SWE-LOOP: TRACE-GUIDED SELF-IMPROVEMENT FOR CODING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents are commonly improved by collecting additional trajectories or tuning a fixed task distribution, leaving open whether their own execution evidence can dynamically steer what they learn next. We present **SWE-Loop**, a trace-guided closed-loop framework for self-improving coding agents. At each iteration, SWE-Loop generates repository-level software-engineering tasks, obtains rollouts from a teacher model and the current student model, diagnoses semantically attributable failures, and updates the next task distribution to target recurring capability deficits. Supported by a multi-agent research team that validates tasks, curates trajectories, and controls the experimental process, SWE-Loop curates execution-verified trajectories as training data while analyzing teacher and student traces to identify shared blind spots and student-specific regressions. The framework disentangles semantic/model failures from system-level failures, including setup, provider, timeout, verifier, and infrastructure errors, and evaluates each iteration on a mainstream software-engineering benchmark and group-disjoint evaluation task families using resolution rate, patch coverage, regression rate, and compute cost. Across a three-iteration evolution loop, we test whether trace-guided task synthesis yields generalizable improvements beyond non-adaptive baselines that train on self-generated tasks.

Keywords: coding agents; trace-guided learning; self-improvement; task synthesis.

1 INTRODUCTION

Coding agents are increasingly evaluated on software-engineering tasks that require more than producing a locally plausible code snippet. Given a natural- language issue and a repository, an agent must locate the relevant behavior, modify one or more files, run commands and tests, interpret execution feedback, and preserve the behavior that was already correct. SWE-bench made this repository-level setting a standard testbed by pairing real GitHub issues with executable codebases and tests (Jimenez et al., 2024). Agent systems such as SWE-agent further showed that the interface through which a model searches, edits, and executes code is itself a consequential part of the problem (Yang et al., 2024). These results motivate a broader question: how should coding agents improve once their current failures, partial repairs, and recovery attempts are observable as execution traces?

The data-generation bottleneck. Post-training and reinforcement-learning approaches have made executable software-engineering environments increasingly useful for improving agents. SWE-Gym, for example, organizes real-world tasks together with codebases, runtime environments, and tests so that both agents and verifiers can be trained from execution (Pan et al., 2024). SWE-RL instead uses open software evolution data to train reasoning models with executable software signals (Wei et al., 2025a). Data-generation pipelines such as SWE-smith scale synthetic repair tasks across many repositories (Yang et al., 2025). Together, these efforts show that executable software evolution can provide substantial supervision. They also expose a remaining limitation: when the task distribution is fixed or generated independently of the current model, additional rollouts do not by themselves specify which capability should be trained next.

The limitation is not simply a shortage of trajectories. A long-horizon SWE trace records where an agent searched, which repository evidence it used, how it reacted to tests, and whether its patch

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.

introduced a regression. But the final pass/fail bit is often an insufficient diagnosis. Two agents may both fail a task for different reasons, or a student may fail a task that its teacher can solve. A semantic mistake, an unavailable provider, a broken setup, a timeout, and a verifier failure can all look like an unsuccessful rollout while calling for different interventions. The research problem is therefore one of *attribution*: when do paired traces justify a capability-gap label, and when should the evidence be withheld?

From self-modifying agents to trace-guided curricula. Prior work has explored self-improvement by allowing an agent to modify its own implementation. A Self-Improving Coding Agent maintains an archive of agent versions and uses a coding agent to propose and evaluate changes to the agent system itself (Robeyns et al., 2025). This demonstrates that benchmark feedback can drive open-ended changes to an agent scaffold, but it does not make the next software-engineering training distribution the primary object of adaptation. More recently, Socratic-SWE converts historical solving traces into structured skills, uses those skills to construct targeted repository tasks, validates the candidates through execution, and repeats the process over multiple iterations (Xiao et al., 2026). This is the closest prior formulation to the loop studied here.

SWE-Loop. We present **SWE-Loop**, a trace-guided closed-loop framework for self-improving coding agents. Each iteration follows *task construction* $\rightarrow$ *paired teacher/student rollout* $\rightarrow$ *evidence attribution* $\rightarrow$ *verified recovery* $\rightarrow$ *SFT* $\rightarrow$ *new task construction*. The paired trace is the unit of diagnosis: a teacher success and student failure suggests a learnable deficit; shared failure suggests a common frontier; a student success followed by failure at the next checkpoint signals regression. None is accepted from the outcome bit alone. A multi-agent researcher team assigns task building, independent review, trace analysis, trajectory curation, and experiment-control roles. Execution checks establish the task contract before the trace analyst attributes model behavior. System exceptions are quarantined.

The central design choice is to gate curriculum updates on the *strength of evidence for a failure cause*, not just its frequency. SWE-Loop is SFT-first: accepted recovery trajectories become training data, while attributable gaps determine which tasks should be constructed next. Uncertain or infrastructure-induced failures remain visible in the denominator but cannot masquerade as model weaknesses. The protocol evaluates each checkpoint on external benchmarks and group-disjoint acceptance families, separating improved behavior on generated tasks from the stronger claim of generalization.

Our contributions are:

- An evidence-attribution protocol for paired teacher and student trajectories, with explicit invalid and uncertain states.
- A pillar-aware capability-gap registry that distinguishes shared blind spots from student-specific failures and regressions.
- A verified-recovery SFT loop whose next task distribution is controlled by reviewed, attributable trace evidence.
- A matched-budget evaluation design that tests three iterations with external resolution, patch coverage, paired regression, and cost measures.

2 Related Work

2.1 Repository-Level Evaluation and Agent Scaffolds

SWE-bench established repository-level issue resolution as a demanding evaluation setting: the agent receives an issue and a codebase, modifies the repository, and is evaluated with tests and patch-level evidence (Jimenez et al., 2024). This formulation captures the need for code localization, multi-file reasoning, tool use, and regression avoidance that is not represented by short-form code-generation benchmarks. SWE-agent studied the agent-computer interface explicitly, showing that repository navigation, editing primitives, and execution tools materially affect performance (Yang et al., 2024). Agentless provides a complementary result: a simple localization–repair–validation pipeline can be competitive with more elaborate autonomous scaffolds on SWE-bench Lite (Xia et al., 2024).

These works motivate treating the execution harness and task contract as part of the scientific object. SWE-Loop inherits the repository-level setting but asks which observed failures actually justify a curriculum decision. SWE-Bench Pro Verified documents how leakage and task-quality defects can inflate apparent software-engineering performance (Zheng et al., 2026); this reinforces the need to validate task contracts before attributing failures to a model. Benchmarks remain independent evaluation gates rather than unconstrained task templates.

2.2 Executable Environments for Post-Training

Executable environments make it possible to train agents and verifiers from observable software outcomes. SWE-Gym packages real-world Python tasks with repositories, runtime environments, and tests, and demonstrates that training on such environments can improve performance on SWE-bench (Pan et al., 2024). SWE-RL uses software evolution records—including code snapshots and changes—as a large-scale source for reinforcement learning and reports transfer beyond the software-engineering training data (Wei et al., 2025a). These approaches establish the value of executable feedback and software evolution as post-training substrates.

SWE-Loop is complementary but changes the adaptation boundary. Rather than assuming that a fixed executable task pool is sufficient, it treats the teacher's and student's traces as evidence for constructing the next pool. It also retains the provenance of each outcome so that a failed provider or verifier does not become a semantic negative example or a capability label.

2.3 Synthetic Bugs, Generated Tasks, and Self-Play

Scaling task supply is another major direction. SWE-smith automatically constructs executable repair environments and synthesizes large numbers of tasks from existing repositories, substantially increasing the scale of SWE training data (Yang et al., 2025). Self-play SWE-RL (SSR) further studies bug injection and repair in a self-play setting, using test patches to define synthetic tasks without requiring human-written issue descriptions (Wei et al., 2025b). These methods reduce dependence on manually curated issue–patch pairs and show that repository-grounded task generation can support iterative learning.

The distinction from SWE-Loop is not simply adaptive generation. Here the generator receives only *adjudicated* gaps grounded in paired teacher/student traces and valid task contracts. Raw failed rollouts and infrastructure exceptions cannot directly increase a task family's sampling weight. Thus the task builder is a downstream consumer of diagnostic evidence, not merely a scale-up mechanism.

2.4 Trace-Driven Self-Improvement

A Self-Improving Coding Agent demonstrates a different form of adaptation: an agent uses benchmark feedback and an archive of prior versions to edit its own codebase, thereby improving the scaffold that performs future tasks (Robeyns et al., 2025). SWE-Loop shares the idea that execution evidence can guide future improvement, but focuses on the evolution of the training task distribution and SFT data rather than unrestricted self-editing of the agent implementation.

The closest related work is Socratic-SWE, which distills solving traces into skills, conditions a generator on those skills, validates repository-grounded tasks, and iterates the Solver–Generator loop (Xiao et al., 2026). SWE-Loop differs in four deliberate respects. First, it makes *paired attribution* rather than distilled single-agent skills the control signal. Second, it separates task validity, model failure, and system errors before registering a gap. Third, its weight update is SFT on verified recoveries, not a solver RL objective. Fourth, it requires paired regression and common-support analysis rather than only an aggregate pass rate. SWE-Bench-CL likewise highlights forgetting and transfer in evolving issue sequences (Joshi et al., 2025); SWE-Loop uses those concerns to set a retention gate, without claiming to solve continual learning generally.

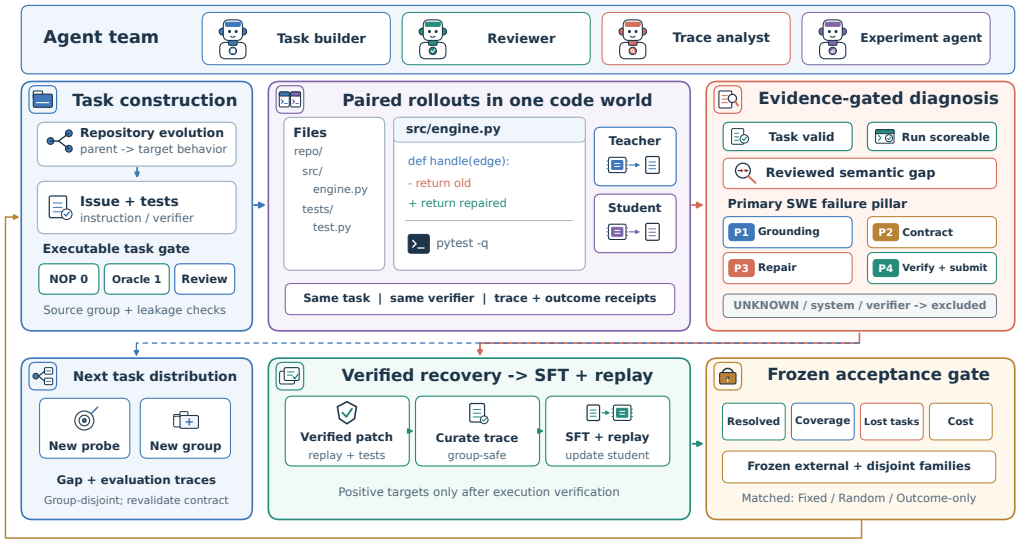


Figure 1: Evidence-gated SWE-Loop. Task and run gates precede paired-trace attribution; reviewed P1–P4 gaps steer new tasks, and verified recoveries supervise SFT. Frozen evaluation closes the loop.

3 METHODOLOGY

3.1 OVERVIEW

Let M_0 be the base student and M_i the checkpoint after iteration i. In iteration i, a builder and reviewer validate a task contract before teacher M_T and parent student M_{i-1} attempt it (Figure 1). Only reviewed semantic gaps in valid, scoreable paired traces steer the next task distribution; verified recoveries supervise SFT. Frozen evaluation supplies the traces for iteration $i+1$. Neither a failed outcome nor a successful trajectory alone identifies the student's missing capability.

3.2 PROBLEM FORMULATION AND EXECUTABLE EVIDENCE

A task $t = (r, c_0, u, v, m)$ contains repository r, pre-change commit c_0, issue instruction u, executable verifier v, and immutable metadata m. The agent interacts with a sandbox and returns a patch p and trajectory $\tau = (o_0, a_1, o_1, \ldots, a_K, o_K, p)$. We retain commands, exit status, selected outputs, diffs, and tool-call provenance, not just the final patch. The observed verifier outcome $r(t, \tau) \in \{1, 0, \perp\}$ records acceptance, completed rejection, or an indeterminate run. A verifier zero is an *outcome*, not a semantic diagnosis. A separate reviewer decision $a(t, \tau)$ records an attributable model failure, an uncertain cause, a task/verifier-contract issue, or a system failure. The paired observation is $z_i(t) = (r_T, r_{i-1}, \tau_T, \tau_{i-1}, e, a)$, where e contains task-validity and environment-parity receipts. An absent reward, even if a transcript looks promising, is not a negative label.

For each candidate task, the unmodified repository must fail the target behavior and a known repair must pass it. Additional checks reject leaked solution text, vacuous tests, invalid environment builds, and tasks that reward unrelated modifications. Oracle/NOP, patch replay, and instruction–test agreement are recorded separately so a reviewer can reject a task without changing its observed solver outcome. A pair is diagnosable only if both attempts share the same instruction and verifier revision or a documented equivalence mapping. Otherwise any apparent teacher/student disagreement may be an artifact of the environment.

3.3 METRICS AND IMPROVEMENT CRITERION

Let E be a frozen external set of N eligible instances. Full resolution is $R(M; E) = N^{-1} \sum_{t \in E} \mathbb{1}[b_E(t, M) = 1]$, where b_E is the benchmark's official per-instance verdict. Offi-

cial errors are reported separately and cannot silently become model failures. Patch coverage is $C = N^{-1} \sum_t \mathbb{1}[p_t \neq \varnothing]$ and conditional accuracy is $A = \sum_t \mathbb{1}[b_E(t, M) = 1] / \sum_t \mathbb{1}[p_t \neq \varnothing]$, with the denominator reported. Coverage and A are inseparable: a system can appear more accurate by submitting fewer patches. We also measure accuracy on the *common support* of tasks where both policies submit a patch.

For paired policies M_0 and M_i, acquired tasks are $G_i = \{t : b_E(t, M_i) = 1, b_E(t, M_0) = 0\}$ and lost tasks are $L_i = \{t : b_E(t, M_i) = 0, b_E(t, M_0) = 1\}$. We report $|G_i|$, $|L_i|$, retained successes, and per-task transitions rather than only $R(M_i) - R(M_0)$. The self-improvement criterion additionally requires a matched-budget fixed-data and random-task control, a group-disjoint acceptance set, retention analysis, and external evaluation after curriculum selection. For the attribution mechanism itself, we report the fraction of attempted pairs with valid evidence, reviewer agreement on primary pillar labels, the share left uncertain, and whether verified recoveries come from shared or student-specific gaps. These denominators prevent selective curation of only easy-to-explain failures.

3.4 Evolution-Native Task Substrate

Candidate tasks begin with real repository evolution: a parent revision, a later target behavior, an issue-like instruction, and tests that distinguish the two. The task builder may strengthen an existing contract or propose a new task when the original change does not exercise the desired capability. We keep a stable source group containing repository family, revision interval, and semantic-near-duplicate key. This group, rather than a trajectory row, is the unit of train/evaluation separation: one issue can produce several rollouts or instruction fixes without becoming several independent tasks.

Each accepted task has a reviewer receipt with source hash, environment image, verifier revision, NOP/Oracle outcomes, and leakage decision. Publication benchmarks are never used as task templates or training labels. A gap-matched task must also specify which behavior would distinguish a repository-grounding error from a contract error, a repair error, or a verification-and-delivery error. This makes task construction a falsifiable probe of a diagnosis, rather than a prose paraphrase of an analyst's label. The same eligible substrate feeds guided and random controls, with random selection stratified by repository, language, length, and approximate difficulty.

3.5 Trace-Derived Capability-Gap Registry

Only a completed run on a valid task enters semantic attribution. We use four operational SWE failure pillars: ***P1 Repository grounding*** (missing relevant code or evidence), ***P2 Behavioral contract*** (misreading required behavior or constraints), ***P3 Repair construction*** (an incorrect or incomplete edit), and ***P4 Verification and delivery*** (not acting on decisive feedback or failing to submit a viable patch under an explicit contract). Each primary label requires a trace span and independent repository or test evidence for the earliest causally relevant error; secondary tags are allowed. Ambiguous P2/P3 cases remain UNCERTAIN. A task with unclear artifact rules or a suspect verifier is routed to task review, not automatically labeled P4.

After validity screening, the two outcomes define four states. $T+/S+$ is solved by both; $T+/S-$ is a potential student-specific gap; $T-/S+$ is teacher-specific and is not counted as a student deficit; $T-/S-$ is a possible shared blind spot. An actual *regression* requires a second, same-task comparison to the previous student checkpoint: success at $i-1$ and failure at i. Outcome cells alone are only candidates. A reviewer must identify the pillar at which evidence diverged and decide whether the task itself is sufficiently specified. Setup, provider/authentication, timeout, verifier, and infrastructure cases are outside the four semantic cells and do not add mass to a gap.

3.6 Gap-Guided Evolution Curriculum

The registry stores each gap's source groups, paired outcome cell, pillar, supporting spans, reviewer decision, recurrence, previous-checkpoint transition, recovery availability, and estimated task cost. Student-specific $T+/S-$ cases are candidates for teacher-to-student recovery data; shared $T-/S-$ cases require an independently verified solution before they can supervise SFT. $T-/S+$ cases can contribute retention examples but should not be promoted as student weaknesses.

Selection weights are defined only over reviewed semantic cases. One predeclared policy can prioritize repeated $T+/S-$ P1 or P4 failures, while reserving a fixed quota for shared frontier tasks with independent recoveries. A candidate task must vary repository and surface form while retaining the pillar-relevant behavior. An outcome-only control may use verifier pass/fail but not paired traces or pillar evidence; fixed and stratified-random controls use neither. All draw the same number of task groups and training tokens. Thus the ablation isolates the *diagnostic decision*, rather than the mere presence of additional teacher examples. Thresholds, quotas, and adjudication rules must be frozen before the next acceptance evaluation.

3.7 Verified Trajectory Construction

The teacher and student roll out under the same task contract. A training example is admitted only if its patch passes the authorized verifier, its trace retains tool-call/response pairing, and its source group does not overlap evaluation. In $T+/S-$ cases, the teacher's verified trajectory may provide a recovery target; in shared failures, a separately verified repair is required. The student failure prefix is linked for attribution but is not copied into the positive SFT target by default. Unverified traces and all system-error cases remain in the analysis ledger.

3.8 Retention-Aware SFT Update

In iteration i, starting from frozen M_{i-1}, we train M_i on verified trajectories plus a capped replay of prior successful tasks. Context length, optimizer, training tokens, update steps, prompt, decoding, and checkpoint selection are held fixed across guided and control branches. Replay explicitly includes tasks where M_{i-1} succeeded but an earlier model failed, so that new gap coverage need not erase acquired capabilities. Group-balanced sampling prevents an issue with many rollouts from dominating retention. Canonical source trajectories and rendered windows have separate manifests.

3.9 Acceptance and Stopping Rules

At each iteration, an evaluator freezes parent checkpoint, task groups, scorer revision, branch budgets, and a blinded sample for diagnosis audit before guided selection. A guided student is accepted only if full resolution improves over parent, fixed, random, and outcome-only controls, the paired transition matrix supports the gain, and coverage/retention do not collapse. We also inspect whether diagnosis precision and uncertain-case rates changed across iterations; lower attribution quality is not a successful loop merely because fewer failures were labeled. Official external evaluation is a separate gate. A failed gate is a diagnostic iteration, not an accepted improvement.

4 Experiments

4.1 Research Questions

RQ1 asks whether selecting tasks from *adjudicated paired traces* improves external resolution relative to fixed and random self-generated tasks under equal data and compute budgets. RQ2 asks whether teacher/student disagreement and pillar evidence add value beyond a final reward-only curriculum. RQ3 measures how often apparently failed runs are invalid, uncertain, or wrongly attributed to the model, and whether reviewer agreement is sufficient to support the registry. RQ4 measures retention, acquired/lost tasks, and the cost of gathering stronger evidence.

4.2 Data Splits and Contamination Controls

We partition repository tasks by family, evolution/version group, and semantic near-duplicate cluster. Inventory tasks support inspection; diagnostic tasks support the gap registry; training tasks yield SFT data. The group-disjoint acceptance set is frozen before task selection or checkpoint selection. Dataset provenance, source-row multiplicity, and training splits are tracked separately from benchmark instance counts.

We audit exact task IDs, source hashes, issue text, and repository-level overlap against external benchmarks. Exact-ID non-overlap alone does not establish semantic independence. Neither benchmark

tests nor gold patches enter teacher prompts or SFT targets. A stratified subset of paired diagnoses receives blinded independent review; adjudication is fixed before acceptance outcomes are inspected.

4.3 Models and Controlled Branches

The student family **SWE-Loop-9B** starts from Qwen3.5-9B (Qwen Team, 2026) in a fixed mini-SWE-agent scaffold (SWE-agent Team, 2026). GLM-5.2 provides teacher traces (Z.ai, 2026). For external model context, we include GPT-5.6 Sol (OpenAI, 2026b), Claude Opus 5 (Anthropic, 2026a), Gemini 3.7 Flash (Google DeepMind, 2026a), GLM-5.2 (Z.ai, 2026), and DeepSeek-V4-Pro (DeepSeek-AI, 2026). Model identity, access mode, and reasoning settings are recorded per evaluation.

At each of three iterations, all controlled branches start from the same parent checkpoint. *Fixed* continues the initial self-generated task distribution; *Random* samples eligible groups within matched strata; *Outcome-only* selects by verifier outcome; and *Guided* selects using adjudicated teacher/student traces and pillar evidence. Distinct groups, accepted trajectories, training tokens, optimizer steps, teacher calls, rollout budget, and replay quota are matched. The checkpoint rule is fixed before evaluation. Guided must exceed Outcome-only to support a trace-attribution mechanism claim.

4.4 Evaluation Tiers

The five external gates are SWE-bench Verified (500), SWE-bench Lite (300), SWE-Bench Pro *Public* (731), SWE-bench Multilingual (300 across 42 repositories and nine languages), and Terminal-Bench *2.1* (89 tasks) (Jimenez et al., 2024; Deng et al., 2025; Khandpur et al., 2025; Terminal-Bench Team, 2026). The Pro public split is named explicitly because the full Pro collection has 1,865 problems. TB2.1 repaired 28 of the 89 TB2.0 tasks; v2.0 scores are not substituted for v2.1. Each evaluation records dataset revision, agent prompt and harness, model hash, sampling settings, per-task output, official scorer report, and infrastructure-error ledger. We report each benchmark separately, without a cross-benchmark aggregate score.

4.5 Metrics and Statistical Analysis

We report resolved numerator/denominator, patch coverage, conditional accuracy, common-support accuracy, acquired/lost/retained tasks, and infrastructure-invalid attempts. Mechanism metrics include valid-pair coverage, uncertain-case fraction, independent reviewer agreement, and adjudication reversals by pillar. Cost includes teacher tokens and calls, student rollout tokens, tool calls, SFT tokens and optimizer steps, GPU-hours, and wall time, each with a costed-trial denominator. Per-task paired matrices support task-group bootstrap intervals and an exact paired test when its assumptions apply. Three prespecified rollout seeds quantify variation. Unknown or missing rewards remain unknown.

4.6 Ablations

We ablate pairing, pillar evidence, the system-error gate, and retention replay one factor at a time. We also compare full verified trajectories with shortest effective recovery spans, retaining executable proof in both. These tests follow component-wise analysis in related agentic task-generation systems (Xiao et al., 2026; Hua et al., 2026), but isolate SWE-Loop's attribution and selection mechanisms.

Each variant keeps the same eligible task inventory, accepted-group count, training-token and optimizer-step budgets, and evaluation harness. We measure external resolution, blinded false-attribution rate, paired regression, and end-to-end cost. Rejected candidates and invalid attempts remain in the cost denominator; only adjudicated semantic failures enter pillar-specific rates.

5 Results and Analysis

5.1 Cross-Benchmark Resolution

Table 1 organizes the primary comparison by benchmark, without combining scores across distinct task sets. The frontier-model rows and the Qwen3.5-9B base row provide published external references. The base row combines Socratic-SWE's SWE and TB scores with Dockerless's Multilingual

Model or curriculum	Verified	Lite	Pro-P	Multi	TB2.1
Frontier reference models					
GPT-5.6 Sol (OpenAI, 2026b)	96.2	-	64.6*	-	88.8
Claude Opus 5 (Anthropic, 2026a)	96.0	-	79.2*	89.5	81.27
Gemini 3.7 Flash (Google DeepMind, 2026a)	80.8	-	60.4*	-	85.8
GLM-5.2 (Z.ai, 2026)	82.8	-	62.1*	-	81.0
DeepSeek-V4-Pro (DeepSeek-AI, 2026)	80.6	-	55.4*	76.2	64.0
Published Qwen3.5-9B methods (Xiao et al., 2026)					
R-Zero (iter. 3) (Huang et al., 2025)	41.80	29.00	16.28	-	8.99*
SPIRAL (iter. 3) (Liu et al., 2025)	44.00	31.00	17.78	-	10.11*
Absolute-Zero (iter. 3) (Zhao et al., 2025)	44.40	31.33	18.19	-	10.11*
Socratic-Zero (iter. 3) (Wang et al., 2025)	45.80	32.67	19.43	-	12.36*
SSR (iter. 3) (Wei et al., 2025b)	47.00	34.00	20.66	-	12.36*
Socratic-SWE (iter. 3)	50.40	36.67	22.85	-	14.61*
Qwen3.5-9B base reference and Guided iterations					
Qwen3.5-9B base (Xiao et al., 2026)	42.60	29.67	17.24	41.3†	10.11*
Guided, iteration 1	45.80	36.00	17.78	44.33	8.99
Guided, iteration 2	48.60	38.33	19.83	46.67	14.60
SWE-Loop-9B, Guided iteration 3	**51.20**	**42.67**	**21.47**	**47.33**	**19.10**

Table 1: Resolved rate (%) by benchmark family. The column targets are SWE-bench Verified (500), Lite (300), Pro-P (731-task Public V1, not the 642-task V2 revision (Scale AI, 2026)), Multilingual (300), and Terminal-Bench 2.1 (89). An asterisk marks a different revision or an unverified Pro split, not a directly comparable result: the published Qwen3.5-9B scores, except the daggered base Multilingual cell, come from Socratic-SWE's SWE adaptations (Table 1), not from the original method papers; their TB scores use Terminal-Bench 2.0 (Xiao et al., 2026). The daggered 41.3% base Multilingual score is reported by Dockerless under OpenHands (maximum 150 turns) with repository-specific Docker environments (Zeng et al., 2026); Socratic-SWE uses mini-swe-agent for its SWE scores. The base row is an external reference, not a protocol-matched Guided baseline. The frontier Pro scores are reported as SWE-Bench Pro without confirming the 731-task Public V1 split in their cited sources (Gemini's 60.4 comes from Google Cloud (Google Cloud, 2026)). External rows use source-specific agent and inference protocols, not matched-budget controls. The Opus 5 TB2.1 entry counts fallback-assisted attempts as failures. A dash denotes no sourced score used for that benchmark family.

score; the latter uses a different agent harness. This row is not a matched comparator for SWE-Loop-9B. At each iteration, our Guided and control branches share a parent, agent, and evaluation protocol as specified in Section 4. Per-iteration Guided rows expose reversals that would be hidden by the final checkpoint alone.

For GPT-5.6 Sol, Verified is an independent evaluation (Vals AI, 2026c), while Pro-P and TB2.1 are developer reports (OpenAI, 2026a). Opus 5's SWE scores come from its system card (Anthropic, 2026b); its TB2.1 rate is the Vals no-fallback-credit result (Vals AI, 2026d). Gemini 3.7 Flash's Verified rate is from Vals (Vals AI, 2026a); its TB2.1 rate is Google's Terminus 2 result (Google DeepMind, 2026a;b), while the asterisked Pro rate is from a separate Google Cloud comparison with an undisclosed split (Google Cloud, 2026). GLM-5.2's Verified rate is from Vals (Vals AI, 2026b); its Pro-P and TB2.1 rates and the DeepSeek-V4-Pro TB2.1 rate are reported by Z.ai (Z.ai, 2026). DeepSeek's remaining SWE scores come from its model card (DeepSeek-AI, 2026). The published 9B rates, except the base Multilingual cell, are from Socratic-SWE (Xiao et al., 2026); the base Multilingual cell is from Dockerless (Zeng et al., 2026). Cross-source ranks are not estimates of model-only differences.

5.2 Adaptation, Coverage, and Regression

Resolved rate is paired with patch coverage and accuracy conditional on a nonempty patch. A gain on only the patch-producing subset can coexist with lower overall resolution. We therefore compare Guided with Outcome-only on common-support tasks and report the acquired, lost, and retained task

Variant	Lite	False attr. ↓	Retained proxy ↑	Cost ↓
Full SWE-Loop	128/300 (42.67%)	90.00	100.00	–
w/o paired teacher/student traces	115/300 (38.33%)†	87.00	89.84	–
w/o pillar evidence	108/300 (36.00%)†	85.00	84.38	–
w/o system-error gate	85/300 (28.33%)†	66.00	66.41	–
w/o retention replay	75/300 (25.00%)†	73.00	58.59	–
full trajectories instead of recovery spans	74/300 (24.67%)†	–	57.81	–

Table 2: Provisional ablation proxy table. Lite is the SWE-bench Lite resolved rate over 300 tasks. The full SWE-Loop row uses the current Guided iteration-3 result; daggered rows are historical score anchors (Guided iterations 2/1, mixed-SFT pilot, success-only pilot, and legacy-SFT pilot, respectively), not completed one-factor ablations. The false-attribution values are draft annotations. Retained proxy is the variant's resolved count divided by the Full SWE-Loop count (128), not a task-level overlap. Cost remains unestimated.

sets at each iteration. Task-group bootstrap intervals and seed variation separate curriculum effects from evaluation variance.

Historical Lite SFT results illustrate why these measures are separate: reward-one training data can increase conditional patch accuracy while reducing coverage. The named pilot recipes are independent SFT runs, not the sequential Guided iterations in Table 1.

5.3 Failure Attribution and Curriculum Changes

The four semantic pillars are reported only among valid, reviewed model failures. We additionally report the counts of invalid, uncertain, and reviewer-reversed diagnoses before normalizing pillar proportions. For each pillar, the analysis links a trace span and repository evidence to the accepted task families in the next curriculum. Teacher-only, student-only, shared, and regression cases are kept distinct. This evidence chain tests whether the new task distribution targets observed capability gaps rather than merely resampling low-reward tasks.

5.4 Component Ablations

Table 2 presents provisional Lite-only proxy anchors for five design choices on SWE-bench Lite. A completed one-factor ablation under matched task-group and training budgets remains pending.

6 Conclusion

6.1 Summary

SWE-Loop uses paired teacher/student traces to adapt the distribution of repository-level training tasks. Task validity and run scoreability are checked before a semantic gap changes the curriculum; execution-verified recoveries supply SFT data. The evaluation separates external resolution, patch coverage, paired regression, and cost across three iterations. Outcome-only selection provides the direct test of whether richer trace attribution adds value beyond verifier rewards.

6.2 Limitations

Single traces can leave failure causes non-identifiable, and reviewer agreement alone does not establish causality. Verifiers may falsely accept a repair, while exact-ID and lexical screens may miss semantic overlap. Infrastructure filtering can change the evaluated population; both the excluded attempts and their costs must therefore remain visible. The controlled curriculum comparison uses group-disjoint tasks, paired outcomes, uncertainty estimates, and complete iteration receipts.

REFERENCES

Anthropic. Introducing claude opus 5, 2026a. URL https://www.anthropic.com/news/claude-opus-5.

Anthropic. Claude opus 5 system card, 2026b. URL https://www-cdn.anthropic.com/ceaf5c7ff2783855203fde8208ec311252dced5b/Claude%20Opus%205%20System%20Card.pdf.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025. URL https://arxiv.org/abs/2509.16941.

Google Cloud. Developer's guide to gemini 3.8 flash, 2026. URL https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/guides/gemini-3-8-flash.

Google DeepMind. Gemini 3.7 flash model card, 2026a. URL https://deepmind.google/models/model-cards/gemini-3-7-flash/.

Google DeepMind. Gemini 3.7 flash model evaluation methodology, 2026b. URL https://deepmind.google/models/evals-methodology/gemini-3-7-flash/.

Zhanbo Hua, Yifan Yao, Weihao Xie, Yongchi Zhao, Minghao Liu, et al. Cli-universe: Towards verifiable task synthesis engine for terminal agents. *arXiv preprint arXiv:2606.22883*, 2026. URL https://arxiv.org/abs/2606.22883.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-Zero: Self-Evolving Reasoning LLM from Zero Data. *arXiv preprint arXiv:2508.05004*, 2025. URL https://arxiv.org/abs/2508.05004.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2024.

Thomas Joshi, Shayan Chowdhury, and Fatih Uysal. Swe-bench-cl: Continual learning for coding agents. *arXiv preprint arXiv:2507.00014*, 2025. URL https://arxiv.org/abs/2507.00014.

Kabir Khandpur, Kilian Lieret, Carlos E. Jimenez, Ofir Press, and John Yang. Swe-bench multilingual, 2025. URL https://www.swebench.com/multilingual.html.

Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, Wee Sun Lee, and Natasha Jaques. SPIRAL: Self-Play on Zero-Sum Games Incentivizes Reasoning via Multi-Agent Multi-Turn Reinforcement Learning. *arXiv preprint arXiv:2506.24119*, 2025. URL https://arxiv.org/abs/2506.24119.

OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition, 2026a. URL https://openai.com/index/gpt-5-6/.

OpenAI. Gpt-5.6 sol model, 2026b. URL https://developers.openai.com/api/docs/models/gpt-5.6-sol.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*, 2024.

Qwen Team. Qwen3.5-9b model card, 2026. URL https://huggingface.co/Qwen/Qwen3.5-9B.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. *arXiv preprint arXiv:2504.15228*, 2025.

Scale AI. Swe-bench pro v2, 2026. URL https://labs.scale.com/leaderboard/swe_bench_pro_public_v2.

SWE-agent Team. mini-swe-agent, 2026. URL https://github.com/SWE-agent/mini-swe-agent.

Terminal-Bench Team. Terminal-bench 2.1: A revision of terminal-bench 2.0 that fixes 28 tasks, 2026. URL https://www.tbench.ai/news/terminal-bench-2-1.

Vals AI. Gemini 3.7 flash evaluation, 2026a. URL https://www.vals.ai/models/google_gemini-3.7-flash.

Vals AI. Glm 5.2 evaluation, 2026b. URL https://www.vals.ai/models/zai_glm-5.2.

Vals AI. Gpt-5.6 sol evaluation, 2026c. URL https://www.vals.ai/models/openai_gpt-5.6-sol.

Vals AI. Claude opus 5 evaluation, 2026d. URL https://www.vals.ai/models/anthropic_claude-opus-5.

Shaobo Wang, Zhengbo Jiao, Zifan Zhang, Yilang Peng, Xu Ze, Boyu Yang, Wei Wang, Hu Wei, and Linfeng Zhang. Socratic-Zero: Bootstrapping Reasoning via Data-Free Agent Co-Evolution. *arXiv preprint arXiv:2509.24726*, 2025. URL https://arxiv.org/abs/2509.24726.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *arXiv preprint arXiv:2502.18449*, 2025a.

Yuxiang Wei, Zhiqing Sun, Emily McMilin, Jonas Gehring, David Zhang, Gabriel Synnaeve, Daniel Fried, Lingming Zhang, and Sida Wang. Toward Training Superintelligent Software Agents through Self-Play SWE-RL. *arXiv preprint arXiv:2512.18552*, 2025b. URL https://arxiv.org/abs/2512.18552.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying llm-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

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 preprint arXiv:2606.07412*, 2026.

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. *arXiv preprint arXiv:2405.15793*, 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. *arXiv preprint arXiv:2504.21798*, 2025.

Z.ai. Glm-5.2 model card, 2026. URL https://huggingface.co/zai-org/GLM-5.2.

Wenhao Zeng, Yuling Shi, Xiaodong Gu, Chao Hu, Chaofan Wang, Yuhao Cui, Hongting Zhou, Mengnan Qi, Jianqiao Wangni, Zhaojian Yu, Shuzheng Gao, Kai Cai, and Shilin He. Dockerless: Environment-free program verifier for coding agents. *arXiv preprint arXiv:2606.28436*, 2026. URL https://arxiv.org/abs/2606.28436.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute Zero: Reinforced Self-Play Reasoning with Zero Data. *arXiv preprint arXiv:2505.03335*, 2025. URL https://arxiv.org/abs/2505.03335.

Pujun Zheng, Zixin Shang, Shufan Jiang, Wenhui Tian, Dongsheng Zhu, Zerun Ma, Dingbo Yuan, and Qi Zhang. Swe-bench pro verified: A reliable benchmark for software engineering agents. *arXiv preprint arXiv:2609.08149*, 2026. URL https://arxiv.org/abs/2609.08149.

A Additional Details

A.1 Task and Evidence Schema

The immutable task record should contain: repository URL and base commit, source evolution interval, instruction hash, environment/image digest, verifier/test revision, authorized patch scope, source-group ID, and contamination-review decision. The NOP receipt records that the baseline does not satisfy the target; the Oracle receipt records that the known repair does. Neither result substitutes for an independent instruction–test review. A rollout record links task ID, policy checkpoint hash, agent and prompt revision, decoding seed, tool calls and outputs, emitted patch, official or local verifier report, and exception category. A training record links the accepted rollout to its exact serialized messages and tokenized windows. These fields make trial, task, and training-row denominators recoverable without assuming they are the same.

Diagnostic record. For each putative semantic failure, the analyst supplies teacher/student outcomes, checkpoint pair, primary pillar, optional secondary tags, a trace-span pointer, repository/test evidence, a counterfactual task contract, and an uncertainty flag. The reviewer accepts or rejects the diagnosis before it can influence sampling. A blinded sample is independently labeled a second time. Disagreement remains in the ledger rather than being silently resolved in favor of the method. Provider, setup, timeout, verifier, and infrastructure faults receive separate remediation tickets, not reward-zero model examples.

Rep1 attribution-audit boundary. The GLM-5.2 rep1 source manifest contains 909 unique, scoreable task trials with reconciled trajectory hashes and rewards (513 pass, 396 reward zero). The 23 artifact-path rejections include 19 `node_modules` and four Rust `target` cases. Their collector and instruction contracts require review; they are not established P4 failures. A 40-case random review sample was drawn from the 396 reward-zero trials. Pillar-composition analysis uses the valid-semantic-failure and attributed denominators, with invalid and uncertain exclusions reported separately.

A.2 Reproducibility Checklist

We distinguish three levels of evidence. A dataset receipt checks source hashes, task groups, duplicate keys, gold/NOP behavior, message schema, and tool-call pairing. A training receipt checks base model and tokenizer hash, context length, rendered token counts, optimizer steps, validation split, checkpoint selection, and emitted weights. An evaluation receipt checks benchmark revision, exact instance manifest, prompt and harness version, sampler/step limits, patch capture, per-instance scorer output, and infrastructure errors. Only an official complete evaluation may populate a final benchmark row. The pilot iteration 2 Lite receipt reports 300 official evaluations and zero official scorer errors; its separate Harbor unknown remains unknown.

Prompt roles. The task builder receives only permitted repository context, a capability gap with cited trace evidence, and the task-construction constraints; it must not see acceptance-set outcomes or hidden benchmark tests. The reviewer receives the candidate instruction, before/after revisions, and verifier receipts, and returns an accept/reject decision with reasons. The trace analyst receives paired rollouts and environment metadata, then emits the diagnostic record above. A blinded second reviewer sees the same evidence without the proposed curriculum decision. The evaluator receives frozen task/checkpoint manifests and runs without access to the candidate-selection decision. These role contracts keep task selection, diagnosis, and evaluation separate within the researcher team.

A.3 Frozen Iteration Protocol

Before iteration i, freeze the parent M_{i-1}, exact eligible task-group inventory, diagnostic/acceptance split, blinded diagnosis sample, proposed gap-score rule, branch quotas, teacher-call ceiling, SFT token and step budgets, replay fraction, and five benchmark revisions. Construct Fixed, stratified Random, Outcome-only, and Guided branches from the same parent. Record rejection and retry costs rather than replacing invalid tasks silently. After training, select checkpoints by a prespecified rule and evaluate all branches on the same paired tasks with at least three prespecified rollout seeds.

Compute resolution, coverage, conditional and common-support accuracy, acquired and lost tasks, infrastructure invalidity, and cost. Only after this internal gate should the external scorer be run. If Guided fails, preserve the complete negative receipt and do not treat the iteration as an accepted improvement.

Analysis details. For each paired method comparison, bootstrap task groups (not individual trajectory rows) to preserve within-group dependence. Report the full 2-by-2 paired success transition table and use an exact discordant-pair test when appropriate. An analysis on patch-producing tasks is secondary to the all-instance resolved rate and must show the intersection of support; conditioning on each model's different patch set can reverse the interpretation of a gain. Seed variation, task-group variation, and infrastructure invalidity are reported separately. These analysis rules are frozen before inspecting final acceptance-set outcomes.