SWE-TRAIL: HISTORY-GUIDED TRAJECTORY COLLECTION FOR CODING AGENT TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents solve repository issues through long sequences of exploration, editing, and validation. These interaction trajectories are increasingly used as post-training data, so how they are collected shapes what the trained model learns. A common approach is to sample multiple trajectories independently for each issue and then select or curate them. However, as more trajectories are sampled independently, later trajectories become increasingly redundant. In fact, earlier trajectories can guide where subsequent collection effort should be spent. A natural way to use this history is to expose it to the coding agent as advice, helping it decide what to do next. But training on such trajectories can make the model rely on advice that is unavailable at test time, creating a train-test mismatch that may reduce downstream performance. Instead, we argue that history should guide *where* collection effort is spent, while the coding agent decides *what* to generate by itself. Based on this principle, we introduce **SWE-Trail**, a trajectory collector that uses earlier trajectories to guide subsequent collection while keeping that history outside the coding agent's context. During a new rollout, SWE-Trail uses history to identify a state where additional collection effort may be useful. When that state is reached, the coding agent proposes candidate next actions from its ordinary context. SWE-Trail briefly executes candidates from the same checkpoint and uses the results to decide which route continues. We compare collection approaches by training student models on the trajectories they produce and measuring downstream performance. For a 124B student initialized from Ling-3.0-flash-base, SWE-Trail improves mean resolution over independent sampling by 2.33 and 1.60 percentage points on SWE-bench Multilingual and SWE-bench Pro, respectively. Without history, the same collection pipeline produces more resolved trajectories than independent sampling, but improves by only 0.11 points on Multilingual. These results show the value of using history to guide collection without exposing it to the coding agent. Our code is available at `https://anonymous.4open.science/r/SWE-Trail-3D13/`.

1 INTRODUCTION

Resolving a repository-level issue requires a coding agent to explore the repository, modify code, and validate its changes (Jimenez et al., 2024; Yang et al., 2024; Wang et al., 2025a). These interactions form trajectories that record not only the final patch but also the agent's reasoning and actions throughout the repair process. Such trajectories are increasingly used as post-training data, so the collection process directly affects their value for training (Chen et al., 2026; Han, 2026).

A common way to collect coding agent training trajectories is to sample several trajectories per issue and keep those that pass tests or a judge (Pan et al., 2025; Ehrlich et al., 2025; Jain et al., 2025). Some work also edits the kept trajectories, masking flawed tool calls or removing redundant steps (Tao et al., 2026; Ma et al., 2026; Han et al., 2026; Xiao et al., 2026b). These methods operate after trajectories are collected, while trajectories for the same issue are still generated independently. However, later independent samples become increasingly redundant as more trajectories are collected. They often revisit repair targets and behaviors already explored earlier. Rather than ignoring this history, subsequent collection can use earlier trajectories to decide where additional collection effort should be spent. For example, disagreements between successful and failed trajectories can reveal decision points where alternative actions are worth comparing. We refer to such evidence as *allocation*

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.

signals and to the earlier trajectories for an issue as its *history*. Section 2 quantifies the redundancy of independent sampling and tests whether earlier trajectories provide such allocation signals. Since earlier trajectories can provide useful information for a new rollout, a natural way to use this history is to place it in the coding agent's context as advice (Chen et al., 2025; Wang et al., 2026b). This can help the current rollout, but the resulting training trajectories record reasoning and actions generated with access to the advice. However, at test time, the trained model must make decisions without that advice. As a result, these trajectories may provide less useful supervision for training, as we evaluate in Section 4.2. This leads to our central question: ***How can history guide where collection effort is spent while the coding agent decides what to generate from its ordinary context?***

To answer this question, we introduce **SWE-Trail**, a trajectory collector that keeps history outside the coding agent's context. Before collection, SWE-Trail uses history to determine which states should receive more collection effort and what evidence should be considered at those states. During a new rollout, when the agent reaches such a state, it proposes candidate next actions from its ordinary context without access to the history. SWE-Trail then compares candidate actions by briefly executing them from the same checkpoint. It selects the route that is better supported by repository evidence. The coding agent then continues from the selected route to complete the trajectory. A concrete example illustrates this process. On a Rust issue, all prior failed trajectories made an edit and then continued reading without running tests. From this history, SWE-Trail determines that this state should receive additional collection effort when it appears again in a new rollout. When the rollout reaches such a state, the coding agent proposes different next actions from its ordinary context, including further reading and running tests. SWE-Trail briefly executes both choices from the same checkpoint and selects the test-running route based on the resulting evidence. The coding agent then continues from the selected route until the rollout ends.

We evaluate whether SWE-Trail collects trajectories that are more useful for downstream training. The comparison uses the same 300 SWE-rebench V2 issues (Badertdinov et al., 2026). Each setting collects five trajectories per issue with GLM-5.1 (GLM-5 Team, 2026; Z.ai, 2026). *SeedBank* provides the independent sampling baseline. *H5* uses the five SeedBank trajectories for each issue as history to collect five new trajectories, while *H0* runs the same collection pipeline without history. We train the 124B Ling-3.0-flash-base model (inclusionAI, 2026) on the trajectories from each condition, with three independent training runs. Under this controlled setup, H5 improves mean resolution over SeedBank by 2.33 and 1.60 percentage points on SWE-bench Multilingual and SWE-bench Pro, respectively. The H0 control helps isolate the contribution of history. It produces more resolved trajectories than SeedBank, but improves the 124B student's mean resolution by only 0.11 points on Multilingual. This contrast shows that collection success alone does not explain the downstream gain. A second student model, Qwen3-14B-Base (Qwen Team, 2025), is also trained on the same H5 trajectories to test whether the gain generalizes across student models. For Qwen3-14B-Base, H5 improves mean resolution over SeedBank by 3.11 and 2.42 percentage points on SWE-bench Multilingual and SWE-bench Pro, respectively. Together, these results support using history to guide trajectory collection without placing that history in the coding agent's context.

2 MOTIVATION

To examine the empirical basis for our motivation, we analyze the independently sampled trajectories in SeedBank. We use *attempt* to refer to one independent try at solving an issue, and examine both how coverage grows with additional attempts and whether earlier attempts provide allocation signals for subsequent collection.

2.1 DIMINISHING COVERAGE GAINS FROM INDEPENDENT ROLLOUTS

Figure 1A shows the expected number of issues solved with k attempts per issue, averaging over all k-attempt subsets of the five SeedBank attempts for each issue. For a given k-attempt subset, an issue is counted as solved if at least one of the k attempts succeeds. With one attempt per issue, 58.4 issues are solved in expectation. Increasing k from 1 to 2 adds 13.1 solved issues, while increasing it to 3, 4, and 5 adds only 8.0, 5.1, and 3.4 more issues, respectively. Thus, each additional attempt still expands issue coverage, but by a smaller amount.

Patch-target coverage shows the same diminishing pattern across different types of history. We use modified files as patch targets and define three history types from each issue's SeedBank outcomes: *stable-solved* if all attempts succeed, *mixed* if both successful and failed attempts occur, and *never-*

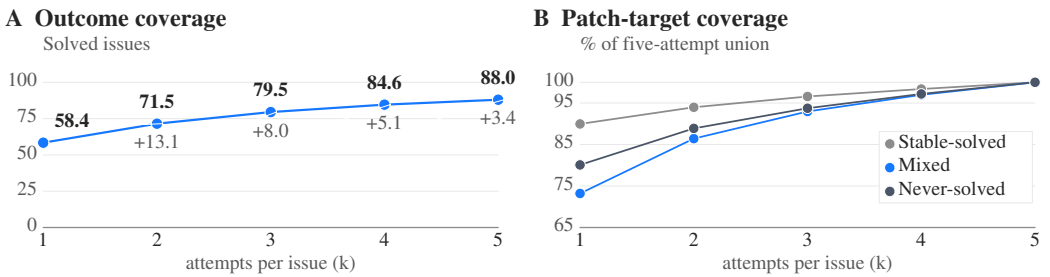


Figure 1: **Diminishing coverage gains from independent rollouts.** A: expected number of issues solved with k attempts per issue. Gray labels show marginal additions. B: mean coverage of each issue's five-attempt patch-target union, grouped by SeedBank outcomes. Both panels average over all k-attempt subsets of the five recorded attempts. The vertical axis in B starts at 65%.

solved if all attempts fail. For each history type, Figure 1B shows the fraction of files modified across an issue's five attempts that are covered by k attempts. We average this fraction over all k-attempt subsets and then across issues. One attempt already covers 90.0% of these files on stable-solved issues, 73.2% on mixed issues, and 80.1% on never-solved issues. These results show substantial target overlap across all three history types. As more attempts are added, later attempts cover fewer new files. Overlap among successful attempts may simply reflect convergence on the same repair targets. However, even when all independent attempts fail, they still modify many of the same files. This pattern shows that independent sampling does not automatically redirect collection effort after repeated failures. Later attempts may continue spending effort on previously explored repair targets rather than exploring new ones. Together with the issue-level results, these findings show that later independent attempts become increasingly redundant.

2.2 History Provides Allocation Signals

Section 2.1 shows that later independent attempts add less coverage and often revisit repair targets explored earlier. The remaining question is whether history can indicate where additional collection effort may be useful. Mixed issues are informative because successful and failed attempts may choose different next actions from similar states. We therefore examine whether such decision points occur.

For each mixed issue, an auditor compares the most similar successful and failed attempts and locates their first meaningful decision difference. We assess whether the two attempts are similar enough to compare their next actions and whether both actions are reasonable at that point. The audit is conducted independently with GPT-5.6-Sol (OpenAI, 2026) and GLM-5.1. The main analysis includes 43 of the 51 mixed issues, excluding eight whose failed attempt ended in a timeout or collector error. GPT-5.6-Sol identifies such cases in 26 issues and GLM-5.1 in 31, with 20 identified by both models. These results show that mixed histories can reveal decision points where additional collection effort may be useful. The detailed results are reported in Appendix A.3.

However, this evidence relies on having both successful and failed attempts. To test whether history remains useful without one, we examine never-solved histories and measure the overlap in modified files across failed attempts on the same issue. The mean pairwise Jaccard overlap is 0.841. Among the 198 issues with a defined overlap score, 93 have a score of 1.0. These results show that failed attempts often focus on the same repair targets. This overlap does not explain why the attempts fail, but it shows where effort is repeatedly spent across failed attempts. Thus, even without a success–failure contrast, history can still indicate where additional collection effort may be useful. Together, these findings show that history can provide allocation signals for further collection. Appendix A gives the full audit procedure and overlap calculations.

3 SWE-Trail

Section 2 shows that later independent attempts add less new coverage, while earlier attempts can indicate where additional collection effort may be useful. Motivated by these observations, we introduce SWE-Trail, a trajectory collector that uses history to decide where additional collection effort should be spent in a new rollout. For issue i, let x_i contain the issue statement and the repository snapshot at the issue commit. The history H_i contains the attempts already collected on that issue. SWE-Trail has access to H_i, but the coding agent does not. Reference patches are unavailable during

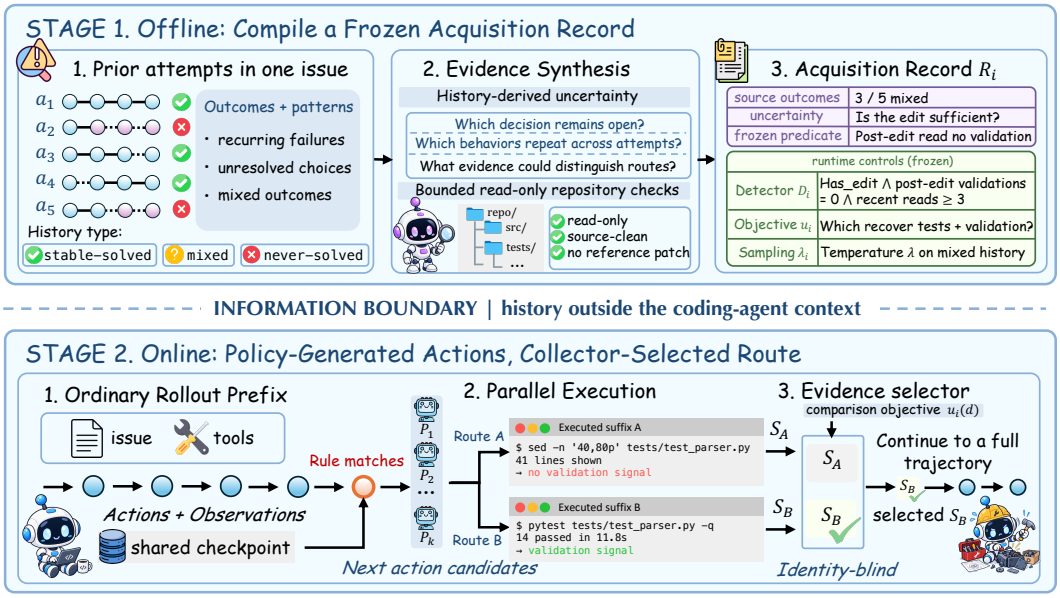


Figure 2: **SWE-Trail in two stages.** Offline, the compiler reads the issue history, runs bounded read-only repository checks, and freezes an acquisition record R_i containing a detector D_i, a comparison objective u_i, and a sampling schedule λ_i. Online, when D_i identifies a state for additional collection effort, the coding agent proposes candidate next actions from its ordinary context. SWE-Trail briefly executes two candidates from the same checkpoint, and an identity-blind selector retains the route better supported by repository evidence under u_i. The coding agent then continues from the selected route. History never enters the coding agent's context.

collection. Figure 2 shows the two stages of SWE-Trail. Before collection, SWE-Trail uses the history and read-only repository checks to determine which states should receive additional collection effort and what evidence should be considered at those states. During a rollout, when such a state is reached, the coding agent proposes candidate next actions from its ordinary context without access to the history. SWE-Trail selects candidates for comparison, briefly executes them in parallel from the same checkpoint, and retains the route better supported by repository evidence. The coding agent then continues from the selected route to complete the trajectory.

3.1 Compiling History into Detectors

Earlier attempts can indicate which states should receive additional collection effort, but a new rollout may reach similar situations through a different sequence of actions. The compiler therefore builds a detector D_i whose rules recognize such states from the rollout's actions and observations rather than relying on fixed step numbers. For example, the detector in Figure 2 recognizes repeated reading after an edit without validation. The runtime checks these rules after each new repository observation and samples candidate actions when a state matches.

To build the detector, the compiler first analyzes (x_i, H_i) to extract allocation signals from the history. These signals include decisions supported by successful attempts, differences across repair routes, and recurring failures. The history may leave open questions about the issue or repository, so the compiler then runs bounded read-only repository checks to resolve these questions. These checks do not modify source code or access the reference patch. The compiler stores the historical evidence, check results, and remaining questions in a private acquisition record R_i. Using R_i, the compiler defines the detector rules and a local comparison objective $u_i(d)$ for each detector state d. The objective specifies what evidence should be considered when comparing candidate routes at that state. Different histories lead the compiler to focus on different kinds of decisions. For stable-solved histories, repeated success shows which decisions already have strong support. The compiler allocates additional effort only when meaningful uncertainty remains. For mixed histories, it focuses on states where successful and failed repair routes differ. For never-solved histories, it focuses on recurring failure patterns, such as edits followed by further reading without validation. The compiler also sets a sampling-temperature schedule λ_i for candidate generation when a detector state is reached. Mixed

and never-solved histories use higher temperatures to encourage more varied next actions when the history shows unresolved choices or repeated failures.

Before collection begins, the compiler validates each proposed detector state against its supporting evidence and replays its rules on the recorded attempts. States without sufficient support are discarded. The admitted detector states, their comparison objectives, and λ_i are then fixed for the entire collection round. Appendices B and F.5 provide the compiler details, admission checks, and example records.

3.2 Sampling Comparable Alternatives

The detector determines where additional collection effort is spent, while the actions to consider come from the coding agent itself. When the rollout reaches a state d recognized by D_i, SWE-Trail samples several candidate next actions using only the agent's ordinary context. The history, acquisition record R_i, and comparison objective $u_i(d)$ are not added to that context. The first sample uses the agent's ordinary temperature and serves as the default. The remaining samples use λ_i.

Different samples may propose equivalent actions, so SWE-Trail first groups equivalent proposals together. It keeps the default candidate and shortlists executable alternatives from the remaining proposals. To keep each intervention local and bounded, SWE-Trail chooses at most one alternative from this shortlist using R_i and $u_i(d)$. If no suitable alternative is available, the rollout continues with the default action. Otherwise, SWE-Trail restores two isolated workspaces from the same checkpoint, one for the default and one for the chosen alternative. Both candidates start from the same conversation and workspace state and run for the same short horizon, producing two *executed suffixes* of actions and observations. To avoid expanding collection into a multi-level search tree, each trajectory performs at most one such comparison. Different rollouts on the same issue may still make their comparison at different detector states.

3.3 Selecting a Route by Executed Evidence

Once both suffixes have been executed, SWE-Trail compares their progress using the local comparison objective $u_i(d)$. The two suffixes are presented to SWE-Trail in shuffled order, hiding which one comes from the default candidate. SWE-Trail uses R_i only as background for the comparison. It prefers a route only when the two executed suffixes provide clear evidence that one makes better progress under $u_i(d)$. Otherwise, it keeps the default. After selection, the coding agent continues only from the selected workspace until the full rollout ends. For training, SWE-Trail keeps only the selected route, preserves the coding agent's original reasoning and actions without rewriting, and removes the unselected suffix and private collector events. This ensures that the training record remains a single continuous trajectory generated by the coding agent and that private collection information does not enter the training process. Algorithm 1 in Appendix B summarizes the procedure, and Appendix H.1 provides a full end-to-end case study.

4 Experiments

4.1 Setting

We test whether training on SWE-Trail trajectories improves student performance on new issues. Across all comparisons, we use the same collection issues and the same number of coding trajectories for training. The background training data and the training and evaluation protocols remain fixed.

Shared setting. Each training dataset contains five coding trajectories for each of the same 300 SWE-rebench V2 issues (Badertdinov et al., 2026), giving 1,500 trajectories in total. None of the collection repositories appears in either evaluation benchmark. All coding rollouts use GLM-5.1 (GLM-5 Team, 2026) with the OpenHands scaffold (Wang et al., 2025a). Collection runs in an offline sandbox with repository commit histories removed. Reference patches and published fixes are unavailable during collection. Appendix C details the provenance of the collection issues and verifies that they share no issues or repositories with either evaluation benchmark.

Methods and variants. SeedBank provides the independent-sampling baseline. We apply controlled adaptations of SWE-Gym (Pan et al., 2025), SWE-TRACE (Han et al., 2026), and SWE-Lego (Tao et al., 2026) to the same SeedBank trajectories. We also adapt SWE-Exp (Chen et al., 2025) to test a different use of history, where the coding agent reads an experience message compiled from the SeedBank attempts during collection. The message is retained in the training record.

Table 1: **H5 improves downstream resolution on both benchmarks.** Mean resolution over three independent training runs. Changes are relative to SeedBank, and ranges show the lowest and highest result. Rows named after prior work use the same SeedBank attempts as controlled adaptations. Table 13 defines all conditions.

	Multilingual			SWE-Pro		
Coding-trajectory condition	Mean		Range	Mean		Range
SeedBank independent rollouts	63.33		[63.00, 63.67]	41.22		[41.18, 41.31]
SWE-Gym (Pan et al., 2025), outcome-conditioned transform	64.00	(+0.67)	[63.33, 65.00]	40.86	(−0.36)	[40.36, 41.59]
SWE-TRACE (Han et al., 2026), critical-path reduction	64.44	(+1.11)	[63.33, 65.33]	40.77	(−0.46)	[40.08, 41.72]
SWE-Lego (Tao et al., 2026), strict tool-error masking	64.56	(+1.22)	[63.00, 66.00]	41.63	(+0.41)	[39.81, 42.95]
SWE-Exp (Chen et al., 2025), agent-visible experience	62.00	(−1.33)	[61.00, 63.00]	38.44	(−2.78)	[38.30, 38.58]
SWE-TRAIL-H0	63.44	(+0.11)	[63.00, 63.67]	41.95	(+0.73)	[41.04, 43.09]
SWE-TRAIL-H5 (random detection)	64.22	(+0.89)	[63.33, 64.67]	41.54	(+0.32)	[41.31, 41.72]
SWE-TRAIL-H5 (random retention)	64.78	(+1.44)	[63.67, 66.00]	41.63	(+0.41)	[41.04, 42.13]
SWE-TRAIL-H5	**65.67**	(+2.33)	[64.00, 67.67]	**42.82**	(+1.60)	[41.72, 43.37]

For SWE-Trail, history remains outside the coding agent's context throughout collection. Its main variant, H5, uses the five SeedBank attempts for each issue as history when collecting five new trajectories. H0 sets $H_i = \varnothing$ and runs the same read-only repository checks and compiler without history. Comparing H0 with SeedBank tests the collection pipeline without history, while comparing H5 with H0 isolates what history adds. We also use two H5 variants to test where SWE-Trail makes comparisons and which route it retains. *Random detection* replaces the history-guided comparison location with a random eligible state while keeping H5's remaining collection procedure unchanged. *Random retention* keeps H5's detector but randomly chooses which executed route to continue. Both variants use the same per-trajectory model-call limits and execution horizon as H5.

To examine history scale, H10 adds five independently sampled attempts to the five attempts already used by H5 for each issue. It otherwise uses the same read-only repository checks and compiler, so comparing H10 with H5 tests whether more history adds value. We also compare SWE-Trail with verifier-based trajectory selection. *Top-5-of-10* selects five of ten independently sampled trajectories per issue. *Top-5-hybrid* applies the same verifier to five SeedBank and five H5 trajectories and selects five from the combined set. Table 13 in Appendix C gives the full experimental conditions. Appendices B.5 and B.6 provide the runtime settings and export checks.

Students and evaluation. Every trajectory in the final training dataset is used, whether resolved or not. The primary student starts from the 124B Ling-3.0-flash-base model (inclusionAI, 2026) and is trained with full-parameter supervised fine-tuning on 128 H800 GPUs. Each dataset is used for three independent training runs. Evaluation uses SWE-bench Multilingual (Yang et al., 2025) with 300 tasks and SWE-bench Pro (Deng et al., 2025) with 731 tasks. We refer to the latter as SWE-Pro below. Resolution is the fraction of tasks whose patches pass the benchmark tests, and the reported score is the mean across the three training runs. A second student, Qwen3-14B-Base (Qwen Team, 2025), is trained to test whether the gains extend to another model. Appendix D gives the training and evaluation protocols and metric definitions.

4.2 Downstream Training Utility

SWE-Trail improves downstream resolution. Table 1 shows that H5 outperforms all other conditions in the table on both benchmarks. H5 improves mean resolution over SeedBank by 2.33 percentage points on Multilingual and 1.60 points on SWE-Pro. Since the dataset size and training setup are fixed, these differences reflect changes in the collected trajectories. Across three independent training runs, every H5 run exceeds every SeedBank run on both benchmarks. An exact one-sided permutation test over the training runs gives $p = 0.05$ on each benchmark. Appendix E reports the individual runs, statistical tests, and results by language. SWE-Exp falls below SeedBank by 1.33 points on Multilingual and 2.78 points on SWE-Pro. This may reflect reliance on advice available during training but absent at evaluation. To examine the role of advice more directly, Appendix H.2 compares two exports of the same trajectories, with the advice either retained or removed. Compared with retaining the advice, removing it improves SWE-Pro but lowers Multilingual. Removing the advice after collection does not remove its influence on how the trajectory was generated. The

Table 2: **H10 costs 12.9% more recorded tokens than H5 for a 0.22/0.14 point gain.** Differences are relative to H0. Acquisition-only tokens exclude the ordinary rollout and total tokens include it. †: H0 omits offline compiler calls, so its compute is a lower bound and the differences are upper bounds.

		Acquisition		Recorded compute		Student utility	
Condition	Hist.	Res.	Issues	Acq.-only	Total	Multi.	Pro
SWE-TRAIL-H0	0	329	92	164M†	≥1.845B†	63.44	41.95
SWE-TRAIL-H5 (main)	5	**345** (+16)	**98** (+6)	307M (+143M†)	1.914B (+69M†)	65.67 (+2.22)	42.82 (+0.87)
SWE-TRAIL-H10	10	341 (+12)	94 (+2)	510M (+346M†)	2.160B (+315M†)	**65.89** (+2.44)	**42.95** (+1.00)

agent produced its reasoning after reading the advice, so some decisions may remain without the information that originally supported them.

H5 has higher mean resolution than H0. Without history, H0 improves mean resolution over SeedBank by 0.11 points on Multilingual and 0.73 points on SWE-Pro. Adding history in H5 raises mean resolution by a further 2.22 and 0.87 points over H0, respectively. The point estimates are positive on both benchmarks, but task-paired bootstrap gives 95% intervals of $[-0.2, 4.6]$ points on Multilingual and $[-0.8, 2.5]$ on SWE-Pro. Appendix E.2 gives the full paired analysis. Section 4.4 compares these downstream differences with the corresponding collection outcomes.

Both collection decisions matter. To test these two choices, we randomize each one separately. *Random detection* replaces H5's history-guided choice of where to compare with a random eligible point and lowers mean resolution by 1.44 points on Multilingual and 1.28 points on SWE-Pro. *Random retention* keeps H5's history-guided comparison location but randomly chooses which executed route continues, lowering mean resolution by 0.89 and 1.19 points, respectively. These decreases support using history to decide where additional collection effort should be spent and execution evidence to decide which route should continue. Both variants remain above SeedBank in mean resolution, while their run ranges overlap H5's on both benchmarks.

The gains extend to another student. We repeat the comparison with Qwen3-14B-Base using the same SeedBank, H0, and H5 trajectory sets. H5 improves mean resolution over SeedBank by 3.11 percentage points on Multilingual and 2.42 points on SWE-Pro. It also exceeds H0 by 1.67 and 1.60 points, respectively. These results show that the downstream gains from SWE-Trail also appear with another student model. Appendix E.4 gives the full results.

4.3 HISTORY SCALE AND COST

More history brings smaller gains. Table 2 compares H0, H5, and H10. Using five prior attempts improves mean resolution over H0 by 2.22 points on Multilingual and 0.87 points on SWE-Pro. Increasing the history from five to ten attempts adds only 0.22 and 0.14 points, respectively. H10 also does not improve the collection outcomes over H5, despite increasing tokens spent beyond the ordinary rollout from 307M to 510M, a 66.0% increase, and total collection tokens by 12.9%. More history therefore provides only small additional mean gains at substantially higher collection overhead. The main H5 setting remains close to H0 in cost. Excluding the cost of collecting prior attempts, H5 uses at most 3.7% more tokens than H0. Its ordinary rollouts use 4.4% fewer tokens than H0. The same diminishing gain appears with Qwen3-14B-Base. H5 improves over H0 by 1.67 and 1.60 points, while H10 adds a further 1.00 and 0.36 points on Multilingual and SWE-Pro, respectively. Appendix G.2 gives the complete cost breakdowns.

Selection alone does not match H5. To test whether selection from a larger candidate pool of independent rollouts can match SWE-Trail, Top-5-of-10 samples ten independent trajectories per issue and selects five for training. As shown in Table 3, it improves mean resolution over SeedBank but remains below H5 by 1.56 points on Multilingual and 0.96 points on SWE-Pro. Thus, selecting from more independent trajectories does not recover the gains from SWE-Trail. The collection process matters beyond post-hoc selection. Detailed results are reported in Table 26.

Table 3: Mean resolution (%) across three independently trained students per dataset.

Dataset	Multilingual	SWE-Pro
SeedBank	63.33	41.22
Top-5-of-10	64.11	41.86
SWE-TRAIL-H5	65.67	42.82
Top-5-hybrid	**66.22**	**43.14**

Selection and SWE-Trail are complementary. Top-5-hybrid applies the same verifier to five SeedBank and five H5 trajectories per issue, then selects five trajectories for training. Like Top-5-of-10, it retains 1,500 trajectories in total. The resulting student reaches 66.22 on Multilingual and 43.14

on SWE-Pro, exceeding H5 by 0.56 and 0.32 points, respectively. These results show that post-hoc selection can complement SWE-Trail when applied to the combined candidate set. Appendix G.3 gives the selection rule and candidate counts.

4.4 WHAT HISTORY ADDS TO COLLECTION

Collection success does not explain student gains. Table 4 compares the collection outcomes. Relative to SeedBank, H0 produces 37 more resolved trajectories, but improves the student by only 0.11 points on Multilingual. Adding history in H5 produces 16 more resolved trajectories and solves six more issues than H0, while improving the student by a further 2.22 points. Across both benchmarks, the downstream gain is not proportional to the increase in resolved trajectories.

History changes the repair process. We examine how H5 and H0 trajectories differ beyond their final outcomes using the process statistics in Table 19 of Appendix F. Relative to H0, H5 reduces the mean number of actions from 40.89 to 39.51 and repeated file reads from 7.59 to 7.01. The share of edited trajectories with a check after the final edit increases slightly from 88.91% to 89.53%. The clearest shift is toward shorter repair processes with fewer repeated reads, while post-final-edit checking changes only slightly. Appendix F.1 gives the full distributions. Similar differences in actions and repeated reads also appear in the students' evaluation trajectories. On both benchmarks, H5-trained 124B students take fewer actions and repeat fewer file reads on average than H0-trained students. Appendix F.2 reports the comparison.

Most collection gains come from mixed histories. Table 21 in Appendix F breaks down the gains in Table 4 by history type. Stable-solved issues show no difference in resolved trajectories between H0 and H5. Although mixed histories account for only 51 of the 300 collection issues, they contribute 13 of H5's 16 additional resolved trajectories and five of its six additional solved issues. Never-solved histories account for the remaining three trajectories and one issue. The gains are therefore concentrated on issues whose prior attempts include both successes and failures. This is consistent with Section 2.2, where such histories reveal decision points that may benefit from additional collection effort.

Table 4: Collection outcomes over 1,500 trajectories per method. Changes in the H5 row are relative to H0.

Method	Resolved	Solved issues	Verified positive
SeedBank	292	88	263
SWE-TRAIL-H0	329	92	272
SWE-TRAIL-H5	**345** (+16)	**98** (+6)	**285** (+13)

4.5 HOW THE COLLECTION MECHANISM CHANGES

History leads to more comparisons. Figure 3 compares H0 and H5 across the stages of collection. With history, candidate sampling occurs in 1,147 of 1,500 rollouts, compared with 664 under H0. Both suffixes are executed in 968 rollouts under H5 and 560 under H0. Panel A shows that candidate sampling increases by 29.8 percentage points on mixed histories and 35.4 on never-solved histories, compared with 17.3 on stable-solved histories. This pattern is consistent with using disagreements and repeated failures to identify where more collection effort may be useful.

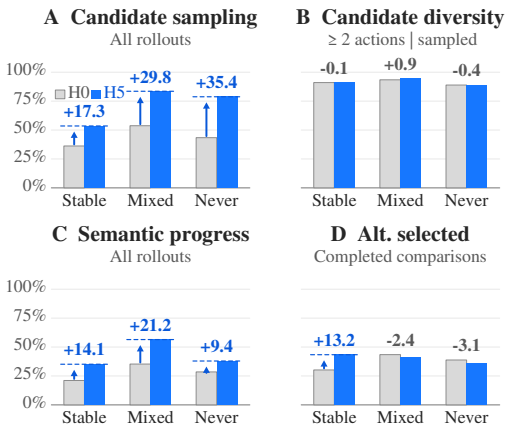


Figure 3: **History leads to more comparisons.** Conditional on candidate sampling, candidate diversity remains similar, as does the alternative-selection rate. Table 22 in Appendix F gives the full funnel and counts by history type.

More comparisons, similar selection rates. Panel B shows similar candidate diversity under H0 and H5, with at least two distinct actions in about 90% of candidate-sampling cases. The increase in comparisons therefore comes mainly from candidate sampling occurring more often. Panel C shows better local progress on one route in 40.7% of H5 rollouts and 28.7% of H0 rollouts. Yet Panel D shows similar alternative-selection rates among completed comparisons, at 37.4% and 39.0%, respectively. Together, Panels C and D show that H5 more often reaches comparisons with clear progress, while the rate of selecting the alternative remains similar.

5 Related Work

Three lines of work relate to SWE-Trail. Appendix I gives the extended discussion.

Test-time search and selection. Search methods spend extra compute to solve the current issue. Some sample several independent rollouts and select among them using tests or model judgments (Ehrlich et al., 2025). Others reuse earlier prefixes or summaries of prior attempts to guide later rollouts on the same issue (Ding & Zhang, 2026; Kim et al., 2026). Tree search explores alternatives within a growing execution tree during the current solve (Kocsis & Szepesvári, 2006; Zhou et al., 2024; Antoniades et al., 2025). Instead, SWE-Trail uses prior attempts on the same issue to guide where additional collection effort is spent during trajectory generation. The coding agent proposes actions from its current context, SWE-Trail briefly compares them from the same checkpoint, and the selected route becomes the training trajectory.

Trajectory generation and curation. Methods for constructing training data generate more tasks and trajectories or increase trajectory diversity through adaptive synthesis (Yang et al., 2025; Chen et al., 2026). Others select resolved trajectories or modify completed trajectories through filtering and process-level editing (Pan et al., 2025; Tao et al., 2026; Ma et al., 2026; Han et al., 2026). Recent work also evolves curation rules using downstream failures (Wang et al., 2026a). Instead, SWE-Trail uses prior attempts on the same issue to guide where additional collection effort is spent during trajectory generation. These comparisons collect new execution evidence rather than only selecting or modifying completed trajectories. Post-hoc selection can still follow, as Section 4.3 shows.

Experience-guided agents. Agents use reflections, distilled insights, workflows, and skills to guide later actions (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2025b; Li et al., 2026). For coding tasks, this experience can come from earlier repair trajectories, human debugging records, or the ongoing attempt (Chen et al., 2025; Wang et al., 2026b; Hayashi et al., 2025; Guo et al., 2025). These methods use experience to guide the agent's reasoning or actions. Instead, SWE-Trail uses history to guide where additional collection effort is spent. The coding agent proposes actions from its ordinary context without receiving that history.

6 Discussion and Limitations

What the results show. The experiments separate collection success from downstream training value. H0 resolves more trajectories than independent sampling but brings little gain on Multilingual, while adding history in H5 produces larger student gains. Randomizing either where additional collection effort is spent or which executed route continues lowers mean resolution relative to H5. The gains also appear with Qwen3-14B-Base. These results support using history as a collector-side signal for improving the training value of new trajectories, rather than as advice shown to the coding agent.

Limitations and future work. Our experiments use two student models but one collection model and one agent scaffold. Future work could test other collection models and scaffolds. Our current setting also keeps history fixed during each collection round and uses the same collection budget for every issue. Future work could update history across rounds and allocate effort differently across issues. The exploratory replay audit in Appendix G.1 also motivates testing more diverse candidate proposals. Extending SWE-Trail beyond coding would require environments that support state restoration and feedback for comparing alternatives. Web navigation is one possible direction.

7 Conclusion

Our results point to a broader lesson: prior attempts are useful not only for what they contain, but also for guiding where new collection effort should be spent. SWE-Trail operationalizes this idea by compiling prior attempts into rules that determine when additional collection effort may be useful during a new rollout. The coding agent proposes actions without seeing the history, while the collector briefly executes candidates from the same checkpoint and uses the resulting evidence to choose which route to continue. With the training setup fixed, SWE-Trail improves the 124B student's mean resolution over independent sampling by 2.33 percentage points on SWE-bench Multilingual and 1.60 points on SWE-bench Pro. The gains also appear with Qwen3-14B-Base. The H0 comparison shows that collection success alone does not explain the downstream gains. Randomizing either where additional collection effort is spent or which executed route continues lowers mean resolution relative to H5. Together, these results show that prior attempts can improve the training value of new trajectories by guiding collection without being shown to the coding agent as advice.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript editing and code implementation. The authors reviewed all AI-assisted content and code and take responsibility for the final manuscript and implementation.

REPRODUCIBILITY STATEMENT

The main text describes the experimental setup and evaluation criteria. Appendix C gives the collection issues and the overlap audit. Appendix B provides the collector algorithm, schemas, and prompts, and Appendix B.6 specifies the export checks. Appendix D describes the training and evaluation procedures, and Appendix D.3 defines the metrics. For each student, training settings are held fixed across trajectory datasets. We use publicly available base models, the OpenHands scaffold, and public issues from open-source repositories.

ETHICS STATEMENT

The study uses public open-source repositories and benchmark issues. It does not involve human subjects or collect private user data. Trajectories may contain security-sensitive code. Any released trajectories will retain repository provenance and license information and will be screened for secrets. Collection runs in an offline sandbox with repository commit histories removed. This prevents retrieval of published fixes through the network or commit history during collection. The method may improve the efficiency of autonomous code modification. Systems trained using such trajectories should retain sandboxing, human review, and repository access controls.

REFERENCES

Wasi Uddin Ahmad, Nikolai Ludwig, Somshubra Majumdar, and Boris Ginsburg. Open-swe-traces: Advancing dual-mode multilingual distillation for software engineering agents. *arXiv*, 2026.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Yang Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. In *ICLR*, 2025.

Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, and Alexander Golubev. Swe-rebench V2: language-agnostic SWE task collection at scale. *arXiv*, 2026.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. Unifying count-based exploration and intrinsic motivation. In *NeurIPS*, 2016.

Guhong Chen, Chenghao Sun, Cheng Fu, Qiyao Wang, Zhihong Huang, Chaopeng Wei, Guangxu Chen, Feiteng Fang, Ahmadreza Argha, Bing Zhao, Xander Xu, Qi Han, Hamid Alinejad-Rokny, Qiang Qu, Binhua Li, Shiwen Ni, Min Yang, HU Wei, and Yongbin Li. Beyond quantity: Trajectory diversity scaling for code agents. In *ACL*, 2026.

Silin Chen, Shaoxin Lin, Xiaodong Gu, Yuling Shi, Heng Lian, Longfei Yun, Dong Chen, Weiguo Sun, Lin Cao, and Qianxiang Wang. Swe-exp: Experience-driven software issue resolution. *arXiv*, 2025.

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, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. Swe-bench pro: Can AI agents solve long-horizon software engineering tasks? *arXiv*, 2025.

Yifeng Ding and Lingming Zhang. Swe-replay: Efficient test-time scaling for software engineering agents. *arXiv*, 2026.

Ryan Ehrlich, Bradley C. A. Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. Codemonkeys: Scaling test-time compute for software engineering. *arXiv*, 2025.

Adam Foster, Desi R. Ivanova, Ilyas Malik, and Tom Rainforth. Deep adaptive design: Amortizing sequential bayesian experimental design. In *ICML*, 2021.

Amirata Ghorbani and James Y. Zou. Data shapley: Equitable valuation of data for machine learning. In *ICML*, 2019.

Yonatan Gideoni, Sebastian Risi, and Yarin Gal. Simple baselines are competitive with code evolution. *arXiv*, 2026.

GLM-5 Team. GLM-5: From vibe coding to agentic engineering. *arXiv*, 2026.

Yifu Guo, Jiaye Lin, Huacan Wang, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, and Mingguang Chen. Se-agent: Self-evolution trajectory optimization in multi-step reasoning with llm-based agents. In *NeurIPS*, 2025.

Hao Han, Jin Xie, Xuehao Ma, Weiquan Zhu, Ziyao Zhang, ZhiLiang Long, Hongkai Chen, and Qingwen Ye. SWE-TRACE: optimizing long-horizon SWE agents through rubric process reward models and heuristic test-time scaling. *arXiv*, 2026.

Yunze Han. A systematic evaluation of trajectory data curation for lora fine-tuning of code agents. In *ICIC*, 2026.

Hiroaki Hayashi, Bo Pang, Wenting Zhao, Ye Liu, Akash Gokul, Srijan Bansal, Caiming Xiong, Semih Yavuz, and Yingbo Zhou. Self-abstraction from grounded experience for plan-guided policy refinement. *arXiv*, 2025.

inclusionAI. Ling-3.0-flash-base. Hugging Face model card, 2026. URL `https://huggingface.co/inclusionAI/Ling-3.0-flash-base`.

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. *arXiv*, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *ICLR*, 2024.

Joongwon Kim, Wannan Yang, Kelvin Niu, Hongming Zhang, Yun Zhu, Eryk Helenowski, Ruan Silva, Zhengxing Chen, Srinivasan Iyer, Manzil Zaheer, Daniel Fried, Hannaneh Hajishirzi, Sanjeev Arora, Gabriel Synnaeve, Ruslan Salakhutdinov, and Anirudh Goyal. Scaling test-time compute for agentic coding. *arXiv*, 2026.

Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. In *NeurIPS*, 2019.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, 2006.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *ICML*, 2017.

Yanzhou Li, Yiran Zhang, Xiaoyu Zhang, Xiaoxia Liu, and Yang Liu. CODESKILL: learning self-evolving skills for coding agents. *arXiv*, 2026.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Xuanjing Huang, Hang Yan, Zhenhua Han, and Tao Gui. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv*, 2026.

Yalan Lin, Yingwei Ma, Rongyu Cao, Binhua Li, Fei Huang, Xiaodong Gu, and Yongbin Li. Llms as continuous learners: Improving the reproduction of defective code in software issues. *arXiv*, 2024.

Murong Ma, Tianyu Chen, Yun Lin, Shuai Lu, Qinglin Zhu, Yeyun Gong, Zhiyong Huang, Peng Cheng, Yan Lu, and Jin Song Dong. From patches to trajectories: Privileged process supervision for software-engineering agents. *arXiv*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *NeurIPS*, 2023.

Fangwen Mu, Junjie Wang, Lin Shi, Song Wang, Shoubin Li, and Qing Wang. Experepair: Dual-memory enhanced llm-based repository-level program repair. *PACMSE*, 2026.

OpenAI. GPT-5.6 system card. Technical report, 2026. URL `https://deploymentsafety.openai.com/gpt-5-6`. Accessed August 2026.

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 *ICML*, 2025.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference. *arXiv*, 2026.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. In *NeurIPS*, 2021.

Qwen Team. Qwen3 technical report. *arXiv*, 2025.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv*, 2025.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *ICLR*, 2018.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

Chaofan Tao, Jierun Chen, Yuxin Jiang, Kaiqi Kou, Shaowei Wang, Ruoyu Wang, Xiaohui Li, Sidi Yang, Yiming Du, Jianbo Dai, Zhiming Mao, Xinyu Wang, Lifeng Shang, and Haoli Bai. Swe-lego: Pushing the limits of supervised fine-tuning for software issue resolving. *arXiv*, 2026.

Dingzirui Wang, Xuanliang Zhang, Keyan Xu, Qingfu Zhu, and Wanxiang Che. Curateevo: Data-curation evolving for agentic post-training. *arXiv*, 2026a.

Qihao Wang, Ziming Cheng, Shuo Zhang, Fan Liu, Rui Xu, Heng Lian, Kunyi Wang, Xiaoming Yu, Jianghao Yin, Sen Hu, Yue Hu, Shaolei Zhang, Yanbing Liu, Ronghao Chen, and Huacan Wang. Memgovern: Enhancing code agents through learning from governed human experiences. *arXiv*, 2026b.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, and et al. Openhands: An open platform for AI software developers as generalist agents. In *ICLR*, 2025a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *ICML*, 2025b.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. Agentgym: Evolving large language model-based agents across diverse environments. *arXiv*, 2024.

Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. Live-swe-agent: Can software engineering agents self-evolve on the fly? *arXiv*, 2025.

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. *arXiv*, 2026a.

Yuan-An Xiao, Pengfei Gao, Chao Peng, and Yingfei Xiong. Reducing cost of LLM agents with trajectory reduction. *PACMSE*, 2026b.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. Doremi: Optimizing data mixtures speeds up language model pretraining. In *NeurIPS*, 2023.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *NeurIPS*, 2024.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. In *NeurIPS*, 2025.

Z.ai. GLM-5.1. Hugging Face model card, 2026. URL https://huggingface.co/zai-org/GLM-5.1.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: LLM agents are experiential learners. In *AAAI*, 2024.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *ICML*, 2024.

Overview of the Appendix

The appendix follows the paper from motivation and collection design to student outcomes, collection diagnostics, and cost. Table 5 maps the main-text questions to their supporting sections.

A Evidence Behind the Motivation. 14

B Collector Algorithm, Schemas, and Prompts 18

C Data and Experimental Conditions 23

D Training, Evaluation, and Metrics. 25

E Complete Downstream Results 27

F Collection Mechanism in Detail 29

G Component Controls, Cost, and Best-of-N 33

H Case Study and Advice Diagnostics 35

I Detailed Related Work 37

Table 5: Where each main-text claim is documented in full.

Main text	Appendix
Section 2.1: marginal coverage of independent rollouts	Appendix A.1
Section 2.2: mixed-outcome audit and its replication	Appendices A.2 and A.3
Section 3: compiler, runtime, selector, export, prompts	Appendix B
Section 4.1: issue pool, conditions, training, evaluation	Appendices C and D
Section 4.2: main runs, paired contrasts, language breakdown	Appendix E
Section 4.2: second student	Appendix E.4
Sections 4.4 and 4.5: yield, process, and comparison by history type	Appendix F
Sections 4.2 and 4.5: randomized component controls and replay audit	Appendix G.1
Section 4.3: history scale, cost, stricter selector, Best-of-N	Appendices G.2, G.1, and G.3
Sections 1, 3.3, and 4.5: running Rust example	Appendix H.1
Sections 1, 4.1, and 4.2: agent-visible advice	Appendices C.3 and H.2
Section 5: related work	Appendix I

A Evidence Behind the Motivation

A.1 Coverage and Overlap of Independent Attempts

We quantify how issue and patch-target coverage grows as more independent attempts are included. Figure 1A reports issue coverage, and Table 6 gives the patch-target coverage shown in panel B.

For issue i with s_i resolved attempts among five, the expected number of issues solved by a k-attempt subset is

$$\sum_i \left[1 - \frac{\binom{5-s_i}{k}}{\binom{5}{k}} \right], \tag{1}$$

with an empty numerator treated as zero. This gives 58.4, 71.5, 79.5, 84.6, and 88.0 expected solved issues for $k = 1, \ldots, 5$, as reported in Figure 1A.

Pairwise patch-target Jaccard averages 0.918 on stable-solved, 0.745 on mixed, and 0.841 on never-solved issues. Among the 198 never-solved issues with a defined overlap value, 133 have an overlap of at least 0.8 and 93 have an overlap of 1.0. The remaining 14 issues have no value because attempts that produce no patch provide no patch-target set to compare. These results show substantial target

Table 6: Expected patch-target coverage over exact subsets of the five SeedBank attempts. Values are the mean fraction of each issue's five-attempt union.

History type	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
Stable-solved	89.96%	93.98%	96.58%	98.38%	100%
Mixed	73.20%	86.42%	92.96%	96.98%	100%
Never-solved	80.09%	88.89%	93.75%	97.19%	100%

overlap even when all attempts fail. They do not explain why the attempts fail, but they show that independent sampling often returns to the same repair targets.

Behavior-level redundancy. Overlapping repair targets do not imply that attempts follow identical action sequences. We therefore also measure behavioral similarity using `SequenceMatcher` over the action-token sequences of all $\binom{5}{2}$ attempt pairs per issue, giving 3,000 pairs in total. Actions are tokenized as in Appendix A.2. Pairwise values are averaged within each issue and then across issues by history type. Table 7 reports the results.

Mixed histories have the lowest mean similarity on every measure, followed by never-solved and stable-solved histories. This ordering matches the patch-target overlap in Figure 1B. Excluding the 348 of 1,500 attempts that fail or time out in the collector preserves the ordering and increases the mixed-to-never-solved d_{full} gap from 0.012 to 0.026. These comparisons use issue-level means. Mixed and never-solved histories reverse order under pooled-pair medians and under medians that exclude timed-out and errored attempts.

On never-solved issues, high patch-target overlap coexists with a full-sequence similarity of 0.503. Failed attempts often return to the same repair targets without simply repeating the same action sequence.

Table 7: Behavior-level redundancy of the five SeedBank attempts. Pairwise similarity is averaged within each issue and then across issues by history type. d_{full} and d_{25} are `SequenceMatcher` ratios over the full and first-25 action-token sequences. J_{explore} and J_{patch} are Jaccard overlaps of explored and modified targets, with J_{patch} computed over issues with a defined value.

History type	Issues	Pairs	d_{full}	d_{25}	J_{explore}	J_{patch}
Stable-solved	37	370	0.605	0.629	0.665	0.918
Mixed	51	510	0.491	0.560	0.625	0.745
Never-solved	212	2,120	0.503	0.571	0.642	0.841

The coverage and overlap results show where additional independent attempts become increasingly redundant. They do not show whether earlier attempts can identify where additional collection effort may be useful. We examine that question next.

A.2 Decision Differences in Mixed Histories

Mixed histories are informative because they contain both successful and failed attempts on the same issue. The audit asks whether these histories reveal decision points where more than one next action is reasonable given the available evidence. It uses outcomes first to locate a meaningful difference between a successful and a failed attempt. It then removes the outcomes and future feedback and asks whether the two states are comparable and whether both next actions are defensible from the information available at that point. This directly tests the allocation signal studied in Section 2.2. The issue set, pair construction, model inputs, label spaces, and aggregation rule were fixed before the results were computed. The audit model was `GLM-5.1` at temperature zero.

Pair construction. The input contains the 51 mixed issues whose five SeedBank attempts include at least one resolved and one unresolved trajectory. Here, *native* means generated by the coding agent from its ordinary view, with no collector text. Canonical native events are converted into 1-based action timelines after planning-only events are removed. Each action records its category, repository target, normalized command or edit, the immediately preceding native reasoning, and the resulting observation and error flag. Whitespace is normalized. Problem statements are capped at 7,000 characters, per-action native reasoning at 900, observations at 800, edit summaries at 1,700, and other action summaries at 700.

For every issue, we score every resolved/unresolved pair. When a naturally completed unresolved trajectory exists, collector failures and timeouts are excluded from the candidate pool. Otherwise, the censored unresolved attempts form the pool. This produces 43 primary pairs and eight explicitly censored pairs.

The audit compares a successful and failed attempt that reached similar observed situations. We select the highest-scoring pair for each issue. Let d_{full} be the `SequenceMatcher` ratio between the full action-token sequences and d_{25} the ratio between their first 25 actions. An action token is its action category and normalized target. Terminal, environment, and validation actions additionally include the recognized command family. Let J_{explore} and J_{patch} be Jaccard overlap of explored and modified repository targets. The score is

$$s = 0.45d_{\text{full}} + 0.30d_{25} + 0.15J_{\text{explore}} + 0.10J_{\text{patch}}. \tag{2}$$

The maximum-scoring pair is selected, with deterministic tie-breaking by fewer coding actions and then run ID.

Stage 1: retrospective localization. The first stage uses the complete trajectories and their outcomes only to locate an early decision difference that may matter for the repair route. The model receives the problem statement, both complete action timelines, their outcomes, the censor flag, and the pair score. It returns a status, one of `found`, `no_meaningful_divergence`, and `not_comparable`, together with a rationale and a confidence. For a found case, it also returns a boundary index for each route, summaries of the shared state and the two route commitments, a divergence type, and an outcome-relevance label. That label is one of `direct`, `plausible`, `weak`, and `none`. Returned indices are rejected if they fall outside the supplied timelines. The audit uses the following exact system instruction.

Stage 1: retrospective-localization instruction

```
Retrospectively compare one resolved and one unresolved native
coding-agent trajectory for the same issue.  Locate the earliest
outcome-relevant persistent route divergence.  Ignore task-tracker
updates, harmless ordering differences, equivalent reads, and wording
changes.  A meaningful divergence changes the working hypothesis,
repository scope, edit semantics, validation strategy, failure
recovery, or stopping decision.  Outcomes are provided only to
locate the boundary; do not infer a gold patch or claim that every
difference caused the outcome.  If the trajectories never reach
comparable states or no meaningful divergence can be supported, say
so.  Cite only supplied action indices.  Return one JSON object and
no prose outside it.
```

The full divergence-type label space is localization, hypothesis, implementation, validation, recovery, finish, resource/infra, and other. Only hypothesis, implementation, and validation occur among the 44 localized cases.

Stage 2: outcome-blind action audit. Only a Stage-1 `found` case proceeds to Stage 2. For each source route, the model sees the prefix strictly before its localized boundary, including the feedback already available at that point. It also sees the immediate candidate action with its observation removed. It does not receive either trajectory outcome, candidate feedback, future actions, the gold patch, the Stage-1 retrospective rationale, or the solved/failed source identity. The two sources are exposed only as routes A and B, with their source identities hidden.

This stage asks whether the visible states are sufficiently comparable and whether both immediate actions are reasonable using only the information available at those states. The audit uses the following exact system instruction.

Stage 2: prospective branch-audit instruction

```
Audit whether two native coding decisions represented a real
prospective fork state.  The two prefixes come from independent
attempts on the same issue, and route labels are opaque.  You do not
know either outcome or any candidate feedback.  First decide whether
```

```
the visible states are sufficiently comparable.  Then judge each
immediate candidate only from its own visible prefix:  is the action
supported, and could it obtain decision-relevant information?  Two
choices are reasonable only when both are prospectively defensible,
semantically distinct, and neither depends on hidden future evidence.
Do not reward mere activity, extra length, or knowledge of a gold
solution.  Return one JSON object and no prose outside it.
```

The structured response records five labels, a reason, and a confidence. State comparability is equivalent, similar with minor differences, or different. Route distinctness is distinct, equivalent, or unclear. Support for each candidate is supported, partially supported, or unsupported. Information gain for each candidate is high, medium, low, or none. The schema calls the final field prospective branchability. Its labels are two reasonable routes, one reasonable route, equivalent routes, non-comparable, and insufficient evidence. The call is then repeated with A/B order reversed.

Execution, aggregation, and missingness. We aggregate the two route-order judgments before counting supported decision points. Two `two_reasonable_routes` votes produce the strict label. One such vote produces the possible-two-routes label. Two one-route votes produce a one-route consensus only when they retain the same source-level support pattern. Otherwise, the result is order-sensitive. Other identical votes retain their shared branchability label, and any remaining disagreement is order-sensitive.

Stage 1 returns valid structured results for 47/51 pairs. These comprise 44 localized divergences and three no-meaningful-divergence cases. The remaining four are collector-censored pairs whose audit calls failed after retries. None is imputed as a negative. Both orderings complete for all 44 localized cases, giving 88 valid Stage-2 responses. In the primary set, this gives 43 complete dispositions, 40 localized cases, and three no-meaningful-divergence cases. The eight censored pairs are tabulated separately.

In the primary set, 21 strict and ten possible cases give 31 decision points with evidence that more than one next action may be worth comparing. Order-sensitive cases do not enter this count. Table 8 gives the complete funnel.

Table 8: Complete primary audit funnel. "Share" retains the original 51-issue denominator, while "Retention" uses the preceding row. The eight collector-censored pairs remain in Table 9 but do not enter the primary funnel after pair selection.

Stage	Count	Share of 51	Retention
Selected mixed-issue pairs	51	100.0%	100.0%
Naturally completed unresolved pair	43	84.3%	84.3%
Complete audit disposition	43	84.3%	100.0%
Localized divergence and dual-order audit	40	78.4%	93.0%
Decision point supporting comparison	31	60.8%	77.5%

A.3 Independent Cross-Model Audit Replication

We next test whether the mixed-history audit depends on the model used to perform it. An independent Auditor B, `gpt-5.6-sol` (OpenAI, 2026), replays the same 51-pair set used in Appendix A.2. Auditor B never receives Auditor A's responses, labels, aggregate counts, or paper claims. Its Stage-1 run uses the same retrospective localization instruction and structured schema. Its Stage-2 run again hides outcomes, candidate feedback, future actions, the gold patch, and source identity, and evaluates both route orders. Calls use medium reasoning effort and provider-default temperature.

The end-to-end replication independently completes all 43 primary pairs. Of these, 41 contain a localized divergence and two contain no meaningful divergence. Auditor B identifies 26 decision points supporting comparison, comprising 16 strict and ten possible cases. Auditor A identifies 31, comprising 21 strict and ten possible cases. These aggregate counts do not by themselves establish agreement on individual cases.

The two auditors agree on `found` versus `no_meaningful_divergence` in 40/43 primary and 4/4 commonly completed censored cases. They both localize 43 cases overall. The solved-route and

Table 9: Complete aggregate audit classifications. Divergence type and outcome relevance partition the Stage-1 `found` rows, while final disposition partitions every pair. "Primary" uses naturally completed unresolved attempts, and "censored" contains collector failures or timeouts.

Family	Classification	All (51)	Primary (43)	Censored (8)
Stage 1	Localized divergence	44	40	4
	No meaningful divergence	3	3	0
	Not analyzed: API long tail	4	0	4
Divergence type	Hypothesis	3	3	0
	Implementation	28	25	3
	Validation	13	12	1
Outcome relevance	Direct	25	21	4
	Plausible	17	17	0
	Weak	2	2	0
Final disposition	Strict two reasonable routes	23	21	2
	Possible two reasonable routes	11	10	1
	One reasonable route, consensus	3	2	1
	Order-sensitive	6	6	0
	Equivalent routes	1	1	0
	No meaningful divergence	3	3	0
	Not analyzed: API long tail	4	0	4

failed-route boundary distances each have median zero. Twenty boundary pairs match exactly, and 26 divergence-type labels match exactly.

To separate disagreement about where the decision difference occurs from disagreement about the actions themselves, we also compare the auditors at fixed boundaries. This comparison uses the boundaries from the first 32 pairs completed by Auditor A, including 28 primary and four censored pairs. Auditor B judges the same outcome-blind inputs in both route orders. On the 28 primary pairs, Auditor B identifies 16 decision points supporting comparison, and both auditors support 15.

Table 10 compares the final judgments on whether both actions are defensible. Binary-label agreement is 60.5% on the end-to-end primary set and 64.3% under fixed boundaries. The auditors jointly support 20 primary cases end to end.

Table 10: Independent cross-model replication of the mixed-history audit. N is the common complete-disposition set for each row. Positive means strict or possible two reasonable routes. "Both" is the conservative positive intersection. Pos./Neg. are positive/negative agreement. Both auditors complete all 43 primary pairs end to end. Auditor A identifies 31 supported comparison points and Auditor B identifies 26. Raw, Pos., Neg., κ, and AC1 describe binary audit labels, not agreement on divergence localization or the full label vocabulary.

Track	Set	N	A+	B+	Both+	Raw	Pos.	Neg.	κ	AC1
End to end	Primary	43	31	26	20	60.5%	70.2%	41.4%	.129	.285
End to end	Censored	4	3	3	3	100.0%	100.0%	100.0%	1.000	1.000
Fixed boundary	Primary	28	24	16	15	64.3%	75.0%	37.5%	.205	.397
Fixed boundary	Censored	4	3	2	2	75.0%	80.0%	66.7%	.500	.529

The audits identify candidate locations for comparison rather than prescribing which action to take. Agreement is stronger on locating meaningful differences than on judging whether both actions are reasonable. The compared prefixes come from independent attempts. During collection, alternatives are instead executed from the same checkpoint. Appendix F.4 reports whether the coding agent proposes distinct actions and whether short execution provides evidence for choosing between them.

B Collector Algorithm, Schemas, and Prompts

This appendix gives the implementation details behind Section 3. The collector uses history only to decide where additional collection effort may be useful and what evidence should distinguish candidate routes. The coding agent receives its ordinary context and generates all candidate actions itself.

B.1 End-to-End Collection Algorithm

Algorithm 1 SWE-Trail collection for one issue. Orange marks collector-private operations; blue marks coding-agent generation from its ordinary visible context.

Require: issue x_i, ordered history H_i, coding agent π, candidate count K, suffix horizon L

$E_i \leftarrow \textsc{AnalyzeHistory}(x_i, H_i)$
$P_i \leftarrow \textsc{RunReadOnlyChecks}(x_i, E_i)$
$(R_i, D_i, u_i, \lambda_i) \leftarrow \textsc{CompileAndFreeze}(x_i, E_i, P_i)$
initialize agent-visible trajectory τ, workspace w, and *compared* $\leftarrow$ false
while τ is not terminal **do**
  $(a, o, w) \leftarrow \textsc{AgentStep}(\pi, \tau, w)$
  append generated action a and observation o to τ
  $d \leftarrow \textsc{MatchDetector}(D_i, \tau, w)$
  if $\neg$*compared* and $d \neq \varnothing$ **then**
    $c \leftarrow \textsc{Checkpoint}(\tau, w)$
    $C \leftarrow \textsc{SampleCandidates}(\pi, \tau, K, \lambda_i)$ ▷ λ_i sets temperatures
    $C' \leftarrow \textsc{SemanticDeduplicate}(C)$
    $a_0 \leftarrow$ default candidate in C'
    $a_1 \leftarrow \textsc{PrivateShortlist}(C', R_i, d)$
    if a_1 exists **then**
      $(s_0, w_0) \leftarrow \textsc{ExecuteSuffix}(c, a_0, L)$
      $(s_1, w_1) \leftarrow \textsc{ExecuteSuffix}(c, a_1, L)$
      $j \leftarrow \textsc{BlindSelect}(u_i, d, R_i, s_0, s_1)$ ▷ R_i is background evidence only
      if j is tie or invalid **then**
        $j \leftarrow 0$
      end if
      $(\tau, w) \leftarrow \textsc{ContinueFrom}(c, s_j, w_j)$
    end if
    compared $\leftarrow$ true
  end if
end while
return $\textsc{ExportContinuousTrajectory}(\tau)$

Algorithm 1 shows the main collection path. For each detector state d, $u_i(d)$ specifies what evidence should be considered when comparing the executed routes. Policy calls, including candidate generation and suffix execution, receive only the ordinary issue, conversation, and repository observations. The schedule λ_i changes sampling temperatures without adding history to the coding-agent prompt. Shortlisting and selection choose among actions already proposed by the coding agent. They do not write new actions or replace the agent's reasoning. Proposal-only rechecking and runtime failures are described in Appendix B.5.

B.2 Collection Cost Accounting

We separate the size of the exported training corpus from the compute used to collect it. Given an existing set of prior attempts, we use two accounting boundaries:

$$\begin{aligned} C_{\text{overhead}} &= C_{\text{compile}} + C_{\text{candidates}} + C_{\text{extra suffix}} \\ &\quad + C_{\text{selection}} + C_{\text{retry}}, \\ C_{\text{bank}} &= C_{\text{rollout}} + C_{\text{overhead}}. \end{aligned} \tag{3}$$

Here C_{overhead} is the additional compute beyond an ordinary rollout, while C_{bank} includes both the ordinary rollout and this overhead. We report model tokens and environment actions under these boundaries. Summed trajectory-hours are reported separately from concurrent wall-clock time. Appendix G.2 gives the measured breakdown. Provider-token accounting is complete for H5 and H10. H0 records runtime calls but not offline compilation, and SeedBank has no source-provider usage record. We do not impute unavailable components.

B.3 Offline Compiler

The compiler turns information from earlier attempts into fixed rules for deciding where additional collection effort may be useful in a new rollout. It follows the same sequence for each issue. All model-produced fields pass schema and content checks before they can affect collection.

1. **History.** Read the ordered source attempts and their observed outcomes.
2. **Evidence.** Extract supported patterns from the history, including decisions repeated across successful attempts, differences between repair routes, recurring failures, exhausted searches, and unresolved questions. Each item cites the source events that support it.
3. **Read-only checks.** Run bounded repository checks when the history leaves an unresolved question. Each check records its command, observation, interpretation, and provenance. Checks cannot edit source code, access a reference patch, or directly create a detector rule.
4. **Rules.** Combine the history evidence and read-only checks into candidate detector rules. Each rule includes observable conditions, a local comparison objective, and explicit abstention conditions.
5. **Admission.** Parse, lint, deduplicate, and replay the candidate rules on the source trajectories. Fix all admitted rules before the main collection begins.

Internally, each evidence item records an identifier, source type, hypothesis or constraint, direct observation, bounded interpretation, uncertainty, history type, and links to supporting or contradicting evidence. The accompanying human-readable representation forms the private acquisition record R_i. Neither representation enters a coding-agent request.

B.4 Detector Schema and Admission Checks

A detector rule can guide collection only after its rule and supporting evidence pass the checks below. An *active detector rule* has passed these checks and is fixed for collection. At runtime, the rule is evaluated against the actions and observations seen so far.

Each detector rule contains the following fields.

Scope	A state and decision family.
Predicates	Observable conditions with a minimum-observation window.
Support	Evidence IDs and the rationale for spending additional collection effort at this state.
Comparison	A local objective for comparing executed routes.
Status	Confidence, replay coverage, and admission status.
Fallback	Explicit abstention and fallback conditions.

Executable predicates may use only features available during the current rollout. They cannot contain a reference patch, hidden tests, historical solution actions, repository-specific instructions to the coding agent, or future outcome labels. H5 and H10 rules must cite history evidence and pass history replay. H0 rules must cite source-clean repository evidence. Parse failure, prohibited content, contradictory rationale, duplicate signatures, or insufficient support causes abstention.

H5 compiles 687 candidate detector rules. Of these, 569 match at least one source trajectory during replay, and 397 pass the admission checks for collection. Appendix H.1 gives the concrete post-edit read-without-validation rule used in the running Rust example.

B.5 Online Runtime Details

The runtime evaluates the detector after each new repository observation. A trajectory executes at most one comparison. If all sampled candidates are equivalent, no comparison is executed. The collector may sample candidates once more after the default action runs.

1. **Checkpoint.** Snapshot the agent-visible conversation, task workspace, and runtime metadata when a detector rule matches the current rollout.
2. **Candidates.** Sample eight one-step continuations from the same model weights and visible prefix. Candidate zero uses temperature 0.7. H0 uses 0.7 for every slot. Within H5 and H10, stable-solved histories use 0.7 throughout, mixed histories alternate 0.7/0.85, and never-solved histories cycle 0.7/0.85/1.0.

3. **Shortlist.** Cluster semantically equivalent proposals. Keep candidate zero and privately shortlist at most one distinct, executable alternative.
4. **Execution.** Restore two isolated workspaces from the same checkpoint and execute the default and alternative candidates for two observations. If restoration or execution fails, retain candidate zero.
5. **Selection.** Shuffle the two route identities and compare their reasoning, actions, workspace changes, and repository observations using the local comparison objective. The private record provides background evidence only. Select a route only when the executed suffixes provide clear evidence that one makes better progress. Otherwise, retain the default route.
6. **Export.** Continue only from the selected workspace and export one continuous trajectory. Preserve the unselected suffix only in the private collection record.

B.6 Information Boundaries and Export Checks

The exported trajectory must preserve the coding agent's original reasoning and actions without adding private collector information. Table 11 summarizes which information each stage may use. The export checks enforce this boundary and verify that the retained actions and tool results form one continuous trajectory.

Table 11: Information boundary enforced during collection and export. "Private" means that information may inform the collector but never enters a coding-agent request. The last column contains the two executed suffixes used by the route selector.

Component	Visible prefix	Prior attempts	Private record	Executed suffix pair
Offline compiler	issue only	H-dependent	yes	no
Candidate generation	yes	no	no	no
Private shortlist	yes	yes	yes	no
Blind suffix selector	yes	private	private	yes
Exported trajectory	selected route	no	no	selected only

The private comparison log records the visible prefix, shared checkpoint, candidate actions, both executed suffixes, and the selector's decision. Isolation checks require identical prefix and pre-execution workspace digests and separate workspaces after restoration. A failed check prevents the alternative route from being selected.

Export canonicalization restores the coding agent's original reasoning attached to each selected action, pairs every tool call with exactly one result, and removes private collector events. It does not rewrite or reconstruct reasoning. A record is rejected if a selected event cannot be traced to the raw stream, tool pairing fails, the exported patch differs from the selected workspace, or private collector content appears in the training trajectory.

B.7 Prompts and Inputs

Table 12 gives the input and output boundary for each model-assisted collector stage. Candidate generation is intentionally absent. It uses the ordinary coding-agent prompt and visible trajectory prefix, with no SWE-Trail instruction and no history. The system instructions below are reproduced exactly as used during collection.

Table 12: Inputs and outputs of the model-assisted collector stages. All outputs remain private to the collector and are never inserted into a coding-agent request.

Stage	Structured inputs	Required output	Downstream consumer
History diagnosis	issue, ordered attempts, outcomes	mechanisms, decision differences, unresolved questions	repository-check planner
Repository exploration	diagnosis, repository snapshot	bounded commands, findings, provenance	evidence auditor

Table 12 (continued)

Stage	Structured inputs	Required output	Downstream consumer
Evidence audit	diagnosis, repository observations	supported/contradicted claims, uncertainty	rule synthesis
State synthesis/compiler	audited evidence, feature schema	states, predicates, objectives, abstentions	private detector
Candidate shortlist	visible prefix, candidates, private record	default plus one existing candidate index	matched executor
Suffix selector	shuffled suffixes, comparison objective, private record as prior	select A, select B, or tie with evidence	trajectory continuation

History diagnosis: system instruction

```
Audit same-issue coding trajectories before any environment
exploration.  Separate successful mechanisms, failed mechanisms,
decision boundaries, and unresolved repository questions.  Failed
trajectories are counterexamples.  Do not generate detector code,
advice, file hints, or a patch.  Return JSON only.
```

Bounded repository exploration: system instruction

```
Act as a bounded source-clean repository explorer.  Start from
the supplied history diagnosis, refine competing hypotheses, and
propose safe read-only or validation probes.  Do not edit source,
use network/history, or generate a detector.  One command per probe,
with no pipes or redirection.  Return JSON only.
```

Evidence audit: system instruction

```
Act as an auditor separate from history diagnosis and repository
exploration.  Admit only claims with exact supplied evidence.
Separate observation, supported inference, negative boundary, and
uncertainty.  Do not generate detector code, patch text, or a gold
answer.  Return JSON only.
```

Acquisition-state synthesis: system instruction

```
Synthesize prospective issue-specific acquisition states only
after reviewing history diagnosis and audited repository probes.
Conditions may use only the supplied public feature schema and must
not encode repository targets.  Explain every condition and cite
supplied history and audited probe evidence.  Return JSON only.
```

History-conditioned detector compiler: system instruction

```
Compile an issue-private prospective branch detector from atomic
same-issue history experiences and, when supplied, audited repository
probe experiences.  History failures are counterexamples.  Probe
facts refine issue knowledge but must not automatically become
detector states.  Preserve the strong native prior for stable-solved
issues.  Use only declared process features, cite exact evidence, and
abstain when branch timing is unsupported.  Return JSON only.
```

For H0, the compiler instead receives the following history-free instruction.

H0 detector compiler: system instruction

```
Compile an issue-private prospective branch detector using only the
issue statement and source-clean repository probe evidence.  No
historical trajectory, outcome regime, or prior detector is available.
Use only declared process features, cite exact supported probe
experiences, and abstain when static repository evidence cannot
justify a prospective resampling point.  Do not encode repository
paths or patch answers.  Return JSON only.
```

Candidate shortlist: system instruction and user contract

```
Shortlist native coding-agent actions generated from one identical
prefix.  Private history and audited probe evidence may only
distinguish targets already proposed by a candidate.  Candidate 0
is the native control and must remain in the matched pair.  Do not
invent a new action.  Return JSON only.
```

User contract. `Return provisional_winner_index, ranked_candidate_indices, reason, confidence, and candidate_assessments. Assess visible support, reasoning/action alignment, expected information gain, historical redundancy, and tool validity. Indices use candidate_index from candidate_actions.`

Executed-suffix selector: system instruction and user contract

```
Compare two coding-agent suffixes executed from the same conversation
and workspace checkpoint.  Branch A and Branch B are opaque shuffled
labels with equal status; do not infer which lineage produced either
branch.  Judge only local observed evidence.  Private history is a
failure/success prior, never a gold answer.  Select a branch only
when its executed suffix demonstrates better semantic progress;
otherwise return tie.  The decision field must be exactly select_a,
select_b, or tie.  semantic_progress must be a JSON boolean.  Return
JSON only.
```

User contract. `Return decision, reason, confidence, semantic_progress, and branch_assessments. Each assessment includes branch, new_evidence, validation_signal, tool_error, unchanged_retry, unsupported_finish, and reason.`

C Data and Experimental Conditions

C.1 Collection Issues and Prior Attempts

The main experiments use 300 repository-level issues from SWE-rebench V2 (Badertdinov et al., 2026), with one issue per repository. Each collection condition produces five trajectories per issue. SeedBank provides the five independently sampled attempts used as history by H5. H10 adds five more eligible attempts to the same five, giving ten prior attempts per issue.

We define three history types from the outcomes of the five SeedBank attempts before the main collection begins. The stable-solved group contains 37 issues for which all five attempts succeed. The mixed group contains 51 issues with both successful and failed attempts. The never-solved group contains 212 issues for which all five attempts fail. These labels describe the observed SeedBank outcomes and are separate from how the compiler interprets each history. Appendix F.5 reports the compiler classes.

Collection runs in an offline sandbox with no network access and no repository commit history. The coding agent sees the working tree at the issue commit but cannot access later commits. The reference patch, fixing commit, and externally published solutions are also unavailable. The read-only repository checks in Section 3.1 follow the same restrictions. Any information already encoded in the model weights is shared across conditions, since all collection conditions use the same model on the same issue set.

C.2 Evaluation Benchmarks and Overlap Audit

We evaluate on SWE-bench Multilingual and SWE-bench Pro. SWE-bench Multilingual, released with SWE-smith (Yang et al., 2025), contains 300 issues across nine languages: C, C++, Go, Java, JavaScript, TypeScript, PHP, Ruby, and Rust. Appendix E.3 reports results by language. SWE-bench Pro (Deng et al., 2025) contains longer-horizon repository issues whose reference patches typically span multiple files. We use its public split of 731 tasks. Both benchmarks use the evaluation protocol in Appendix D.2.

We verify that the collection and evaluation sets do not share issues or repositories. Exact instance IDs and canonical lowercase owner/repository names are compared across the sets. The 300 Multilingual tasks span 41 repositories, and the 731 SWE-Pro tasks span 437 repositories. Neither benchmark shares an issue or repository with the collection pool.

C.3 Experimental Conditions

The conditions separate three ways of using existing trajectories. The three curation conditions are controlled adaptations rather than full reproductions of the original methods. They modify the same completed SeedBank trajectories without collecting new interactions. SWE-Exp uses prior attempts to create information that is shown to the coding agent during a new rollout. SWE-Trail instead keeps prior attempts private to the collector and uses them to guide where additional collection effort is spent.

Our SWE-Gym adaptation (Pan et al., 2025) conditions each record on its outcome. Successful records receive the outcome annotation while preserving their original messages and actions. Unresolved records remain unchanged. The adaptation does not apply a condition-specific loss mask.

Our SWE-Lego adaptation (Tao et al., 2026) applies deterministic tool-error masking to successful records. It masks only assistant turns whose tool result indicates strict tool misuse, including invalid arguments, missing paths, permission errors, or shell syntax errors. Failed tests and bug reproduction output remain trainable. Unresolved records remain unchanged.

Our SWE-TRACE adaptation (Han et al., 2026) removes noncritical messages from successful records. It retains interactions involving patch-modified files, validation, final repository inspection, and the final answer. Repeated non-mutating observations are deduplicated. Messages are removed rather than loss-masked. Unresolved records remain unchanged.

Our SWE-Exp adaptation (Chen et al., 2025) compiles the same five prior attempts into an experience message. The coding agent receives this message while generating the new trajectory, and the message remains in the exported training record.

The SWE-Trail conditions keep prior attempts outside the coding agent's context. H0 runs the same collection pipeline without history. H5 uses five prior attempts, while H10 uses ten. Random detection replaces the history-guided comparison location with a random eligible state while leaving the remaining collection procedure unchanged. Random retention keeps H5's history-guided comparison location but randomly chooses which executed route continues. The randomized controls use the same per-trajectory collection budget and execution horizon as H5.

Table 13: Experimental conditions. Every condition produces 1,500 training records. The controlled adaptations use the same SeedBank attempts and training protocol as our main experiments.

Condition	Source	History visible to agent	New collection	Training record	Role
SeedBank	independent rollouts	no	complete rollout	generated trajectory	sampling baseline
SWE-Gym (Pan et al., 2025), outcome-conditioned transform	SeedBank	no	no	outcome-conditioned record	outcome control
SWE-Lego (Tao et al., 2026), strict tool-error masking	SeedBank	no	no	masked record	step control
SWE-TRACE (Han et al., 2026), critical-path reduction	SeedBank	no	no	compressed record	compression control
SWE-Exp (Chen et al., 2025), agent-visible experience	five attempts	yes	complete rollout	experience retained	visibility control
SWE-Trail-H0	no history	no	history-free collection	selected trajectory	no-history control
SWE-Trail-H5 (random detection)	five attempts	no	random comparison location	selected trajectory	location control
SWE-Trail-H5 (random retention)	five attempts	no	history-guided comparison	random executed route	retention control
SWE-Trail-H5	five attempts	no	history-guided collection	selected trajectory	main method
SWE-Trail-H10	ten attempts	no	history-guided collection	selected trajectory	history-scale control

D Training, Evaluation, and Metrics

D.1 Controlled SFT Protocol

The primary student is initialized from Ling-3.0-flash-base, a 124B-parameter base model with 5.1B activated non-embedding parameters (inclusionAI, 2026). To compare the training value of different trajectory sets, we keep the remaining training setup fixed across conditions. Table 14 summarizes these controls.

Each main training corpus contains 1,500 coding trajectories. Every trajectory is used whether or not it resolves the issue. Process-valid and verified-positive subsets are reported only as diagnostics and do not determine which trajectories enter training.

Each condition is trained independently three times from the same initialization on 128 H800 GPUs using the same training schedule. No condition receives separate hyperparameter tuning, and evaluation results are never used to select a trained model. Each resulting model is evaluated once on every task under the fixed protocol in Appendix D.2.

Second student. We repeat the comparison with Qwen3-14B-Base (Qwen Team, 2025). The model is trained with full-parameter supervised fine-tuning on the SeedBank, H0, H5, and H10 trajectory sets. Within this student, all training settings other than the coding trajectories are again fixed. Each dataset is trained independently three times, and every resulting model is evaluated once under the same benchmark protocol. Appendix E.4 reports the complete results.

D.2 Evaluation Protocol

Within each benchmark, all students use the same evaluation setup. This includes the autonomous OpenHands scaffold and prompt, available tools, sampling policy, context and interaction budgets,

Table 14: Shared training settings for the 124B student. All listed settings are fixed across trajectory conditions.

Field	Controlled setting
Initialization	Ling-3.0-flash-base
Model scale	124B total; 5.1B activated non-embedding parameters
Coding trajectories	1,500 per condition; all outcomes retained
Background mixture	identical
Tokenizer and tool representation	identical
Context length and packing	identical
Optimizer and learning-rate schedule	identical
Update budget, passes, and precision	identical
Randomization policy	identical
Condition difference	coding trajectories only
Agent reasoning supervision	identically enabled
Model-selection rule	identical and fixed before evaluation
Training repeats	three independent runs per condition
Evaluation repeats	one complete evaluation per trained model
Reported score	mean resolved count across the three runs

repository construction, and containerized verifier. The trained student is the only object that varies across conditions. No condition receives inference-time tuning.

Every task remains in the denominator, including errors and timeouts under their recorded outcomes. Per-run percentages are rounded only for display. The reported condition mean is computed from the underlying resolved counts before rounding.

For paired comparisons, each evaluation task contributes its mean binary outcome across the three independently trained students in a condition. We resample complete tasks with 20,000 bootstrap replicates and report the percentile 95% interval for the paired difference. We also count tasks for which the first condition resolves more, fewer, or the same number of the three trained models. These intervals describe uncertainty across evaluation tasks conditional on the three trained models. Variation across training runs is reported separately in Appendix E.1. Table 16 in Appendix E.2 gives the complete paired contrasts.

D.3 Metric Definitions

This section defines the metrics used in Section 4 and the appendix tables.

Outcome and collection yield. A trajectory is resolved when its final patch passes the benchmark tests. Resolution is the fraction of evaluated tasks that are resolved. A collection issue is counted as solved when at least one of its five newly collected trajectories resolves. A verified positive is a resolved, process-valid, nonempty selected trajectory that also passes the export checks. The focused Top-5 analysis in Appendix G.3 uses the same exact strict-positive definition for both candidate banks.

Process. Repeated reads count repeated accesses to the same normalized file target. Validation counts executable checks. Post-final-edit validation records whether at least one executable check occurs after the final edit.

Collection mechanism. Candidate sampling counts a rollout in which a detector rule matches the current state and the collector samples next-action candidates. Sampling alone does not imply that a comparison is executed. Equivalent proposals or a failed workspace restoration can prevent execution.

An executed comparison requires both selected candidate suffixes to run from the same checkpoint. Candidate diversity is the fraction of candidate-sampling cases in which the eight proposals contain at least two semantically distinct actions. Alternative selection is the fraction of completed comparisons in which the non-default route is retained.

Semantic progress records whether the selector finds clear evidence that one executed suffix makes better local progress under the comparison objective $u_i(d)$. We report this quantity over all rollouts.

Candidate coverage asks whether a useful action appears among the eight sampled proposals at all. Appendix G.1 evaluates this with exact-prefix replay.

Tokens and time. Token counts use provider-reported prompt and completion tokens. Reported reasoning tokens are a subset and are not added a second time. Collection-overhead usage includes compiler, candidate-generation, additional suffix, selector, and retry calls while excluding the ordinary coding rollout. Where tables use the term *acquisition-only*, it refers to this same boundary. Summed trajectory-hours add elapsed time across individual tasks and are not concurrent suite wall-clock time.

Statistical tests for collection outcomes. The five trajectories collected for one issue are not independent observations, so collection-level bootstrap intervals resample whole issues. Win/loss/tie counts compare the number of resolved trajectories out of five for each issue. Appendix F additionally reports an exact sign-flip test over nonzero issue-level differences. These collection-level analyses describe differences in the collected data. The downstream training endpoint is reported separately in Section 4.2.

E Complete Downstream Results

E.1 Per-run resolution

Table 15 reports all three independent training runs for the 124B student. Every H5 run exceeds every SeedBank run on both benchmarks. On Multilingual, the lowest H5 result, 64.00, also exceeds the highest H0 result, 63.67. The H5 and H0 ranges overlap on SWE-Pro.

For the H5–SeedBank comparison in Section 4.2, we use an exact one-sided permutation test over the six independently trained models. The test statistic is the difference in mean resolution,

$$T = \bar{s}_{\mathrm{H5}} - \bar{s}_{\mathrm{SeedBank}}.$$

Under the null hypothesis that the six run-level scores are exchangeable between conditions, there are $\binom{6}{3} = 20$ ways to assign three runs to each condition. The exact p-value is the fraction of assignments whose mean difference is at least T_{obs}. On both benchmarks, the three H5 scores are the three highest scores, so only the observed assignment reaches T_{obs}. This gives a one-sided $p = 1/20 = 0.05$ on each benchmark. With three runs per condition, this is the smallest value attainable by this test.

This test uses independently trained models as the statistical unit. Appendix E.2 examines differences across evaluation tasks while keeping the trained models fixed.

Table 15: Independent downstream training runs, three per condition. Means are computed from integer resolved counts before rounding. Table 13 defines the experimental conditions.

	Multilingual				SWE-Pro			
Condition	R1	R2	R3	Mean	R1	R2	R3	Mean
SeedBank independent rollouts	63.00	63.67	63.33	63.33	41.18	41.31	41.18	41.22
SWE-Gym (outcome-conditioned transform)	63.67	65.00	63.33	64.00	40.36	41.59	40.63	40.86
SWE-TRACE (critical-path reduction)	65.33	64.67	63.33	64.44	41.72	40.08	40.49	40.77
SWE-Lego (strict tool-error masking)	66.00	63.00	64.67	64.56	39.81	42.13	42.95	41.63
SWE-Exp (agent-visible experience)	62.00	61.00	63.00	62.00	38.44	38.58	38.30	38.44
SWE-Trail-H0	63.67	63.00	63.67	63.44	41.04	41.72	43.09	41.95
SWE-Trail-H5 (random detection)	64.67	63.33	64.67	64.22	41.59	41.72	41.31	41.54
SWE-Trail-H5 (random retention)	66.00	64.67	63.67	64.78	41.04	42.13	41.72	41.63
SWE-Trail-H5	67.67	65.33	64.00	65.67	41.72	43.37	43.37	42.82
SWE-Trail-H10	65.67	65.00	67.00	65.89	43.64	42.68	42.54	42.95

E.2 Task-paired contrasts

We next examine downstream differences on the same evaluation tasks while keeping the trained models fixed. The contrasts cover H5–H0, H10–H0, H10–H5, and the stricter-selector condition of Appendix G.1 against H5. Table 16 and Figure 4 use the task-paired bootstrap protocol defined in Appendix D.2. Comparisons with SeedBank use the run-level results and mean differences in Appendix E.1.

The H5–H0 point estimates are positive on both benchmarks, at 2.22 points on Multilingual and 0.87 points on SWE-Pro. Their 95% task-bootstrap intervals, however, both include zero. Increasing the history from five to ten attempts adds only 0.22 and 0.14 points over H5. The stricter-selector condition is below H5 on both benchmarks.

Table 16: Task-paired downstream contrasts conditional on the trained models. W/L/T counts tasks for which the first condition resolves more, fewer, or the same number of the three independently trained models. Intervals resample evaluation tasks rather than training runs.

Contrast	Multilingual Δ [95% CI]	W/L/T	SWE-Pro Δ [95% CI]	W/L/T
H5 − H0	2.22 [−0.2, 4.6]	55/35/210	0.87 [−0.8, 2.5]	130/111/490
H10 − H0	2.44 [−0.1, 4.9]	61/39/200	1.00 [−0.7, 2.7]	134/112/485
H10 − H5	0.22 [−2.0, 2.5]	51/49/200	0.14 [−1.3, 1.6]	117/114/500
Stricter selector − H5	−1.78 [−4.0, 0.4]	42/58/200	−0.91 [−2.6, 0.8]	105/125/501

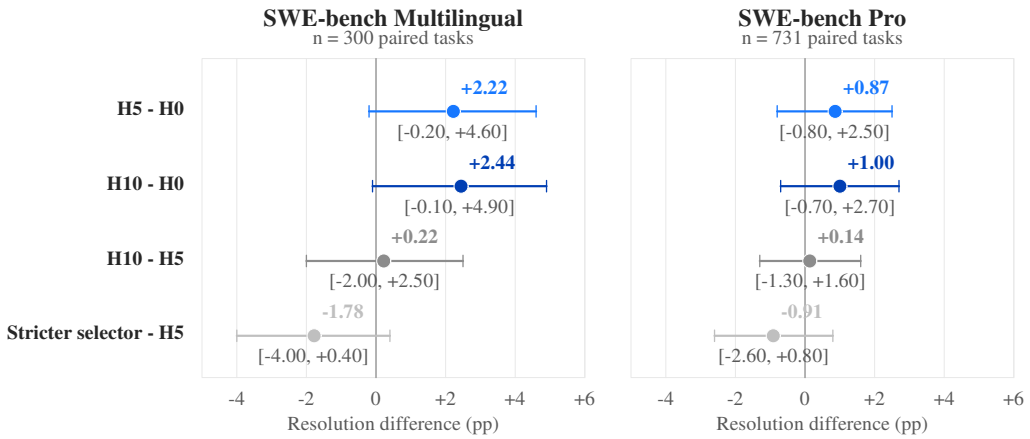


Figure 4: Task-paired downstream contrasts conditional on the trained models. Points show mean differences across the three trained models, and bars show task-cluster 95% bootstrap intervals.

E.3 Multilingual results by language

Table 17 breaks down Multilingual resolution by repository language using the same three-run aggregation as the overall results. Each language contains only 12–44 tasks, so the breakdown is descriptive.

H5 is no lower than H0 on seven of the nine languages. The two decreases, on JavaScript and PHP, correspond to roughly one third of a task per run each. The largest increase over H0 occurs on C.

We also compare this pattern with the language composition of the collection set. It contains only three C issues out of 300 and no Ruby issues, while nearly one third of its issues are in Python, which does not appear in Multilingual. The observed gains do not simply follow the language composition of the collection set.

E.4 Results for the Qwen3-14B Student

We repeat the SeedBank, H0, H5, and H10 comparison with Qwen3-14B-Base to test whether the downstream gains also appear with another student model. Table 18 reports three independent training runs under the protocol of Appendix D.1.

H5 improves mean resolution over SeedBank by 3.11 percentage points on Multilingual and 2.42 points on SWE-Pro. It also exceeds H0 by 1.67 and 1.60 points, respectively. Every H5 run exceeds every SeedBank and H0 run on both benchmarks.

Table 17: Multilingual resolution by repository language, in percent. Δ columns compare H5 with SeedBank and H0.

Language	Tasks	SeedBank	H0	H5	Δ SeedBank	Δ H0
C	30	60.0	56.7	68.9	(+8.9)	(+12.2)
C++	12	63.9	69.4	69.4	(+5.6)	0.0
Go	42	34.1	36.5	37.3	(+3.2)	(+0.8)
Java	43	72.9	72.1	72.9	0.0	(+0.8)
JavaScript	31	74.2	72.0	71.0	(−3.2)	(−1.1)
PHP	43	69.8	72.9	72.1	(+2.3)	(−0.8)
Ruby	44	61.4	62.9	64.4	(+3.0)	(+1.5)
Rust	43	74.4	71.3	75.2	(+0.8)	(+3.9)
TypeScript	12	55.6	55.6	61.1	(+5.6)	(+5.6)
All	300	63.33	63.44	65.67	(+2.33)	(+2.22)

Increasing the history from five to ten attempts adds 1.00 point on Multilingual and 0.36 points on SWE-Pro. Every H10 run exceeds every H5 run on both benchmarks. These additional gains are smaller than the H5–H0 gains.

Table 18: Resolution of the Qwen3-14B-Base student in three independent training runs per trajectory set. Values are percentages over the 300 Multilingual and 731 SWE-Pro tasks. Means are computed from resolved counts before rounding, and Δ is relative to SeedBank in percentage points.

	Multilingual					SWE-Pro				
Training dataset	R1	R2	R3	Mean	Δ	R1	R2	R3	Mean	Δ
SeedBank	11.67	11.00	11.33	11.33		10.94	11.22	11.08	11.08	
SWE-TRAIL-H0	12.67	13.00	12.67	12.78	+1.44	11.90	11.76	12.04	11.90	+0.82
SWE-TRAIL-H5	14.33	14.67	14.33	14.44	+3.11	13.27	13.54	13.68	13.50	+2.42
SWE-TRAIL-H10	15.33	15.33	15.67	15.44	+4.11	13.82	13.95	13.82	13.86	+2.78

F COLLECTION MECHANISM IN DETAIL

F.1 ACTIONS, REPEATED READS, AND VALIDATION

We first examine how the collected trajectories differ in their repair process, not only in their final outcomes. To avoid attributing differences only to a change in the resolved/unresolved mixture, we compare resolved and unresolved trajectories separately. Figure 5 reports repeated-read and validation distributions relative to SeedBank.

Table 19 complements these outcome-stratified distributions with means over the full corpus. It also reports whether an executable check follows the final edit, rather than the number of validation actions. Mean actions and repeated reads decrease from SeedBank to H0 and again from H0 to H5, while post-final-edit validation increases. H10 partially reverses each of these changes.

Table 19: Process means over all 1,500 collected trajectories per condition. Post-edit validation is the share of edited trajectories whose final edit is followed by an executable check.

Condition	Mean actions	Mean repeated reads	Post-edit validation
SeedBank	43.37	8.78	86.62%
SWE-TRAIL-H0	40.89	7.59	88.91%
SWE-TRAIL-H5	39.51	7.01	89.53%
SWE-TRAIL-H10	40.72	7.46	88.65%

F.2 STUDENT BEHAVIOR ON HELD-OUT ISSUES

The preceding analysis describes the trajectories used for training. We next ask whether similar process differences appear when the trained students solve new issues. We compare evaluation

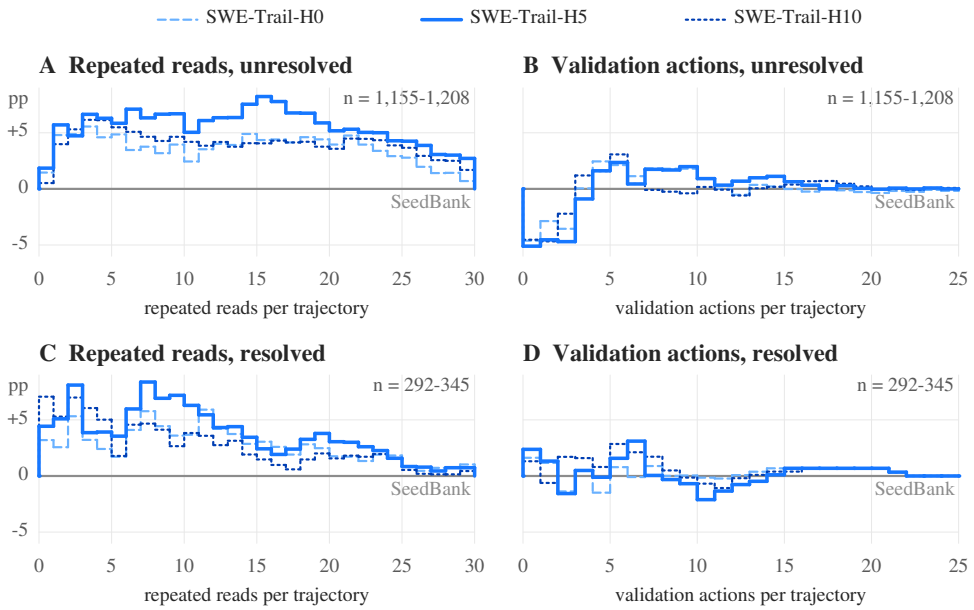


Figure 5: **H5 shows the largest shift toward fewer repeated reads, while validation counts remain close to SeedBank.** At each count, the curve gives the share of trajectories at or below that count minus the corresponding SeedBank share, in percentage points. SeedBank is the zero line. A positive value means that more trajectories fall at or below the displayed count. Panels A and C show repeated reads. H5 reaches 8.2 points above SeedBank for unresolved trajectories and 8.4 points for resolved trajectories, compared with 5.6 and 5.9 for H0. Panels B and D show validation actions, where all curves remain within about five points of SeedBank. The rightmost bin includes values at or above the displayed cap. Coding-action distributions are omitted for legibility.

trajectories from the 124B students trained on H0 and H5 on both Multilingual and SWE-Pro. All evaluation tasks are included, including failed and incomplete executions.

Coding actions are tool calls other than planning and finish calls. Repeated reads count file-read occurrences beyond the first read of each normalized target, including reads after an edit. Repeated-read share divides this count by the number of recognized file-read occurrences in the trajectory. Post-final-edit checking is measured only for trajectories that contain an edit and records whether a recognized executable check occurs after the final edit, regardless of whether that check passes.

Table 20: Descriptive evaluation behavior of the 124B students. Metrics are computed separately for each independent training run and then averaged across the three runs per condition. Δ is H5 minus H0. Differences in rates are in percentage points.

	Multilingual			SWE-Pro		
Metric	H0	H5	Δ	H0	H5	Δ
Actions / trajectory	84.55	78.30	−6.25	77.84	75.34	−2.50
Repeated reads / trajectory	16.41	14.89	−1.52	15.17	14.52	−0.65
Repeated-read share (%)	48.12	46.61	−1.51	44.77	44.17	−0.61
Post-final-edit check rate (%)	81.43	80.57	−0.86	86.39	87.43	+1.04

Table 20 shows that H5-trained students take fewer actions and repeat fewer file reads on both benchmarks. Repeated-read share is also lower. The decreases in actions and repeated reads follow the same direction as the training-corpus differences in Appendix F.1. Post-final-edit checking decreases on Multilingual but increases on SWE-Pro. These results describe differences in student behavior and do not identify the cause of the resolution gains.

F.3 Collection Yield by History Type

For collection yield, we compare H0 and H5 across the three SeedBank history types. Table 21 breaks down the 16 additional resolved trajectories and six additional solved issues by the three SeedBank history types.

Table 21: H0/H5 collection yield by SeedBank history type. An issue is counted as solved when at least one of its five newly collected trajectories resolves.

	Resolved trajectories			Solved issues		
History type	H0	H5	Δ	H0	H5	Δ
Stable-solved	176	176	0	37	37	0
Mixed	130	143	+13	40	45	+5
Never-solved	23	26	+3	15	16	+1
All	329	345	+16	92	98	+6

Mixed histories account for 13 of the 16 additional resolved trajectories and five of the six additional solved issues. H5 solves every mixed issue solved by H0 and five additional ones. Never-solved histories account for the remaining three additional resolved trajectories and one net additional solved issue. Within this group, H5 solves nine issues missed by H0, while H0 solves eight missed by H5. The gain on mixed histories therefore preserves H0's solved-issue coverage, whereas the smaller never-solved gain comes from different sets of solved issues.

With issues as clusters, the H5−H0 difference in resolved trajectories is $+1.07$ percentage points, with a 95% CI of $[-0.33, 2.47]$ and an exact sign-flip $p = 0.16$. The difference in solved issues is $+6$, or $+2.0$ percentage points, with a 95% CI of $[-1.0, 5.0]$ pp and $p = 0.29$. These collection-level comparisons are descriptive and do not identify which trajectories account for the downstream training difference in Section 4.2.

F.4 Runtime Comparison Stages

We next examine where the H0 and H5 collection processes differ. Table 22 expands Figure 3 by history type and gives the denominator for each stage.

Candidate sampling occurs in 664 H0 rollouts and 1,147 H5 rollouts. At least two semantically distinct actions remain in 598 and 1,030 of these cases, respectively. Both candidate suffixes are successfully executed in 560 H0 and 968 H5 rollouts. The non-default route is retained in 218 and 362 completed comparisons.

The largest difference occurs in how often rollouts reach candidate sampling. Once candidates are sampled, H0 and H5 have similar candidate diversity, execution rates, and alternative-selection rates. Because the two collectors sample candidates at different points in their rollouts, similar conditional rates do not imply that they generate the same candidate distributions.

Table 22: H0/H5 collection stages by SeedBank history type. H0 history labels are joined post hoc by issue ID and were never available during H0 collection. Candidate diversity is conditional on candidate sampling, executed comparison is conditional on candidate diversity, and alternative selection is conditional on a completed comparison. Candidate sampling, semantic progress, and resolution use all trajectories.

Condition	History type	Traj.	Candidate sampling	Distinct actions	Executed comparison	Alt. selected	Semantic progress	Resolved
H0	stable-solved	185	36.22%	91.04%	86.89%	30.19%	21.08%	95.14%
H5	stable-solved	185	53.51%	90.91%	92.22%	43.37%	35.14%	95.14%
H0	mixed	255	53.73%	93.43%	95.31%	43.44%	35.29%	50.98%
H5	mixed	255	83.53%	94.37%	94.53%	41.05%	56.47%	56.08%
H0	never-solved	1,060	43.40%	88.91%	94.13%	38.80%	28.49%	2.17%
H5	never-solved	1,060	78.77%	88.50%	94.05%	35.68%	37.92%	2.45%
H0	all	1,500	44.27%	90.06%	93.65%	39.00%	28.73%	21.93%
H5	all	1,500	76.47%	89.80%	93.98%	37.40%	40.73%	23.00%

F.5 Compiler Classes and Representative Records

The SeedBank history types above describe only the outcomes of the five prior attempts. The compiler separately interprets the evidence in each history and may abstain when that evidence does not support additional collection effort. Table 23 reports these compiler classes and the number of detector rules admitted for collection.

Among histories assigned to the stable-solved, mixed, and never-solved compiler classes, mixed and never-solved histories have more active rules per issue, at 1.61 and 1.47, than stable-solved histories at 0.62. The compiler also abstains more often on stable-solved histories. Twelve issues are assigned to the insufficient-evidence class, for which no rule is admitted.

Table 23: H5 acquisition-record statistics by compiler class. Per-class means are rounded to two decimals. Appendix B.4 gives the exact totals of 687 compiled and 397 active detector rules.

Compiler class	Issues	Mean rules	Active rules	Abstention
Stable-solved	37	2.38	0.62	37.84%
Mixed	41	2.22	1.61	7.32%
Never-solved	210	2.29	1.47	13.81%
Insufficient	12	2.33	0.00	100.00%

Representative compiler records. The following records illustrate three different compiler outcomes: retaining uncertainty around an otherwise successful history, allocating additional collection effort when competing repair choices remain, and abstaining when the available evidence is insufficient. The never-solved example in Appendix H.1 follows one admitted rule through online execution. Predicates and objectives below are reproduced verbatim. Omitted fields are marked with `...`. The status `active` means that the rule passed compiler admission. It does not indicate that the rule matched a new rollout. All objectives remain private to the collector.

Stable-solved | shepmaster__snafu-237

```
source outcomes: 5/5 resolved
open uncertainty u1: whether other files in the snafu-derive crate contain quote!
blocks with unhygienic std:: or core:: references, since the probes inspected only
src/lib.rs
frozen predicate: has_repository_evidence = true AND unique_target_count >= 4 AND
has_validation = false AND last_category = "search"
objective: Preserve both viable editing routes until commitment; avoid premature route
elimination
status: active (1 of 2 proposed states activated; the other stays dormant)
```

Mixed | cloudflare__cloudflare-go-1066

```
source outcomes: 4/5 resolved
competing hypotheses: h1 adopt the repository's existing custom UnmarshalJSON pattern;
h2 change PctComplete from int to string with no custom unmarshaling
frozen predicate: has_repository_evidence = true AND has_edit = false AND validation_count
= 0 AND search_count >= 3 AND semantic_phase = "hypothesis"
objective: Prefer proceeding to minimal type-change edit over continued search for
custom unmarshaling patterns
status: active (3 of 3 proposed states activated)
```

Insufficient | adobe__elixir-styler-203

```
source outcomes: 0/5 resolved
open uncertainties: u1 whether the Zipper module exposes parent-node access; u2 whether
any existing style module demonstrates parent-context detection; u3 the AST shape of
first..last//step ...
proposed predicate: edit_count = 0 AND recent10_read_search_count >= 4 AND action_count >=
6
objective: Prefer implementing the template-driven easy task (List.zip) over continued
exploration or entangled multi-task planning
status: dormant (0 of 2 proposed states activated; the compiler abstains)
```

G Component Controls, Cost, and Best-of-N

G.1 Randomized Controls and Selector Replay

The two randomized controls test where comparisons occur and which executed route continues. Both use H5's five-attempt history, sampling-temperature schedule, training protocol, and evaluation protocol. They also use the same per-trajectory collection budget and execution horizon as H5. This budget allows eight candidate proposals, one shortlist call, one selector call, and at most two two-observation suffix executions. Realized token counts may still differ across trajectories.

Random detection. Random detection replaces H5's history-guided comparison location with a random eligible state while leaving the remaining collection procedure unchanged. Eligible states are repository-observation boundaries at which the observation has completed, the workspace can be restored, no earlier comparison has been executed, and enough budget remains for both short suffixes. The state is sampled uniformly from these eligible locations without using history or a detector score. If a trajectory contains no eligible state, it completes as an ordinary rollout without a comparison.

Random retention. Random retention keeps H5's history-guided comparison location and executes the same candidate-generation, shortlisting, and suffix-comparison procedure. Once both suffixes execute successfully, a fair binary draw determines which workspace continues, regardless of the selector's judgment. If restoration or suffix execution fails, the default route continues.

Separate collection with a stricter selector. We also test whether requiring stronger execution evidence improves the training value of the collected trajectories. This condition runs a separate H5-style collection in which an alternative route is preferred only when its executed suffix provides stronger evidence about the root cause or working hypothesis. The collection resolves 343 of 1,500 trajectories across 97 issues, compared with 345 trajectories across 98 issues for H5. Students trained on these trajectories reach 63.89% on Multilingual and 41.91% on SWE-Pro, 1.78 and 0.91 points below H5. Thus, the stricter criterion leaves collection yield nearly unchanged but produces lower student means on both benchmarks.

Fixed-comparison diagnostic replay. To examine the selector without changing candidate generation or execution, we replay 18 H5 comparisons under the stricter criterion. The visible prefix, workspace, candidates, and both executed suffixes remain fixed, so only the selection judgment changes. These comparisons come from seven issues that the compiler classifies as mixed.

The stricter criterion changes eight decision labels and six retained routes. Some changed decisions become ties and therefore retain the default route, which explains the difference between these counts. Ties increase from 5 of 18 comparisons to 12 of 18. The stricter criterion mainly converts uncertain preferences into abstentions.

Candidate coverage is the most frequent non-exclusive limitation in this sample, appearing in 13 of 18 comparisons. Seven have partial coverage and six have poor coverage. Selection is identified as a limitation in 4 of 18 comparisons, followed by suffix horizon and post-selection drift at 3 of 18 each. A selector cannot recover a useful action that was not proposed, which motivates testing more diverse candidate proposals. These counts describe the 18 replayed comparisons rather than all 968 executed comparisons.

G.2 History Scale and Collection Cost

We compare H0, H5, and H10 to measure how the benefit and cost change as more history is provided. Costs cover new collection given existing prior attempts, under the accounting boundary in Appendix B.2. Figure 6 combines the recorded collection cost with downstream results for the 124B student.

H5 records 1.914B total collection tokens, including 307.3M acquisition-only tokens. H0 records at least 1.845B total tokens, including 164.3M acquisition-only runtime tokens. Its offline compiler tokens were not recorded. Relative to this lower bound, H5 uses at most 3.7% more total tokens than H0. Removing acquisition-only usage leaves 1.607B ordinary rollout tokens for H5 and 1.681B for H0, a 4.4% reduction.

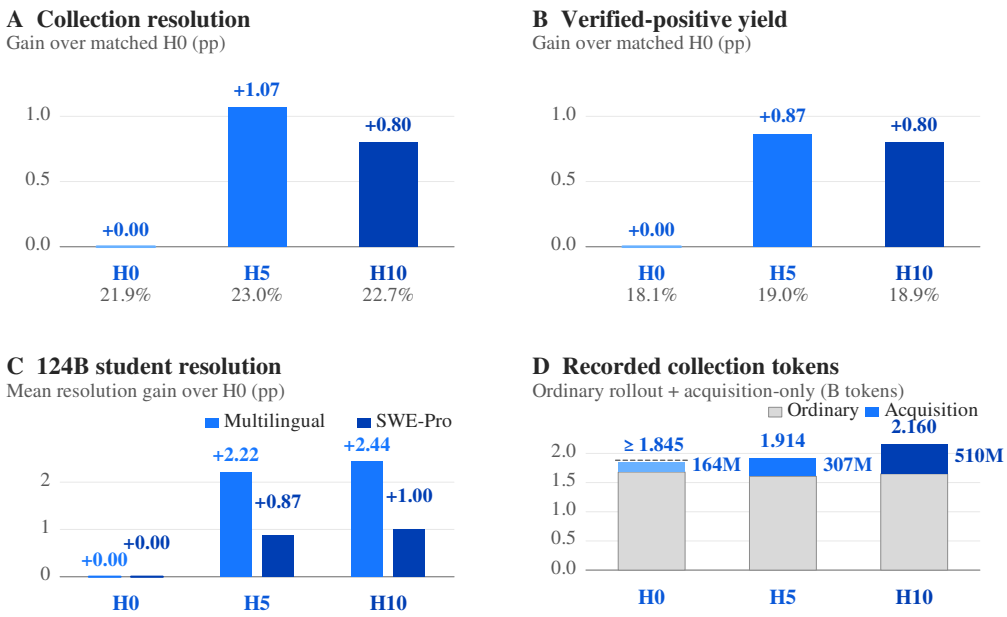


Figure 6: **H10 adds little mean improvement over H5 while collection cost continues to increase.** Collection outcomes and 124B-student gains are shown relative to H0. Recorded tokens are divided into ordinary rollout and acquisition-only usage. H0 is a lower bound because its offline compiler calls were not recorded.

H10 uses 2.160B total tokens, including 510.0M acquisition-only tokens. These are 12.9% and 66.0% above H5, respectively. Ordinary rollouts use 1.650B tokens.

To show where H5's 307.3M acquisition-only tokens are spent, we break them down by component and history type. Candidate proposals use 238.4M tokens, executed suffixes 43.1M, selection 16.8M, and offline compilation 9.1M. Across history types, stable-solved histories use 22.32M tokens, or 7.3%. Mixed histories use 51.97M, or 16.9%, and never-solved histories use 232.85M, or 75.8%.

Table 24 gives the corresponding environment-side costs. Trajectory-hours sum elapsed task time and do not represent concurrent wall-clock time.

Table 24: Environment-side totals over the 1,500 collected trajectories in each condition.

Condition	Coding actions	Validations	Trajectory-hours
SeedBank	65,056	6,814	355.2
SWE-TRAIL-H0	61,329	6,760	196.2
SWE-TRAIL-H5	59,260	6,700	239.4
SWE-TRAIL-H10	61,085	6,721	208.1

G.3 VERIFIER-BASED TRAINING CORPUS SELECTION

We next test whether post-hoc selection can replace history-guided collection or complement it. The first comparison selects from ten independently sampled SeedBank trajectories per issue. The second applies the same selection rule to five SeedBank and five H5 trajectories. Both retain five trajectories per issue for student training.

Ranking rule. For each issue, the verifier ranks ten candidates and retains five. Format-admissible candidates rank ahead of inadmissible candidates, followed by the binary SWE-rebench resolved label. Remaining ties use the same deterministic SHA256-based rule in both comparisons. Reference patches, trajectory length, and process metrics are not used for ranking.

Budget and selected corpus. Top-5-of-10 uses twice as many independent rollouts as the five-rollout SeedBank baseline. Its collection cost is not matched exactly to H5. Both selection conditions rank 3,000 candidate trajectories and retain 1,500 for training.

Table 25: Verifier-based Top-5 selection. Both candidate sets contain 3,000 trajectories and retain 1,500 trajectories for training.

Candidate set	Res.	Solved issues	Strict +	Never traj.	Never issues
10 SeedBank	394	99	313	15	11
5 SeedBank + 5 H5	413	104	321	26	16

Table 25 compares the two selected corpora. The hybrid set has a 1.27 percentage-point higher trajectory-resolution rate under issue-cluster pairing, with a 95% CI of $[-0.07, 2.67]$ and a 29/17/254 win/loss/tie record. All disagreements in solved-issue coverage occur on never-solved histories, with ten issues solved only by the hybrid set and five only by the independent set. Both selected sets preserve coverage on stable-solved and mixed histories. The selected banks show complementary issue coverage under the same ranking rule.

Student results. Table 26 reports students trained on the selected trajectories under the protocol in Appendix D.

Table 26: Student resolution after verifier-based selection. Each corpus has three independent training runs. Ranges give the lowest and highest run. SeedBank and H5 repeat the results in Table 1.

	Multilingual		SWE-Pro	
Trajectory corpus	Mean	Range	Mean	Range
SeedBank	63.33	[63.00, 63.67]	41.22	[41.18, 41.31]
Top-5-of-10	64.11	[63.33, 65.00]	41.86	[41.45, 42.13]
SWE-TRAIL-H5	65.67	[64.00, 67.67]	42.82	[41.72, 43.37]
Top-5-hybrid	66.22	[65.33, 66.67]	43.14	[42.82, 43.37]

Top-5-of-10 improves over SeedBank but remains below H5 by 1.56 points on Multilingual and 0.96 points on SWE-Pro. Thus, selecting from a larger pool of independent trajectories does not recover the gains from SWE-Trail in this comparison.

Top-5-hybrid reaches 66.22 on Multilingual and 43.14 on SWE-Pro, exceeding H5 by 0.56 and 0.32 points, respectively. These results show that post-hoc selection can complement SWE-Trail when applied to a candidate set that includes its trajectories.

H CASE STUDY AND ADVICE DIAGNOSTICS

H.1 A REPAIR CASE AFTER FIVE FAILED ATTEMPTS

This case shows how information from earlier failed attempts guides collection without entering the coding agent's context. For `gresau__schemars-304`, all five SeedBank attempts fail after making a one-line `SERDE_KEYWORDS` edit without validating the change. Under H5, four of five newly collected trajectories resolve the issue. We follow one H5 trajectory from the compiled record through candidate execution and route selection. Figure 7 shows the two executed suffixes.

`schemars-304` | never-solved | H5 attempt 4

1. Earlier attempts identify where more collection effort may be useful. All five prior attempts are unresolved. They repeatedly identify or edit `rename_all_fields`, then return to repository reading without validating the edit. A read-only repository check locates the `SERDE_KEYWORDS` list that controls which serde attributes are forwarded, together with its definition and two uses. The compiled record keeps the sufficiency of the one-line edit as an open question.

Acquisition record

```
source outcome:  0/5 resolved
uncertainty:  is the edited keyword sufficient?
probe:  locate the keyword list that gates attribute forwarding
observed:  SERDE_KEYWORDS definition and two uses, contents unobserved
failure boundary:  post-edit reading without validation
```

2. The compiler turns this pattern into an observable rule. The rule `post_edit_read_loop_without_validation` matches after an edit when no subsequent validation has occurred and at least three of the ten most recent actions are reads or searches. When this pattern appears in the new rollout, the coding agent proposes eight next actions from its ordinary context. These proposals form three semantic groups.

3. Short execution provides the evidence for route selection. Candidate 0 continues reading the tests directory and `enum.rs`. Candidate 6 instead runs the two relevant Rust test suites from the same checkpoint.

Suffix evidence

```
A / candidate 0:  no validation signal
B / candidate 6:  schemars_derive -- 1 test passed
B / candidate 6:  schemars -- all tests and doc-tests passed
blind selector:  retain B
unblinded mapping:  B = candidate 6
```

The collector continues only from candidate 6's workspace. The exported record remains one continuous trajectory generated by the coding agent, and its one-file patch resolves under the SWE-rebench harness. History determines where additional collection effort is spent, while the coding agent itself proposes the test command that provides the deciding evidence.

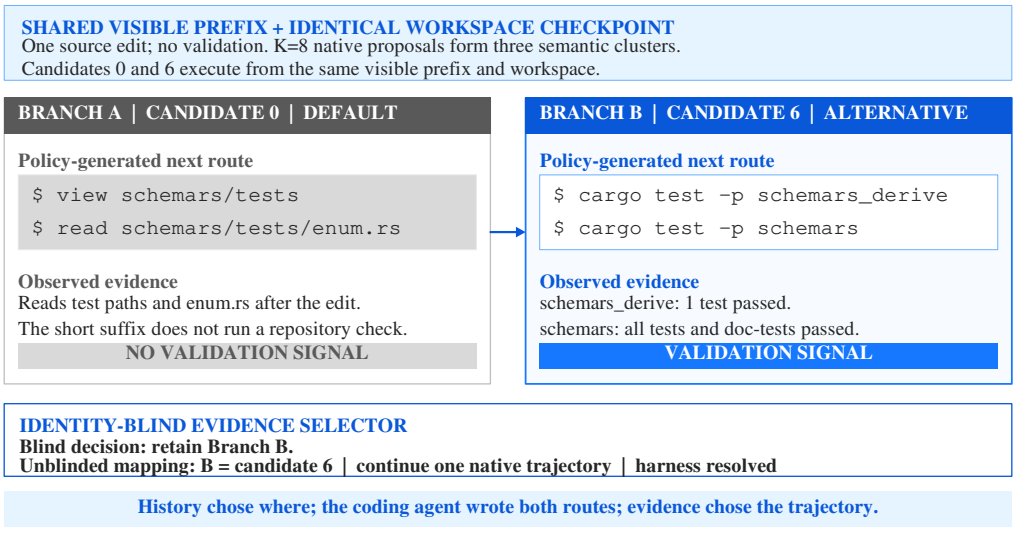


Figure 7: **History guides where additional collection effort is spent.** In the `schemars-304` example, both candidate suffixes are generated by the same coding agent from the same visible checkpoint. Candidate 0 continues reading, while candidate 6 runs the relevant Rust tests. The selector uses the resulting repository evidence to choose which route continues.

H.2 Retaining and Removing Advice in Training Trajectories

We separately examine whether removing advice from an advice-conditioned trajectory changes its value for training. This experiment differs from the SWE-Exp adaptation in Section 4.2. Here, reviewer-generated advice is inserted into the coding agent's conversation during the rollout.

Two exports of the same trajectories. We create two training corpora from the same 1,500 advice-conditioned trajectories and train students under the protocol in Appendix D. The two exports preserve trajectory membership and order, coding actions, tool arguments, and tool results.

The advice-retained export keeps each aligned advice message and places it before the agent-generated reasoning in the corresponding assistant tool-call turn. The advice-removed export removes the advice messages and only the reasoning fragments that explicitly attribute an action to that advice. All remaining agent reasoning and actions are unchanged. Both exports retain actions generated while advice was visible to the agent. Removing advice changes the training record, not the context in which those actions were generated.

Table 27: Three-run SFT means for two exports of the same advice-conditioned trajectories.

Training corpus	Multilingual	SWE-Pro
Independent SeedBank	63.33	41.22
Advice removed	63.00	42.23
Advice retained	64.56	41.63
SWE-Trail-H5	65.67	42.82

Table 27 reports the resulting student means. Retaining advice improves over SeedBank on both benchmarks. Removing it lowers Multilingual resolution but raises SWE-Pro resolution relative to retaining it. Advice removal does not consistently improve downstream performance in this comparison.

I Detailed Related Work

Section 5 groups the closest work into search and selection, trajectory construction and curation, and experience-guided agents. Here we give a broader comparison centered on two questions: who reads prior attempts, and how those attempts affect later computation. Table 28 summarizes the main interfaces.

Table 28: How prior attempts are reused. The main distinction is whether history enters the coding agent's context or remains private to an external collection procedure.

Use of prior attempts	Consumer	Effect on later computation	Primary goal
Retrieved reflection or memory	Coding agent	Adds previous feedback or experience to the visible context	Improve the current solve
Distilled skill or workflow	Coding agent or workflow	Supplies reusable procedural guidance	Improve the current solve
Inference-time replay or refinement	Search harness and coding agent	Reuses prefixes or conditions generation on prior summaries	Improve or select a current solution
History-guided collection (ours)	Collector	Guides where additional collection effort is spent while history remains outside the agent context	Collect training trajectories

I.1 Executable Coding Agents and Trajectory Corpora

SWE-bench established executable repository-level repair, while SWE-agent and OpenHands provided widely used coding-agent scaffolds (Jimenez et al., 2024; Yang et al., 2024; Wang et al., 2025a). SWE-Gym, R2E-Gym, SWE-smith, and Open-SWE-Traces expand the available tasks, environments, rollouts, and languages for training (Pan et al., 2025; Jain et al., 2025; Yang et al., 2025; Ahmad et al., 2026). These efforts increase the supply of coding trajectories. Our focus is different: given several prior attempts on the same issue, we ask how that history can guide the collection of additional trajectories.

I.2 Inference-Time Search and Selection

A common inference-time strategy samples several complete rollouts and selects or combines them using tests, execution outcomes, or model judgments (Ehrlich et al., 2025). Other methods reuse earlier computation more directly. They may restore a previous trajectory prefix and continue from an earlier point, or summarize prior attempts and use those summaries to condition later rollouts (Ding & Zhang, 2026; Kim et al., 2026). These methods use additional computation to improve the solution returned for the current issue.

Tree-search methods provide another form of adaptive inference. MCTS and related approaches compare alternatives within a growing online search tree and allocate computation using value estimates, process signals, or terminal outcomes (Kocsis & Szepesvári, 2006; Zhou et al., 2024; Antoniades et al., 2025). They can revisit a shared prefix and explore alternatives that independent full rollouts may miss.

SWE-Trail differs in both source and objective. Its guidance comes from independent attempts collected before the new rollout, and the goal is to collect training trajectories rather than to return the best solution from a search tree. It compiles fixed rules before collection and executes at most one comparison during a rollout, without recursive tree expansion or value backup. The H0 control in Section 4.1 evaluates this bounded collection procedure without history, rather than a full tree-search policy.

I.3 Trajectory Construction and Post-Hoc Curation

Several methods improve coding-agent training data after trajectories have already been generated. SWE-Gym and related pipelines retain trajectories based on executable outcomes, while other methods mask flawed tool calls, remove redundant interactions, reconstruct better paths, or use process-level signals to identify more useful supervision (Pan et al., 2025; Tao et al., 2026; Ma et al., 2026; Han et al., 2026; Xiao et al., 2026b). Data valuation and training-dynamics methods similarly estimate the usefulness of examples that already exist (Koh & Liang, 2017; Ghorbani & Zou, 2019; Paul et al., 2021; Xie et al., 2023).

These methods are complementary to SWE-Trail. Post-hoc curation decides which parts of completed data should be retained or emphasized. SWE-Trail instead uses prior attempts while a new trajectory is being collected. The comparison can produce new repository evidence that would not exist if the alternative action were never executed. Curation can still be applied afterward. Appendix G.3 evaluates one such combination using verifier-based selection over SeedBank and H5 trajectories.

I.4 Training-Time Diversity and Adaptive Curation

Recent work treats the training corpus itself as an object to optimize. TDScaling increases trajectory diversity through blueprint-driven synthesis and adaptive task evolution rather than only sampling more trajectories (Chen et al., 2026). Han studies filtering of an existing SWE trajectory pool and reports an interaction between data quality, quantity, and model scale (Han, 2026). CurateEvo represents curation as executable code and updates the curation procedure from downstream failures to construct SFT, RL, and memory data from a fixed raw corpus (Wang et al., 2026a).

SWE-Trail shares the goal of improving the training distribution but acts at a different stage. It changes how new trajectories are collected. Earlier attempts on the same issue guide where additional collection effort is spent, while the coding agent generates the candidate actions without receiving that history.

I.5 Experience-Guided Agents

Another line of work converts previous interaction into information that helps an agent act later. Reflexion and ExpeL store feedback or reusable experience across attempts (Shinn et al., 2023; Zhao et al., 2024). Agent Workflow Memory, CODESKILL, and Socratic-SWE derive workflows, skills, or procedural knowledge from previous executions (Wang et al., 2025b; Li et al., 2026; Xiao et al., 2026a). For software repositories, experience may come from earlier repair trajectories, human debugging records, or trajectories produced during the current solve (Chen et al., 2025; Lin et al., 2024; Mu et al., 2026; Wang et al., 2026b; Hayashi et al., 2025; Xia et al., 2025). Other systems recombine agents or revise failed paths during inference (Ehrlich et al., 2025; Guo et al., 2025).

The common pattern is that distilled experience changes the coding agent's visible context, workflow, or behavior. SWE-Trail keeps that information outside the coding agent. History guides the collector in deciding where additional collection effort may be useful, while the coding agent proposes its next actions from the ordinary context. This separation is central to our training-data setting because the exported trajectory does not depend on history being available to the student at test time.

I.6 Adaptive Acquisition and Harness Optimization

The broader idea of allocating computation using previous observations appears in active learning and experimental design, which choose informative examples or experiments (Sener & Savarese, 2018; Kirsch et al., 2019; Foster et al., 2021). Intrinsic-motivation methods similarly direct interaction toward novel or prediction-reducing states (Bellemare et al., 2016; Pathak et al., 2017). Agent self-improvement and harness-optimization methods adapt prompts, workflows, tools, or control procedures from execution feedback (Madaan et al., 2023; Xi et al., 2024; Robeyns et al., 2025; Pan et al., 2026; Lin et al., 2026).

SWE-Trail applies a narrower form of adaptive allocation within one issue. The history of that issue is compiled before the new collection round, and the resulting rules remain fixed during the rollout. They guide where additional collection effort is spent and how executed alternatives are compared, without changing the coding-agent prompt, tools, or model weights.

Recent results show that simple search can remain a strong baseline (Gideoni et al., 2026). We compare against independent sampling, a history-free version of the same collection pipeline, randomized controls, and verifier-based selection in Section 4.1.