. This trajectory suggests a stage transition in the data factory: repeated local revisions before the jump were insufficient, while the later strategy exposed a much stronger reasoning-training regime. The score sequence alone does not isolate which component caused the jump, so we treat it as process evidence for qualitative strategy change rather than a single-factor ablation.

Behavior-aligned trajectories outperform synthetic completion shortcuts in the SWE-BENCH Pro case. To test whether environment-successful repairs provide sufficient supervision, we compare how Codex `gpt-5.6-sol` represents task completion on SWE-BENCH Pro. The agent begins with repair trajectories that succeed in execution, then uses evaluation failures to build a curriculum that teaches the model to review its final diff, verify its edits, and submit the task. Its valid selection scores progress from 3.00% to 4.00%, 6.00%, 6.00%, and 8.00%; a variant supervised on synthetic finish actions regresses to 7.00%, whereas grounding the closing behavior in real submissions from successful trajectories reaches 9.00% and is selected for official evaluation. This contrast associates stronger SWE performance with supervision that represents the complete tool-use and task-closing behavior rather than only the existence of a correct patch.

Failure recovery and focused supervision separate validity from performance in Terminal-Bench 2.0. Codex `gpt-5.6-sol`'s Terminal-Bench 2.0 trajectory first produces a candidate that fails before yielding a usable selection score. After recovery, the first valid candidate reaches 7.87%. A more focused second valid candidate raises the selection score to 15.73% and is selected, while

Table 3: Protocol-level comparison of benchmarks across data-synthesis research, closed-loop evolution, and controlled infrastructure. A "✓" denotes a criterion explicitly required and enforced by the benchmark protocol, rather than merely enabled by available tools.

	Data-Synthesis Research		Closed-loop Evolution		Controlled Infrastructure	
Benchmark	Executable Experience Synthesis	Reusable Data-Synthesis Policy	Capability-gap Diagnosis	Feedback-driven Revision	Train/Eval Data Isolation	Service-isolated Train/Serve/Eval
DataComp	✗	✗	✗	✗	✗	✗
DataComp-LM	✗	✗	✗	✗	✗	✗
AgoraBench	✗	✗	✗	✗	✗	✗
DCA-Bench	✗	✗	✗	✗	✗	✗
DataEnvGym	✗	✗	✓	✓	✗	✗
Curation-Bench	✗	✗	✗	✓	✗	✗
PostTrainBench	✗	✗	✓	✓	✗	✗
Agent^2 RL-Bench	✓	✗	✗	✗	✗	✗
RSIBENCH-DATA (ours)	✓	✓	✓	✓	✓	✓

a later stop-oriented modification regresses to 13.48%. The selected checkpoint obtains 20.22% officially (18/89), compared with the 1.12% base-model reference. This trajectory separates three abilities: recovering a failed training run creates a valid candidate, focused supervision advances the frontier, and historical-best preservation prevents a later targeted modification from overwriting the stronger checkpoint.

C Related Work

Data curation and generation benchmarks. Early data-centric benchmarks control the surrounding learning stack to study which examples should be selected or curated from a fixed corpus (Gadre et al., 2023; Li et al., 2024). To move beyond a provided candidate pool, subsequent work evaluates generated datasets or whether agents can diagnose hidden data defects (Kim et al., 2025; Huang et al., 2024). DataEnvGym and Curation-Bench are the closest to our closed-loop setting: they introduce student-model or benchmark feedback so that agents can adapt curricula and curation strategies across rounds (Khan et al., 2025; Kang et al., 2026). RSIBENCH-DATA extends this direction by evaluating one researcher agent that owns the data-synthesis hypothesis and revision process while the outer training and evaluation loop remains fixed.

Recursive self-improvement and agent training. To address the shortage of grounded training experience for tool-using agents, early work (Song et al., 2024; Xu et al., 2024) collects interaction trajectories or provides executable environments for generating such experience (Pan et al., 2025; Jain et al., 2025). Building on these resources, recent systems (Chang et al., 2026; Xiao et al., 2026; Shalev et al., 2026) automate experience acquisition, using knowledge exploration, historical execution traces, or self-generated interactions to construct supervision and improve the agent across rounds (Hu et al., 2026; Zhang et al., 2026). These feedback-driven systems provide mechanisms relevant to recursive self-improvement, but generally evaluate a particular method rather than comparing how different LLM agents perform the same data-centric researcher role under controlled experimental services. RSIBENCH-DATA instead isolates the data-centric research component through which a frontier researcher agent improves a separate target model.

Automated post-training and agentic evaluation. Automated post-training benchmarks test whether agents can improve other models by granting control over data, optimization, hyperparameters, and implementation (Rank et al., 2026; Chen et al., 2026b). Complementary work studies persistent multi-agent evolution, develops fresh contamination-resistant evaluations, and provides long-horizon SWE, terminal, and tool-use targets (Qu et al., 2026; Badertdinov et al., 2025; Zhang et al., 2025; Jimenez et al., 2024; Merrill et al., 2026; Yao et al., 2025). Together, these benchmarks measure whether an end-to-end system improves and whether the resulting model succeeds, but their broad scope obscures the contribution of data-centric research decisions.

Existing benchmarks make a fundamental tradeoff: some isolate individual data operations, while others give agents broad control over the post-training stack. RSIBENCH-DATA occupies the middle ground: it evaluates whether an LLM agent can act as a data-centric researcher while the surrounding

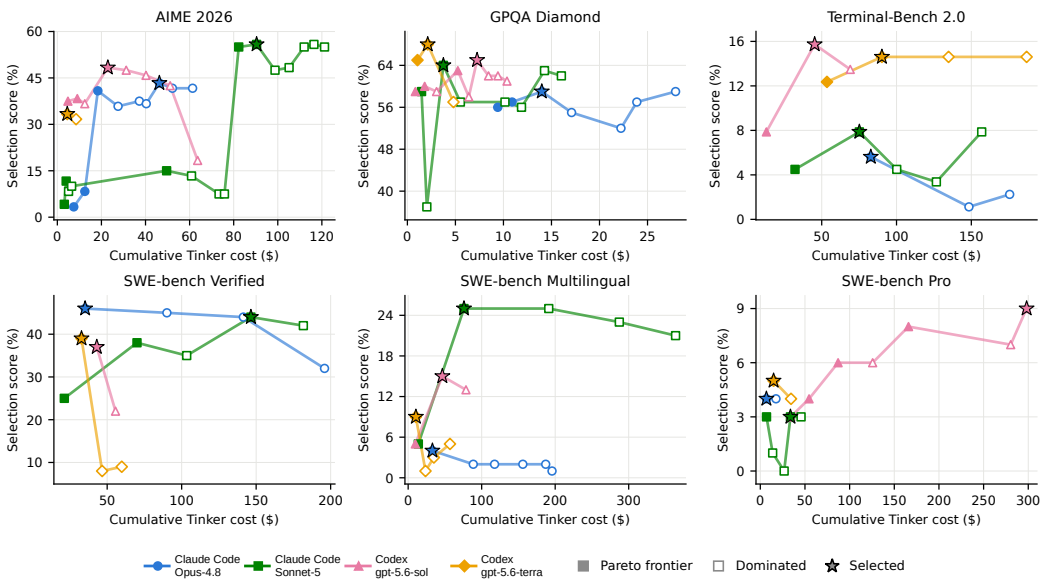


Figure 5: Selection score versus cumulative TINKER cost across benchmarks. Filled markers indicate candidates that establish a new within-run best, hollow markers indicate the remaining candidates, and stars mark selected checkpoints.

training, serving, and official evaluation stack remains fixed (Table 3). Rather than scoring only the resulting checkpoint, it records the hypotheses, training-data strategies, feedback-driven revisions, and checkpoint choices that produce it. Shared TINKER training and serving and fixed HARBOR–E2B evaluation make this research process auditable.

D BUDGET UTILIZATION AND SEARCH EFFICIENCY

Similar final scores can require very different search costs. Agents differ substantially in how much search they use. Across the 24 settings, a run has a median of 4.5 valid candidates (range: 2–15), costs $62.51 ($4.80–$363.77), and takes 6.55 hours (1.14–14.91). Final score alone therefore does not measure efficiency: two runs can reach similar scores even if one uses much more time and money. Figure 5 compares each candidate's selection score with the cumulative TINKER cost incurred up to that attempt and highlights candidates that establish a new within-run best.

Productive search can turn into costly post-peak regression. To test whether early frontier gains remain productive, we examine the completed Terminal-Bench 2.0 trajectories before and after their best candidate. Codex `gpt-5.6-sol` first recovers from a failed attempt, improves from 7.87% to 15.73% by $45.28 cumulative cost, and then falls to 13.48% after reaching $69.07; its selected checkpoint nonetheless reaches 20.22% officially with the lowest valid-candidate TINKER cost among the four Terminal-Bench runs. Claude Code `Sonnet-5` improves from 4.49% to 7.87% by $75.16 but spends another $81.77 without refreshing the frontier and obtains a 5.62% official score, while Codex `gpt-5.6-terra` reaches a 14.61% selection plateau by $90.23 and then spends another $96.74 for a 12.36% official score. Search can therefore discover better candidates, yet continued spending after a peak may fail to refresh the frontier, and higher cost need not imply a stronger final checkpoint.

Checkpoint selection is itself an efficiency decision. To separate efficient discovery from efficient deployment, we compare the selected checkpoint with the historical-best selection score. Across the 12 GPQA, AIME, and Terminal-Bench settings, every run selects a checkpoint that attains its best observed selection score, so the development-stage selection rule does not miss the measured frontier. Selection peaks nevertheless differ from official evaluation scores by 3.89 percentage points on average: Claude Code `Sonnet-5` on GPQA falls from a 64.00% selection peak

Table 4: Cross-case mechanisms suggested by the current process evidence. These are process-level observations, not single-factor causal ablations.

Mechanism	Observation	Implication for researcher agents
Diagnose the real capability gap	Claude Code `Sonnet-5` on AIME stays below 15.00% for eight valid attempts before jumping to 55.00% and then 55.83%.	Agents should recognize when local revisions are not changing the effective data strategy and search for a qualitatively different capability hypothesis.
Embed validation signals	AIME uses answer-agreement filtering, SWE-BENCH Pro uses environment-successful trajectories, and Terminal-Bench 2.0 uses sandbox execution with explicit task outcomes.	Agents should use executable or decidable checks rather than relying only on model self-judgment.
Match supervision to target behavior	Real submit behavior improves SWE-BENCH Pro; on Verified, final-action data reaches a 39.00% selection score while all-turn and safe variants reach 8.00% and 9.00%; Terminal-Bench 2.0 improves for Codex `gpt-5.6-sol` after a more focused candidate but regresses after a later stop-oriented modification.	Supervision granularity and action source are data-design variables, although training changes prevent a single-factor claim.
Preserve historical best candidates	Claude Code `Sonnet-5` on AIME reaches its first 55.83% peak at the tenth valid candidate and later returns to but does not exceed it; Codex `gpt-5.6-sol` on SWE-BENCH Pro peaks at the ninth attempted round; Codex `gpt-5.6-sol` on Terminal-Bench peaks at its second valid candidate before regressing.	Agents need both continued search and explicit stop or rollback policies.
Reasoning effort changes search depth and resource allocation	On SWE-BENCH Verified with Claude Code `Sonnet-5`, max effort improves the first candidate from 25% to 36%, the best selection score from 44% to 49%, and the official score from 35% to 52%.	Effort should be analyzed as a research control that changes the search policy, not only as extra agent-side computation.

to 52.00% officially, while Codex `gpt-5.6-sol` on AIME rises from 48.33% to 53.33% and Codex `gpt-5.6-sol` on Terminal-Bench rises from 15.73% to 20.22%. The trajectories further show that selection discipline can save substantial post-peak spending: the Terminal-Bench runs spend \$23.79–\$96.74 after first reaching their best selection score without improving it. These cases show that stopping and checkpoint choice determine whether search expenditure is converted into the best discovered model.

E Supplementary Mechanism Details

F Early Experiment of RSI

Setting. We run an exploratory same-family RSI experiment outside the main matrix. A Claude Code harness powered by `kimi-k2.6` serves as the researcher agent, the target is the instruction-tuned `moonshotai/Kimi-K2.6` model on TINKER, and Claude Opus 4.8 is the fixed rollout model. Candidates are trained with LoRA SFT and evaluated by MINI-SWE-AGENT through HARBOR–E2B on a fixed 100-task SWE-BENCH Pro subset, with concurrency 64 and a 200-step limit. The backend uses a 32,768-token context window, a 28,671-token prompt limit, and up to 4,096 output tokens. The run has a 20-hour and \$2,000 budget and uses 12.3 hours and an estimated \$738.69. Figure 7 reports within-run candidate evaluations.

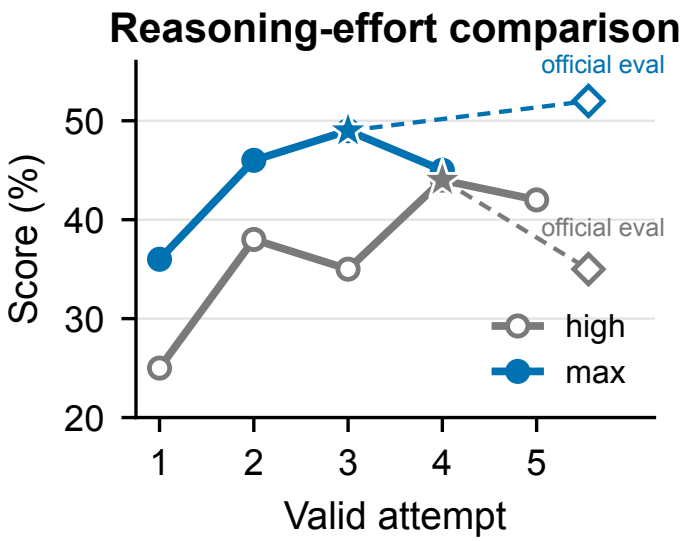


Figure 6: Reasoning-effort diagnostic on SWE-BENCH Verified with the Claude Code `Sonnet-5` researcher agent. The figure compares within-run selection scores under high and max effort. Stars mark the first historical-best candidate, and diamonds show the selected checkpoint's official evaluation score.

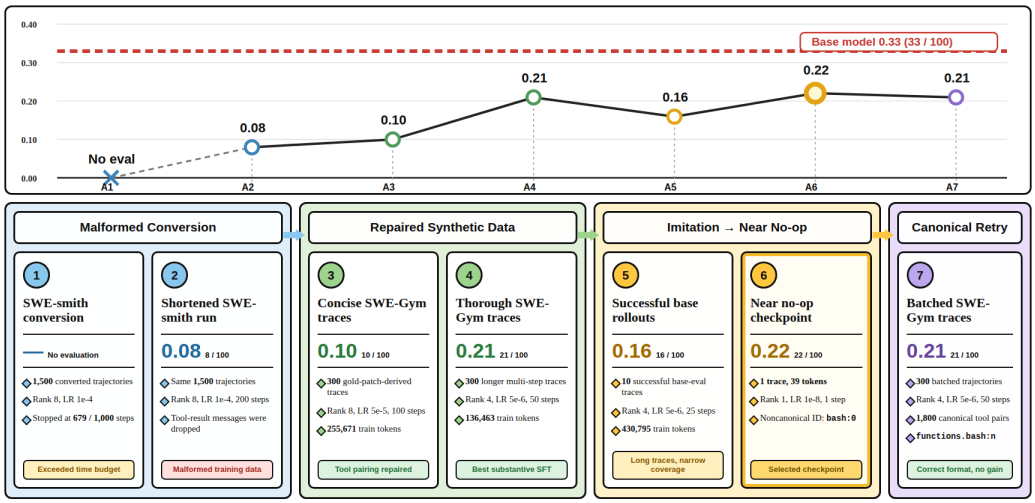


Figure 7: Early RSI trajectory on SWE-BENCH Pro. The Kimi K2.6 researcher evolves its synthetic-data strategy across seven TINKER LoRA attempts, improving the candidate score but remaining below the 33% unadapted-model reference.

Results. The trajectory shows clear strategy evolution but no effective model improvement. The agent first converts 1,500 resolved SWE-smith trajectories into full assistant–tool conversations and schedules 1,000 high-learning-rate steps, but stops at step 679 without evaluation. It then shortens the same recipe to 200 steps, scoring 8%; inspection shows that the converter dropped tool-result messages and produced malformed supervision. The next two attempts switch to 300 gold-patch-derived SWE-Gym trajectories with repaired tool pairing: concise traces score 10%, while longer multi-step traces combined with a lower learning rate and LoRA rank improve to 21%. The agent then extracts 10 successful unadapted-model rollouts as imitation data, but their narrow coverage reduces the score to 16%. To test whether training itself is destructive, it creates a near-no-op adapter from one 39-token example, one step, rank 1, and a negligible learning rate; this reaches 22% and

is selected. A final attempt returns to 300 SWE-Gym trajectories, batches commands into 1,800 canonical tool pairs, and changes identifiers from forms such as `bash:0` to `functions.bash:n`, but scores 21%.

In conclusion, Although agent has improved the synthetic data pipeline and improve the performance from 8% to 21%, every checkpoint remains below the 33% unadapted reference. This likely because the target is already a capable instruction model on an agentic benchmark and the current researcher's RSI capability remains insufficient to discover an improving training distribution.