# Can LLMs Detect Benchmark Defects? A Meta-Benchmark from Benchmark Updates

Jongwon Park [1]

## Abstract

Every major coding benchmark has released updated versions acknowledging substantial defect rates, yet quality assurance remains largely manual. We propose Task Verification Bench, a meta-benchmark that repurposes author-acknowledged defects from benchmark updates as ground truth and evaluates whether LLM agents can detect them given only the original artifacts. Because a defect diagnosis must be actionable to guide a fix, we evaluate with root-cause matching — requiring the agent to identify the correct defect, not just flag one — which reveals that verdict-level recall overestimates true detection by up to 54pp. A GPT-5.4+Codex verify-only baseline achieves 35–60% root-cause-matched recall across SWE-bench and two non-overlapping splits of Terminal-Bench — meaningful for triage, but far from fully automated auditing. Pipeline ablation uncovers self-attribution anchoring in GPT-5.4 — harness access causes the agent to blame its own code rather than the benchmark — while Gemini tends to resist this effect, showing that model choice matters more than pipeline design. These results suggest that LLM-assisted auditing can meaningfully reduce human review effort for text-layer defects, while infrastructure-layer verification remains a challenge requiring human expertise.

## 1. Introduction

Code generation benchmarks play a central role in assessing progress in LLM-based software engineering. However, these benchmarks are themselves software artifacts subject to bugs. Notably, every major coding benchmark has released a v2 that acknowledges defects in v1. SWE-bench Verified (Chowdhury et al., 2024) was created after OpenAI audited 1,699 SWE-bench (Jimenez et al., 2024) tasks with 3 independent annotators per task, filtering out 1,160 (68%) as defective. Terminal-Bench (Merrill et al., 2026) collected 229 community-contributed tasks and curated 89 into Terminal-Bench 2 through multi-stage review — three human reviewers per task, LLM-assisted audits, and adversarial testing.

Yet defects survive curation. When frontier models saturated SWE-bench Verified, auditing the remaining failures revealed that 59% stemmed from flawed tests rather than model limitations (OpenAI, 2025). Terminal-Bench 2 required 28 post-release fixes (31%) despite its extensive review process. This is not a lack of effort but an inherent difficulty of manual auditing at scale (Cao et al., 2025). As benchmarks continue to grow in both volume and complexity (Deng et al., 2025; Chu et al., 2026; Yang et al., 2024; Badertdinov et al., 2026a;b; Rank et al., 2026; Elfeki et al., 2026), this gap will only widen, making automated auditing not just useful but increasingly necessary. LLM coding agents have grown capable enough to fix real-world bugs—can they also detect benchmark defects?

We introduce Task Verification Bench (TVB), a meta-benchmark that repurposes these benchmark updates — defects acknowledged by benchmark creators themselves — as ground truth to evaluate LLM verification agents. We curate labels from two benchmark families — SWE-bench (129 defect tasks after filtering for $\geq 2/3$ annotator agreement and excluding build failures) and Terminal-Bench, split into the curated 89-task subset TB2 (26 defect tasks) and the remaining non-overlapping TB1 tasks (60 defect tasks) — spanning patch-based and interactive evaluation paradigms (Section 3.1). Given only the original problem statement, test suite, and gold patch, the agent must determine whether each task contains a defect. We evaluate with root-cause matching rather than binary defect detection: an agent that flags a task as defective but for

[1]Delphik. Correspondence to: Jongwon Park <jongwon.park@posttrain.dev>.

Preprint. May 20, 2026.

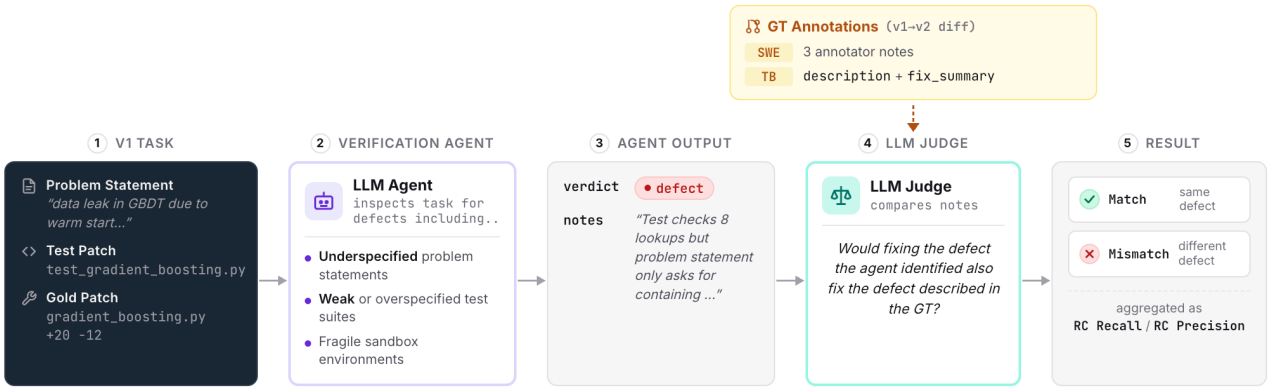


Figure 1. Task Verification Bench evaluation pipeline. (1) The agent receives only the original v1 task artifacts: problem statement, test patch, and gold patch. (2) A verification agent inspects the task for benchmark defects such as underspecified problem statements, weak or overspecified tests, and fragile environments. (3) The agent outputs a binary verdict with free-text notes explaining the identified defect. (4) An LLM judge compares the agent's notes against GT annotations derived from the benchmark's v1→v2 version diff, asking whether fixing the agent's identified defect would also fix the GT defect. (5) The result is either a root-cause match or mismatch, aggregated into RC recall and RC precision.

the wrong reason provides no actionable signal for fixing it.

A verify-only baseline with GPT-5.4 achieves 35–60% root-cause-matched recall across all evaluation sets — meaningful for triage but insufficient for automated auditing; verdict-level recall overestimates performance by up to 54pp, confirming that root-cause matching is essential (Section 4.1). The agent excels at text-layer defects (all 6 sampled TPs) but exhibits an infrastructure-layer blind spot: in all 8 sampled TB miss/mismatch cases involving environment issues or Dockerfile leakage, the agent either misses the defect entirely or identifies a plausible but incorrect text-layer issue instead (Section 4.1). Pipeline ablation on SWE-bench uncovers self-attribution anchoring in GPT-5.4: when solving and verifying with harness access, the agent attributes test failures to its own implementation rather than the benchmark — an effect Gemini resists; model choice ultimately matters more than pipeline design (Section 4.2). Manual verification of 77 cases validates judge accuracy and reveals structural limitations of version-diff ground truth, including over-labeling in annotation-based GT and under-labeling in PR-based GT (Section 4.3).

## 2. Related Work

Benchmark Defects Are Endemic. Benchmark quality problems span domains and decades. Northcutt et al. (2021) found at least 3.3% label errors across 10 widely-used CV, NLP, and audio test sets, showing that even ImageNet contains ∼6% errors that destabilize model rankings. In LLM evaluation, Gema et al. (2025) re-annotated 5,700 MMLU questions and estimated a 6.5% error rate, with some subsets reaching 57%. Code generation benchmarks are no exception: Siddiq et al. (2024) found spelling errors, unclear intent, and improper documentation across 3,566 prompts from 9 function-level benchmarks — surface-level defects that differ from the structural issues (underspecified requirements, weak tests, oracle bugs) we study in task-level benchmarks like SWE-bench and Terminal-Bench. A survey of 572 code benchmarks (Cao et al., 2025) found that test coverage neglect is growing as benchmark production scales, suggesting the problem is worsening rather than self-correcting. The ABC checklist (Zhu et al., 2026) evaluated 10 agentic benchmarks and found that 7 violate task validity and 7 violate outcome validity, with relative performance estimation errors up to 100%. BenchJack (Wang et al., 2025a) found all 8 audited coding benchmarks exploitable, with agents achieving 73–100% scores through trivial means. Most of these efforts either propose guidelines for building better benchmarks (Joaquin et al., 2025) or report how many defects are there, but none measure how reliably LLMs can detect defects in existing benchmarks — the question we address.

Automated Benchmark Auditing. Recent work shows that LLMs can find benchmark defects that human reviewers miss. In non-coding domains, automated auditing is an active area: BenchGuard (Tu et al., 2026) found 12 author-confirmed defects in ScienceAgentBench despite prior expert review, and ELT-Bench-Verified (Zanoli et al., 2026) showed that benchmark quality issues underestimate agent capabilities in data

pipelines — with the same defect categories (rigid tests, ambiguous specs) we observe in coding benchmarks. McAleese et al. (2024) addressed data quality for RLHF: they inserted synthetic bugs into ChatGPT code responses and trained LLM critics to detect them. We address the analogous problem for RLVR, where coding benchmarks serve as reward signal: instead of synthetic bugs, we repurpose defects documented during benchmark updates and measure how reliably LLMs can detect them. AutoTriage (Hu et al., 2026) automates failure attribution (task defect vs. agent error) for RL training data curation, notably finding that the benefit of execution access is model-dependent, a result we corroborate in our harness ablation — but focuses on classifying failures rather than proactively detecting defects. Our work bridges this gap: we directly audit coding benchmark tasks against held-out ground truth from benchmark updates, enabling measurement of detection reliability rather than relying on self-discovered GT or single-benchmark annotations.

Defect Ground Truth. Several studies have uncovered defect evidence in both patch-based (Aleithan et al., 2024; Wang et al., 2025b; Yu et al., 2025; Li et al., 2026) and interactive benchmarks (Bercovich, 2026); we use their findings for valid-set curation (Section 3.3) but they are difficult to repurpose as held-out evaluation GT. Success-case analyses (Aleithan et al., 2024; Wang et al., 2025b) have found widespread evaluation weaknesses in SWE-bench, but only cover tasks where a model already passes. Test-strengthening approaches (Yu et al., 2025; Li et al., 2026) have broader coverage — STING finds weakness in 77% of Verified instances — but flag weakness broadly without per-task defect labels suitable for evaluation. Our approach addresses both gaps: it produces author-acknowledged defect labels that require no passing patches, no test generation, and no self-discovered GT. Complementary work on test augmentation (Liu et al., 2023; Yu et al., 2026; Harman et al., 2025; Chen et al., 2026) strengthens existing tests rather than detecting which tasks are fundamentally broken; our work asks the prior question.

## 3. Task Verification Bench

### 3.1. Ground Truth from Benchmark Updates

Our key insight is that benchmark updates produce defect labels as a byproduct of maintenance — annotations, PR descriptions, and fix summaries written by benchmark authors and contributors, not for evaluation purposes. When benchmark authors release v2, each modified task comes with documentation of what was wrong in v1. We collect labels from three GT sources across two benchmark families (Table 1).

Table 1. Ground truth sources from two benchmark families. TB2 is a curated subset of the Terminal-Bench repository; TB1 comprises the remaining non-overlapping tasks.

| Source | Method | Defect / Total | Defect rate |
|---|---|---|---|
| SWE-bench | 3-annotator audit | 1,160 / 1,699 | 68.3% |
| TB2 | Community fix PRs | 26 / 89 | 29.2% |
| TB1 | Community fix PRs | 60 / 141 | 42.6% |

Table 2. SWE-bench defect type distribution (multi-label, $N = 1{,}160$).

| Defect axis | Prevalence |
|---|---|
| false negative (FN, valid solution rejected) | 95.3% |
| underspecified (US, problem statement insufficient) | 88.4% |
| both (FN + US) | 83.9% |
| other major issues | 11.6% |

SWE-bench. OpenAI hired software engineers to audit 1,699 SWE-bench tasks with 3 independent annotators per task. Each task received structured scores from each annotator for underspecification (0–3), false negative likelihood (0–3), and other issues (0–1). 1,160 tasks were flagged as defective. The annotation is multi-label: 84% of defective tasks are flagged for both false negatives and underspecification (Table 2).

Terminal-Bench 2. 26 of 89 tasks were revised in a consolidated PR, aggregating community and author fixes with per-task defect descriptions. We use an LLM (Claude Opus 4.7) to classify each defect into six types based on the PR description and diff; a single task may have multiple defect types when the PR addresses several issues.

Terminal-Bench 1 (non-overlapping split). TB2's tasks were selected from a broader repository of 229 contributed tasks; 88 of 89 overlap. We use the 141 non-overlapping tasks as a separate evaluation set, applying community-contributed PRs as GT. 96 PRs fixed 106 tasks across the full repository; after restricting to TB1-only tasks and excluding two bulk-fix PRs for timeout and path normalization, 60 defect tasks remain. These are the most organic labels — contributors submitted "fixes" without framing them as defect reports. We apply the same LLM-based classification to each PR's diff and description, extracting multi-label defect types per task (93 labels across 60 tasks).

Table 3 shows the Terminal-Bench defect type distribution. Environment issues and instruction underspecification are the two dominant categories across both splits.

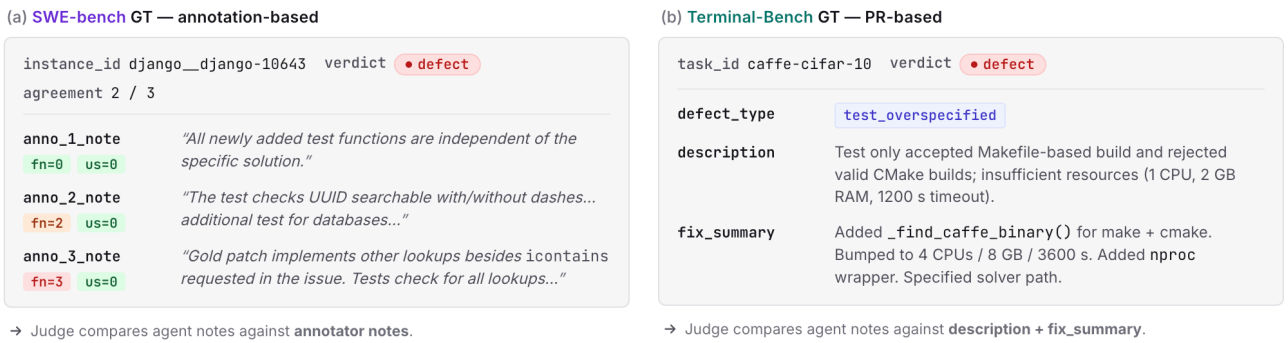


Figure 2. Representative ground-truth entries from the two benchmark families used in this work. (a) SWE-bench GT is assembled from multiple human annotators per task; each annotator independently rates false negative likelihood (fn) and under-specification (us) on a 0–3 scale and writes a free-form note. (b) Terminal-Bench GT is derived from the upstream pull request that fixed the defect, yielding a defect type, description, and fix_summary. Both formats feed the same downstream LLM judge, which compares agent notes against these GT annotations.

Table 3. Terminal-Bench defect types (multi-label; % = fraction of defect tasks exhibiting each type, so columns sum to >100%).

| Type | TB2 ($n$=26) | TB1 ($n$=60) | Total ($n$=86) |
|---|---|---|---|
| environment issue | 16 (61.5%) | 18 (30.0%) | 34 (39.5%) |
| instruction underspecified | 10 (38.5%) | 26 (43.3%) | 36 (41.9%) |
| test overspecified | 7 (26.9%) | 15 (25.0%) | 22 (25.6%) |
| information leakage | 2 (7.7%) | 16 (26.7%) | 18 (20.9%) |
| oracle bug | 4 (15.4%) | 6 (10.0%) | 10 (11.6%) |
| test underspecified | 0 (0%) | 8 (13.3%) | 8 (9.3%) |

From these raw sources, we construct per-task GT entries consisting of a binary defect/valid label and structured defect descriptions that can be compared against agent outputs (Figure 2). For SWE-bench, each GT entry contains the annotators' free-text notes for each defect axis; for Terminal-Bench, it contains the defect type, description, and fix summary extracted from the PR. These descriptions serve as the reference for root-cause matching, which we define next. Details of the parsing pipeline are in Appendix C.

### 3.2. Evaluation

We evaluate at two levels of granularity.

Verdict-level (binary). Each task receives a binary verdict (defect/valid). We report defect recall, precision, and F1 score.

Root-cause match. A correct verdict with the wrong reasoning is insufficient — if the agent flags a defect for the wrong reason, the diagnosis cannot guide a fix. We require the agent's identified root cause to match the ground truth annotation; a verdict-level TP where the root cause does not match is reclassified as a false negative. Since both SWE-bench and Terminal-Bench tasks can have multiple co-occurring defects, the agent need only match any one GT defect label to count as a root-cause match. Formally, root-cause (RC) recall = RC match / total GT defect, and RC precision = RC match / all predicted defect, where the denominator includes both mismatches and false positives. RC metrics are upper-bounded by verdict-level metrics since RC matches $\subseteq$ verdict TPs.

LLM Judge. For each verdict-level TP, an LLM judge compares the agent's defect notes against the GT annotations to determine whether they identify the same benchmark problem. For SWE-bench, the judge compares against the annotators' free-text notes, with the test patch as context. For Terminal-Bench, the judge compares against the defect description and fix summary, with the task instruction and test code as context. The core criterion is: would fixing the defect the agent identified also fix the defect described in the GT? Same root cause described differently counts as a match; entirely different benchmark issues count as a mismatch. We use GPT-5.4 full at temperature = 0, selected over mini after manual verification of 10 disagreement cases: full was correct in 8, mini in 1, and 1 was borderline (Appendix A). Three-run agreement was 97.3% on SWE-bench, so we adopt single-run evaluation for efficiency.

### 3.3. Curation

SWE-bench (258 tasks: 129 defect + 129 valid). We initially curated diverse subsets by severity, difficulty, and defect type, totaling 434 tasks for pilot experiments (Appendix B). Pilot analysis revealed that tasks with only 1-of-3 annotator agreement contained severe GT noise — 91% of false-negative verdicts were GT noise rather than agent errors — so we introduced a $\geq$2/3 annotator agreement filter, removing 169 low-

agreement defect labels. After excluding 41 tasks with Docker build failures, 129 defect tasks remain. For valid tasks, we sample 129 from the Verified pool (filter_out=False), excluding tasks flagged by PatchDiff (Wang et al., 2025b) or UTBoost (Yu et al., 2025) for test weakness.

TB2 (52 tasks: 26 defect + 26 valid). All 26 defect tasks from PR #53, with multi-label extraction (39 defect labels across 26 tasks). 26 valid tasks randomly sampled from the remaining tasks.

TB1 (74 tasks: 37 defect + 37 valid). From the 60 TB1-only defect tasks, we extract multi-label defect descriptions (93 labels across 60 tasks). Since community PRs sometimes fix minor quality improvements rather than genuine defects — e.g., reformatting output or tightening a timeout that was already sufficient — we used Claude Code + Opus 4.7 to assess whether each GT label reflects a real task defect on a 5-point confidence scale; noise tasks were confirmed after manual inspection and excluded. After excluding Docker build failures, 37 defect tasks remain. Valid tasks are randomly sampled from the TB1-only pool to match defect count, yielding 74 evaluable tasks (37 defect + 37 valid).

## 4. Results

We ablate across three axes: model choice, pipeline design, and harness access. Our primary metric is RC recall — the fraction of GT defects for which the agent identifies the correct root cause — since a benchmark audit must surface actionable defects, not just flag tasks as problematic. We use GPT-5.4 and Gemini 3.1 Pro as verification models to test whether findings generalize across model families.

Pipeline design. In verify-only (VO), the agent receives all task artifacts and directly renders a defect/valid verdict. In solve-then-verify (STV), the agent first attempts to solve the task with only the problem statement (Phase 1), the host runs the benchmark's test harness (Phase 2), and the agent then verifies using the test results, gold patch, and its solve experience (Phase 3). Phases 1–2 always use GPT-5.4 with Codex CLI in Docker, regardless of the ablation condition; only Phase 3 varies across experiments. The STV design is motivated by AutoTriage (Hu et al., 2026), which showed that failure attribution is more accurate when grounded in execution experience — an agent that has attempted the task may better diagnose why a test is unfair. Pipeline and prompt details are in Appendix E.

Harness access applies to the verification phase (Phase 3) only. With harness (H), the verifier runs inside a Docker sandbox with full repository access via a model-specific agentic CLI — Codex CLI for GPT-5.4 and Gemini CLI for Gemini 3.1 Pro.[1] Without harness (NH), all task artifacts — problem statement, test patch, gold patch, solve-phase agent patch, and Phase 2 test output — are inlined in a single API call. NH necessarily omits the full repository source code, so defects that require reading specific implementation files may be structurally undetectable in this mode. Since H requires per-task Docker containers and agentic CLI execution while NH is a single API call, determining when harness access is worth the cost is a practical question for large-scale benchmark auditing.

Table 4. Cross-benchmark baseline results (GPT-5.4 + Codex, verify-only with harness). RC = root-cause-matched via LLM judge (GPT-5.4 full, $t = 0$).

| | SWE-bench | TB2 | TB1 |
|---|---|---|---|
| Type | Patch-based | Interactive | Interactive |
| Tasks (def./valid) | 258 (129/129) | 52 (26/26) | 74 (37/37) |
| Verdict-level | | | |
| Recall | 62.0% | 88.5% | 94.6% |
| Precision | 72.7% | 54.8% | 52.2% |
| F1 | 0.669 | 0.677 | 0.673 |
| Root-cause-matched | | | |
| RC recall | 59.7% | 34.6% | 48.6% |
| RC precision | 70.0% | 21.4% | 26.9% |
| RC F1 | 0.644 | 0.265 | 0.346 |
| Verdict→RC gap | 2pp | 54pp | 46pp |

Section 4.1 presents cross-benchmark baselines with VO-H. Section 4.2 ablates all three axes on SWE-bench. Section 4.3 validates GT quality through manual verification.

### 4.1. Cross-Benchmark Baseline Results

Table 4 presents baseline results using GPT-5.4 in verify-only mode with harness access. We use VO-H as the common baseline across all benchmarks: TB defects frequently involve environment issues, oracle bugs, or information leakage that require Docker execution to observe, and applying the same condition to SWE-bench ensures cross-benchmark comparability.

Qualitative analysis of 18 TP and FN cases (6 per benchmark, stratified by error category; details in Appendix) reveals consistent strengths and two failure patterns.

[1]GPT-5.4 via Azure API (gpt-54); Gemini 3.1 Pro Preview via Vertex AI (gemini-3.1-pro-preview).
Harness: Codex CLI @openai/codex v0.41, Gemini CLI @google/gemini-cli v0.41.
NH calls use Azure Responses API and Vertex GenAI respectively.

What the agent gets right. Across all three benchmarks, the agent excels at static contract analysis — comparing instruction text against test assertions and oracle logic to identify explicit mismatches. On SWE-bench, it reliably catches test-overfit-to-gold-patch defects where tests verify behavior absent from the problem statement. On Terminal-Bench, it catches instruction-test mismatches such as unstated API constraints and format incompatibilities.

What the agent misses. The same analysis reveals two failure patterns: (1) Subtle semantic gap rationalization (SWE): the agent accepts tests that check a different context or abstraction level as "reasonable generalizations" of the problem statement. (2) Infrastructure-layer blind spot (TB): environment issues, Dockerfile leakage, and oracle regressions require longer reasoning chains than text comparison. The split is sharp: all 6 TP cases across three benchmarks involve text-layer defects, while all 8 TB miss/mismatch cases involve infrastructure-layer defects. In the 4 TB mismatch cases, the agent does not simply fail — it finds a plausible but incorrect text-layer defect instead, despite having Docker access to investigate the actual infrastructure issue. For example, in build-pov-ray and financial-document-processor, the GT defect is stale package version pinning in the oracle, but the agent flags test overspecification in the instruction-test contract.

Verdict–RC gap. Verdict-level recall overestimates root-cause-matched performance by 2–54pp across benchmarks. The gap is small on SWE-bench (3.8% mismatch rate among verdict TPs) because its defects lie in the agent's strongest layer — PS–test text comparison. On Terminal-Bench, the mismatch rate is 49–61%: the agent often finds a real defect but in the wrong layer, yielding a correct verdict with an incorrect root cause.

### 4.2. Pipeline Ablation on SWE-bench

We restrict the ablation to SWE-bench, where 95% of defects are text-layer (false negative or underspecified) and both H and NH conditions can detect them — on TB, over 50% of defects require Docker execution, confounding the harness axis. Table 5 presents the full 2×2 ablation of solve and harness across both models.

For GPT, solve and harness interact rather than add. Harness alone is neutral: VO-H and VO-NH both yield 59.7% RC recall. Solve alone helps: adding solve without harness raises recall to 78.3% (+18.6pp), because the agent's failed attempt provides empirical evidence that the test rejects a valid approach, converting abstract suspicion into a concrete defect diagnosis. But combining solve with harness drops recall to 53.9% (−24.4pp vs. STV-NH) — worse than any other GPT condition.

We call this pattern self-attribution anchoring. In STV-H, the agent first fails to solve the task, then verifies with sandbox access to its own failed patch — creating conditions for self-blame. In STV-NH, the agent receives the same solve experience as static text but cannot interactively explore its failure; this constrains attention to PS–test alignment rather than self-debugging. We qualitatively analyze 19 tasks where STV-NH and other GPT conditions disagree (12 where STV-NH is uniquely correct, 7 where it is uniquely wrong) and find a consistent mechanism: the STV-H agent reviews its failed patch, attributes the test failure to its own implementation rather than to the test, and downgrades or dismisses defects it would otherwise flag. In django-14182 and django-12212, the STV-H agent identifies the exact same defect as STV-NH but concludes "valid" after inspecting its solve trace.

The reverse cases (7 tasks where STV-H outperforms STV-NH) follow the same mechanism in the opposite direction: harness grounds the analysis with concrete evidence and prevents STV-NH's tendency to rationalize subtle gaps as "probably fine." The mechanisms are symmetric; the frequencies are not. Per-task counts confirm this: adding solve flips 31 tasks forward and 7 backward (4.4:1); removing harness flips 36 forward and 4 backward (9:1). The net asymmetry arises because 95% of SWE-bench defects are false negative or underspecified — text-layer gaps detectable by abstract PS–test comparison, where self-attribution is the dominant failure mode. The few cases where harness helps are ambiguous enough that abstract reasoning rationalizes them away.

Gemini is stable across all conditions (70–76% RC recall). We qualitatively compare GPT and Gemini on 12 tasks where they disagree in at least one condition (of 129 evaluated; Gemini wins 8, GPT wins 4). Three behavioral patterns emerge. First, Gemini's static analysis is stricter: in its outputs, it consistently asks whether any correct implementation could fail the test and flags mismatches as "overfit to gold patch," whereas GPT asks whether the test checks reasonable behavior and tends to accept borderline cases as "reasonable generalizations." Gemini correctly identifies all 3 tasks that GPT-VO-NH misses, confirming that its stricter criteria are sufficient without solve experience. Second, Gemini resists self-attribution anchoring: it maintains separation between "my patch failed" and "is the test fair?" On 3 tasks where both models

Table 5. SWE-bench pipeline ablation (129 defect + 129 valid). RC = root-cause-matched via LLM judge (GPT-5.4 full). STV = solve-then-verify, VO = verify-only, H = with agentic harness, NH = no harness (API single-shot).

| Condition | | | Verdict | | | Root-Cause Matched | | |
|---|---|---|---|---|---|---|---|---|
| Model | Harness | Solve | Recall | Prec. | F1 | Recall | Prec. | F1 |
| GPT-5.4 | NH | VO | 61.2 | 85.9 | 0.715 | 59.7 | 83.7 | 0.697 |
| | | STV | 80.6 | 74.8 | 0.776 | 78.3 | 72.7 | 0.754 |
| | H | VO | 62.0 | 72.7 | 0.669 | 59.7 | 70.0 | 0.644 |
| | | STV | 64.1 | 73.9 | 0.687 | 53.9 | 62.2 | 0.577 |
| Gemini 3.1 | NH | VO | 77.2 | 81.0 | 0.791 | 70.9 | 75.0 | 0.729 |
| | | STV | 77.5 | 76.9 | 0.772 | 71.3 | 70.8 | 0.710 |
| | H | VO | 81.8 | 77.6 | 0.796 | 76.4 | 76.4 | 0.764 |
| | | STV | 81.0 | 75.4 | 0.781 | 74.4 | 69.2 | 0.717 |

fail to solve and verify with harness, Gemini attributes the failure to the benchmark while GPT attributes it to its own implementation. In django-11281, both agents fail identical tests, but Gemini concludes "the task is underspecified — tests demand exact matches on an undocumented string set" while GPT concludes "the task is hard but valid — my patch was incomplete." Third, GPT-STV-NH finds 3 hard cases that Gemini misses across all conditions — Gemini's tendency to fill ambiguity with assumed intent prevents it from flagging the gap. In matplotlib-21542, the PS says "use new-style format strings" but the test also requires old-style backward compatibility; GPT flags this as a false negative, while Gemini infers backward compatibility was intended and accepts the test.

Model choice matters more than pipeline design. Gemini achieves a high floor (stable 70–76%); GPT achieves a high ceiling (78.3% in the right condition) but a low floor (53.9% in the wrong one). GPT-STV-NH peaks because GPT is lenient by default (accepting borderline tests as "reasonable"), but a failed solve attempt provides direct evidence that the test rejects a valid approach — converting abstract suspicion into a concrete defect diagnosis — while the absence of harness prevents self-attribution from discounting that evidence. Gemini VO-NH (70.9%) outperforms GPT VO-H (59.7%) with no pipeline tricks, suggesting that intrinsic model capability for specification reasoning is the dominant factor. Solve and harness effects are model-dependent interactions, not consistent improvements, precluding general pipeline recommendations.

### 4.3. GT Quality and Manual Verification

We manually verify 77 cases across all error categories to validate both judge accuracy and GT quality (Table 6; methodology in Appendix D).

Table 6. Manual verification breakdown (77 cases).

| Category | SWE | TB2 | TB1 | Total |
|---|---|---|---|---|
| TP | 9 | 5 | 13 | 27 |
| FP | 5 | 5 | 5 | 15 |
| TN | 3 | 2 | 3 | 8 |
| FN (verdict) | 10 | 2 | 3 | 15 |
| FN (RC) | 5 | 3 | 4 | 12 |
| Total | 32 | 17 | 28 | 77 |

Judge accuracy. All 27 sampled TP cases confirm judge MATCH as appropriate. Of 15 sampled FP cases, the agent escalates minor inconsistencies into defect verdicts or misattributes its own solve failure as a benchmark defect. Of 8 sampled TN cases, all confirm valid tasks correctly identified. Of 15 FN (verdict) cases (agent says valid, GT says defect), the majority are genuine agent misses, but a subset reflects GT noise rather than agent error. Of 12 FN (RC) cases (agent says defect but identifies the wrong root cause): All 5 SWE cases are agent errors. Notably, in sympy-13978, the agent's reasoning concludes "fair" but its JSON verdict outputs "defect" due to a hedging sentence — a reminder that even frontier models make elementary mistakes. TB cases (7) are 6 agent errors and 1 GT noise — the agent found a real but unlabeled defect, confirming GT under-labeling rather than agent error. Both FN categories (verdict + RC = 27 cases) are where GT quality directly affects reported metrics — a noisy GT label penalizes a correct agent — so we analyze GT noise in these cases below.

GT noise. We define GT noise as cases where the GT label is incorrect or incomplete — either the flagged concern is not a functional defect, or the GT misses a real defect the agent found. GT noise manifests in two directions. Over-labeling (FN verdict: agent says valid, GT says defect): GT noise rates are SWE-bench 2–3/10 (20–30%, all in 2/3-agreement tasks;

3/3-agreement tasks had 0% noise), TB2 0/2, TB1 1/3. SWE-bench noise reflects annotators flagging mild concerns as defects — in these cases, the agent's "valid" verdict is more defensible than the GT label. Underlabeling (FN RC): 1 of 12 cases is a genuine multi-defect where the agent found a real but unlabeled defect, confirming GT incompleteness rather than agent error.

Separately, we analyze 33 SWE-bench tasks excluded by our $\geq 2/3$ agreement filter (i.e., only 1/3 annotators flagged a defect): 91% of verdict FN cases in this set are GT noise, validating the filtering threshold.

Version-diff GT limitations. SWE-bench GT tends toward over-labeling: annotators flag tasks conservatively, and $\geq 2/3$ agreement filtering reduces but does not eliminate noise. TB GT tends toward underlabeling: PR-based GT captures only the defect the authors fixed, missing co-occurring defects. Overall, reported RC recall is a slight lower bound, but the dominant source of RC mismatch is agent error (5/5 SWE + 6/7 TB = 11/12), not GT incompleteness.

## 5. Discussion

Limitations. Version-diff GT has a fundamental constraint: it captures only defects the authors fixed, so unfixed defects may remain. SWE-bench GT tends toward over-labeling because annotators conservatively flagged any task with mild concerns to curate a clean subset, while PR-based TB GT tends toward underlabeling. We evaluate only two models (GPT-5.4 and Gemini 3.1 Pro); open-source and smaller models may exhibit different patterns. The LLM judge is not infallible: in our 10-case ablation (Appendix A), the selected judge (GPT-5.4 full) was wrong in 1 case, so a small fraction of RC matching decisions may be incorrect. Most qualitative analyses have small sample sizes because manual verification of each task requires substantial human time; claims based on these analyses should be interpreted as recurring patterns rather than statistically robust findings.

Model choice matters more than pipeline design. Solve experience and harness access interact in model-dependent ways that preclude general recommendations. Gemini achieves stable 70–76% RC recall across all conditions; GPT ranges from 54–78% depending on pipeline configuration. Performance is sensitive to prompt design (Appendix E); developing systematic methods for prompt optimization in verification tasks is an important open problem. At least for SWE-bench, harness and sandbox access are not essential: NH matches or exceeds H at a fraction of the cost for text-layer defects.

Human-AI complementarity. Agents excel at static contract analysis (PS–test text comparison) but struggle with infrastructure-level defects (environment issues, Dockerfile leakage). The optimal workflow is agent-first triage followed by human review of infrastructure-heavy tasks — LLMs prioritize human review rather than replace it.

Future work. Beyond detection, measuring whether agents can fix identified defects is the natural next step. Current RC matching is binary; a richer actionability spectrum would measure whether a diagnosis can guide a fix, even when the root-cause label does not exactly match. The GT in TVB could also serve as training signal for verification-specialized critics via RLVR, analogous to how McAleese et al. (2024) trained code-bug critics via RLHF. Scalable GT quality assurance across benchmark types remains an open challenge.

## 6. Conclusion

We present Task Verification Bench, a meta-benchmark that repurposes defects documented during benchmark updates as ground truth across two benchmark families. Our two-tier evaluation — verdict followed by root-cause matching — reveals that verdict-level recall overestimates true defect detection by up to 54pp, establishing RC matching as a necessary evaluation layer for benchmark auditing. A verify-only baseline achieves 35–60% RC recall, meaningful for triage but insufficient for full automation. Pipeline ablation on SWE-bench uncovers self-attribution anchoring: in STV-H, GPT-5.4 attributes test failures to its own implementation rather than the benchmark, dropping RC recall to 53.9%; model choice ultimately matters more than pipeline design. Manual verification of 77 cases confirms that the dominant source of RC mismatch is agent error (11/12), not GT incompleteness, though version-diff GT exhibits structural over- and under-labeling. Agents excel at text-layer defect detection but struggle with infrastructure-layer defects, suggesting agent-first triage followed by targeted human review as the practical workflow.

## References

Aleithan, R., Xue, H., Mohajer, M. M., Nnorom, E., Uddin, G., and Wang, S. Swe-bench+: Enhanced coding benchmark for llms. arXiv preprint arXiv:2410.06992, 2024.

Badertdinov, I., Golubev, A., Nekrashevich, M.,

Shevtsov, A., Karasik, S., Andriushchenko, A., Trofimova, M., Litvintseva, D., and Yangel, B. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. Advances in Neural Information Processing Systems, 38, 2026a.

Badertdinov, I., Nekrashevich, M., Shevtsov, A., and Golubev, A. Swe-rebench v2: Language-agnostic swe task collection at scale. arXiv preprint arXiv:2602.23866, 2026b.

Bercovich, I. What makes a good terminal-agent benchmark task: A guideline for adversarial, difficult, and legible evaluation design. arXiv preprint arXiv:2604.28093, 2026.

Cao, J., Chan, Y.-K., Ling, Z., Wang, W., Li, S., Liu, M., Qiao, R., Han, Y., Wang, C., Yu, B., He, P., Wang, S., Zheng, Z., Lyu, M. R., and Cheung, S.-C. Rigor, reliability, and reproducibility matter: A decade-scale survey of 572 code benchmarks. arXiv preprint arXiv:2501.10711, 2025.

Chen, Z., Sun, Z., Shi, Y., Peng, C., Gu, X., Lo, D., and Jiang, L. Rethinking the value of agent-generated tests for llm-based software engineering agents. arXiv preprint arXiv:2602.07900, 2026.

Chowdhury, N., Aung, J., Shern, C. J., Jaffe, O., Sherburn, D., Starace, G., Mays, E., Dias, R., Aljubeh, M., Glaese, M., Jimenez, C. E., Yang, J., Ho, L., Patwardhan, T., Liu, K., and Madry, A. Introducing SWE-bench verified. https://openai.com/index/introducing-swe-bench-verified/, 2024.

Chu, E., Agarwal, R., Thangamuthu, A., Graham, B., Mattern, J., Jiang, F., Cento, P., Jain, S., Abbasi, M., Rezaei, M. H., Wang, G., Zhang, A., Guo, S., Nguyen, K., Bidgoli, A., et al. FrontierSWE. https://www.frontierswe.com/blog, 2026.

Deng, X., Da, J., Pan, E., He, Y. Y., Ide, C., Garg, K., Lauffer, N., Park, A., Pasari, N., Rane, C., et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? arXiv preprint arXiv:2509.16941, 2025.

Elfeki, M., Trinh, T., Luu, K., Luo, G., Hunt, N., Montoya, E., Marwaha, N., He, Y., Wang, C., Crabedo, F., et al. Hil-bench (human-in-loop benchmark): Do agents know when to ask for help? arXiv preprint arXiv:2604.09408, 2026.

Gema, A. P., Leang, J. O. J., Hong, G., Devoto, A., Mancino, A. C. M., Saxena, R., He, X., Zhao, Y., Du, X., Madani, M. R. G., et al. Are we done with mmlu? In Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pp. 5069–5096, 2025.

Harman, M., Ritchey, J., Harper, I., Sengupta, S., Mao, K., Gulati, A., Foster, C., and Robert, H. Mutation-guided llm-based test generation at meta. In Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, pp. 180–191, 2025.

Hu, J., Shukla, P., and Huang, K. Verifying the verifiers: Failure attribution for agentic benchmark diagnostics and training data curation. In ICLR 2026 Workshop on Lifelong Agents: Learning, Aligning, Evolving, 2026. URL https://openreview.net/forum?id=B6QfT7SHh8.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. Swe-bench: Can language models resolve real-world github issues? In International Conference on Learning Representations, volume 2024, pp. 54107–54157, 2024.

Joaquin, A. S., Gipiškis, R., Staufer, L., and Gil, A. Deprecating benchmarks: Criteria and framework. arXiv preprint arXiv:2507.06434, 2025.

Li, C., Xu, Y., Wang, Z., Tan, S. H., et al. Are benchmark tests strong enough? mutation-guided diagnosis and augmentation of regression suites. arXiv preprint arXiv:2604.01518, 2026.

Liu, J., Xia, C. S., Wang, Y., and Zhang, L. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. Advances in neural information processing systems, 36:21558–21572, 2023.

McAleese, N., Pokorny, R. M., Uribe, J. F. C., Nitishinskaya, E., Trebacz, M., and Leike, J. Llm critics help catch llm bugs. arXiv preprint arXiv:2407.00215, 2024.

Merrill, M. A., Shaw, A. G., Carlini, N., Li, B., Raj, H., Bercovich, I., Shi, L., Shin, J. Y., Walshe, T., Buchanan, E. K., et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. arXiv preprint arXiv:2601.11868, 2026.

Northcutt, C. G., Athalye, A., and Mueller, J. Pervasive label errors in test sets destabilize machine learning benchmarks. arXiv preprint arXiv:2103.14749, 2021.

OpenAI. Why SWE-bench verified no longer measures frontier coding capabilities. https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/, 2025.

Rank, B., Bhatnagar, H., Prabhu, A., Eisenberg, S., Nguyen, K., Bethge, M., and Andriushchenko, M. Posttrainbench: Can llm agents automate llm post-training? arXiv preprint arXiv:2603.08640, 2026.

Siddiq, M. L., Dristi, S., Saha, J., and Santos, J. C. The fault in our stars: Quality assessment of code generation benchmarks. In 2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM), pp. 201–212. IEEE, 2024.

Tu, X., Wang, T., Huang, K., Qu, Y., Mostafavi, S., et al. Benchguard: Who guards the benchmarks? automated auditing of llm agent benchmarks. arXiv preprint arXiv:2604.24955, 2026.

Wang, H., Mang, Q., Cheung, A., Sen, K., and Song, D. BenchJack: Benchmarking hackability of code benchmarks. https://github.com/benchjack/benchjack, 2025a.

Wang, Y., Pradel, M., and Liu, Z. Are" solved issues" in swe-bench really solved correctly? an empirical study. arXiv preprint arXiv:2503.15223, 2025b.

Yang, J., Jimenez, C. E., Zhang, A. L., Lieret, K., Yang, J., Wu, X., Press, O., Muennighoff, N., Synnaeve, G., Narasimhan, K. R., et al. Swe-bench multimodal: Do ai systems generalize to visual software domains? arXiv preprint arXiv:2410.03859, 2024.

Yu, B., Zhu, Y., He, P., and Kang, D. Utboost: Rigorous evaluation of coding agents on swe-bench. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 3762–3774, 2025.

Yu, B., Cao, Y., Zhang, Y., Lin, L., Xu, J., Zhong, Z., Xu, Q., Wang, G., Cao, J., Cheung, S.-C., et al. Swe-abs: Adversarial benchmark strengthening exposes inflated success rates on test-based benchmark. arXiv preprint arXiv:2603.00520, 2026.

Zanoli, C., Giovannini, A., Jin, T., Klimovic, A., and Perlitz, Y. Elt-bench-verified: Benchmark quality issues underestimate ai agent capabilities. arXiv preprint arXiv:2603.29399, 2026.

Zhu, Y., Jin, T., Pruksachatkun, Y., Zhang, A., Liu, S., Cui, S., Kapoor, S., Longpre, S., Meng, K., Weiss, R., et al. Establishing best practices in building rigorous agentic benchmarks. Advances in Neural Information Processing Systems, 38, 2026.

## A. Judge Model Ablation

We compared GPT-5.4 mini and full as judge models. On SWE-bench, mini achieved 30.5% RC recall with 87.3% three-run agreement, while full achieved 38.5% RC recall with 97.3% agreement. On TB2, mini achieved 26.9% and full 42.3%, both with 100% agreement. Manual verification of 10 cases where mini and full disagreed showed full was correct in 8, mini in 1, and 1 was borderline. The primary failure mode of mini is classifying semantic supersets or alternative perspectives as mismatches — it applies stricter literal matching rather than reasoning about whether the same underlying defect is identified. We adopt GPT-5.4 full with single-run evaluation for all results.

## B. SWE-bench Subset Curation

We initially curated 9 overlapping subsets from 1,699 annotated tasks to test specific hypotheses about defect detection: random (baseline), easy/hard (difficulty), agree/disagree (annotator agreement), severe/mild (severity), and other/main types (defect type). Each subset contains 24–30 tasks, totaling 434 unique tasks (213 defect + 221 valid) for pilot experiments. Pilot analysis on this set revealed that 1/3-agreement tasks contained 91% GT noise, motivating the $\geq$2/3 agreement filter described in Section 3.3.

After filtering, per-subset RC recall (GPT-5.4, VO-H) shows that more severe and harder-to-fix defects are easier to detect:

Table 7. Per-subset RC recall (GPT-5.4, VO-H, $\geq$2/3 agreement).

| Factor | Subset | $n$ | RC recall |
|---|---|---|---|
| Difficulty | easy (<15min) | 22 | 27.3% |
| | medium (15min–1hr) | 51 | 58.8% |
| | hard (>1hr) | 53 | 73.6% |
| Severity | mild (max 1–2) | 49 | 44.9% |
| | severe (max $\geq$3) | 77 | 68.8% |

More severe defects produce clearer PS–test mismatches that the agent's static analysis can detect; easy-to-fix tasks tend to have subtler gaps that require domain knowledge.

## C. GT Source to GT Entry Parsing

SWE-bench. We parse the per-annotator CSV (samples_with_3_annotations_public.csv) by grouping rows by instance_id. Each GT entry contains the binary label (filter_out=True $\rightarrow$ defect), annotator agreement count, and each annotator's free-text notes

for false negative, underspecification, and other issues. A task is classified as defect if ≥2 of 3 annotators flag it (fn≥2 or us≥2 or other≥1). The judge uses the concatenated annotator notes as the GT reference.

Terminal-Bench. For both TB2 (PR #53) and TB1 (96 PRs), we use Claude Opus 4.7 to read each PR's diff and description, classifying defects into six types: instruction underspecified, environment issue, test overspecified, information leakage, oracle bug, and test underspecified. A single task may receive multiple labels when the PR addresses several issues (e.g., caffe-cifar-10 has 3 labels). Each GT entry contains the defect type, a natural-language description, and a fix summary; the judge uses the description and fix summary as the GT reference.

## D. Manual Verification Methodology

For each of the 77 sampled cases, we use Claude Code with Opus 4.7 to compile a structured summary containing: (1) the task's problem statement and instructions, (2) the agent's solve-phase trajectory and patch (if STV), (3) Phase 2 test output, (4) the agent's verification verdict and reasoning, and (5) the GT label and judge's RC matching decision.

The human reviewer reads this summary and interrogates details via interactive dialogue with Claude Code — e.g., requesting specific file diffs, checking whether a claimed environment issue is reproducible, or tracing the agent's reasoning chain step by step. The final verdict (correct/incorrect, GT noise/valid) is made by the human reviewer; Claude Code serves as an information retrieval and summarization tool, not as a judge.

Each case produces a markdown analysis file documenting the GT claim, the agent's reasoning, the reviewer's assessment, and the final classification.

## E. Prompt Design and Ablation

### Verification Pipeline

The split-phase architecture runs in Docker containers built from each benchmark's official images.

Phase 0: Start container, install agent runtime. Phase 1: Agent solves the task with only the problem statement and repository code; test patch and gold patch are physically absent. Phase 2: Host executes the benchmark's test harness (SWE-bench eval.sh or Terminal-Bench test.sh). Phase 3: Agent receives test results, gold patch, and test patch via a structured verification prompt and renders a verdict.

Phase 1 context is preserved into Phase 3 via session resumption, allowing the agent to reference its solve attempt during verification.

### Prompt Iteration

Prompts were iterated independently for SWE-bench (6 versions) and TB2 (5 versions) on small pilot sets, using an LLM judge to compare agent verdicts against GT annotations and identify failure modes after each iteration. Key transitions:

v3→v4 (single→split phase): Test/gold patch physically removed from container. Prevents TDD degradation. RC recall on pilot set: 0%→20%.

v5→v6 (anchoring removal + checklist): "Forget your patch" instruction + PS–test fairness checklist. RC recall: 20%→51%.

v6→v8 (failure analysis + sufficiency): Instead of "forget your patch," the agent analyzes why its patch failed. Adds PS sufficiency checklist. RC recall: 51%→54%. Net gain is small (+33 gains, −28 regressions) — Phase 2 failure utilization is a double-edged sword.

SWE v8 vs TB v5: SWE-bench uses Phase 2 failure as Step 1 diagnostic evidence. TB2 uses "forget your solve result" + oracle bug detection via solve.sh execution + information leakage checklist, addressing TB-specific defect types.

## F. Per-Repo Analysis

Table 8 shows RC-matched recall by repository on SWE-bench.

Table 8. SWE-bench RC-matched recall by repository (GPT-5.4, VO-H, ≥2/3 agreement).

| Repository | $n$ | RC recall |
|---|---|---|
| pydata (xarray) | 5 | 80% |
| scikit-learn | 16 | 75% |
| sympy | 24 | 75% |
| psf (requests) | 4 | 75% |
| matplotlib | 9 | 56% |
| django | 51 | 51% |
| astropy | 4 | 50% |
| sphinx | 8 | 38% |
| pytest | 4 | 25% |

Three defect tasks lack judge results due to agent output parsing failures, reducing the per-repo total to 126 of 129. Django has the most defect tasks (51) and recall closest to the overall average (51% vs. 59.5%), suggesting small-sample repos may have inflated or deflated recall due to noise.

## G. Full-Set Results Including Low-Agreement GT

Table 9 reports SWE-bench results on the unfiltered GT set (213 defect + 221 valid = 434 tasks), including 88 defect tasks where only 1 of 3 annotators flagged the task as problematic. In this unfiltered set, 91% of verdict FN cases among 1/3-agreement tasks are GT noise, motivating the agreement filter used in the main paper. Main-paper results use the $\geq 2/3$ agreement subset with build failures excluded (129 defect + 129 valid = 258 tasks) for cleaner evaluation.

Table 9. SWE-bench results on unfiltered GT (including 1/3 agreement defect labels).

| | GPT-5.4 (H) | GPT-5.4 (NH) |
|---|---|---|
| Tasks (def./valid) | 434 (213/221) | |
| Verdict recall | 53.5% | 60.1% |
| RC recall | 38.5% | 37.1% |
| RC F1 | 0.485 | 0.461 |