TERMINAL SELF-DEGRADATION: CODE AGENTS END WORSE THAN THEY REACHED

Anonymous authors
Paper under double-blind review

ABSTRACT

Code agents are scored on the patch they finally submit. That score hides a failure we can only see by executing what came before it: in a failed trajectory, an earlier executable state can pass a *strict superset* of the tests the submitted patch passes. We call this *terminal self-degradation* (TSD). Its strongest form, *solved-then-broke*, is a trajectory that reached a state passing the entire suite and then submitted one that does not. To measure it we replay every reconstructable intermediate state of fresh full-benchmark SWE-bench Verified rollouts through the official harness. A scaffold is admitted only if its reconstructed states pass a submission-time parity check, and a bash-only scaffold that fails it is excluded. On our primary cell (two frontier GPT-5.6 models on SWE-Agent), at least 4.9% of all failures are TSD. Nine are solved-then-broke: a harness-certified resolving patch was on disk and something else was submitted, and in seven of the nine the agent had itself evaluated that state before editing it away. It is *more pronounced* on six open-weight model lines (5.7–14.6% of all failures, every one above the frontier floor), and also holds on a second scaffold and eight non-Python language groups, surviving triple re-execution. Degradation proceeds mostly through `PASS_TO_PASS` regressions that a stable or rising pass count masks. The agents' own tests never separated the lost state from the submitted one. Most primary-cell losses are clean un-fixes that differ from the final only on the held-out target, so a reference-free selector built on repository tests recovers almost none of them. We establish TSD's existence, generality, and structure, report lower bounds rather than prevalences, and argue that the gap between failing to *generate* a fix and failing to *retain* one is a diagnostic axis that final-patch scoring, and any signal derived from it, cannot currently report.

1 INTRODUCTION

A code agent solving a GitHub issue (Jimenez et al., 2023) passes through many executable states. It edits, runs tests, edits again, and eventually submits. Benchmarks and leaderboards score one of those states, the last. That is a fine proxy if the final state is the best one the agent reached. We ask whether it is.

Replaying intermediate states through the benchmark harness, we find that often it is not. In a failed trajectory, an earlier working-tree state can pass a strict superset of the tests the final patch passes. The agent reached a strictly better-tested state and then edited away from it. We call this *terminal self-degradation* (TSD). Its strongest form, *solved-then-broke* (STB), is an intermediate state that passed the *entire* suite, a resolving patch, before the final regressed. On our primary cell (SWE-Agent, two frontier GPT-5.6 models, fresh full-500 SWE-bench Verified rollouts) TSD affects at least 4.9% of all failures ($15/305$, a lower bound; equivalently 10.4% of the 134 *assessable* via ≥ 2 executable checkpoints, Section 2.1). Nine of the 15 are solved-then-broke. A resolving patch was on disk and something else was submitted (Figure 1A). The fifteenth event, the one-checkpoint case `django-15554`, is TSD but not STB.

Throughout, a *checkpoint* is the working-tree state between two consecutive edit actions, executed by the harness exactly as the submitted patch is. It is a measurement unit rather than an agent decision (Section 2.1). TSD says that a strictly better-tested state *existed* on the agent's path and was not the one submitted, and we use "retention" and "loss" in this operational sense.

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.

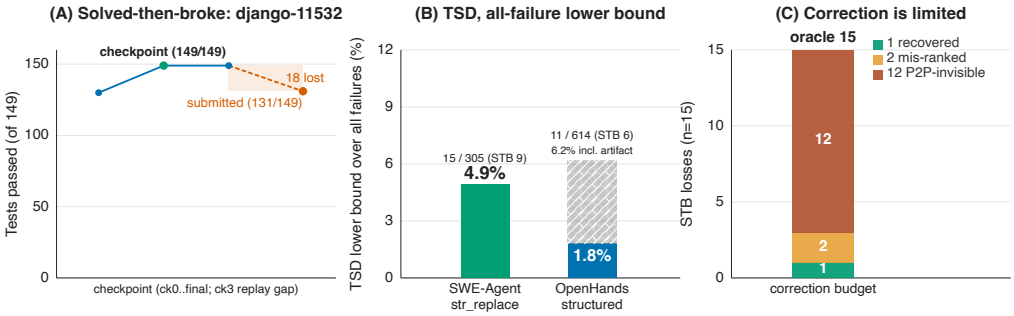

Figure 1: Terminal self-degradation: the phenomenon, its measured floor, and the limit of reference-free correction. **(A)** A real solved-then-broke case, `django-11532`: the trajectory rises to a full $149/149$ pass (`ck1`–`ck2`) and the submitted final regresses to 131 (`ck3` lost to a replay gap). **(B)** TSD over *all* failures, the only cross-scaffold-comparable quantity, from full-500 rollouts on the two fidelity-verified scaffolds (both lower bounds). SWE-Agent: $15/305{=}4.9\%$. OpenHands: the solid bar is the artifact-free $11/614{=}1.8\%$; the hatched extension to $38/614{=}6.2\%$ is 27 `sphinx-doc` events that are a confirmed patch-extraction artifact (Section 4.2). **(C)** The 15 solved-then-broke losses whose final scores above zero, split by what a reference-free selector can see (Section 6).

Why a small phenomenon matters. Measured against all instances TSD looks small: a floor of 4.9% of primary-cell failures, 1.5% of runs. The relevant comparison is different. On six open-weight lines the floor is 5.7–14.6% of all failures (Section 4.3), and even the frontier cell is not exempt: its nine solved-then-broke runs are instances the agent demonstrably solved, with a harness-certified resolving patch on disk, and then lost, 0.9 points of pooled Pass@1 given away. These failures differ in kind from the rest. The agent produced the solution and the harness certifies it, yet final-patch scoring, and any signal derived from it (a score, an RL reward, an SFT trajectory filter), cannot distinguish them from instances the agent never solved. It conflates failing to *generate* a fix with failing to *retain* one. We argue the two are worth separating.

What kind of failure is it? TSD rises with checkpoint count, as any event with more opportunities does. Successful runs pass through transient regressions at similar rates and recover; what distinguishes TSD is failure to recover. When a protected `PASS_TO_PASS` test breaks, a stable or rising visible pass count usually *masks* it (34 of 54). In none of the 14 primary-cell cases did the agent's own tests separate the lost state from the submitted one, and in six of the nine solved-then-broke cases its own test was *red* at the resolving state. The loss is an active over-edit that rewrites, extends, or reverts a state that already passed (Section 5).

Can it be recovered without the reference patch? An oracle freezing any fully-passing checkpoint would recover the 33 STB cases across both fidelity-verified scaffolds. Of these, 18 (all OpenHands sphinx) are a patch-extraction artifact (Section 4.2), which leaves 15 genuine losses whose final scores above zero as the hard ceiling. A reference-free selector that restores the checkpoint maximizing the repository's own `PASS_TO_PASS` tests recovers 1. The reason is that 12 of the 15 differ from the final *only* on the held-out `FAIL_TO_PASS` target: a clean un-fix, invisible to the repository's tests, to the agent's tests, and to any selector built on them. A preregistered target-aware signal recovers one more (Section 6).

Contributions. This is a measurement-and-diagnosis paper; we propose no new agent. We do not estimate a prevalence. 91 of 305 primary-cell failures could not be assessed, so every rate is a lower bound (Section 2.3), and the primary-cell characterization rests on 14 events, so its proportions are indicative rather than precise (Section 8).

- **An execution-authenticated evaluation axis.** We define TSD/STB as a strict-set endpoint on harness-executed checkpoints, establish it at $\geq 4.9\%$ of all failures on the primary cell with preregistered replication across six open-weight lines and eight non-Python languages, and show it is not harness nondeterminism, budget truncation, or packaging artifact (Section 4).

- **A fidelity requirement for checkpoint analysis.** A submission-time parity check separates faithful checkpoint reconstruction (native diffs; deterministic structured edits) from unfaithful (bash replay) and removes most of a bash-only scaffold's apparent TSD (Section 2.2); any checkpoint-based method (selection, verification, process reward) inherits this requirement.
- **A characterization and an observability boundary.** TSD is the non-recovered tail of ordinary edit volatility, driven by masked PASS_TO_PASS regressions that the agent's own tests never expose. On the primary cell most losses ($12/15$) are clean un-fixes invisible to any repository-test-based selector (the share is model-dependent across a wider census); a preregistered target-aware probe recovers one further case (Sections 5, 6).

2 Defining and Measuring Terminal Self-Degradation

2.1 The endpoint

Fix a benchmark instance with test universe $U = \texttt{FAIL_TO_PASS} \cup \texttt{PASS_TO_PASS}$. A trajectory produces a sequence of executable checkpoints $k = 1, \ldots, K$ and a submitted final; let $S_k \subseteq U$ be the set of tests passed when checkpoint k is applied to the base commit and executed by the official harness, and S_{final} likewise for the submission (an empty submission is scored at the base state).

Definition 1 (Terminal self-degradation)**.** A failed trajectory ($S_{\text{final}} \neq U$) exhibits *TSD* if $S_{\text{final}} \subsetneq S_k$ for some checkpoint k. It exhibits *solved-then-broke* (STB) if moreover $S_k = U$ for some such k.

Since the harness marks a patch *resolved* exactly when every FAIL_TO_PASS test passes and no PASS_TO_PASS test regresses, $S_k = U$ is equivalent to checkpoint k being a resolving patch, and a failed trajectory is one whose submission is not. The strict-set requirement is deliberately conservative: it excludes trade-offs (a checkpoint that passes more tests by count but misses one the final passes), equal counts with different passing tests, and transient regressions repaired before submission. A count-only relaxation adds only 6 cases on SWE-Agent ($10.4 \rightarrow 14.9\%$) and 1 on OpenHands, all genuine trade-offs (Appendix B.4); every number below uses the strict endpoint.

What a checkpoint is, and is not. A checkpoint is the working-tree state after each distinct edit action (Section 2.2). The endpoint uses the benchmark's held-out tests *retrospectively*. It is a measurement the agent never sees. A fix applied in three `str_replace` calls produces three checkpoints, so a fully passing state at call two may be a point the agent passed through on the way to a worse call three. We therefore report the per-case mechanism (rewrite, add, revert; Appendix Table 5) and, for the 9 solved-then-broke cases, whether the last fully-passing checkpoint was *completed* (the next action evaluated it: a test run, `git diff`, or `submit`) or *transitional* (the next action was another edit): 7 are completed and 2 transitional. Mostly the agent reached the solution, checked it, and edited it away. Over all 14 primary-cell events, 10 are completed and 4 transitional, so a *completed-only* endpoint, which admits a checkpoint only if the agent's next action evaluated it, still gives $10/134 = 7.5\%$ of eligible failures and 7 solved-then-broke; we report the full endpoint below because a transitional state is no less real to the harness, but the completed-only figure bounds how much of the phenomenon a stricter checkpoint semantics could remove.

Eligibility. Checkpoints $k = 1, \ldots, K$ are the *intermediate* (pre-submission) states; the final is compared against them and not counted among the K. A failed trajectory is *eligible* if it has $K \geq 2$ harness-executable intermediate checkpoints; this gate was fixed before the preregistered replications and is the denominator for conditional rates. It is stricter than the definition requires, so on the primary cell we also executed every one-checkpoint failure and count its events in the all-failure rate (Section 2.3).

2.2 Checkpoint fidelity: the submission-time parity check

A TSD verdict is only as trustworthy as the checkpoint it executes, so we admit a scaffold only if we can certify that its reconstructed states existed on the agent's working tree. **SWE-Agent** needs no reconstruction: with the `diff_state` bundle, every step persists the cumulative working-tree diff, and each distinct consecutive diff is a checkpoint. Reconstruction is exact, but execution can still fail: a persisted diff that produces no scoreable harness report (apply failure, error, or timeout) is dropped

from the checkpoint sequence rather than imputed (ck3 in Figure 1A is such a gap). **OpenHands** archives only condensed histories and finals, so we re-ran it from scratch (CodeActAgent, history condenser disabled) so that every edit action is persisted, replayed the saved edit actions inside each instance's container at base_commit, and captured the working-tree diff after each edit. Because structured file edits are deterministic functions of the action log, a faithful final replay implies faithful intermediate replays, provided the working tree is modified only through those edits (a bash action that writes a file which a later structured edit overwrites would leave the final intact but an intermediate state wrong). We therefore required, per instance, that the reconstructed final reproduce the harness pass-vector of the actually-submitted patch; mismatches were excluded, not counted. We audited the bash exposure on the OpenHands-SOL cell, whose full action histories we retain: across all 500 instances only 4 run actions could touch a tracked file, and just one (sympy-22080, a solved-then-broke event) unambiguously modifies one (git checkout - sympy/printing/. . .); the other three are sed edits to agent-created reproduce_* scripts or guarded no-ops. That case passed the parity check, so only its intermediate checkpoints carry residual risk.

LiveSWEAgent is bash-only, so its checkpoints must be reconstructed by replaying saved bash commands, which can depend on run-time state. Under the same parity check the reconstructed "final" typically passes 1–2 fewer tests than the true submission: of 29 apparent TSD events, 17 have a true submission that *resolved* the task, 6 more are empty or near-empty submissions reconstructed as non-empty, and the $\sim$6 remaining rest on *intermediate* checkpoints that bash replay cannot certify. We exclude LiveSWEAgent from every TSD number (Appendix B.1); the lesson generalizes: a final-state check certifies intermediates only if the edit primitive is deterministic.

2.3 Reading the rates: attrition and lower bounds

Table 1 traces the attrition from all runs to detected TSD. On the primary cell, 305 of 1000 runs fail; 80 have fewer than two executable checkpoints, 91 are technically excluded (89 non-empty finals whose patch applies but whose run errors, times out, or emits no report, plus 2 missing trajectories), 134 are eligible, and 14 exhibit TSD. The 80 sub-eligible failures split into 27 with *no* executable checkpoint (TSD impossible) and 53 with exactly one; we executed all 53 directly and only 1 (django-15554) exhibits TSD. The *measured* all-failure count is thus $15/305 = 4.9\%$, a lower bound and, because eligibility selects a different subpopulation on each scaffold, the only cross-scaffold-comparable quantity. If the 91 excluded failures exhibit TSD at anywhere from zero to the eligible-cell rate, the all-failure rate lies in $[4.9, 8.0]\%$; this is a sensitivity range, not a bound, because finals that apply but error or time out may well be *more* TSD-prone than eligible ones (the assumption-free worst case, $106/305$, carries no information about the true rate and we do not use it; Appendix F). For OpenHands (one-checkpoint failures not separately executed) the range is $[6.2, 26.2]\%$, wide because 240 of its 614 failures are replay-excluded (Appendix F). We therefore claim nothing about magnitude beyond these floors; Sections 4–6 concern existence, generality, and the structure of the events that do occur, none of which depends on the unassessed failures. (The cross-family panel applies the $K \geq 2$ gate only and lists this cell at $14/305 = 4.6\%$.)

Table 1: Attrition from all runs to detected TSD (full-500 SWE-bench Verified, pooled SOL+TERRA). "<2 ckpt": fewer than two executable intermediate checkpoints; "+1-ckpt": TSD found by executing one-checkpoint failures directly (SWE-Agent only); "Tech. excl.": replay failures, missing trajectories, finals without a harness report. The all-failure rate counts every excluded failure as a non-event and is the only cross-scaffold-comparable quantity; the conditional rate is survivor-conditioned and, for OpenHands, reported outside sphinx-doc (Section 4.2). OpenHands totals 999 runs (one TERRA instance produced no scoreable rollout).

Scaffold	Runs	Failed	<2 ckpt	Tech. excl.	Eligible	TSD	+1-ckpt	all-fail (LB)	cond.
SWE-Agent	1000	305	80	91	134	14	1	$\mathbf{15/305 = 4.9\%}$	10.4%
OpenHands	999	614	281	240	93	38	—	$\mathbf{38/614 = 6.2\%}$	16.9%[†]

[†]Outside sphinx-doc: $11/65$ over 7 repositories. The pooled conditional rate including sphinx-doc is $38/93$; we do not use it (Appendix B.2).

3 EXPERIMENTAL SETUP

Models. The primary cell uses two frontier models, GPT-5.6 SOL and GPT-5.6 TERRA, served through a hosted commercial gateway; we report each as a separate cell and, within a scaffold, their pooled rate, and never pool across scaffolds. Preregistered breadth panels use six further open-weight lines on SWE-Agent (Qwen3-Coder-30B, Qwen3-32B, devstral-2-123B, DeepSeek-V3.2, GPT-OSS-120B, Mistral-Large-3-675B) and three on OpenHands (DeepSeek-V3.2, Mistral-Large-3, GPT-OSS-120B). Decoding uses each scaffold's defaults; closed-API runs are not bit-deterministic, so we report per-cell comparisons and test harness nondeterminism directly (Section 4.5).

Scaffolds and harness. SWE-Agent 1.1.0 (Yang et al., 2024) with the `str_replace` editor and `diff_state` bundle (80-call budget); OpenHands v0.22.0 CodeActAgent (Wang et al., 2024) with the condenser disabled (100 iterations); LiveSWEAgent (Xia et al., 2025b) (mini-swe-agent 1.14.4, bash-only; excluded per Section 2.2). All checkpoints and finals are scored with the official SWE-bench Docker harness pinned to `swebench 3.0.15`, dataset revision `c104f840`, over the fixed `FAIL_TO_PASS`∪`PASS_TO_PASS` universe (Appendix G).

Benchmarks. SWE-bench Verified (Jimenez et al., 2023) (500 Python instances, 12 repositories) is the primary frame; SWE-bench Multilingual (300 instances, 37 eligible repositories, eight non-Python language groups) tests whether TSD is a Python artifact. An instance is resolved if the submitted patch passes all `FAIL_TO_PASS` tests without regressing any `PASS_TO_PASS` test.

Preregistration. The cross-family, OpenHands cross-family, multilingual, and correction-intervention analyses were preregistered with frozen SHA-256 hashes (`PREREG_*` files in the artifact); every cell that passed its technical smoke gate is reported regardless of outcome (Appendix C.2). Analyses labeled *post-hoc* were not preregistered.

4 HOW OFTEN DOES TSD OCCUR?

Table 2: TSD from full-500 fresh SWE-bench Verified rollouts with per-checkpoint harness authentication, on the two fidelity-verified scaffolds. n is the eligible denominator (failed finals with ≥ 2 executable checkpoints); STB is solved-then-broke; conditional rates are not comparable across scaffolds; CIs in text; the six preregistered open-weight cells are in Appendix Table 6. [§]27 of the 38 pooled OpenHands events are a `sphinx-doc` patch-extraction artifact (Section 4.2); figures including it are in Appendix Table 3 and not used.

Scaffold	Model	n	TSD	STB	Provenance
SWE-Agent	SOL	60	5 (8.3%)	4	full-500 fresh rollout
SWE-Agent	TERRA	74	9 (12.2%)	5	full-500 fresh rollout
SWE-Agent	**pooled (primary cell)**	**134**	**14 (10.4%)**	**9**	2 models, full-500
OpenHands	**pooled, excl. `sphinx-doc`[§]**	**65**	**11 (16.9%)**	**6**	2 models, full-500, 7 repos.

4.1 PRIMARY CELL: SWE-AGENT, FULL POPULATION

We ran fresh SWE-Agent rollouts over all 500 SWE-bench Verified instances for both models (Pass@1 $348/500 = 69.6\%$ SOL, $347/500 = 69.4\%$ TERRA; 695 of 1000 pooled) and executed every reconstructable checkpoint and every submitted final. Among failed finals with ≥ 2 executable checkpoints, TSD occurs in $5/60 = 8.3\%$ (SOL) and $9/74 = 12.2\%$ (TERRA); pooled, $14/134 = 10.4\%$ (Table 2; repository-clustered bootstrap 95% CI $[5.6, 15.1]$; Wilson $[6.3, 16.8]$). Nine of the 14 are solved-then-broke: an earlier checkpoint passes the *entire* test universe, so these nine runs (4 SOL, 5 TERRA) are 0.9 points of pooled Pass@1 that the agents reached and did not keep; under the completed-only endpoint of Section 2.1 the cell reads $10/134 = 7.5\%$ with 7 STB. An independent re-execution over an archived 250-instance-per-cell frame gives a concordant $18/164 = 11.0\%$, and leave-one-repository-out leaves the rate at $\geq 9.4\%$ (Appendix B.3); with the one-checkpoint event of Section 2.3, the measured all-failure rate is $15/305 = 4.9\%$.

The events are ordinary submissions rather than crashes: the TSD finals modify *source* files (plus 5 agent-created `reproduce_*.py` scripts; no test-file or build-configuration edits), and 13 of 14 produce a non-empty final at 14–36 steps, far under the 80-call budget (7 by explicit `submit`; 6 captured by the scaffold's format-error or step limit, for which "submitted" means "left on disk at termination"), while the fourteenth reverted its fix and exited with an empty submission, scored at base; neither test-file staging nor budget truncation explains the primary cell.

4.2 Second scaffold: OpenHands replicates the phenomenon, with one confounded repository

On the re-instrumented OpenHands population, TSD is detected in 38 of 93 eligible failures, but 27 of them are in one repository, `sphinx-doc`, and these are a confirmed patch-extraction artifact rather than a retention failure. Sphinx is the only SWE-bench Verified repository whose harness `pre_install` edits *tracked* files (`tox.ini`, `setup.py`); those edits remain uncommitted in the OpenHands runtime image, so the git-diff final carries them, fails to apply on the already-patched clean testbed, and scores 0, whereas checkpoints replayed from `base_commit` omit the hunk and pass. That is exactly why they read as STB. The 18 OpenHands STB cases whose final scores 0 are exactly the 18 sphinx STB cases, and the non-empty sphinx-TSD finals modify `conf.py`, build configuration, or test files (Appendix B.4). We confirmed the artifact by re-scoring: stripping the harness-injected `pre_install` hunk and running the agent's own `sphinx/` source edits through the harness resolves *all* 10 re-scorable SOL sphinx STB finals ($10/10$), so these are genuine Pass@1 successes the extraction corrupted, not retention failures. We therefore remove the sphinx events from the OpenHands failure set; the exclusion is conservative for both the TSD count and Pass@1. Outside `sphinx-doc`, TSD occurs in $11/65 = 16.9\%$ with 6 STB over 7 repositories (Wilson $[9.7, 27.8]$); the pooled figure including sphinx (40.9%) is recorded in Appendix Table 3 and not used in any claim. What OpenHands establishes is that TSD, including solved-then-broke, occurs on a second scaffold with a different edit primitive; its artifact-free all-failure floor is $11/614 = 1.8\%$ (Section 2.3).

The repository concentration is a property of the scaffold, not of GPT-5.6: three preregistered open-weight families through the identical pipeline are each modal in `sphinx-doc`, with sphinx-excluded conditional rates of 52.6% (DeepSeek-V3.2), 40.6% (Mistral-Large-3), and 8.8% (GPT-OSS-120B) (Appendix C.3). Because eligibility, repository mix, and budgets differ, we do not attribute scaffold differences to the edit interface.

4.3 Cross-family replication

We ran the identical full-500 SWE-Agent pipeline on six further open-weight lines, preregistered before scoring. All six show TSD: 15.0–29.1% of eligible failures and, on the cross-scaffold-comparable all-failure denominator, 5.7–14.6% of all completed failures. Every open-weight floor sits at or above the frontier cell's 4.6% under the same $K \geq 2$ gate, each with a repository-clustered lower bound above zero and robust to leave-one-repository-out (Appendix Table 6). The primary preregistered cell, Qwen3-Coder-30B, gives $34/227 = 15.0\%$ ($[10.5, 19.0]$; 7 STB; 9 repositories); two further full-500 rollouts of the same cell give 13.1% and 16.8%, so the rate is stable across stochastic rollouts. We read the open-weight rates as replication, not as an ordering of models: those cells fail more often and emit more multi-edit trajectories, so their eligible populations differ in ways we cannot match post hoc.

4.4 Cross-language replication

On SWE-bench Multilingual (Qwen3-Coder-30B, SWE-Agent, identical pipeline, preregistered), TSD occurs in $42/201 = 20.9\%$ of eligible failures (repository-clustered $[13.3, 28.9]$), with 11 STB, spanning 20 of 37 eligible repositories and eight language groups (Java 10, PHP 10, JavaScript/TypeScript 7, Ruby 5, C++ 4, Rust 3, Go 2, C 1); excluding the modal repository (`lombok`) leaves $34/190 = 17.9\%$. TSD is not an artifact of the Python-only Verified ecology; two further declared cells did not complete for infrastructure reasons and are disclosed, not scored (Appendix C.2).

4.5 Robustness of the endpoint

Re-executing all 51 TSD-positive checkpoint/final pairs from two cells (GPT-5.6-TERRA on Verified; Qwen3-Coder-30B on Multilingual) three times each reproduces the strict-superset relation in 50 of 51, including all single-test margins (the exception is a flaky Java test), so harness noise does not produce TSD. Leave-one-repository-out minima stay at 9.4% (GPT-5.6), 13.8% (Qwen3-Coder-30B), and 18.8% (Qwen3-32B) on SWE-Agent (Appendix Table 4), so it is not one repository, and Sections 4.1 and 4.3 rule out packaging artifacts and frontier idiosyncrasy.

5 What Kind of Failure Is TSD?

The endpoint says a better state existed. This section asks what the loss looks like. All analyses here are on the fidelity-verified cells; those marked post-hoc were not preregistered.

Opportunity rather than length. TSD incidence rises with checkpoint count ($1/32$, $3/52$, $10/50$ for two, three–four, five-or-more); more observed states create more detection opportunities, so this is not evidence that longer reasoning causes degradation.

The lost states are real solutions. By construction the STB subset has a checkpoint passing the entire suite, so those instances are solvable by the agent itself (9 of 14 on SWE-Agent; 6 genuine of 24 on OpenHands). The sibling model (SOL for TERRA and vice versa) resolves 5 of the 14 and 3 of the 38, most already STB, so the non-STB tail is only sparsely confirmed solvable and partly reflects genuine difficulty.

The loss is an active over-edit. Diffing each best checkpoint against its submitted final (Appendix Table 5), all 14 SWE-Agent TSD losses are *active* edits past a better state (one then exits with an empty submission); none is budget exhaustion, though in 6 the degraded state was captured by the scaffold's format-error or step limit rather than explicitly submitted, so "active" describes the edit that left the better state, not a decision to submit. Nine discard a *complete* resolving state (STB); five degrade a partial improvement. The degrading transition is a rewrite of an already-passing edit (5), an added edit that regresses previously-passing tests (4), or a revert of the good hunks (5); on the preregistered Qwen3-32B cell (58 TSD: 38 added, 13 rewrote, 7 reverted; none a truncation) *adding* is modal. Agents lose the state by continuing to edit it, not by giving up; rewrite cases are the most consistent with a transitional winning checkpoint, whereas add and revert cases return to a passing state and change it.

Degradation is masked; successful runs dip too, but recover. On an archived sample of 305 checkpoint-to-checkpoint transitions from 163 SWE-Agent GPT-5.6 trajectories (successful and failed), a previously passing `PASS_TO_PASS` test regresses in 54 (18%); 34 of those (63%) are *masked*: the visible net pass count holds or rises while a protected test breaks, so a "more tests pass" monitor would miss them. Resolved runs pass through a transient `PASS_TO_PASS` regression in $9/68$ (13%) and recover; failed runs in $23/80$ (29%), a gap not robust to matching on checkpoint count (Appendix B.4). Transient regression is ordinary edit volatility; what distinguishes TSD is that the agent did *not* recover, and we do not claim it is a distinct mechanism.

Did the agent test at the winning checkpoint? We parsed the agents' own test executions on the 14 primary-cell TSD trajectories. At the best checkpoint the agent's own test was *red* in 8 (6 of the 9 STB): it had written a reproduction or unit test stricter than, or different from, the held-out suite (e.g. a `repr` argument order in `scikit-learn-14983`), kept editing to satisfy it, and broke the held-out target. In 5 no test was run there. In the one green case (`django-13023`) the agent submitted the resolving patch, trimmed it during the post-submit review, and resubmitted a state on which the same module tests *also* passed; the regressed tests lie outside that module. In no case did the agent's own tests separate the lost state from the submitted one, in 8 of 14 they told it the correct state was wrong, and a *last-green* selector (freeze the last self-tested green state) recovers 0 of 14.

Relation to localization (post-hoc). On the preregistered Qwen cells, TSD is enriched among failures that reached and edited the reference-relevant code (Qwen3-32B 21% vs 9%, Fisher $p =$

0.0016, retained under checkpoint-count stratification; Appendix A.1), an association; we make no causal claim.

6 Can the Lost Solution Be Recovered Without the Reference Patch?

TSD is recoverable *in principle*: an oracle that froze any fully-passing checkpoint would recover the 33 STB finals across the two GPT-5.6 fidelity-verified cells (9 SWE-Agent, 24 OpenHands). Of these, 18 (all OpenHands, all sphinx) are the patch-extraction artifact of Section 4.2, whose final fails to apply and passes *no* tests. Stripping that harness-injected `pre_install` hunk and re-scoring the agents' own `sphinx/` edits resolves all 10 SOL cases we could re-score: they are recoverable Pass@1 the extraction destroyed, not retention failures, which both confirms the artifact and shows the recovery it hides is real. We therefore exclude them rather than count them as recoverable losses, leaving 15 finals that score above zero (14 with a genuine non-empty patch, one empty submission scored at base) as the hard ceiling (Appendix Table 8); the question is how much of it a *reference-free* method can reach.

A Pareto-safe selector recovers one. We evaluated an offline selector that, among all extracted checkpoints and the final, restores the state maximizing the repository's own `PASS_TO_PASS` (P2P) pass count, overriding the final only when P2P strictly increases. "Reference-free" means it never sees the reference patch or the held-out `FAIL_TO_PASS` tests; it does use the benchmark's `PASS_TO_PASS` list, which in deployment would be the repository's existing test suite. It is Pareto-safe: a resolved final already passes all P2P tests and is never overridden, so 0 are harmed. It reconstructs the resolving checkpoint for all 15 cases (0 replay gaps there) and recovers 1.

Most losses are invisible to the repository's tests. Of the 15, only 3 are P2P-*visible* (some checkpoint passes more P2P than the final): the selector recovers 1 and mis-orders 2. The remaining 12 are P2P-*invisible*: the resolving checkpoint and the failed final pass identical P2P and differ *only* on the held-out `FAIL_TO_PASS`. The accounting closes as $15 = 1 + 2 + 12$; the invisible share is $12/15$ (Wilson $[55, 93]\%$), and the two cells agree in direction ($6/9$ SWE-Agent, $6/6$ OpenHands). A clean un-fix is P2P-invisible by definition; the empirical content is that STB regressions *usually are* clean un-fixes rather than collateral breakage. `scikit-learn-14983` is the paradigm: on OpenHands-TERRA the resolving checkpoint and the final pass the identical 105 `PASS_TO_PASS` tests and differ only on the two held-out targets; the same instance is independently solved-then-broke on SWE-Agent-SOL with the same profile. The limit is therefore selector-independent: it binds any method built on `PASS_TO_PASS` and, by the last-green result, any method built on the agent's own tests, though not all signals. Invisibility recurs beyond GPT-5.6 across eight further model×scaffold×benchmark cells ($37/87$ combined; Appendix Table 9), but its share is model- and task-dependent, not a monotone function of capability: DeepSeek-V3.2 ($7/7$) and GPT-OSS-120B ($3/3$) on SWE-Agent are *fully* P2P-invisible like the frontier, whereas on the Qwen SWE-Agent cells the same selector recovers 9 of 12 because those finals break visible `PASS_TO_PASS` tests. What replicates is the recurrence of P2P-invisible losses; what does *not* is the frontier's high *share*.

Target-aware signals can see, but barely act (preregistered). An execution-feedback-refined procedure synthesizes a reproduction test from the issue text alone, iterates it against the buggy base until it fails for a bug-related reason (91% on-target, $84/92$ OpenHands failed multi-edit finals), and adds its verdict to the P2P score. It recovers exactly one more case, the OpenHands-TERRA loss on `scikit-learn-14983` (so the boundary is not absolute), harms none, but falls far short of its frozen $+15$ gate: on most failed finals ($37/49$ SOL) the refined test passes at *both* an earlier checkpoint and the final, which already satisfies the issue's surface behavior while failing the hidden target. A blinded two-model semantic guard over 305 transitions ranks regressions above chance (AUROC 0.701) but misses its preregistered top-10% capture gate ($6/26$) (Appendix D, Table 11).

What this implies. Acting on TSD divides at an observability boundary, and our results are mainly negative: the small visible part needs better checkpoint ranking, the large invisible part cannot be reached by any signal built from the repository's or the agent's own tests, and the one target-aware signal we tested recovers a single case, so we do not recommend a checkpoint selector. What the

results support is narrower: (i) final-patch scores cannot distinguish retention failures from generation failures, and reporting an STB count alongside Pass@1 costs only harness time on already-persisted diffs; (ii) any remedy must cover the held-out target (stronger oracles (Li et al., 2026), issue-derived tests (Mündler et al., 2024), or learned verifiers) and meet the fidelity requirement of Section 2.2; (iii) agents must retain executable checkpoints for it to act on.

7 Related Work

Trajectory analysis of code agents. Concurrent reference-patch-grounded analyses derive progress-critical actions for SFT curation (Ma et al., 2026a), train action-level process reward models (Dihan & Khan, 2026), or give function-level diagnostics (Shu et al., 2026); they assign progress by *proximity* to the reference patch, whereas we *execute* reconstructed intermediate states and score a strict test-pass superset, needing neither. AgentLens (Sahoo et al., 2026) studies "Lucky Pass," incidental success in passing runs, the complement of TSD (Appendix E).

Self-correction, overthinking, and best-of-N. TSD is the within-trajectory analogue of a documented failure of iterative refinement: without external feedback, self-correction of reasoning often degrades answers (Huang et al., 2024), and self-repair of code is weak unless fed a strong signal (Olausson et al., 2024); those studies measure an explicit "revise" step, whereas we measure the same direction of change over a tool-using agent's ordinary edit sequence, with the harness as arbiter. Cuadron et al. (2025) show that agents which reason more and act less resolve fewer issues; TSD is the complementary action-side evidence.

Selection, verification, and test-time compute. Best-of-N reranking with generated tests or learned verifiers (Xia et al., 2025a; Antoniades et al., 2025; Kim et al., 2026a) and self-correction under test-time compute (Stechly et al., 2025; Kumar et al., 2025; Snell et al., 2025) optimize the choice among candidates; TSD measures the cost of *not* choosing within a single trajectory, and Section 6 shows the same observability boundary applies to reranking its checkpoints as to reranking N finals. Concurrent work on test adequacy (Li et al., 2026) finds accepted patches exploit suite gaps, so a stronger oracle would render part of our invisible majority visible; structured edit interfaces (Kim et al., 2026b) and stateful architectures (Gautam et al., 2025) could change checkpoint semantics (Appendix E).

8 Conclusion

Final-patch scoring assumes the final patch is the best state a code agent reached. Executing intermediate states shows it is not always: in at least 4.9% of primary-cell failures, 5.7–14.6% across six open-weight lines, and on eight non-Python languages, the agent passed through a strictly better-tested state, usually a fully resolving one, and submitted something worse. The loss is the non-recovered tail of ordinary edit volatility, proceeds through masked regressions the agent's own tests never expose, and on the primary cell is mostly invisible ($12/15$) to any repository-test-based selector. Terminal self-degradation is thus a diagnostic axis, generation versus retention, that benchmarks cannot currently report and reference-free selection cannot currently close. The immediate implication is to report it alongside Pass@1; the longer-term one is that closing it requires target-aware verification, not better ranking of the states agents already produce.

Limitations

The characterization rests on 14 TSD trajectories and 15 losses, so the proportions in Sections 5–6 are indicative, and the invisible-majority finding is scoped to the primary cell. Prevalence is weakly identified: 91 primary-cell and 240 OpenHands failures could not be assessed and exclusion may correlate with TSD, so we report lower bounds only. The primary cell is closed-weight, most cells are single rollouts, and the cross-language result is one model on one scaffold (Appendix F).

AI USE STATEMENT

Generative AI was used to improve the grammar, clarity, and coding assistants. Where language models are part of the method itself (the semantic-guard critics and the synthesized reproduction tests of Appendix D), they are research instruments whose outputs are reported as model outputs, never as human labels. The research questions, study design, analyses, and conclusions are the authors' own decisions. The authors reviewed and verified all AI-assisted content and take full responsibility for the paper.

ETHICS STATEMENT

This work analyzes execution trajectories from existing code agents on public benchmarks (SWE-bench Verified and SWE-bench Multilingual; the appendix lens additionally uses SWE-PolyBench) derived from open-source GitHub repositories under their original licenses. No human subjects, personal data, or private repositories were involved; all trajectories were produced by automated agents in sandboxed environments. Two risks: the reference-free selector is exploratory and its net gain is near-null, so it should not be read as a deployable recovery method; and the reference-aligned diagnostics of Appendix A measure localization, not correctness, and could be gamed by matching reference trajectories rather than solving problems, so they should not be optimized against directly. We release the analysis pipeline as an evaluation tool, not a training signal.

REPRODUCIBILITY STATEMENT

The anonymized artifact (Anonymous, 2026) contains the code and data needed to reproduce our results: the per-instance TSD replay outputs and `reproduce_tsd.py`, which regenerate Table 2 under `swebench 3.0.15`, revision `c104f840`; the LiveSWEAgent submission-time parity check and per-instance verdicts (Section 2.2); the semantic-guard archive with per-checkpoint harness reports; the preregistration files with frozen SHA-256 hashes; the stage-extraction code and feature records of Appendix A; and the OpenHands re-instrumentation and gateway fixes. Configuration details are in Appendix G; the artifact's contents are enumerated in Appendix G.1.

REFERENCES

Anonymous. Anonymous artifact for trajeval. https://anonymous.4open.science/r/TRAJEVAL-C082/, 2026. Anonymous repository for double-blind review.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=G7sIFXugTX.

Islem Bouzenia and Michael Pradel. Understanding software engineering agents: A study of thought-action-result trajectories, 2025.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail?, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, Nicholas Thumiger, Aditya Desai, Ion Stoica, Ana Klimovic, Graham Neubig, and Joseph E. Gonzalez. The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks, 2025.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. Trail: Trace reasoning and agentic issue localization, 2025.

Mahir Labib Dihan and Md Ashrafur Rahman Khan. Swe-shepherd: Advancing prms for reinforcing code agents, 2026.

Dhruv Gautam, Spandan Garg, Jinu Jang, Neel Sundaresan, and Roshanak Zilouchian. Refactorbench: Evaluating stateful reasoning in language agents through code. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=NiNIthntx7.

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://openreview.net/forum?id=IkmD3fKBPQ.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2022.

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*, 2023.

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

Myeongsoo Kim, Joe Hsu, Dingmin Wang, Shweta Garg, Varun Kumar, and Murali Krishna Ramanathan. Codestruct: Code agents over structured action spaces, 2026b.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, JD Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=CjwERcAU7w.

Chenglin Li, Yisen Xu, Zehao Wang, Shin Hwei Tan, and Tse-Hsun Chen. Probe to generate: Program variant-guided test augmentation for repository-level repair benchmarks, 2026.

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

Wei Ma, Zhi Chen, Jingxu Gu, Tianling Li, Shangqing Liu, and Lingxiao Jiang. Same signal, different semantics: A cross-framework behavioral analysis of software engineering agents, 2026b.

Oorja Majgaonkar, Zhiwei Fei, Xiang Li, Federica Sarro, and He Ye. Understanding code agent behaviour: An empirical study of success and failure trajectories, 2025.

Niels Mündler, Mark Niklas Müller, Jingxuan He, and Martin Vechev. Swt-bench: Testing and validating real-world bug-fixes with code agents, 2024.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=y0GJXRungR.

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, 2025. URL https://arxiv.org/abs/2504.08703.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis, 2020.

Priyam Sahoo, Gaurav Mittal, Xiaomin Li, Shengjie Ma, Benjamin Steenhoek, Pingping Lin, and Yu Hu. Agentlens: Revealing the lucky pass problem in swe-agent evaluation, 2026.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Rui Shu, Chun Yong Chong, Xin Zhou, Yun Peng, Zihan Wu, Xu Han, Zeyang Zhuang, Guowen Yuan, and Yuan Wang. What resolve rate hides: Trajectory structure diagnostics for coding agents, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4O0v4s3IzY.

Chenyu Wang, Yunbo Lyu, Junda He, Zhou Yang, Chenxing Zhong, Yaniv Harel, and David Lo. Fail-fast, restart-smart: Early failure prediction and restart for swe agentic tasks, 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying llm-based software engineering agents. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025a. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754.

Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. Live-swe-agent: Can software engineering agents self-evolve on the fly? *arXiv preprint arXiv:2511.13646*, 2025b.

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. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems?, 2025.

A TRAJEVAL: THE REFERENCE-ALIGNED LENS THAT SURFACED TSD

TRAJEVAL is a training-free, reference-patch-aligned lens that maps each trajectory to stage-specific sets of files and functions and compares them with the reference patch. It is the diagnostic that first pointed us to retention failure: its *Edit-Quality residual* — failures that reach and edit the reference-relevant code yet still fail — is where execution-verified TSD is enriched (Appendix A.1). The TSD endpoint of the main text is authenticated by execution alone and does not depend on this lens; we summarize the lens only to the extent needed to read that linkage.

Stage recall. For a trajectory T producing candidate patch $\hat{P}$, the reference patch P^* defines the *golden context*: the files $\mathcal{F}^*$ and functions $\mathcal{H}^*$ it modifies (on average 1.4 files and 2.1 functions per instance). *Search* recall R_s is the fraction of $\mathcal{F}^*$ among the files viewed during T; *Read* recall R_r the fraction of $\mathcal{H}^*$ among functions appearing in agent observations; *Edit* recall R_e the fraction of $\mathcal{H}^*$ among functions modified in $\hat{P}$. Read measures exposure, not comprehension, and Edit measures *localization*: an agent may achieve perfect edit recall with an incorrect patch, so correctness is left to the harness. The strict reference (files and functions the reference patch actually modifies) is well-defined and benchmark-portable, and among 350 Verified instances resolved by two or more models, 93.7% show exact function-level convergence of the modified functions, so low recall against it rarely reflects a valid alternative fix.

Failure-mode partition. Each feature-extractable failed trajectory is assigned to one of four mutually exclusive stage-level failure modes by a cascading 0.5 threshold on recall: **Search** ($R_s < 0.5$), **Read** ($R_s \geq 0.5$, $R_r < 0.5$), **Edit-Target** ($R_r \geq 0.5$, $R_e < 0.5$), and **Edit-Quality** ($R_e \geq 0.5$: the agent reached and modified the reference functions but the patch is wrong). Edit-Quality is a recall-based localization label, deliberately not a precision claim. On a seven-model feature corpus (GPT-5, GPT-5-mini, Qwen3-8B/32B/235B, Qwen3-Coder-30B/480B; SWE-Agent, OpenHands, and LiveSWEAgent, the last retained here because stage-level analysis involves no checkpoint reconstruction; SWE-bench Verified and SWE-PolyBench Verified (Rashid et al., 2025); $16{,}121$ trajectories) Edit-Quality is the modal failure mode for every model resolving $\gtrsim 45\%$ of Verified (64–69% of SWE-Agent failures; robust to $\tau \in \{0.3, 0.5, 0.7\}$), while edit precision in those failures is near zero (median 0.026). The residual therefore says only that *reaching and touching* the reference code is not the bottleneck for capable models; we use it as a pointer to the population in which TSD is enriched, not as a result in itself.

A.1 LINKING THE EDIT-QUALITY RESIDUAL TO TSD

The Edit-Quality residual and TSD are produced by separate pipelines, so we test their association directly. This is an *exploratory, post-hoc* analysis (not preregistered; we reuse the preregistered Qwen cells' data). For each failed trajectory in the Qwen cross-family cells we join its TRAJEVAL bucket (function-level $\tau{=}0.5$ cascade) to its execution-verified TSD label. On the full-500 Qwen3-32B cell (372 failed trajectories with complete stage features), TSD incidence rises across the localization stages: Search $6/88{=}7\%$, Read $5/36{=}14\%$, Edit-Target $9/89{=}10\%$, Edit-Quality $34/159{=}21\%$. Edit-Quality is enriched relative to the other three combined (21% vs 9%; Fisher exact $p{=}0.0016$). This is not a checkpoint-opportunity artifact: Search trajectories carry the most checkpoints (mean 6.1) yet the least TSD, and a Cochran–Mantel–Haenszel test stratified by checkpoint count (2, 3–4, 5+) retains the association ($\chi^2{=}11.1$, $p{=}0.0009$), with Edit-Quality above the rest in every stratum (33% vs 21%; 12% vs 4%; 33% vs 13%). Qwen3-32B is a weak model on this benchmark ($\leq 15\%$ Pass@1); on the stronger Qwen3-Coder-30B cell the direction replicates (15% vs 11%) but is not individually significant ($p{=}0.44$). Pooling both Qwen SWE-Agent cells and stratifying by model, the enrichment is significant (Mantel–Haenszel $\chi^2{=}8.9$, $p{=}0.0029$). We cannot test the frontier GPT-5.6 line this way, because its release carries file-level recall labels rather than the function-level stage features the bucketing requires. The failures the lens flags as reaching and editing the reference-relevant code are therefore precisely those most prone to terminal self-degradation. This is an association within failed trajectories, not a causal claim; the CMH stratifies by checkpoint count but not repository, so we report an association rather than a clustered-adjusted effect.

B TSD MEASUREMENT: FIDELITY, ATTRITION, AND ROBUSTNESS

B.1 WHY LIVESWEAGENT IS EXCLUDED: THE SUBMISSION-TIME PARITY CHECK

Checkpoint replay fidelity differs by scaffold. SWE-Agent checkpoints are the per-step working-tree diffs the trajectory already persists (no reconstruction), and OpenHands checkpoints are validated per instance by a final-state harness pass-vector parity check (mismatches excluded; Section 2.2). LiveSWEAgent is bash-only: checkpoints are reconstructed by replaying the agent's saved bash commands in-container, and bash actions can depend on run-time state. We subjected it to the *same* submission-time parity check: for every candidate TSD/STB instance we re-scored the *actually-submitted* patch (from the rollout's `preds`) through the pinned harness (`swebench 3.0.15`) and compared its pass-vector to the reconstructed final. The reconstruction fails parity on the majority of cases: the replayed "final" typically passes 1–2 fewer tests than the true submission, and in most flagged instances the true submission actually *resolved* the task. Of the 29 apparent TSD events, 17 (10 TERRA, 7 SOL) have a true submission that resolved the task; 6 further are artifacts (empty or near-empty submissions the replay reconstructed as non-empty, plus one whose true final equals its best checkpoint); leaving $\sim$6 genuine solved-then-broke events. Even those rest on *unverified intermediate checkpoints*: the parity check certifies only the final state, and for a bash replay a correct final does not imply correct intermediates. OpenHands is retained because its structured edits are deterministic functions of the action log, so a faithful final replay implies faithful intermediate replays; its pipeline already excludes any instance whose reconstructed final mismatches the submitted patch's pass-vector, and a $10/10$ spot-check of its non-empty and empty STB finals confirms it. We therefore exclude LiveSWEAgent from all TSD and observability numbers and report this as a negative methodology result: a submission-time parity check is mandatory for any reconstruction-based checkpoint pipeline, and a final-state check does not certify intermediates unless the edit primitive is deterministic. The parity-check code and per-instance verdicts ship with the artifact.

Scope of the OpenHands bash-exposure audit. The bash-exposure audit of Section 2.2 (how often a `run` action modifies a tracked file, which a later structured edit could then overwrite and leave the final intact but an intermediate checkpoint wrong) is SOL-only, because only that cell's full action histories were retained; on SOL, 4 of 500 instances issue such an action and only one (`sympy-22080`, solved-then-broke) unambiguously modifies a tracked file, and it passed the parity check. Bash is available in every OpenHands cell through the same CodeActAgent interface, but how often a model uses it on tracked files is a model behavior, so we do *not* claim the SOL rate transfers to TERRA or the open-weight cells. What transfers is the guard: the per-instance submission-time parity check is applied to every cell, so it confines the residual fidelity risk to intermediate checkpoints of the (few) instances in which a bash action modified a tracked file that a later structured edit overwrote — the submitted final of any such instance is still certified against the harness pass-vector or the instance is excluded.

B.2 ATTRITION AND THE OPENHANDS FIGURES INCLUDING `sphinx-doc`

Table 1 in the main text gives the per-scaffold attrition from all runs to detected TSD. The OpenHands pooled conditional rate *including* `sphinx-doc` is $38/93 = 40.9\%$ (SOL $18/50 = 36.0\%$, Wilson $[24, 50]$; TERRA $20/43 = 46.5\%$, Wilson $[33, 61]$; repository-clustered bootstrap $[6, 72]$ and $[13, 76]$). We do not use this figure in any claim because 27 of the 38 events are in one repository where a packaging interaction is confounded with retention (Section 4.2, Appendix B.4); we report the rate outside that repository, $11/65 = 16.9\%$ (Wilson $[9.7, 27.8]$). The OpenHands run count of 999 is 500 SOL + 499 TERRA: one TERRA instance produced no scoreable rollout and is dropped under a pre-declared "no scoreable final" rule. The eligible subpopulation (failed final with ≥ 2 harness-executable intermediate checkpoints) is differently selected on each scaffold, so no conditional rate is comparable across scaffolds; the cross-scaffold quantity is the all-failure lower bound, which for SWE-Agent includes the one event found by executing the 53 one-checkpoint failures directly (`django-15554`; $15/305 = 4.9\%$). The cross-family panel of Appendix C.1 applies the $K \geq 2$ gate uniformly and therefore lists this cell at $14/305 = 4.6\%$. The sensitivity analysis for the all-failure rate is in Appendix F.

Table 3: OpenHands GPT-5.6 TSD figures *including* `sphinx-doc` (full-500 re-instrumented rollouts; eligibility as in Table 2). Recorded for completeness only: 27 of the 38 pooled events are in `sphinx-doc`, where a packaging interaction is confounded with retention (Section 4.2), so none of these rates is used in any claim.

Scaffold	**Model**	n	**TSD**	**STB**	**Wilson** 95%
OpenHands	SOL, incl. sphinx	50	18 (36.0%)	11	[24, 50]
OpenHands	TERRA, incl. sphinx	43	20 (46.5%)	13	[33, 61]
OpenHands	pooled, incl. sphinx	93	38 (40.9%)	24	—

B.3 Repository Robustness: Leave-One-Repository-Out

TSD events cluster by repository, so for each cell we recompute the TSD rate after removing, in turn, every repository, and report the *minimum* resulting rate (dropping the most TSD-enriched repository). Table 4 shows that the SWE-Agent endpoint is repository-robust across the GPT-5.6 and Qwen lines: the leave-one-out minimum stays at 9.4–18.8%. The OpenHands rate is the exception: on GPT-5.6-SOL it is `sphinx`-concentrated (36.0% → 11.1% dropping `sphinx-doc`; 14 of 18 events are `sphinx`), which is why we report per-cell conditional rates and never a pooled prevalence.

Table 4: Leave-one-repository-out (LORO) robustness. **LORO-min** is the TSD rate after removing the single most TSD-enriched repository.

Cell	Scaffold	Full	LORO-min	(dropped repo)
GPT-5.6 (sol+terra)	SWE-Agent	10.4%	9.4%	astropy
Qwen3-Coder-30B	SWE-Agent	15.0%	13.8%	django
Qwen3-32B	SWE-Agent	20.5%	18.8%	astropy
GPT-5.6 SOL	OpenHands	36.0%	11.1%	**sphinx-doc**

B.4 Opportunity, Solvability, Endpoint Sensitivity, and Mechanism

The analyses below are exploratory, *post-hoc* (not preregistered), on the full-500 GPT-5.6 (SOL+TERRA) per-instance replay results, using the same eligible populations as the main panel (134 SWE-Agent, 93 OpenHands).

Opportunity standardization. The raw per-instance TSD rate rises with checkpoint count (SWE-Agent $1/32$, $3/52$, $10/50$ for 2, 3–4, and 5+ checkpoints; OpenHands $17/52$ and $21/41$ for 3–4 and 5+; the OpenHands eligible set contains no exactly-two-checkpoint finals). Because more observed states create more detection opportunity, this rise cannot be read as a causal effect of trajectory length. We deliberately do *not* convert this into a per-transition rate: TSD is a trajectory-level event, so dividing events by transitions would mix observational levels and ignore within-trajectory dependence. We treat the conditional rates as opportunity-confounded and rely on the all-failure bounds for prevalence.

Solvability of TSD instances. By construction the STB subset (33 of the 52 TSD events on the two fidelity-verified scaffolds: SWE-Agent 9, OpenHands 24) has an earlier checkpoint that passes the *entire* suite, so those instances are provably solvable by the agent itself. For the non-STB events we check whether the sibling run (the other of SOL/TERRA) resolves the instance. Over all TSD events the sibling resolves $5/14$ SWE-Agent and $3/38$ OpenHands instances, most of which are STB events already solvable by construction, so the non-STB tail is only sparsely confirmed solvable by another run.

Endpoint sensitivity: strict-superset vs count-only. A weaker count-only criterion flags any checkpoint passing *more* tests than the final by count. On the full-500 GPT-5.6 cells, moving from strict-superset to count-only adds only 6/1 cases: SWE-Agent 14 → 20 (10.4 → 14.9%), OpenHands 38 → 39 (40.9 → 41.9%). The added SWE-Agent cases are genuine trade-offs (higher count but not

a superset), which we exclude by design; the conservative definition does not materially undercount the phenomenon.

Diff-level check: genuine regression vs packaging artifact. We classify each TSD event's final-patch files as source, test, or build/config. On the SWE-Agent cell, the 14 TSD finals touch *source* files and 5 agent-created `reproduce_*.py` scripts only (zero test-file or build/config edits), so an inadvertent `git add -A` of test/config is not the cause; 13 of 14 produce a non-empty submission at 14–36 steps, far under the 80-call budget (7 by explicit `submit`, 6 captured at the scaffold's format or step limit); the fourteenth, `sphinx-7454`, reverted its source edit and exited on error with an *empty* submission (its working tree held only a reproduce script, but nothing was submitted), scored at base ($27/28$). We retain it among the 15 ceiling losses (Table 8), whose criterion is a final scoring above zero (not non-emptiness); the one excluded Qwen empty-final STB and the 18 OpenHands artifacts all score 0. Budget truncation therefore does not explain them either. The OpenHands `sphinx` concentration, by contrast, is confounded: every *non-empty* GPT-5.6-TERRA sphinx-TSD final modifies `conf.py`/build config, the non-empty SOL sphinx-TSD finals are large diffs collectively including $\sim$1,600 test-file edits, and all 18 sphinx STB finals *fail to apply* and score 0: a patch-extraction artifact in which the submitted git-diff carries the harness `pre_install` edits to tracked `tox.ini`/`setup.py` (sphinx is the only Verified repository whose `pre_install` touches tracked files), which conflict on the already-patched clean testbed, while base-commit–replayed checkpoints omit the hunk and pass. The OpenHands sphinx concentration is therefore a confirmed extraction artifact, not a retention failure, reinforcing our use of SWE-Agent as the clean primary cell.

Per-case mechanism. Table 5 diffs the best checkpoint against the submitted final for each of the 14 SWE-Agent TSD cases. In all 14 the agent *actively* edits past a better state; none is a budget failure. The patterns are *rewriting* an already-passing edit (5), *adding* further edits that regress (4), and *reverting* the good hunks (5, one of which, `sphinx-7454`, then exited with an empty submission). The table also records whether the best checkpoint was *completed* (C: the agent's next action evaluated that state by a test run, `git diff`, or `submit`) or *transitional* (T: the next action was another edit), and what the agent's own test reported there. Over all 14, 10 are completed and 4 transitional; over the 9 STB cases, 7 and 2. At the best checkpoint the agent's own test was red in 8 cases (6 STB), absent in 5, and green in 1: in 13 of 14 the agent had no green signal at the state it lost, and where it had a red signal it was satisfying a self-authored test stricter than or different from the held-out suite and regressed the target while doing so. The one green case, `django-13023`, ran its module tests green (`Ran 11 tests ... OK`), submitted, then removed `ValueError` from the `except` clause during the post-submit review, ran the same module green again, and resubmitted the $9/11$ patch: the two regressed harness tests lie outside the module it ran, so a last-green selector recovers 0 of 14.

Cell	Instance	best ck	final	Mechanism	STB	ck position	own test at ck
SOL	`django-15098`	90	88	revert good hunks	✓	C	red
SOL	`matplotlib-25479`	263	262	add regressing edit		C	red
SOL	`matplotlib-25775`	96	84	add regressing edits	✓	T	none
SOL	`scikit-learn-14983`	107	105	rewrite passing edit	✓	C	red
SOL	`sympy-19495`	9	8	revert good hunks	✓	C	red
TERRA	`astropy-13398`	68	63	add regressing edits		T	none
TERRA	`astropy-14598`	176	174	rewrite passing edit	✓	C	red
TERRA	`django-11532`	149	131	rewrite passing edit	✓	T	none
TERRA	`django-13023`	11	9	revert good hunks	✓	C	green
TERRA	`django-14315`	9	7	revert good hunks		T	none
TERRA	`django-15098`	88	87	add regressing edit		C	red
TERRA	`sphinx-7454`	28	27[†]	revert, empty submission	✓	C	red
TERRA	`sympy-20801`	97	96	rewrite passing edit	✓	C	red
TERRA	`sympy-21930`	50	46	rewrite passing edit		C	none

Table 5: Per-case mechanism for the 14 SWE-Agent GPT-5.6 TSD cases: harness pass count at the best reconstructable checkpoint vs the submitted final, the classified degrading transition, the position of the best checkpoint (C = completed, T = transitional), and what the agent's own test reported at that state. For STB cases the best checkpoint is the last fully-passing one; for non-STB ties the latest is used. [†]Exited with an error and an empty submission, scored at base.

Masked regressions at degrading transitions. On an archived sample of 305 checkpoint-to-checkpoint transitions from 163 SWE-Agent GPT-5.6 trajectories (including successful ones), a previously passing PASS_TO_PASS test regresses in $54/305$ (18%) transitions. Of those 54, 34 (63%) are *masked*: the visible net test-pass count holds or rises even as a protected test breaks. Separately, 14 transitions flip a previously resolved instance to unresolved (a transition-level count, distinct from the final-level STB events), and 9 of the 14 do so through a PASS_TO_PASS regression. This probe covers SWE-Agent trajectories only and mixes successful and failed runs, so it characterizes regression dynamics generally rather than the TSD population specifically.

Base rate of transient regression in successful runs. On the same sample (148 transition-bearing trajectories), among runs that *end resolved*, $8/68$ (12%) pass through a transient drop in test-pass count and $9/68$ (13%) through a PASS_TO_PASS regression, then recover; among runs that *end failed*, $16/80$ (20%) and $23/80$ (29%). The gap is largely checkpoint-count-confounded: matched on checkpoint count, dip incidence rises with count in both outcomes and the resolved/failed gap is not robust (resolved/failed any-dip: 0%/12% at 2–3 checkpoints, 38%/31% at 4–5, 62%/38% at 6+; strata small, mean checkpoints 3.3 resolved vs 3.7 failed). The distinguishing feature of TSD is therefore *failure to recover* from the dip, not a higher dip incidence. (Resolved finals pass the entire suite, so they cannot themselves be strictly dominated; the comparison is on transient mid-trajectory dips.)

C Preregistered Replications

C.1 Cross-Family SWE-Agent Breadth Panel

We preregistered (PREREG_MULTIFAMILY_TSD, PREREG_XMODEL_TSD; frozen SHA-256) a breadth panel of six further open-weight model lines, using the *identical* full-500 SWE-Agent checkpoint-replay pipeline, and report every model that passed the technical smoke gate regardless of rate. Table 6 shows TSD in all six (15.0–29.1% over the $\geq$ 2-checkpoint eligible denominator, 5.7–14.6% of all completed failures), each with a repository-clustered lower bound above zero and robust to leave-one-repository-out. Two gateway-served models declared in the panel (`qwen3-coder-next`, `deepseek-r1`) were excluded before scoring under the technical gate (0 str_replace edit trajectories in the smoke; a scaffold-incompatibility exclusion, not a result-based one). Open-weight cells have larger eligible denominators than GPT-5.6 because they emit more multi-edit trajectories. Qwen3-Coder-30B was sharded; its count 34 uses a pre-fixed lower-shard-wins deduplication rule for 24 overlap instances and is robust to the tie-break (33–35).

Table 6: Preregistered cross-family SWE-Agent TSD breadth panel (full-500 SWE-bench Verified). **Elig.** = failed multi-edit finals with $\geq$ 2 executable checkpoints; **CI** is the repository-clustered bootstrap 95% (percentile; asymmetric because `django` supplies over half of the eligible instances at a below-average rate, so resamples that omit it run high); **LORO** is the leave-one-repository-out minimum; **all-fail** is TSD over all completed failures under the uniform $K \geq 2$ gate; for GPT-5.6 the denominator is the 305 failed runs of Table 1 (including 2 with missing trajectories, which changes the rate by $<$0.1 pp), giving $14/305$=4.6%, whereas the main-text figure $15/305$=4.9% additionally counts the one event found by executing the one-checkpoint failures (Section 2.3). All rates are per-cell (not pooled).

Model	Family	Elig.	TSD (rate)	repo-clustered CI	LORO	all-fail
GPT-5.6 (sol+terra)	OpenAI frontier	134	14 (10.4%)	[5.6, 15.1]	9.4%	4.6%
Qwen3-Coder-30B	Qwen	227	34 (15.0%)	[10.5, 19.0]	13.8%	13.9%
devstral-2-123B	Mistral (code)	164	27 (16.5%)	[10.9, 27.6]	14.9%	12.4%
DeepSeek-V3.2	DeepSeek	158	28 (17.7%)	[12.1, 23.8]	15.4%	14.6%
Qwen3-32B	Qwen	283	58 (20.5%)	[14.3, 40.6]	18.8%	13.4%
GPT-OSS-120B	OpenAI open	81	20 (24.7%)	[18.3, 40.5]	22.7%	5.7%
Mistral-Large-3-675B	Mistral	117	34 (29.1%)	[21.4, 40.0]	27.1%	8.3%

C.2 Cross-Language External Validity: SWE-bench Multilingual

We preregistered a second-benchmark replication (PREREG_MULTILINGUAL_TSD, frozen SHA-256 before scoring) on SWE-bench Multilingual with the identical SWE-Agent checkpoint-replay pipeline, strict-set endpoint, and per-checkpoint harness authentication; only the dataset and its language-specific harness images change. The preregistration declared three sentinel cells; we report each as-is.

Qwen3-Coder-30B (primary cell, complete). Over the 300-instance frame, Qwen3-Coder-30B resolves 68 (22.7% Pass@1). Of the 232 failed finals, 201 reach ≥ 2 harness-executable checkpoints. TSD occurs in $\mathbf{42/201 = 20.9\%}$ (Wilson 95% $[15.8, 27.0]$; repository-clustered bootstrap $[13.3, 28.9]$, seed 20260922), with 11 solved-then-broke. The 42 events span 20 of the 37 eligible repositories and eight language groups: Java 10 (lombok, lucene), PHP 10 (laravel, phpspreadsheet, php-cs-fixer, carbon), JavaScript/TypeScript 7 (preact, axios, docusaurus, immutable-js), Ruby 5 (jekyll, rubocop, fastlane, fluentd), C++ 4 (fmt), Rust 3 (tokio, coreutils), Go 2 (gin, hugo), and C 1 (redis). Excluding the modal repository (lombok, 8/11 eligible) leaves $34/190 = 17.9\%$. The rate is at or above this model's Verified SWE-Agent rate (15.0%, Table 6).

Incomplete declared cells (disclosed, not results). (i) *SWE-Agent* × *GPT-5.6*-SOL: the rollout did not complete (repeated Docker Hub pull failures for the language-specific eval images limited coverage to $\sim 130/300$), so no rate is claimed. (ii) *OpenHands* × *DeepSeek-V3.2*: OpenHands was not ported to the Multilingual harness and the pre-declared two-instance smoke gate was not run. Per the preregistration's anti-cherry-picking commitment, neither incompletion is presented as evidence for or against TSD.

C.3 OpenHands Cross-Family Panel

The high OpenHands TSD rate (GPT-5.6, 36.0–46.5% including sphinx-doc) could be a GPT-5.6 idiosyncrasy. We preregistered (PREREG_OH_XFAMILY_PANEL) three open-weight cells using the identical OpenHands re-instrumentation pipeline (CodeActAgent, condenser disabled, per-checkpoint action replay + harness authentication, strict-set TSD, ≥ 2-checkpoint eligibility), reported regardless of rate (Table 7). Every cell is modal in sphinx-doc, and the rates fall sharply when sphinx is excluded: 52.6% (DeepSeek), 40.6% (Mistral), and 8.8% (GPT-OSS-120B, whose OpenHands TSD is almost entirely sphinx: 11 of 14 events). Both the OpenHands elevation and its sphinx-doc concentration are thus cross-family properties of the scaffold, not GPT-5.6 artifacts. We report OpenHands conditional rates per cell and never pool them; the cross-family generality claim rests on the repository-robust SWE-Agent breadth panel.

Table 7: OpenHands cross-family TSD panel (SWE-bench Verified). The three open-weight cells are preregistered; each is modal in sphinx-doc, and the *excl. sphinx* column gives its rate on the remaining repositories. The GPT-5.6-SOL row is the frontier cell shown for reference; its pooled SOL+TERRA sphinx-excluded rate is 11/65=16.9% (Section 4.2).

Model	Family	Elig.	TSD (rate)	Wilson 95%	excl. sphinx
GPT-5.6-SOL	OpenAI frontier	50	18 (36.0%)	[24, 50]	—
GPT-OSS-120B	OpenAI open	45	14 (31.1%)	[19.5, 45.7]	3/34 = 8.8%
Mistral-Large-3-675B	Mistral	112	70 (62.5%)	[53.3, 70.9]	28/69 = 40.6%
DeepSeek-V3.2	DeepSeek	29	20 (69.0%)	[50.8, 82.7]	10/19 = 52.6%

D Correction Analyses

Reference-free offline selector. Among all extracted checkpoints and the submitted final, the selector restores the state maximizing the repository's own PASS_TO_PASS pass count, overriding the final only when strictly better with no PASS_TO_PASS regression (tie-break: latest). Reference-patch-free selection does not imply independence from benchmark test-selection information: the selector uses the benchmark-specified PASS_TO_PASS list while excluding the held-out FAIL_TO_PASS target.

Of the 33 STB across the two GPT-5.6 fidelity-verified cells, 18 (all OpenHands, all sphinx) are a patch-extraction artifact whose final fails to apply and scores 0 (Section 4.2); we exclude them, leaving 15 finals that score above zero (14 non-empty, one empty scored at base) as the ceiling. Both cells have verified checkpoints, so the selector reconstructs the resolving checkpoint for all 15 (0 replay gaps at the resolving checkpoint). Only 3 are PASS_TO_PASS-detectable: the selector recovers 1 (SWE-Agent) and mis-orders 2. The remaining 12 are P2P-invisible, differing from the final *only* on the held-out FAIL_TO_PASS; the accounting closes as $15 = 1 + 2 + 12$ ($12/15{=}80\%$, Wilson $[55, 93]$), and the two cells agree ($6/9$ SWE-Agent, $6/6$ OpenHands; Table 8). The 18 sphinx artifact finals are excluded from this ceiling (Section 4.2), not counted as losses the selector fails to recover.

Scaffold	STB (n)	artifact	score > 0	P2P-invisible	recovered
SWE-Agent	9	0	9	6	1
OpenHands	24	18	6	6	0
Total	33	18	15	12	1

Table 8: Composition of the 33 GPT-5.6 solved-then-broke (STB) cases on the two fidelity-verified scaffolds and the 15-instance oracle ceiling (finals scoring above zero). "artifact" finals (all OpenHands, all sphinx-doc) fail to apply and score 0 (a patch-extraction artifact; Section 4.2); "P2P-invisible" ceiling losses differ from the final only on the held-out FAIL_TO_PASS; "recovered" is the reference-free P2P selector's recoveries among ceiling losses.

Preregistered refined reproduction-test augmentation. We preregistered (PREREG_REFINED_REPRO, gate frozen before running) an execution-feedback-refined augmentation that synthesizes a gold-free reproduction test from the issue statement alone (GPT-5.6, no reference patch or benchmark tests), guarantees pytest in-container, and iterates the test against the buggy base checkout on its own pytest output until it fails for a bug-related reason. It then scores each state by (P2P pass count) + (1 if the synthesized test passes). Refinement raised the on-target rate to 91% ($84/92$ OpenHands failed multi-edit finals); on-target means the test fails on the buggy base for a bug-related reason, not that it reproduces the hidden FAIL_TO_PASS. (An earlier single-shot $0/112$ was largely a testbed pytest-invocation artifact: SWE-bench testbeds do not all ship pytest.) The augmentation recovered exactly one further instance over the P2P selector: the OpenHands-TERRA loss on scikit-learn-14983, whose resolving checkpoint and submitted final pass identical repository PASS_TO_PASS ($105/105$ each) and differ only on the 2 held-out FAIL_TO_PASS (the same instance is independently a solved-then-broke case on SWE-Agent-SOL, Table 5, with the same $107{\to}105$ profile). Of the 4 instances where the refined test passed at a checkpoint but failed at the final, 2 held a resolving checkpoint (scikit-learn-14983, sphinx-10323), but sphinx-10323 is one of the 18 sphinx-doc finals outside the 15-case ceiling (Table 8) and is not counted; the other 2 passed at non-resolving checkpoints (neutral, 0 false-freezes). Net $+1$, 0 harmed, below the frozen $+15$ gate. The refined test passes at *both* an earlier checkpoint and the submitted final on a majority of failed finals ($37/49$ SOL): the final already satisfies the issue's surface behavior while failing the hidden target.

Cross-family replication of the observability boundary. Applying the identical offline P2P selector to the retained per-checkpoint and final harness reports of the two Qwen SWE-Agent cells (no new harness runs): of the 12 non-empty solved-then-broke cases (7 Qwen3-Coder-30B, 5 Qwen3-32B; one empty-final case excluded), the selector recovers 9, and 3 are P2P-invisible (django-10880, django-7530, requests-2317); there are 0 mis-ordered cases. P2P invisibility is therefore not a GPT-5.6 artifact, but the proportions differ: the Qwen cases are dominated by finals that dropped some PASS_TO_PASS the selector can see. (The django-7530 classification is inferred from its universe sizes, $|\text{P2P}|{=}59$, $|\text{F2P}|{=}1$, and 59 passed tests, its final report not having been retained.) Table 9 consolidates the partition across further cells; the two Qwen SWE-Agent Verified rows (Qwen3-32B $1/5$, Qwen3-Coder-30B $2/7$) are the per-model view of the replication just described ($3/12$ combined), whereas the second Qwen3-Coder-30B row is the distinct Multilingual cell. Per-cell rates range 0–100% and the cells are small, so we report them separately; the size-weighted combined figure ($37/87{=}43\%$) spans clean-un-fix–dominated open-weight cells (DeepSeek-V3.2 and GPT-OSS-120B on SWE-Agent are *fully* P2P-invisible, $7/7$ and $3/3$, matching the frontier) and visible-regression–dominated ones (the Qwen SWE-Agent cells; Mistral on OpenHands). Invisibility

is thus severe on some open-weight lines too, not a GPT-5.6 artifact, but its share is model- and task-dependent rather than a monotone function of capability.

Table 9: P2P-invisible cases across further cells (resolving checkpoint and failed final pass identical PASS_TO_PASS). Reconstructable STB = non-empty solved-then-broke finals with a replayable resolving checkpoint.

Model	Scaffold	Benchmark	P2P-invisible / recon. STB
Qwen3-32B	SWE-Agent	Verified	1/5
Qwen3-Coder-30B	SWE-Agent	Verified	2/7
Devstral-2	SWE-Agent	Verified	2/3
DeepSeek-V3.2	SWE-Agent	Verified	7/7
GPT-OSS-120B	SWE-Agent	Verified	3/3
Qwen3-Coder-30B	SWE-Agent	Multilingual	5/11
Mistral-Large-3	OpenHands	Verified	17/47
GPT-OSS-120B	OpenHands	Verified	0/4
Combined			**37/87 = 43%**

Preregistered semantic checkpoint guard (did not pass its gate). We tested whether a target-aware but gold-free LLM critic could detect test-count regressions between consecutive checkpoints. Before generating any score, we froze a population of 305 consecutive test-evaluated transitions from 163 checkpoint-bearing SWE-Agent trajectories (11 repositories, SOL/TERRA); $26/305$ pass strictly fewer total tests than their preceding checkpoint. Each blinded critic input contains only the issue, previous and current cumulative patches, and the intervening agent actions/thoughts/observations — never the reference patch, harness outcomes, hidden tests, repository identity, or later states. Critics are Claude Sonnet 4.5 and Qwen3-Coder-480B at temperature zero, with their mean probability predeclared as an ensemble. The frozen gate required $\geq 95\%$ completion; AUROC ≥ 0.65 with repository-bootstrap lower bound > 0.50; AUPRC ≥ 0.20; top-10% capture $\geq 8/26$ with precision ≥ 0.20; and AUROC > 0.55 in both models. All three candidates complete $305/305$ rows; the ensemble clears the aggregate AUROC/AUPRC conditions but misses the top-budget capture condition (Table 10), so no candidate advanced to a prospective intervention. The predeclared PASS_TO_PASS-failure-increase endpoint is similar (ensemble AUROC 0.686, AUPRC 0.176, top-10% capture $5/22$). For auditability: 139 Claude responses hit the original 500-output-token cap before their final JSON and were retried, before labels were opened, at a 1,200-token cap with identical inputs; the first parseable response per row/model is retained.

Table 10: Preregistered semantic checkpoint guard on 305 blinded transitions (26 regressions). Intervals are repository-clustered bootstrap 95% CIs. **Top 10%**: regressions captured within the top-10% score budget ($\sim$30 transitions), with in-budget precision in parentheses; the frozen gate requires $\geq 8/26$ at precision ≥ 0.20.

Candidate	**AUROC [95% CI]**	**AUPRC**	**Top 10%**	**Gate**
Claude Sonnet 4.5	0.722 [0.586, 0.814]	0.181	5/26 (16.7%)	not passed
Qwen3-Coder-480B	0.643 [0.484, 0.813]	0.135	6/26 (20.0%)	not passed
Mean ensemble	0.701 [0.538, 0.856]	0.216	6/26 (20.0%)	not passed

E Extended Related Work

Process supervision and code agents. Outcome-only metrics (Pass@k (Chen et al., 2021), reference-similarity (Ren et al., 2020)) discard trajectory information, motivating process reward models (Wei et al., 2022; Huang et al., 2022; Schick et al., 2023). TSD is an evaluation-side counterpart: it scores the trajectory's states by execution rather than by proximity to a reference, and complements patch-derived trajectory selection/SFT (Ma et al., 2026a) and action-level PRMs (Dihan & Khan, 2026). Agentless (Xia et al., 2025a) uses a fixed localize→repair→validate pipeline; our search/read/edit decomposition (Appendix A) is applied descriptively to any trajectory and is not prescribed as an architecture.

Table 11: Denominators and per-cell results for every tested correction. Populations differ by design and are reported per-intervention; none is pooled across scaffolds.

Intervention	Population (denominator)	Result
Reference-free P2P selector (offline)	Solved-then-broke finals scoring above zero on the two fidelity-verified scaffolds; 18 sphinx artifact finals (apply-fail) excluded	Recovers 1 (Pareto-safe, 0 harmed); $12/15$ ceiling losses are P2P-invisible; oracle upper bound 15
Refined gold-free reproduction-test augmentation (*preregistered*)	92 of the 93 OpenHands failed multi-edit finals (49 SOL+43 TERRA)	91% on-target ($84/92$); $+1$ over P2P (`scikit-learn-14983`), 0 harmed; $<$ frozen $+15$ gate
Semantic guard (*preregistered*)	305 evaluated transitions, 26 test-count regressions, two critics	AUROC 0.701, AUPRC 0.216; $6/26$ at top-10% budget $<$ frozen $8/26$ gate; prospective run not executed

Failure attribution, restart, and test generation. Mündler et al. (2024) benchmark issue→test synthesis, the signal family our refined-reproduction probe draws on. Zhang et al. (2025) attribute failure across agents in multi-agent systems; Wang et al. (2026) pair early failure prediction with a restart policy, the deployable counterpart to our diagnosis. Broader taxonomies (MAST (Cemri et al., 2025), TRAIL (Deshpande et al., 2025)) catalog failure *types*; we isolate one code-repair phenomenon grounded in test execution.

Test-suite strengthening and structured edit interfaces. Concurrent work on benchmark test adequacy (Li et al., 2026) augments SWE-bench suites with program-variant-guided probes and finds that accepted patches often exploit test-suite gaps. This directly complements our observability boundary: a stronger oracle would render part of the P2P-invisible majority visible; how much is due to the oracle gap rather than intrinsic is an open question a variant-augmentation study on our invisible cases could answer. Structured, AST-level edit interfaces (Kim et al., 2026b) replace brittle string edits with syntax-validated transformations and could reduce regressive edits; we do not test this interaction.

Trajectory and behavior analysis. Bouzenia & Pradel (2025) analyze thought-action-result patterns at the action-type level; Majgaonkar et al. (2025) study SWE-bench failures via outcome statistics; Ma et al. (2026b) study cross-framework rule transfer. None executes intermediate states. Concurrent work by Sahoo et al. (2026) introduces "Lucky Pass" for incidental success; TSD captures the complementary event in which test-verified progress is later lost.

F Full Limitations

Logs capture actions, not reasoning. Our trajectories record the agent's actions and observations, not its internal reasoning. The mechanism analyses characterize *what* an agent did (which edits it made and destroyed), not *why*; we make no claim about intent behind a degrading edit, and a checkpoint is a measurement unit rather than a candidate the agent weighed (Section 2.1).

Harness nondeterminism is not the source of TSD. We re-executed all 51 TSD-positive checkpoint/final pairs from two cells (GPT-5.6-TERRA on Verified; Qwen3-Coder-30B on Multilingual) three times each. In 50 of 51 pairs the strict-superset relationship reproduces in all three runs, including all three cases whose margin is a single test; the lone exception is a flaky Java repository test.

TSD rates are conditional on survivors, and exclusion could correlate with the outcome. The per-scaffold TSD denominators are failed finals reaching ≥ 2 harness-executable checkpoints. A large minority of failures are excluded beforehand (for OpenHands, 240 of 614 failures because of replay failures or missing/unexecutable finals). We report per-cell conditional rates and detected lower bounds against all failures because we cannot rule out that the excluded runs differ in TSD

propensity from the retained ones; the rate magnitudes are weakly identified and should not be read as population prevalences. *Sensitivity analysis.* Failures reaching < 2 executable checkpoints (80/281 for SWE-Agent/OpenHands) split into those with *no* executable checkpoint (27/220; TSD impossible) and exactly one (53/61). For SWE-Agent we evaluated all 53 single-checkpoint failures directly: only 1 (`django-15554`) exhibits TSD, giving a measured all-failure count of $15/305{=}4.9\%$, a lower bound. Treating the remaining 91 technically-excluded failures at zero-to-eligible-rate propensity gives an all-failure rate in $[4.9, 8.0]\%$. This is an assumption-based sensitivity range, not a bound: the 89 no-report finals are patches that apply but error or time out ("patch does not apply" yields a report and is scored at base, so it is not among them), and they are heavily `django`-skewed (50 of 89, versus 41 of 149 in the assessed population; `sphinx-doc` 9, `sympy` 8, `astropy` 6, `matplotlib` 5, `scikit-learn` 4, five other repositories 7), consistent with large test suites erroring or timing out under the harness; such a population may be *more* TSD-prone than eligible failures, and the assumption-free worst case is $106/305{=}34.8\%$. For OpenHands we did not separately evaluate its 61 single-checkpoint failures, so counting them with the 240 technically-excluded at zero-to-eligible-rate propensity gives $[6.2, 26.2]\%$ (worst case $339/614{=}55.2\%$), its width dominated by replay attrition. The existence, generality, and observability-boundary results are unaffected by this conditioning: generality concerns whether the loss occurs at all in each family, and the boundary analysis is an information property of the specific 33 STB finals (and 15 non-empty subset) that do occur.

Other limitations. The primary cell is closed-weight and served through a non-deterministic gateway, so its headline numbers are not bit-reproducible (the open-weight cells are); most cells are single rollouts; OpenHands is `sphinx-doc`-confounded; the cross-language result is one model on one scaffold; the OpenHands bash-exposure audit (Appendix B.1) covers only the SOL cell, whose full action histories were retained; and the reference-aligned lens of Appendix A measures localization only, so two agents editing the same function with different patches receive identical stage metrics.

G Reproducibility

G.1 Artifact Contents

The anonymized artifact (Anonymous, 2026) contains: the per-instance TSD replay outputs (checkpoint and final test-pass counts, TSD/STB verdicts) and `reproduce_tsd.py`, which regenerates Table 2 and the preregistered cross-family SWE-Agent cell; the LiveSWEAgent submission-time parity check and per-instance verdicts (Appendix B.1); the preregistration files with frozen SHA-256 hashes; the semantic-guard archive (preregistration, reconstructed candidates, 305 evaluated transitions, labels, two-model scores; seed 20260915); the stage-extraction code and the 16,121-trajectory feature records of Appendix A; per-checkpoint `FAIL_TO_PASS`/`PASS_TO_PASS` reports for the replayed populations; and the OpenHands re-instrumentation and LiteLLM gateway fixes. Our generated trajectories, features, replay labels, and code are provided under a permissive open-source license, subject to the upstream benchmark and repository licenses they derive from.

G.2 Frontier (GPT-5.6) Configuration

Models and inference endpoint. We use two variants of a frontier OpenAI-family reasoning model, `gpt-5.6-sol` and `gpt-5.6-terra`, served through a hosted commercial inference gateway addressed via LiteLLM. Running them required four provider-specific adjustments in each scaffold's LiteLLM call, without which every request errors or silently no-ops: (i) `litellm.drop_params=True`; (ii) strip `cache_control` (top-level and nested `content[]` entries) for non-Anthropic providers; (iii) omit `temperature`/`top_p` when the model id contains `gpt-5` or `openai.` (the reasoning model rejects them, and `drop_params` alone is insufficient because LiteLLM lacks the schema for these custom ids); (iv) add `additional_drop_params=['stop','stopSequences']` for these ids (the gateway rejects the `stopSequences` field). Sampling is left at each scaffold's default with temperature dropped as above; agents are stochastic, so per-instance outcomes vary across runs.

Scaffolds and versions. Table 12 lists the exact scaffold versions and configurations.

Benchmark and evaluation harness. SWE-bench Verified (`princeton-nlp/SWE-bench_Verified`, 500 instances; dataset revision `c104f840`). All

Scaffold	Version / config / hyperparameters
SWE-Agent	1.1.0; config `config/benchmarks/250522_anthropic_filemap_simple_review.yaml` (tool bundles: `edit_anthropic`, `review_on_submit_m`, `diff_state`, `filemap`); `per_instance_call_limit=80`; LiteLLM 1.99.0.
OpenHands	v0.22.0-2496 (CodeActAgent); `max_iterations=100`; history condenser disabled; Poetry 2.1.3 (2.4.x mis-parses the lock); per-instance runtime image build; `EVAL_SKIP_MAXIMUM_RETRIES_EXCEEDED=true`; the (i)–(iv) LiteLLM fixes applied in `openhands/llm/llm.py`.
LiveSWEAgent	mini-swe-agent 1.14.4; config `livesweagent_swebench.yaml`; `model_kwargs={}`, `temperature` removed, `drop_params=True`; runner `python -m minisweagent.run.extra.swebench`. Excluded from TSD numbers (Appendix B.1).

Table 12: Scaffold versions and configurations for the GPT-5.6 study.

checkpoints and finals are scored with the official SWE-bench Docker harness pinned to `swebench==3.0.15` in a dedicated environment separate from the rollout environment (SWE-Agent 1.1.0's default `swebench==5.0.2` fails with `KeyError: 'image'` on Verified). Pass@1 is resolved$/500$; unresolved and incomplete both count as failures. Evaluation command template:

```
python -m swebench.harness.run_evaluation \
  --predictions_path <preds.json> \
  --run_id <tag> \
  --dataset_name princeton-nlp/SWE-bench_Verified \
  --instance_ids <ids> --namespace swebench \
  --max_workers 6
```

Checkpoint replay applies each reconstructed cumulative working-tree diff inside the instance's SWE-bench Docker image at `base_commit` and runs the same harness; states are keyed by model, instance, checkpoint index, and patch hash.

Compute and concurrency. Rollouts ran on general-purpose cloud VMs (4–16 vCPU class; 400 GB–4 TB SSD depending on scaffold; OpenHands per-instance runtime images require the larger disks). High aggregate concurrency ($\sim$96 simultaneous gateway calls) triggers rate-limiting/timeouts and truncates completion; we mitigate with low per-endpoint concurrency (num_workers 3–6), spreading across two gateway endpoints, and an on-box completion-retry loop that re-runs patch-less instances. Harness evaluation is CPU/Docker-only (no model calls). The feature-corpus models (GPT-5 family via the OpenAI API; Qwen3 via OpenRouter; both through a LiteLLM client; default decoding; analysis seed 42) are unchanged from Appendix A.