SWE-RESIDUAL: EVALUATING STARTING-STATE ROBUSTNESS OF CODING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Software engineering (SWE) benchmarks are the primary way to measure the capabilities of coding agents: an agent is placed in a code repository or machine environment, given a request, and judged by hidden tests. These benchmarks almost always start the agent from a clean state that contains no earlier work on the task. In practice, however, agents often continue work that a developer or another agent has already done and reported as complete, with no indication of what remains incorrect or missing. We introduce SWE-Residual, a benchmark for *Starting-State Robustness*: the ability to maintain performance on a task when the starting state already contains earlier work on that task. SWE-Residual contains 198 Residual States drawn from 111 tasks in DeepSWE and Terminal-Bench, where the earlier work modifies a code repository or a whole machine environment. Each Residual State is the state a predecessor agent left behind after submitting its work as complete, although the hidden tests still fail. The evaluated agent receives only the original request and this state, without the predecessor's trajectory or any sign that the attempt failed; because the request and hidden tests are unchanged, the starting state is the only difference from the source benchmark. Across five frontier agents, success drops by 20.6–53.5 percentage points on DeepSWE and 20.6–34.9 points on Terminal-Bench relative to a Clean Start, and agent rankings on DeepSWE reverse. Failed continuations show a pattern we call *acceptance bias*: agents treat the inherited work as the definition of what remains to be done instead of checking it against the request. On DeepSWE, in 58.9–88.2% of failed runs, they submit with exactly the same tests failing, although they pass most of these tests from a Clean Start. SWE-Residual thus exposes a capability gap that clean-start evaluation does not capture.

1 INTRODUCTION

Coding agents are language models that read files, run commands, and edit code, and they are now used for software tasks that take hours of work (Yang et al., 2024; Deng et al., 2025; Huang et al., 2026). As these agents have become more capable, a growing number of benchmarks have been built to measure their capabilities, from resolving issues in code repositories to operating a whole machine through the command line (Jimenez et al., 2024; Huang et al., 2026; Merrill et al., 2026). These benchmarks differ in their tasks, but they evaluate an agent in the same way. The agent is placed in a starting state, such as a code repository or a machine, and given a request; it works until it declares the task done, and it succeeds if a set of hidden tests then passes. Scores obtained in this way are now the main evidence of how capable a coding agent is.

However, these benchmarks typically start the agent from a clean initial state, one that contains no earlier work on the task the agent is given. We call this a *Clean Start*. Yet in practice, agents often start from states that already contain earlier work on the same task. Development is routinely handed over between developers (Khatoonabadi et al., 2023), and increasingly between agents: a user may return in a new session to work that an earlier agent declared done, without the earlier session's context, and in multi-agent systems one agent builds on output that another has submitted as final. In each case, the agent inherits the artifacts of earlier work, such as modified code, newly created files, or installed software. Agents frequently declare success on tasks they have not completed (Advani, 2026), so the work handed to the next agent or session often looks finished but is not. We study handoffs in which this work was submitted as complete but does not satisfy the request, and the

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.

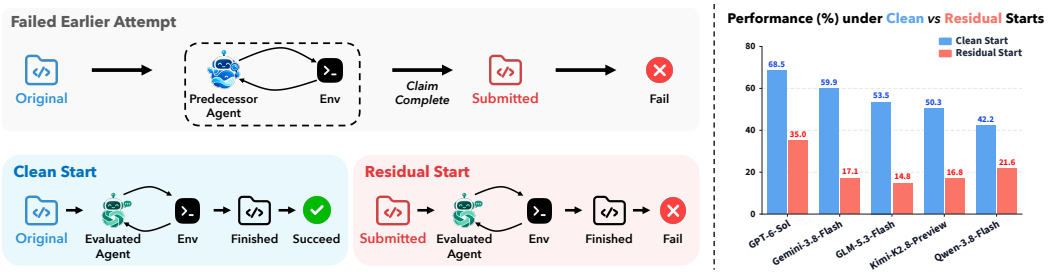


Figure 1: **Left:** Standard benchmarks evaluate an agent from the original repository (*Clean Start*). SWE-Residual evaluates the same agent on the same task from the repository left by a predecessor agent that submitted its work as complete but failed the hidden tests (*Residual Start*), with the same request and hidden tests and no information about the earlier attempt. **Right:** Success rates of the five evaluated agents under both settings, averaged over DeepSWE and Terminal-Bench.

agent receives no indication of what is incomplete or incorrect: it must work out from the state alone what has been done and what is still missing.

We call the ability to complete a task from a state that already contains such earlier work *Starting-State Robustness*. Most benchmarks cannot measure this ability, because a Clean Start never presents the agent with earlier work, so two agents with the same benchmark score may still differ widely in whether they can finish what someone else has started, and, as we show, their ranking can even reverse. This ability cannot be taken for granted. On one DeepSWE task, for example, an agent inspects the inherited changes, runs the package tests, and submits without changing a single file, although the same agent passes the 11 failing hidden tests from a Clean Start.

To address this gap, we introduce **SWE-Residual**, a benchmark for Starting-State Robustness, as shown in Figure 1. A *Residual State* is the state a predecessor agent leaves behind after submitting its work on a task from an existing benchmark as complete, although the hidden tests still fail; starting an agent from it is a *Residual Start*. Such a state looks finished: the work is coherent, often passes the predecessor's own checks, and on DeepSWE the median Residual State already passes 98.4% of the hidden tests; yet parts of the request remain missing or incorrectly implemented, and the evaluated agent must determine which parts of the apparent solution to trust and what work remains. Each instance of SWE-Residual pairs a Residual State with the original request and hidden tests. Because these are unchanged, the same agent can attempt each task from a Clean Start and from a Residual Start. We report success from Residual Starts and the change from Clean Start to Residual Start on the same tasks.

We build SWE-Residual on DeepSWE, where the inherited work modifies a code repository, and Terminal-Bench, where it modifies a machine's files, installed software, and configuration, using Residual States left by predecessor agents from three model families (DeepSeek, Qwen, and GLM). Across five frontier agents, success drops by 20.6–53.5 percentage points on DeepSWE and 20.6–34.9 points on Terminal-Bench relative to Clean Start, and agent rankings on DeepSWE reverse, so strong clean-start performance does not guarantee Starting-State Robustness. We observe a recurring pattern we call *acceptance bias*: the evaluated agent treats the inherited work as the definition of what remains to be done, instead of checking it against the original request. It inspects the inherited changes but keeps most of them and rarely changes the files the predecessor left out. In 58.9–88.2% of failed continuations on DeepSWE, exactly the same tests fail at the end as at the start, even though the same agent passes most of these tests from a Clean Start.

In summary, our contributions are as follows.

- We identify Starting-State Robustness, the ability to maintain performance on a task when the starting state already contains earlier work on that task, as an evaluation dimension that complements conventional clean-start evaluation.
- We release SWE-Residual, comprising 198 Residual States produced by predecessor agents from three model families (DeepSeek, Qwen, and GLM) on tasks from DeepSWE and Terminal-Bench. We also provide Residualization, a pipeline for constructing and validat-

ing these states that enables the benchmark to be extended to additional predecessor agents and source benchmarks.

- Across five frontier agents, success drops substantially from a Clean Start to Residual States, and failed continuations show a pattern we call *acceptance bias*: agents treat the inherited work as the definition of what remains to be done instead of checking it against the original request.

2 Related Work

Coding-agent benchmarks. Rapid progress in LLMs has spurred substantial research on coding agents for automated software engineering. Such agents are commonly evaluated by placing them in a predefined software environment, providing a task specification, and evaluating whether their resulting changes satisfy execution-based tests. The SWE-bench series (Jimenez et al., 2024; Deng et al., 2025) established this paradigm for resolving real-world repository-level issues, and subsequent benchmarks have extended it to more programming languages (Zan et al., 2025; Rashid et al., 2025; Guo et al., 2025), multimodal inputs (Yang et al., 2025), and longer-horizon tasks (Huang et al., 2026; Zhou et al., 2026; Miserendino et al., 2025; Le et al., 2026). Beyond code repositories, Terminal-Bench (Merrill et al., 2026) extends agent evaluation to whole machine environments in which agents must manipulate files, dependencies, configurations, and command-line tools. A related line gives an agent a sequence of requests on one repository and studies how the code accumulated for earlier requests affects later ones (Chen et al., 2026; Lam et al., 2026; Shastry et al., 2026; Orlanski et al., 2026). Despite differing in task scope and environment, these benchmarks share the assumption that each task begins from a clean starting state containing no prior work on the task itself. This leaves performance on states modified by an earlier attempt at the same task unmeasured.

Benchmarks beyond clean-start evaluation. Recent research has begun to relax the clean-start assumption by evaluating coding agents in code or machine environments already modified by prior attempts. Handoff Debt (KC & Budathoki, 2026) and Handoff Tax (Ganz et al., 2026) study takeover from intermediate agent checkpoints, examining how different forms of predecessor context affect the handoff cost of resuming a task. OmniCode (Sonwane et al., 2026) and FeedbackEval (Dai et al., 2025) assess whether agents can comprehend and repair code given explicit compiler, test, or LLM feedback. Recovery-Bench (Tan et al., 2025) replays failed Terminal-Bench runs of a weaker agent and evaluates whether another agent can recover from the resulting environment, finding that more context from the failed attempt can hurt. FixedBench (Gloaguen et al., 2026) studies the reverse situation, in which the reported issue is already resolved, and finds that agents often make unnecessary changes. Together, these works show that prior attempts can substantially affect subsequent agent behavior and completion cost. Table 1 summarizes how their settings differ from SWE-Residual, which studies a handoff after a predecessor has submitted its work as complete, although the original task remains unsolved, and pairs each Residual State with a Clean Start of the same agent on the same task.

Relation to Recovery-Bench. Recovery-Bench asks whether an agent can recover from a failed attempt: its states come from failed Terminal-Bench runs, whether or not the predecessor submitted its work, and the task is framed as recovery from that attempt. SWE-Residual instead keeps only final states that were submitted as complete and still fail the original verifier; the submission is a selection criterion, and the evaluated agent receives only the request and the inherited state, without the submission or the verdict. The question therefore shifts from recovering after a known failure to finding, in work that appears done, what the original request still requires.

Why agents accept flawed work. Our analysis of acceptance bias relates to known limitations of language models. Without external feedback, models often fail to correct their own reasoning (Huang et al., 2024; Kamoi et al., 2024), and they locate errors less reliably than they fix them once the location is given (Tyen et al., 2024). They also struggle to notice what is missing from an input (Fu et al., 2025), and they tend to build on earlier mistakes instead of revisiting them (Zhang et al., 2024; Laban et al., 2026), even declaring success on tasks they have not completed (Advani, 2026). Human developers who reuse an existing artifact show a similar anchoring effect (Parsons

Table 1: A comparison of SWE-Residual to existing coding agent benchmarks. *Prior work*: the starting state contains residual artifacts from another agent's earlier attempt. *Submitted as done*: the predecessor submitted the prior work as complete, rather than being interrupted or selected without regard to submission. *Failed*: the prior work does not fully satisfy the task specification. *Blind continuation*: the evaluated agent receives only the resulting state and task request, with no explicit feedback, trajectory, or results of the prior attempt. ✓ yes, ✗ no, – not applicable.

Benchmark	Environment	Prior work	Submitted as done	Failed	Blind continuation
SWE-bench (Jimenez et al., 2024)	Repository	✗	–	–	–
Terminal-Bench (Merrill et al., 2026)	Machine	✗	–	–	–
Handoff Debt (KC & Budathoki, 2026)	Repository	✓	✗	✗	✗
Handoff Tax (Ganz et al., 2026)	Repository	✓	✗	✗	✗
FeedbackEval (Dai et al., 2025)	Repository	✓	✗	✓	✗
OmniCode (Sonwane et al., 2026)	Repository	✓	✗	✓	✗
Recovery-Bench (Tan et al., 2025)	Machine	✓	✗	✓	✗
SWE-Residual (ours)	Repository & Machine	✓	✓	✓	✓

& Saunders, 2004). SWE-Residual measures what these tendencies cost when one agent continues another agent's work.

3 SWE-RESIDUAL

This section formalizes the SWE-Residual evaluation setting and describes how we construct and validate its starting states.

3.1 TASK FORMULATION

Clean Start. An evaluation of task T_i consists of a request q_i, a starting state S_i, and a verifier V_i. The starting state describes the environment that the agent can observe and modify when it begins: in DeepSWE, this is the code repository; in Terminal-Bench, it is the entire machine environment, including files, installed software, configurations, and running processes. The verifier V_i evaluates the final state using a set of hidden tests that are never exposed to the agent.

For an agent A, let $F_A(q_i, S_i)$ denote the final state produced by agent A in a run when the agent either declares the task complete or exhausts the evaluation budget. We define the outcome of the run as

$$y_{A,i}(S_i) = V_i\left(q_i, F_A(q_i, S_i)\right) \in \{0, 1\}, \tag{1}$$

where 1 indicates success and 0 otherwise. We denote the clean starting state of task i by $S_{0,i}$, which contains no work from a prior attempt on q_i. A run under the standard clean-start setting therefore has outcome $y_{A,i}(S_{0,i})$.

Residual States. A *Residual State* is the final state left by a prior attempt at the task that was submitted as complete but failed. Let $\tilde{A}$ denote the predecessor agent that attempts task T_i from the clean state $S_{0,i}$. We define a Residual State as

$$S_{r,i} = F_{\tilde{A}}(q_i, S_{0,i}), \quad \text{where } \tilde{A} \text{ declares the task complete and } y_{\tilde{A},i}(S_{0,i}) = 0. \tag{2}$$

That is, $S_{r,i}$ is the state left by the predecessor agent $\tilde{A}$ that attempted the same request q_i and submitted its work as complete, but did not pass the hidden tests.

Starting-State Robustness. We use *Starting-State Robustness* to refer to an agent's ability to maintain its performance when the same request q_i is attempted from a Residual State $S_{r,i}$ rather than the clean state $S_{0,i}$. The verifier is held fixed, so the starting state is the only change.

3.2 BENCHMARK CONSTRUCTION

We call our process for constructing validated Residual States from attempts by predecessor agents *Residualization*. Residualization consists of four stages. We apply this process to DeepSWE (Huang et al., 2026) and Terminal-Bench (Merrill et al., 2026), while keeping each task's request and verifier unchanged.

Stage I: Trajectory generation. We use three predecessor agents based on DeepSeek-V4-Flash-0731, Qwen-3.8-27B, and GLM-5.3: recent open-weight models of different sizes that are inexpensive to run at scale and capable enough that their work resembles what agents hand over in practice. For each task T_i, a predecessor agent $\tilde{A}$ starts from the clean state $S_{0,i}$ and attempts the request q_i under a generous time limit of eight hours per run, producing a final state $F_{\tilde{A}}(q_i, S_{0,i})$ with outcome $y_{\tilde{A},i}(S_{0,i})$. For each run, we record the full trajectory, termination reason, whether the predecessor agent declared completion, and the verifier outcome, together with the information needed to reconstruct its final state: a binary-safe patch of all repository changes on DeepSWE, and the ordered sequence of terminal interactions on Terminal-Bench.

Stage II: Automatic filtering. Not every generated run yields a suitable candidate Residual State: some make no persistent changes, some terminate because of infrastructure failures, and others exhaust the time limit before the agent declares the task complete. We therefore apply rule-based filtering before detailed human review. For each generated run, we retain only those that satisfy three criteria: **(a) Submitted but failing:** the predecessor agent declares the task complete, but $y_{\tilde{A},i}(S_{0,i}) = 0$. **(b) Changed:** the final state $F_{\tilde{A}}(q_i, S_{0,i})$ differs from $S_{0,i}$ after removing incidental execution artifacts such as build outputs, caches, and logs. **(c) No infrastructure failure:** the run does not terminate because of failures unrelated to the task, such as model-request errors, container crashes, or unavailable network resources. Duplicate final states produced for the same task are retained only once.

Stage III: Human review. Agent trajectories can span hundreds of steps, making direct step-by-step inspection impractical. Two human reviewers therefore independently assess each candidate using the request q_i, the changes from the clean state $S_{0,i}$ to the candidate state $F_{\tilde{A}}(q_i, S_{0,i})$, and a GLM-5.3 summary of the predecessor trajectory that describes the intentions behind the agent's actions and its overall solution strategy. A candidate is retained only if it satisfies two criteria: **(a) Task-directed:** the summarized strategy reflects a coherent attempt to satisfy the requirements in q_i, rather than unrelated or incidental changes. **(b) Task–verifier consistent:** the request and hidden tests specify consistent requirements for task completion.

Stage IV: State reconstruction. The retained candidates are saved artifacts rather than reusable environments, so we reconstruct each one from the clean state $S_{0,i}$ and store it in a reproducible form. On DeepSWE, we apply the saved patch at the task's base commit without creating a commit, so the evaluated agent observes the modified working tree without the commit history that produced it. On Terminal-Bench, we replay the recorded interactions in a fresh task environment and checkpoint the resulting machine state, including live processes. We validate that each reconstructed state remains usable and that the original verifier still returns $V_i(q_i, S_{r,i}) = 0$ (Appendix B). During evaluation, each rollout starts from an independent instance of $S_{r,i}$ and receives only the original request q_i; the prior trajectory, predecessor outputs, and failure information are not explicitly supplied. We refer to this protocol as *blind continuation*.

4 BENCHMARK CHARACTERISTICS

SWE-Residual contains 198 validated Residual States derived from 111 tasks across DeepSWE and Terminal-Bench. Each state comes from a prior attempt submitted as complete, passes our inclusion review, and is verified to fail the original task verifier after reconstruction. Table 2 summarizes the benchmark's coverage, predecessor composition, prior-attempt effort, and initial test outcomes.

SWE-Residual covers 90 of DeepSWE's 113 tasks, spanning 74 repositories and five programming languages, and 21 of Terminal-Bench's 89 tasks, spanning nine task categories. The retained states reflect substantial prior attempts: the median predecessor run takes 134 agent steps on DeepSWE

Table 2: Statistics of SWE-Residual. Task counts refer to unique source tasks. Prior-attempt steps and execution time are medians over retained states; steps count recorded agent turns. Test pass rates include all tests used for task acceptance and are summarized with equal weight per state.

	DeepSWE	Terminal-Bench
Source benchmark tasks	113	89
Source tasks covered	90	21
Residual States	165	33
from DeepSeek	62	17
from Qwen	72	11
from GLM	31	5
Tasks with one / two / three states	31 / 43 / 16	11 / 8 / 2
Prior-attempt length (steps)	134	24
Predecessor execution time (min)	47.7	15.5
Test pass rate (mean / median, %)	91.6 / 98.4	47.6 / 60.0
States with 1–3 tests still failing	88 (53.3%)	33 (100%)

and 24 on Terminal-Bench, and DeepSWE states change a median of nine files. On Terminal-Bench, the inherited work extends beyond files: retained states include installed software, a compiled application, and running QEMU instances that the evaluated agent must work around or build on. Many states are close to passing: on DeepSWE, the median state passes 98.4% of the verifier's tests and 53.3% of states have only one to three tests still failing, yet every retained state fails overall. Appendix A gives further statistics.

5 Experiments

We first measure how success changes when the starting state moves from the clean state $S_{0,i}$ to a Residual State $S_{r,i}$, across five frontier agents and across predecessor agents. We then ask whether Clean Start successes carry over to Residual Starts, and finally analyze how agents behave when working from Residual States.

5.1 Experimental Setup

Agents and harnesses. We evaluate five frontier models as coding agents: Gemini-3.8-Flash, Qwen-3.8-Flash, GLM-5.3-Flash, GPT-6-Sol, and Kimi-K2.8-Preview. We use DeepSWE v1.1 and Terminal-Bench 2.1 as the source benchmarks. Following their evaluation setups, we use mini-swe-agent (Yang et al., 2024) as the agent harness for DeepSWE and Terminus-2 (Merrill et al., 2026) for Terminal-Bench, with all experiments orchestrated through Harbor (Team, 2026). Each model is evaluated with the highest available reasoning effort. Per-run time limits and other execution budgets follow the default settings of the corresponding source benchmark. Additional details, including sampling parameters, model-specific runtime settings, and infrastructure configurations, are provided in Appendix C.

Metrics. Our primary metric is the success rate under Residual Start. For tasks with multiple states, we first average their binary success outcomes within the task, then average across tasks so that each source task receives equal weight. As a matched reference, we report Clean Start success on the same source tasks. The request, verifier, harness, prompt, and evaluation budget are identical between the two conditions; only the starting state changes. We define *Residual Gap* as the Clean Start success rate minus the Residual Start success rate, measured in percentage points. A positive gap indicates lower success under Residual Start. We refer to a run that starts from a Residual State as a *continuation*.

Table 3: Performance on the two SWE-Residual tracks. Residual reports success averaged first over Residual States within each task, then equally across tasks. Ref. Clean reports success on the same underlying tasks. Gap is Ref. Clean minus Residual, in percentage points, computed before rounding. Bold indicates the highest residual and clean success rates within each track.

Track	Agent	Residual (%)	Ref. Clean (%)	Gap
SWE-Residual (DeepSWE)	Gemini-3.8-Flash	20.0	**72.2**	52.2
	Qwen-3.8-Flash	30.6	51.1	20.6
	GLM-5.3-Flash	15.4	68.9	53.5
	GPT-6-Sol	**33.5**	65.6	32.0
	Kimi-K2.8-Preview	13.0	57.8	44.8
SWE-Residual (Terminal-Bench)	Gemini-3.8-Flash	14.3	47.6	33.3
	Qwen-3.8-Flash	12.7	33.3	20.6
	GLM-5.3-Flash	14.3	38.1	23.8
	GPT-6-Sol	**36.5**	**71.4**	34.9
	Kimi-K2.8-Preview	20.6	42.9	22.2

5.2 Performance on SWE-Residual

We evaluate five frontier agents under blind continuation. Table 3 reports Residual Start success with each source task weighted equally, alongside the matched Clean Start reference and the Residual Gap. Residual Start success ranges from 13.0% to 33.5% on DeepSWE and from 12.7% to 36.5% on Terminal-Bench. GPT-6-Sol performs best under Residual Start on both tracks, with success rates of 33.5% and 36.5%, respectively. Even the highest-scoring agent therefore leaves a large fraction of inherited states unresolved.

Inheriting earlier work costs every agent 20.6 to 53.5 points. Every evaluated agent performs worse under Residual Start than under the matched Clean Start reference. The Residual Gap ranges from 20.6 to 53.5 percentage points on DeepSWE and from 20.6 to 34.9 points on Terminal-Bench. For example, Gemini-3.8-Flash falls from 72.2% to 20.0% on DeepSWE, while GPT-6-Sol falls from 71.4% to 36.5% on Terminal-Bench.

The decline occurs although the median DeepSWE state already passes 98.4% of the verifier's tests (Section 4), and it appears on both tracks, from inherited repository changes to modified machine environments.

Clean-start rankings reverse on DeepSWE. On DeepSWE, Gemini-3.8-Flash and GLM-5.3-Flash rank first and second under Clean Start, but third and fourth under Residual Start, behind GPT-6-Sol and Qwen-3.8-Flash. In particular, GLM-5.3-Flash outperforms Qwen-3.8-Flash by 17.8 percentage points under Clean Start, whereas Qwen-3.8-Flash outperforms GLM-5.3-Flash by 15.2 points under Residual Start. The DeepSWE ranking reversal shows that clean-start and residual-start success can rank agents differently. Absolute residual performance and the drop from Clean Start also capture different aspects of the results: GPT-6-Sol achieves the highest residual success rate on both tracks, while Qwen-3.8-Flash has the smallest Residual Gap.

The drop holds for states from every predecessor agent. To examine whether the performance decline persists across predecessor agents, we group all Residual States from DeepSWE and Terminal-Bench by the agent that produced them. Within each group, we compare Residual Start success with the matched Clean Start reference on the same underlying tasks, weighting each task equally. Table 8 in Appendix E.1 shows that all five evaluated agents perform worse under Residual Start in every predecessor group, with Residual Gaps ranging from 19.0 to 51.8 percentage points. The decline therefore occurs across states produced by all three predecessor agents. Since the groups cover different task sets, differences in their absolute scores may also reflect task difficulty.

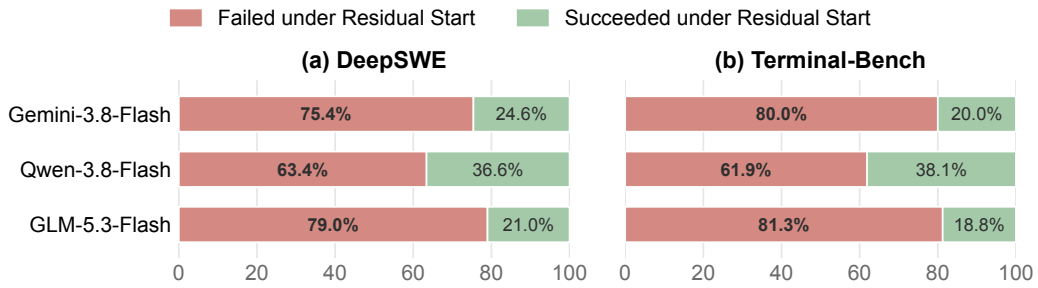


Figure 2: Residual Start outcomes on tasks that the same agent successfully completes under Clean Start. Red indicates failure and green indicates success. Outcomes are averaged over Residual States within each task, then equally across tasks. Each agent is evaluated on its own set of observed Clean Start successes.

5.3 Success Does Not Carry Over from Clean Start

We next ask whether an agent can still complete a task from a Residual State when it succeeds on the same task from Clean Start. For each agent, we select its observed Clean Start successes and examine its Residual Start outcomes on those tasks. Figure 2 shows the resulting failure and success rates.

We run this analysis on three agents: Gemini-3.8-Flash, Qwen-3.8-Flash, and GLM-5.3-Flash. Each has a conditional failure rate above 60% on both tracks, with rates from 63.4% to 79.0% on DeepSWE and from 61.9% to 81.3% on Terminal-Bench. For example, Gemini-3.8-Flash fails at a rate of 75.4% on DeepSWE tasks it completes under Clean Start, while GLM-5.3-Flash fails at a rate of 81.3% on its corresponding Terminal-Bench tasks.

On tasks each agent completes from a Clean Start, most task-weighted Residual Start outcomes are failures. Success on the original task often fails to carry over once earlier work is inherited. Gemini-3.8-Flash and GLM-5.3-Flash, the strongest of the three under Clean Start, fail most often from inherited states on both tracks.

5.4 Acceptance Bias

From a Clean Start, agents pass most of the tests that a Residual State fails; from the Residual State itself, most of these tests still fail at the end. We observe a recurring pattern in their continuations, which we call *acceptance bias*: once earlier work is present, the agent accepts it as the definition of what remains to be done, in place of the request. It continues inside the earlier work and leaves what that work got wrong or left out as it found it. We examine this on DeepSWE, whose per-test verifier outcomes, final patches, and reference solutions make each part of this pattern measurable (Table 4).

Inherited failures survive, although agents can fix them. In 58.9% to 88.2% of failed continuations, exactly the same verifier tests fail at the end as at the start. Across failed continuations, agents repair 10.4% to 20.1% of the initially failing tests and break almost none of the passing ones. These tests are not beyond the agents: on the same tasks, the same agent passes 72.8% to 88.4% of them from a Clean Start, against 14.4% to 47.7% from the Residual State. The tests become hard only once a failing version of the work is present.

The earlier work sets the scope of the continuation. Agents do not overlook the earlier work: every continuation inspects the inherited changes, through `git status`, `git diff`, or the modified files. They then work inside it. Discarding the inherited changes and starting over from the base commit is always possible, and no continuation does so. Failed continuations keep 86.4% to 98.6% of the lines the predecessor agent wrote, and, except for GPT-6-Sol, whose ratio is 1.02, continuations take 0.56 to 0.84 times as many steps as runs from a Clean Start. The clearest evidence comes from the files that the earlier work left out, that is, files changed by the reference solution but

Table 4: Acceptance bias on DeepSWE, with agents sorted by Residual Gap. *Same failing*: failed continuations that end with exactly the tests that failed in the Residual State. *Initially failing*: share of those tests that the agent passes from each start. *Kept*: lines written by the predecessor agent that remain after failed continuations. *Left-out files*: runs that change a file changed by the reference solution but not by the predecessor agent, over states where both runs are normally submitted. *Steps*: median ratio of Residual to Clean steps. Normally submitted runs; task-level averages in % (Gap in points).

			Initially failing			Left-out files		
Agent	Gap	Same failing	Clean	Residual	Kept	Clean	Residual	Steps
GLM-5.3-Flash	53.5	78.9	87.3	23.4	95.7	40.5	6.7	0.57
Gemini-3.8-Flash	52.2	76.8	88.4	27.5	97.2	43.5	9.4	0.70
Kimi-K2.8-Preview	44.8	88.2	77.6	14.4	98.6	45.7	8.1	0.56
GPT-6-Sol	32.0	58.9	84.1	47.7	91.1	41.2	15.0	1.02
Qwen-3.8-Flash	20.6	72.0	72.8	39.1	86.4	47.9	14.6	0.84

not by the predecessor agent. On the same states, agents change these files in 40.5% to 47.9% of their Clean Start runs but in only 6.7% to 15.0% of their Residual Start runs, a difference of 26.2 to 37.6 points. The files are not unseen: agents open them in 37.6% to 64.0% of Residual Start runs, and leave them unchanged.

The more an agent accepts, the more it loses. The three agents with the largest gaps, GLM-5.3-Flash, Gemini-3.8-Flash, and Kimi-K2.8-Preview, show every signature most strongly: they keep the most of the earlier work, most often leave the same tests failing, change the left-out files least, and take the fewest steps. GPT-6-Sol and Qwen-3.8-Flash, whose gaps are 32.0 and 20.6 points, show each signature more weakly. The same holds within agents: for Gemini-3.8-Flash, Qwen-3.8-Flash, and GLM-5.3-Flash, continuations that keep more of the earlier work than the agent's median succeed 14.0 to 22.7 points less often than those that keep less. GPT-6-Sol fails differently. In 11 of 30 sampled failed continuations it names the defect in the earlier work and tries to repair it, which Gemini-3.8-Flash, Qwen-3.8-Flash, and GLM-5.3-Flash each do in at most 2 of 30. Among these four agents, its failed continuations also repair the most initially failing tests (20.1%, against 10.4% to 13.7%) and least often end with the same failures (58.9%). Its failures are more often failed repairs than accepted work. Its Residual Gap is nonetheless 32.0 points: recognizing that the inherited work is wrong recovers part of the lost performance, and repairing it accounts for the rest.

Case study. A run of Gemini-3.8-Flash on `onedump` shows the pattern in its plainest form. The Residual State fails 11 verifier tests, all of which the same agent passes from a Clean Start. From the Residual State, the agent inspects the inherited changes, runs the package tests, which all pass, and submits after 43 steps without changing a single file. Its final message describes the earlier work as its own: "I have implemented the requested `encryption` package." The reference solution changes `encryption/encryption.go`, a file the predecessor agent never touched; the evaluated agent leaves it unchanged, while its own Clean Start run changes it. Qwen-3.8-Flash and GLM-5.3-Flash also fail from this state, each leaving the same 11 tests failing.

6 Conclusion

SWE-Residual measures how well coding agents continue work that a predecessor submitted as complete but that fails the hidden tests, given only the original request and the inherited state. Across five frontier agents, inheriting such work lowers success by 20.6 to 53.5 percentage points on the same tasks, most Clean Start successes do not carry over, and clean-start rankings can reverse. The failures share a pattern we call acceptance bias: agents treat the inherited work as the definition of what remains to be done and leave what it got wrong or left out in place; the more an agent accepts, the more it loses. Clean-start scores therefore overstate how well an agent can take over work in progress, and Starting-State Robustness deserves to be evaluated alongside them. Residualization applies to any benchmark with hidden tests, and SWE-Residual provides a testbed for agents that check inherited work against the request before building on it.

AI USE STATEMENT

Generative AI tools were used for typo revising, and were not used for method design or experimental analysis. In the benchmark construction, GLM-5.3 generates summaries of predecessor trajectories to support human review (Section 3.2); all inclusion decisions are made by human reviewers.

ETHICS STATEMENT

Residual States contain unsuccessful implementations produced during software task attempts. The evaluation protocol uses isolated environments for executing these research artifacts. Distribution follows the terms of the source repositories and benchmarks. Upon publication, we will release all Residual States, the artifacts needed to reconstruct them, the Residualization pipeline, the evaluation code, and per-run outcomes.

REPRODUCIBILITY STATEMENT

Section 3 specifies the construction and paired evaluation protocol. Appendix A reports benchmark statistics, Appendix B describes state transfer, and Appendices C–E give the experimental configurations and additional analyses. Reconstruction manifests, filtering records, and per-run outcomes will be publicly released upon publication.

REFERENCES

Laksh Advani. From Confident Closing to Silent Failure: Characterizing False Success in LLM Agents. *arXiv preprint arXiv:2606.09863*, 2026. URL https://arxiv.org/abs/2606.09863.

Jialong Chen, Xander Xu, Hu Wei, Chuan Chen, and Bing Zhao. SWE-CI: Evaluating Agent Capabilities in Maintaining Codebases via Continuous Integration. *arXiv preprint arXiv:2603.03823*, 2026. URL https://arxiv.org/abs/2603.03823.

Dekun Dai, MingWei Liu, Anji Li, Jialun Cao, Yanlin Wang, Chong Wang, Xin Peng, and Zibin Zheng. FeedbackEval: A Benchmark for Evaluating Large Language Models in Feedback-Driven Code Repair Tasks. *arXiv preprint arXiv:2504.06939*, 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, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can AI Agents Solve Long-Horizon Software Engineering Tasks? *arXiv preprint arXiv:2509.16941*, 2025. URL https://arxiv.org/abs/2509.16941.

Harvey Yiyun Fu, Aryan Shrivastava, Jared Moore, Peter West, Chenhao Tan, and Ari Holtzman. Absence Bench: Language Models Can't See What's Missing. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2506.11440.

Roy Ganz, Mor Shpigel Nacson, Adi Kalyanpur, and Ron Litman. The Handoff Tax: Continuing Non-Native Trajectories in LLM Agents. *arXiv preprint arXiv:2608.24358*, 2026. URL https://arxiv.org/abs/2608.24358.

Thibaud Gloaguen, Niels Mündler, Mark Müller, Veselin Raychev, and Martin Vechev. Coding Agents Don't Know When to Act. *arXiv preprint arXiv:2605.07769*, 2026. URL https://arxiv.org/abs/2605.07769.

Lianghong Guo, Wei Tao, Runhan Jiang, Yanlin Wang, Jiachi Chen, Xilin Liu, Yuchi Ma, Mingzhi Mao, Hongyu Zhang, and Zibin Zheng. OmniGIRL: A multilingual and multimodal benchmark for GitHub issue resolution. *Proc. ACM Softw. Eng.*, 2(ISSTA), June 2025. doi: 10.1145/3728871. URL https://doi.org/10.1145/3728871.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large Language Models Cannot Self-Correct Reasoning Yet. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.01798.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring Frontier Coding Agents on Original, Long-Horizon Engineering Tasks. *arXiv preprint arXiv:2607.07946*, 2026. URL https://arxiv.org/abs/2607.07946.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When Can LLMs Actually Correct Their Own Mistakes? A Critical Survey of Self-Correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024. URL https://arxiv.org/abs/2406.01297.

Dipesh KC and Anjila Budathoki. Handoff Debt: The Rediscovery Cost When Coding Agents Take Over Interrupted Tasks. *arXiv preprint arXiv:2606.02875*, 2026. URL https://arxiv.org/abs/2606.02875.

SayedHassan Khatoonabadi, Diego Elias Costa, Rabe Abdalkareem, and Emad Shihab. On wasted contributions: Understanding the dynamics of contributor-abandoned pull requests–a mixed-methods study of 10 large open-source projects. *ACM Transactions on Software Engineering and Methodology*, 32(1):1–39, 2023.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. LLMs Get Lost In Multi-Turn Conversation. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2505.06120.

Man Ho Lam, Chaozheng Wang, Hange Liu, Jingyu Xiao, Haau-sing Li, Jen-tse Huang, Terry Yue Zhuo, and Michael R. Lyu. SWE-Chain: Benchmarking Coding Agents on Chained Release-Level Package Upgrades. *arXiv preprint arXiv:2605.14415*, 2026. URL https://arxiv.org/abs/2605.14415.

Tue Le, Minh V. T. Thai, Dung Nguyen Manh, Huy Phan Nhat, and Nghi D. Q. Bui. SWE-EVO: Benchmarking Coding Agents in Long-Horizon Software Evolution Scenarios. *arXiv preprint arXiv:2512.18470*, 2026. URL https://arxiv.org/abs/2512.18470.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. SWE-Lancer: Can Frontier LLMs Earn $1 Million from Real-World Freelance Software Engineering? *arXiv preprint arXiv:2502.12115*, 2025. URL https://arxiv.org/abs/2502.12115.

Gabriel Orlanski, Devjeet Roy, Alexander Yun, Changho Shin, Alex Gu, Albert Ge, Dyah Adila, Nicholas Roberts, Frederic Sala, and Aws Albarghouthi. SlopCodeBench: Benchmarking How Coding Agents Degrade Over Long-Horizon Iterative Tasks. *arXiv preprint arXiv:2603.24755*, 2026. URL https://arxiv.org/abs/2603.24755.

Jeffrey Parsons and Chad Saunders. Cognitive Heuristics in Software Engineering: Applying and Extending Anchoring and Adjustment to Artifact Reuse. *IEEE Transactions on Software Engineering*, 30(12):873–888, 2004. URL https://doi.org/10.1109/TSE.2004.94.

Muhammad Shihab Rashid, Christian Bock, Yuan Zhuang, Alexander Buchholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, Anoop Deoras, Giovanni Zappella, and Laurent Callot. SWE-PolyBench: A Multi-Language Benchmark for Repository Level Evaluation of Coding Agents. *arXiv preprint arXiv:2504.08703*, 2025. URL https://arxiv.org/abs/2504.08703.

KN Ajay Shastry, Ganesh Senrayan, Shrey Satapara, Pranoy Panda, and Chaitanya Devaguptapu. Beyond Isolated Tasks: A Framework for Evaluating Coding Agents on Sequential Software Evolution. *arXiv preprint arXiv:2604.03035*, 2026. URL https://arxiv.org/abs/2604.03035.

Atharv Sonwane, Eng-Shen Tu, Wei-Chung Lu, Claas Beger, Carter Larsen, Debjit Dhar, Simon Alford, Rachel Chen, Ronit Pattanayak, Tuan Anh Dang, Guohao Chen, Gloria Geng, Kevin Ellis, and Saikat Dutta. OmniCode: A Benchmark for Evaluating Software Development Agents. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 40634–40661, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.2020. URL https://aclanthology.org/2026.findings-acl.2020/.

Shangyin Tan, Kevin Lin, Koushik Sen, and Matei Zaharia. Recovery-Bench: Evaluating Agentic Recovery from Mistakes. In *NeurIPS Workshop on Evaluating the Evolving LLM Lifecycle*, 2025. URL https://people.eecs.berkeley.edu/~matei/papers/2025/recovery_bench_neurips_workshop.pdf.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments. *Zenodo*, 2026.

Gladys Tyen, Hassan Mansoor, Victor Carbune, Peter Chen, and Tony Mak. LLMs Cannot Find Reasoning Errors, but Can Correct Them Given the Error Location. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. URL https://aclanthology.org/2024.findings-acl.826/.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida I. Wang, and Ofir Press. SWE-bench Multimodal: Do AI Systems Generalize to Visual Software Domains? In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.03859.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Aoyan Li, Lu Chen, Xiaojian Zhong, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Ming Ding, and Liang Xiang. Multi-SWE-bench: A multilingual benchmark for issue resolving. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-2111. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/5afa9cb1e917b898ad418216dc726fbd-Paper-Datasets_and_Benchmarks_Track.pdf.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. How Language Model Hallucinations Can Snowball. In *International Conference on Machine Learning*, 2024. URL https://proceedings.mlr.press/v235/zhang24ay.html.

Qixing Zhou, Jiacheng Zhang, Haiyang Wang, Rui Hao, Jiahe Wang, Minghao Han, Yuxue Yang, Shuzhe Wu, Feiyang Pan, Lue Fan, Dandan Tu, and Zhaoxiang Zhang. FeatureBench: Benchmarking Agentic Coding for Complex Feature Development. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=41xrZ3uGuI.

A BENCHMARK STATISTICS

This appendix expands the summary in Section 4; Table 2 lists the main statistics.

Coverage and provenance. SWE-Residual covers 90 of DeepSWE's 113 tasks and 21 of Terminal-Bench's 89 tasks. The DeepSWE tasks span 74 repositories and five programming languages: TypeScript, Go, Python, JavaScript, and Rust. The Terminal-Bench tasks span nine official categories, including software engineering, scientific computing, data science, security, and system administration. Both tracks contain states produced by all three predecessor families: DeepSeek, Qwen, and GLM. Of the 111 source tasks, 42 have one retained state, 51 have two, and 18 have three. Tasks with multiple states provide different inherited implementations or environments under the same original request and verifier.

Extent of prior work. The retained states reflect substantial prior attempts. The median predecessor run contains 134 agent steps on DeepSWE and 24 on Terminal-Bench, with interquartile ranges of 102–167 and 15–49, respectively. Median agent execution times are 47.7 and 15.5 minutes, including model and tool execution but excluding environment setup and verification. These measurements describe the prior attempts that produced the retained states.

DeepSWE states change a median of nine files, with medians of 1,164 added text lines and 10 deleted text lines. The median patch size is 44.6 KiB. These changes can include source code, tests, documentation, and configuration. In Terminal-Bench, inherited work can additionally include installed software and running processes; retained examples include a compiled application and running QEMU instances. The median prior attempt contains 45 terminal interactions, including polling and control inputs. The two tracks therefore capture both repository-level changes and work that extends into the machine environment.

Progress in Residual States. We characterize each inherited state by the fraction of the original verifier's acceptance tests that it passes before continuation. For DeepSWE, these include both target tests for the requested functionality and regression tests for existing behavior; for Terminal-Bench, they include all task acceptance tests. We compute the pass rate for each state and summarize these rates with equal weight across states. The mean and median are 91.6% and 98.4% on DeepSWE, and 47.6% and 60.0% on Terminal-Bench, respectively.

On DeepSWE, 88 of 165 states (53.3%) have one to three tests still failing, while 147 states (89.1%) preserve all tests that passed in the clean state. On Terminal-Bench, 25 of 33 states (75.8%) pass at least one test. Twenty-five states have one failing test and eight have two. Of the eight states that pass no tests, seven are evaluated by a single test, which provides no finer-grained measure of partial progress.

These results show that many inherited states already satisfy part of the original verifier, although every retained state still fails overall. The pass rates describe the current state rather than the fraction of work completed by the predecessor agent, since some tests already pass in the clean state. Test suites also differ in granularity and coverage, and even a single failing test may require substantial further work. Our evaluation examines whether agents can identify and complete the work that remains.

B RESIDUAL STATE CONSTRUCTION AND TRANSFER

This appendix describes how each Residual State is reconstructed from a predecessor run and what the evaluated agent can observe when it starts from that state. Under both Clean Start and Residual Start, the evaluated agent receives the same request, system prompt, harness, and budget; only the starting state differs.

B.1 DEEPSWE

During Stage I, we save all changes that the predecessor agent made relative to the task's base commit as a binary-safe patch, covering modified, added, and deleted files, including binary files. The patch retains everything the predecessor left in the repository, including build outputs, caches, and

logs; such incidental artifacts are ignored only when Stage II decides whether a run made substantive changes, and they are not removed from the reconstructed state. To reconstruct the state, we start from the task's original Docker image at the base commit and apply the patch to the working tree without creating a commit. The Git history therefore matches the clean state. The inherited changes appear as uncommitted modifications, visible through commands such as `git status` and `git diff`, and any commits the predecessor made during its run are flattened into these modifications, so their commit messages are not transferred. We impose no restriction on Git operations, so an evaluated agent can discard the inherited changes and reimplement the task from the base commit, within the same budget as any other strategy. None of the recorded DeepSWE continuations discarded the inherited work in its entirety.

Files that the predecessor wrote inside the repository, including its own tests, documentation, and notes, remain part of the Residual State, since they are part of the earlier work. The hidden tests used by the verifier are stored outside the repository and are never exposed to the evaluated agent. Tests written or modified by the predecessor remain in the working tree, and the evaluated agent may inspect or change them. Final outcomes are determined by the source benchmark's verifier and acceptance tests, independently of any tests modified by either agent.

B.2 Terminal-Bench

In Terminal-Bench, earlier work can change the whole machine, including files, installed packages, configuration, and running processes. During Stage I, we record the ordered sequence of terminal commands and interactions sent by the predecessor agent. To reconstruct the state, we replay this sequence in a fresh task environment and create a Docker checkpoint using CRIU[1], which preserves the container's filesystem and live process state. Each continuation restores an independent instance from this checkpoint. For states with a running QEMU instance, such as those described in Section 4, we verify after restoration that the VM process and task-relevant services remain operational.

The evaluated agent receives the original request and the restored machine state. Shell history and other artifacts remain accessible as part of that state; explicit handoff notes and verifier feedback are absent.

B.3 Validation

Each reconstructed state is validated before evaluation. A state is usable if it can be instantiated as an independent environment for a rollout: the harness can connect to it, the agent can execute commands and work on the task, and the original verifier can be run on the resulting state. We also check that the original verifier still fails on it, i.e., $V_i(q_i, S_{r,i}) = 0$. A failure must be a valid verdict, meaning that the verifier executed the tests and returned a result; a zero reward caused by an infrastructure or verification error is not treated as a failed attempt, either for a predecessor run or for a reconstructed state. Table 5 reports how many candidates were excluded at each step. Every rollout starts from an independent copy of the validated state.

Table 5: Reconstruction and validation of the candidates that passed human review.

	DeepSWE	Terminal-Bench	Total
Passed human review	170	37	207
Patch application or command replay failed	2	1	3
Environment unusable for rollouts	1	2	3
No valid verifier result, or no longer failing	2	1	3
Included in SWE-Residual	165	33	198

B.4 Information Available to the Evaluated Agent

Table 6 summarizes what the evaluated agent receives under each condition. The two conditions differ only in the starting state and in the earlier work it contains.

[1] `https://criu.org`

Table 6: Information available to the evaluated agent under Clean Start and Residual Start. Both conditions use the same request, system prompt, harness, and budget.

Item	Clean Start	Residual Start
Starting state	Original repository or machine	Residual State
Request	Original request	Original request
System prompt, harness, and budget	Source benchmark defaults	Source benchmark defaults
Earlier work inside the state	None	Retained, including code, tests, documentation, and notes written by the predecessor
Predecessor trajectory, summaries, or handoff notes	–	Not explicitly supplied; traces left in the state, such as shell history, may be accessible
Indication that an earlier attempt exists or failed	–	Not provided
Stage III review materials (trajectory summaries)	–	Not provided
Hidden tests and reference solution	Not provided	Not provided

C Evaluation Configuration and Metrics

C.1 Agents and Harnesses

Table 7 lists the models used as predecessor agents and as evaluated agents. Evaluated agents use mini-swe-agent on DeepSWE and Terminus-2 on Terminal-Bench, with all runs orchestrated through Harbor, following the evaluation setups of the source benchmarks. Predecessor agents use the same harnesses in Stage I. Evaluated agents run under the default per-run budget of each source benchmark, identical for Clean Start and Residual Start. Predecessor agents instead run under a generous time limit of eight hours per run, so that as many attempts as possible reach a submission.

We distinguish task failures from infrastructure failures. If an evaluated agent exhausts its budget without solving the task, the run counts as a task failure. If a run is interrupted by an infrastructure problem, such as a transient network, model-service, or container failure, we restart it from an independent instance of the same initial state. If verification fails for infrastructure reasons after the agent has finished, we rerun verification on the saved final state when possible and otherwise repeat the run from its initial state. A verifier error is never recorded as a task failure. The same policy applies to Clean Start and Residual Start.

Table 7: Models used in SWE-Residual. Predecessor agents produce Residual States in Stage I; evaluated agents are run from Clean Starts and Residual Starts.

Role	Model	Reasoning and sampling settings
Predecessor	DeepSeek-V4-Flash-0731	Highest reasoning effort; default sampling (temperature 1.0)
Predecessor	Qwen-3.8-27B	Highest reasoning effort; default sampling (temperature 1.0)
Predecessor	GLM-5.3	Highest reasoning effort; default sampling (temperature 1.0)
Evaluated	Gemini-3.8-Flash	Highest reasoning effort; default sampling (temperature 1.0)
Evaluated	Qwen-3.8-Flash	Highest reasoning effort; default sampling (temperature 1.0)
Evaluated	GLM-5.3-Flash	Highest reasoning effort; default sampling (temperature 1.0)
Evaluated	GPT-6-Sol	Highest reasoning effort; default sampling (temperature 1.0)
Evaluated	Kimi-K2.8-Preview	Highest reasoning effort; default sampling (temperature 1.0)

C.2 Metrics

Let $\mathcal{T}$ be the source tasks covered by a track and $N = |\mathcal{T}|$. For task i, let $\mathcal{R}_i$ be its set of Residual States, with $K_i = |\mathcal{R}_i| \in \{1, 2, 3\}$, and let $y_{A,i}(S) \in \{0, 1\}$ be the verifier outcome of a run of agent A from state S (Section 3.1). The Residual Start success rate averages outcomes within each

task and then across tasks, so that every source task receives equal weight:

$$\mathrm{Res}(A) = \frac{1}{N} \sum_{i \in \mathcal{T}} \frac{1}{K_i} \sum_{S \in \mathcal{R}_i} y_{A,i}(S). \tag{3}$$

The matched Clean Start reference uses the same tasks:

$$\mathrm{Clean}(A) = \frac{1}{N} \sum_{i \in \mathcal{T}} y_{A,i}(S_{0,i}). \tag{4}$$

The Residual Gap is $100 \cdot (\mathrm{Clean}(A) - \mathrm{Res}(A))$ percentage points, computed before rounding.

Conditional failure rate. Section 5.3 restricts the analysis to tasks that the agent completes from a Clean Start, $\mathcal{C}_A = \{i \in \mathcal{T} : y_{A,i}(S_{0,i}) = 1\}$, and reports

$$\mathrm{CondFail}(A) = \frac{1}{|\mathcal{C}_A|} \sum_{i \in \mathcal{C}_A} \frac{1}{K_i} \sum_{S \in \mathcal{R}_i} \big(1 - y_{A,i}(S)\big). \tag{5}$$

Because $\mathcal{C}_A$ depends on the agent, each agent is evaluated on its own task set.

Predecessor groups. Table 8 pools DeepSWE and Terminal-Bench and weights each source task equally, so the Overall column covers all 111 source tasks. For a predecessor family g, the group contains the tasks with at least one Residual State from g; within each such task, only the states produced by g are averaged, and the Clean Start reference is computed on the same task set. Groups therefore cover different tasks, and their absolute scores are not directly comparable.

D Measuring Acceptance Bias

The analysis in Section 5.4 is limited to DeepSWE, where per-test verifier outcomes, final patches, and reference solutions are available for every run.

D.1 Run Selection

Table 4 uses normally submitted runs, i.e., runs that end with the agent declaring the task complete and for which the verifier returns a valid result. These include both successful runs and valid failures, and exclude runs that exhaust the budget or end in an infrastructure failure. In the success rates of Table 3, runs that exhaust the budget count as task failures. Each metric is computed per run, averaged within each task, and then averaged across tasks. Steps is the exception and is reported as a median ratio.

D.2 Metric Definitions

For a state S, let $\mathcal{F}(S)$ denote the set of verifier tests that fail on S. A continuation from a Residual State S_r ends in a final state S' and is a failed continuation if the verifier outcome is 0.

Same failing. The fraction of failed continuations with $\mathcal{F}(S') = \mathcal{F}(S_r)$, i.e., that end with exactly the tests that failed in the Residual State.

Initially failing. For each Residual State S_r, the fraction of tests in $\mathcal{F}(S_r)$ that pass at the end of the agent's Clean Start run on the same task (Clean) and at the end of its continuation from S_r (Residual). This column covers all normally submitted runs, including successful ones; the repair rate reported in Section 5.4 is the same quantity restricted to failed continuations.

Kept. For a continuation from S_r, let P be the set of text lines that the predecessor added or replaced relative to the base commit; lines the predecessor deleted are not included. Every DeepSWE Residual State has a non-empty P. Kept is the fraction of lines in P that remain in the diff of the final state against the base commit, using exact, order-preserving line matching within each file. Table 4 reports Kept over failed continuations.

Left-out files. For a Residual State S_r, let L be the set of files changed by the reference solution but not by the predecessor. We consider states with non-empty L for which both the agent's Clean Start run and its continuation from S_r are normally submitted. Clean is the fraction of Clean Start runs that change at least one file in L; Residual is the fraction of continuations that do so. A continuation opens a file in L if one of its actions exposes the file's contents to the agent. This includes content-reading commands such as `cat`, `sed`, `head`, and `tail`, `grep` or `rg` output that shows file contents, and editor views recorded by the harness. Commands that only list paths, such as `ls`, `find`, `rg --files`, or `grep -l`, do not count. Neither do `git status` and `git diff --stat`; a `git diff` counts only if its output shows the file's contents.

Steps. For each continuation, the ratio of its step count to the step count of the agent's Clean Start run on the same task, where a step is one recorded agent turn, as in Table 2. We average the ratios over the Residual States of each task and report the median over tasks, using normally submitted continuations.

Inspection of inherited changes. A continuation inspects the inherited changes if it runs `git status` or `git diff`, or reads a file that the predecessor modified.

Split by Kept. To compare continuations within an agent, we compute Kept for every normally submitted continuation, including successful ones, and split them at the agent's median.

E Additional Results

E.1 Performance by Predecessor Agent

Table 8 reports the per-predecessor results discussed in Section 5.2: Residual Start and matched Clean Start success for each evaluated agent, overall and within the states produced by each predecessor agent.

Table 8: Performance by predecessor agent, pooling DeepSWE and Terminal-Bench. All values are success rates (%). Each group uses its own matched Clean Start reference, with each source task weighted equally. Overall includes all Residual States; the remaining groups include states produced by the indicated predecessor agent. DeepSeek, Qwen, and GLM denote DeepSeek-V4-Flash-0731, Qwen-3.8-27B, and GLM-5.3, respectively.

Evaluated agent	Overall		Predecessor agent					
			DeepSeek		Qwen		GLM	
	Clean	Residual	Clean	Residual	Clean	Residual	Clean	Residual
Gemini-3.8-Flash	67.6	18.9	62.0	19.0	68.7	16.9	44.4	5.6
Qwen-3.8-Flash	47.7	27.2	40.5	21.5	51.8	32.5	38.9	8.3
GLM-5.3-Flash	63.1	15.2	57.0	11.4	62.7	15.7	33.3	2.8
GPT-6-Sol	66.7	34.1	63.3	32.9	62.7	34.9	47.2	19.4
Kimi-K2.8-Preview	55.0	14.4	51.9	12.7	55.4	9.6	38.9	11.1

F Limitations

Our measurements cover 198 states from 111 tasks, two source benchmarks, and three predecessor families. Extending the collection to human-produced work and other handoff settings would show how the observed gaps vary with the source of inherited work. The acceptance-bias analysis uses DeepSWE, where per-test outcomes and final patches are available.