FAILS BEFORE REPAIR IS NOT ENOUGH: AUDITING THE VALIDITY OF EXECUTABLE REPAIR BENCHMARKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Executable repair benchmarks are trusted because they run. A task counts as valid when the buggy artifact fails its tests and the reference fix passes them, and this two-sided check is close to the only validity evidence most benchmarks report. We show it misses three distinct failures, using a suite of 81 Verilog repair tasks and three instruments that apply to any executable repair benchmark. First, answer leakage: in 89.5% of our tasks the fix was written into the task inputs, and no task withheld it. Removing it costs a 1.5B-parameter model 16.9 points of pass@1 (95% CI [9.5, 24.7]) and four API models between -0.6 and 3.3 points, and with the answer withheld the share of tasks that separate two frontier models rises from 14.8% to 32.1% (paired +17.3 points, [7.4, 28.4]). Second, oracle weakness: our testbenches detect only 61.6% of injected single-site mutants, and four detect none. Applying the same mutation instrument to two benchmarks we did not build locates the cause in oracle design. RTLLM, whose testbenches are directed like ours, scores 52.9%; VerilogEval, whose testbenches compare the candidate against a reference cycle by cycle, scores 91.8%. Third, resolution: even with the answer withheld, 36 of 81 tasks are solved by all four API models on every sample. We also report that mining frontier-model failures into new tasks produced no novel tasks, with or without the answer available, and why. We release the suite, the harness and all three instruments.

1 INTRODUCTION

Code-generation benchmarks for hardware description languages are close to saturated. VerilogEval (Liu et al., 2023) and RTLLM (Lu et al., 2024) ask a model to write a module from a specification, and the strongest current models pass most of them. Evaluation has therefore moved toward repair: give an agent a design with a defect, a testbench, and simulator access, and ask it to make the testbench pass. HWE-Bench (Cui et al., 2026) is the most ambitious example, with 417 tasks mined from real bug-fix pull requests across six open-source hardware projects. Software engineering went through the same transition earlier with SWE-bench (Jimenez et al., 2024), Defects4J (Just et al., 2014a) and QuixBugs (Lin et al., 2017).

A repair benchmark carries a validity argument that a generation benchmark does not. The task has to contain a real defect, the oracle has to detect it, a correct repair has to exist, and the task inputs must not already contain the repair. The standard check covers the first three only partly. It runs the oracle on the buggy artifact (it should fail) and on the reference fix (it should pass). We call this the *two-sided invariant*. SWE-bench Verified (OpenAI, 2024) exists because the automated version of this check let through a large share of underspecified or unfair tasks, which is evidence that the invariant alone is weak. How weak, and in which directions, has not been measured.

We measured it on our own benchmark. Fail2Bench is a suite of 81 Verilog repair tasks run under a real toolchain (Icarus Verilog, Verilator, Yosys) with a tool-using agent harness. Every task satisfies the two-sided invariant, every verdict is deterministic across five repeats, and an agent that applies the stored reference fixes scores 100%. By the usual standard the suite is valid. It still had three problems that the invariant cannot see, and each needed its own instrument:

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.

1. **Answer leakage** (§6). 77 of 86 task directories state the literal fix, in a source comment or in a `## Fix` section of the specification the agent can query. The cause was our own task-authoring helper. A mechanical audit finds it; a derived hint-free suite removes it while preserving both invariants; a paired ablation prices it per model.
2. **Oracle weakness** (§5). Mutation testing (Jia & Harman, 2011) shows our testbenches catch 61.6% of injected faults. Run unchanged over RTLLM and VerilogEval, the same instrument gives 52.9% and 91.8%. The two suites with directed testbenches sit within nine points of each other and the one with a differential oracle is thirty points above both, so the gap tracks oracle design rather than who wrote the tests.
3. **Low resolution** (§7). With four API models and the answer withheld, only 45 of 81 tasks separate any two models. Whole-suite pass@1 averages in 36 tasks that every model solves on every sample, and this inflates the weakest model by 26.7 points relative to its score on the tasks that carry signal.

Before these instruments existed, a fourth kind of check found that 23 of the 52 tasks in an earlier hand-built version of the suite (44%) were defective, including an oracle that rewarded the *absence* of the feature it tested (§4). None of these defects was visible to the project's 334 unit tests.

Our suite is smaller than HWE-Bench and its tasks are authored rather than mined, and we say plainly in §9 where that costs us. The contribution is not the 81 tasks. It is a set of measurements that any executable repair benchmark can run on itself, evidence from three benchmarks that the measurements discriminate, and a record of what they found in one suite whose authors were trying to get it right.

2 Related work

Hardware generation and repair benchmarks. VerilogEval (Liu et al., 2023) and RTLLM (Lu et al., 2024) evaluate specification-to-RTL generation. RTLFixer (Tsai et al., 2024) repairs syntax errors using compiler feedback; we target functional defects judged by simulation. HWE-Bench (Cui et al., 2026) is the closest work: 417 repair tasks from real pull requests in Verilog, SystemVerilog and Chisel, validated through each project's native regression flow, with a best agent at 70.7%. It is larger, better sourced and harder than our suite. As far as we can tell from the paper, it does not report oracle adequacy or leakage of the solution into the task inputs, which are the two properties we measure. We see the instruments here as complementary to HWE-Bench and describe in §10 how they would apply to it.

Software repair benchmarks. SWE-bench (Jimenez et al., 2024) established repository-level agentic repair from real issues. SWE-bench Verified (OpenAI, 2024) added human screening after automated validity checks proved insufficient. Defects4J (Just et al., 2014a) pairs real Java faults with developer-written triggering tests. Our two-sided invariant is the check these benchmarks share in some form; our results concern what it leaves unchecked.

Contamination and saturation. LiveCodeBench (Jain et al., 2024) and LiveBench (White et al., 2024) counter contamination by refreshing problems after training cutoffs. Answer leakage is a related but different failure: the solution is not in the training data but in the task itself, and recency does not help.

Test adequacy. Mutation testing measures whether a test suite detects small syntactic faults (Jia & Harman, 2011), and mutant detection correlates with real-fault detection in software (Just et al., 2014b). We apply it to benchmark oracles rather than to the code under test. Our pass@k estimator is the unbiased one of Chen et al. (2021), and our failure minimizer uses delta debugging (Zeller & Hildebrandt, 2002).

3 The suite and the harness

Tasks. A task is a directory with a buggy Verilog artifact, a self-checking testbench, a natural-language `spec.md`, a `task.yaml`, and a reference fix that the agent cannot read. The 81 evaluated tasks split into 42 *toy* tasks (small blocks such as counters, encoders, CRC units and simple bus slaves) and 39 *design-derived* tasks written from a referenced design or specification, for example

a RISC-V immediate decoder, a store-queue forwarding unit, an AXI burst address generator, and an interrupt controller modelled on PicoRV32. Design-derived tasks are written by us from those references; they are not mined historical bugs. Four design-derived tasks are sequence-dependent by construction: their testbench has a phase that both the buggy and fixed designs pass, so only multi-cycle, order-dependent stimulus separates them. Bugs fall into 11 classes (for example missing logic, width or sign errors, wrong operators, off-by-one) across 9 functional categories (Appendix D). An earlier easy/medium/hard label did not predict pass rates in three separate checks, so we do not use it.

Validity invariants. All 81 tasks fail before repair and pass with the reference fix. Verdicts are identical across five repeated runs for all 82 discovered tasks, including one excluded task whose "bug" turned out to be correct behaviour. The pass criterion is the testbench's exit status under Icarus Verilog; no model judges correctness.

Agent harness. Each attempt is a tool-use loop of at most six turns. Later arms also have a 200-second wall-clock limit per attempt; §9 lists which arms and what it cost. The agent can call `read_file`, `list_files`, `write_file`, `write_patch` (a context-anchored unified diff), `run_sim`, `run_lint`, `run_synth` and `query_spec`. The reference fix is hidden at three layers and a regression test checks that it cannot be listed or read. The same scaffold runs every model.

Models and statistics. We evaluate Claude Opus 5, Claude Sonnet 5, DeepSeek V4-Pro, Gemini 3.5 Flash, and qwen2.5-coder-1.5B served locally. Each model draws s independent samples per task, with $s = 3$ unless stated. We report pass@1 with 95% percentile-bootstrap intervals resampled over tasks (10,000 resamples); comparisons between conditions are paired per task. We mark a difference * when its interval excludes zero. The Opus hint-free arm was re-derived from saved artifacts after its live run was interrupted and has no cost record; Appendix E lists the provenance of every arm.

4 What the two-sided invariant already misses

Before any of the instruments below, we wrote a reference fix for each of the 52 tasks in an earlier hand-built version of the suite and ran both halves of the invariant. Twenty-three tasks (44%) were defective: 22 of the 42 toy tasks and 1 of the 10 design-derived ones. The existing 334 unit tests passed throughout.

The defects were not cosmetic. Three tasks were not bugs: the artifact already met its specification. Four testbenches demanded values no implementation can produce; for a CRC-8 task we searched all 256 polynomials in both bit orders and none matched, and an LFSR testbench implied four different feedback masks for the same bit. Two testbenches checked the same input twice against two different expected values. Three oracles could not detect the bug their own task shipped. Seven testbenches raced the design at time zero. One pipeline-stall testbench was inverted: it passed only a design that ignored the stall signal, and failed both a specification-conformant design and a partial one.

Most of these defects are invisible to the fails-before-repair check alone. A testbench that does not compile, races the design, or expects impossible values fails every design, buggy or not, so the buggy artifact fails exactly as it should. They surface only when a reference fix is written and refuses to pass. That is why we treat the second half of the invariant as mandatory, and why our authoring tool now refuses to write a task unless both halves hold. None of the 30 design-derived tasks written after that tool existed has needed a change to its testbench or artifact since. The tool also rejected one of our own tasks, a clock divider whose reference produced an 8/72 duty cycle instead of 50/50.

The harness needed the same scrutiny as the tasks. We found and fixed eight bugs in it, and most of them lowered measured pass rates without any visible error. The worst was the patch applier: it applied diff hunks at their stated line offsets without checking context, so a slightly misnumbered patch could delete an unrelated port list and still report success. Fixing it moved Claude Sonnet 5 from 0.397 to 0.825 pass@1 on an earlier 63-task version of the suite. A later incident is instructive for anyone running paid sweeps: an Opus run exhausted its API credit mid-sweep, the provider returned HTTP 400 rather than a quota code, and the harness scored 162 unreached task-samples as model failures. It reported 21.4% pass@1; on the 52 tasks it actually reached, 51 passed. The harness now aborts on billing and authentication errors and marks any sample that passes zero tasks as invalid. Appendix B lists all of these.

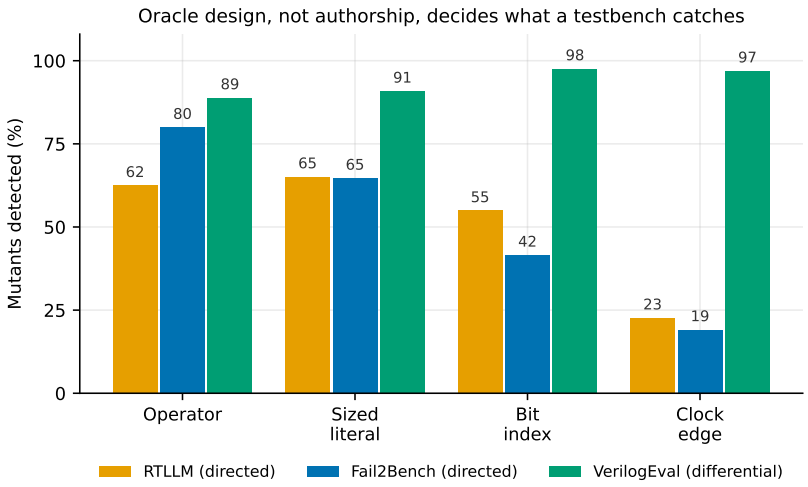


Figure 1: Mutation score by operator class on three benchmarks, one operator set. RTLLM (39 of 50 tasks scored) and Fail2Bench (78 of 81) use directed testbenches; VerilogEval (124 of 156) instantiates the reference beside the candidate and compares outputs every cycle. Overall: RTLLM 52.9%, Fail2Bench 61.6%, VerilogEval 91.8%. Clock-edge mutants survive about four times in five against both directed suites and are almost always caught by the differential one.

5 Oracle strength

The two-sided invariant certifies that the oracle catches the one defect the task ships. It says nothing about whether the oracle would catch a different defect in the same module, and that matters: an agent can pass a task by producing a design that is still wrong somewhere the testbench does not look.

Instrument. We generate every single-site mutant of each reference implementation using four classes of operator: operator substitution (arithmetic, relational, equality, logical, bitwise, shift), sized-literal perturbation, bit-index shift, and clock- or reset-edge inversion (full list in Appendix C). A mutant that does not compile is discarded rather than counted as surviving. The mutation score is the fraction of viable mutants the task's own testbench rejects.

Result on our suite. Our testbenches kill 446 of 724 viable mutants over the 78 tasks that admit one, a score of 61.6%. Twenty-one tasks score below 50%, and four oracles (`mul_pipe_flush`, `shift_reg`, `sync`, `wishbone`) kill no mutant at all. Each of those four satisfies the two-sided invariant.

The same instrument on benchmarks we did not build. A mutation score on a suite we wrote, bugged and repaired could simply reflect how we write testbenches. We therefore ran the unchanged instrument over RTLLM (Lu et al., 2024) and VerilogEval (Liu et al., 2023). Both are generation benchmarks, so we mutate their reference designs and score their shipped testbenches. RTLLM's testbenches drive directed vectors, as ours do. VerilogEval's instantiate the reference and the candidate side by side and count cycle-level mismatches.

Figure 1 shows the result. RTLLM scores 52.9%, our suite 61.6%, VerilogEval 91.8% (613 of 668). The spread matters for two reasons. It shows the instrument discriminates: it does not return roughly 60% for whatever it is pointed at. And the ordering follows oracle design. The two directed suites, written by different groups for different purposes, are nine points apart; the differential suite is thirty points above both. Clock-edge inversion is the sharpest case: 22.6% and 19.1% detected by the directed suites, 96.9% by the differential one. We had earlier excluded clock-edge mutants from our own score on the grounds that they might be equivalent, which would have raised it to 69.2%. VerilogEval's result shows they are observable by a testbench that checks every cycle, so we withdrew that adjustment. Bit-index shifts, which are observable by construction, survive our oracles 58.5% of the time.

Table 1: Pass@1 on the same 81 tasks with the fix stated and with it withheld. Δ is the paired per-task difference in points (positive means the model scored higher with the answer available); intervals are 95% bootstrap over tasks. s is samples per task in each arm. The Sonnet, Gemini and Opus with-hints arms ran without a wall-clock limit and their hint-free arms with one (§9).

Model	s	With fix stated	Fix withheld	Δ (pts)	95% CI
Claude Opus 5	1	97.5%	98.2%	-0.6	[-3.7, +1.8]
Claude Sonnet 5	3	95.6%	93.0%	+2.5*	[+0.2, +5.3]
DeepSeek V4-Pro	3	93.8%	91.0%	+2.9	[-0.8, +6.6]
Gemini 3.5 Flash	3	70.0%	66.7%	+3.3	[-4.5, +11.1]
qwen2.5-coder-1.5B	3	21.8%	4.9%	+16.9*	[+9.5, +24.7]

Two caveats limit what this licenses. VerilogEval can afford a differential oracle because every generation task has a reference; a repair benchmark built from real defects cannot always assume a trustworthy reference exists. And none of the three suites is clean: RTLLM has 18 of 39 scored tasks below 50% with five killing nothing, and VerilogEval has three below 50% with two killing nothing. Running the two-sided invariant on RTLLM turned up something else: 6 of its 50 reference designs fail their own testbench under our toolchain, and 5 more do not compile.

6 Answer leakage

Finding. Sonnet solved all four sequence-dependent tasks on the first sample, hours after we wrote them to be hard. Diffing its work directories showed the repairs were real and minimal, and also showed the agent deleting a comment block that began `// BUG:` and explained the defect. A mechanical audit of all 86 task directories on disk (the 81 evaluated, the excluded task, and four staged tasks) classified each file as *prescriptive* (states the fix), *marking* (points at the defect without stating the fix) or *clean*. Seventy-seven tasks (89.5%) were prescriptive in at least one file and the other nine were marking. None was clean. There were two channels: comments such as `// BUG: should be a - b` in the artifact, and `## Known Bug` and `## Fix` sections in `spec.md`, often containing the corrected Verilog, which the agent can retrieve with `query_spec`. Both came from the helper script we used to author tasks, which inserted them by construction.

Remedy. We derive a hint-free suite by stripping both channels and then re-verifying every task. All 81 tasks rebuild with zero leakage, and all 81 still fail before repair and pass with the reference fix. The one task the builder could not produce is the excluded non-bug, which the builder flagged independently. Our authoring tool now rejects any task the leakage audit flags, as a third gate after the two invariants.

What the leak was worth. Table 1 gives the paired effect for all five models. For the 1.5B model most of its score came from reading the answer: 21.8% with the fix stated, 4.9% without. On the four sequence-dependent tasks it went from 58.3% to zero. For the API models the effect is at most three points. It excludes zero only for Sonnet, and only narrowly: two independent three-sample hint-free runs both give 93.0% and +2.5 points, with intervals [+0.2, +5.3] and [-0.2, +5.5]. Opus shows no measurable effect. The leak mostly inflates weak systems. A small effect on a strong model does not make the leak harmless; it means the model can solve most of these tasks without the hint, which is a separate problem (§7).

Effort moved more than score. The 1.5B model took 1.70 times as long per sample without hints, Sonnet 1.56 times and Gemini 1.50 times, while DeepSeek was slightly faster (0.94 times).

Channels substitute for each other. We also built a variant that keeps the hint-free inputs but replaces the testbench's diagnostic messages with bare pass/fail lines, leaving every check byte-identical. We expected this to cost a frontier model points by removing localisation help. At three samples per arm it did not: Sonnet scored 95.1% with the terse oracle and 93.0% with the verbose one (Δ -2.1, [-4.9, +0.8]). What changed was behaviour: `query_spec` calls rose from 56 to 232. Denied the oracle's explanation, the model read the specification instead. Closing one information channel and measuring the score will understate how much a task gives away if another channel carries the same information, which is why the hint-free suite closes the comment and specification

Table 2: Hint-free pass@1 over the whole suite and over the 45 tasks that discriminate among the four API models. Signal-only intervals are 95% bootstrap over those 45 tasks. Opus has $s = 2$ in this arm; the others $s = 3$.

Model	Whole suite (81)	95% CI	Signal-only (45)	95% CI
Claude Opus 5	98.2%	[95.1, 100.0]	96.7%	[91.1, 100.0]
Claude Sonnet 5	93.0%	[88.1, 97.1]	87.4%	[78.5, 94.8]
DeepSeek V4-Pro	91.0%	[87.2, 94.2]	83.7%	[77.8, 88.9]
Gemini 3.5 Flash	66.7%	[57.6, 75.3]	40.0%	[28.9, 51.1]

channels together. We make no claim here about fault localisation as a bottleneck. An earlier single-sample version of this ablation showed a significant +6.2-point effect, which disappeared at three samples per arm.

7 How much of the suite carries signal

A task tells you something about the ranking of two models only if they perform differently on it. We call a task *discriminating* for a model set if at least two models in the set differ in pass rate on it, and *saturated* if every model passes every sample.

Most of the suite does not discriminate. With the answer withheld and all four API models, 45 of 81 tasks discriminate, 36 are saturated, and none is failed by every model. Table 2 shows what the saturated tasks do to reported scores. Restricting to the discriminating subset costs Opus 1.5 points and Gemini 26.7. Whole-suite pass@1 compresses differences between models because it averages in tasks that every model solves. The three strongest models also have overlapping intervals on both columns, so this suite cannot rank Opus, Sonnet and DeepSeek, and we do not.

Leakage reduces resolution. Discriminative power is a property of a model set and a sampling budget: adding models or samples makes more tasks look discriminating even if nothing about the suite changes. Our first comparison between conditions matched the model set but not the number of samples, and overstated the effect (we had reported roughly a threefold increase). Matching both, for Sonnet and DeepSeek at three samples each, withholding the answer raises the discriminating share from 14.8% to 32.1%, a paired difference of +17.3 points [+7.4, +28.4]. With all four models at one sample each, the difference is +7.4 [-3.7, +19.8]: same sign, not significant. Pairwise gaps between models tell a consistent but weaker story. All six pairs are wider with the answer withheld, by a median of 2.0 points, and only the Opus-Sonnet widening (+3.2, [+0.1, +7.0]) is individually significant. Leakage shows up much more clearly in how many tasks carry signal than in the size of the gaps.

Stable means, unstable verdicts. Two independent three-sample hint-free runs of Sonnet gave identical pass@1, 93.00%, and disagreed on 12 of 81 tasks. Any analysis that consumes per-task verdicts (partition breakdowns, paired ablations, discrimination buckets) inherits that per-task noise, not the stability of the mean. In one run the hint effect was significant on the design-derived partition and in the other on the toy partition. We therefore report no partition-level hint effects, and we would encourage benchmark papers that report per-category deltas to replicate a run and check.

8 A negative result: mining failures into new tasks

Fail2Bench began as an attempt to grow a benchmark from agent failures: take a failed repair, which is a design that did not exist before, minimise it with delta debugging, diagnose it, deduplicate it against the suite, and validate it as a new task. The pipeline works end to end and has produced a valid, novel card from a failure of the 1.5B model. From stronger models it produces nothing. From 59 failures in five with-hints samples of Sonnet on an earlier 63-task version of the suite, it produced zero cards: all 58 compilable failures were rejected as duplicates of existing tasks. Re-run on the 4 failures from one hint-free Sonnet sample over the current 81 tasks, which removes the possibility that the model had copied the stated fix, it again produced zero cards, rejected at the same step.

The reason is structural. Failed repairs from a capable model are rare and tend to leave the design unchanged. In one with-hints sample on the 63-task suite, 8 of Sonnet's 11 failures left the artifact byte-identical to the seed; of the 4 hint-free failures, 2 were byte-identical and the other 2 were more than 99% similar. A stronger model fails less often and produces less new material when it does. Growing a benchmark this way works best with the agents least worth benchmarking, and we do not claim the loop as a contribution.

9 LIMITATIONS

Scale and provenance. 81 tasks is a fifth of HWE-Bench, and our design-derived tasks are authored from references rather than mined from real bug histories. HWE-Bench is the stronger benchmark for measuring repair capability. Our claims concern measurement, and the external mutation results are the main evidence that they are not artifacts of our suite.

Circularity. We wrote the tasks and the oracles with an AI coding assistant, Claude (Appendix statement on AI use), and when we found oracles defective we repaired them the same way. The assistant belongs to the same model family as two of the evaluated models, which could favour those models on tasks it helped write; we have not measured whether it does. The external mutation audit addresses circularity for oracle strength only.

Headroom. With the answer withheld the three strongest models score 91–98%. The suite has little headroom at the top, and our four sequence-dependent tasks did not change that for the strongest models, though all four discriminate.

Contamination. A verbatim continuation probe on 57 of our reference designs gave 18.2% mean line recall, with the longest verbatim run anywhere being 5 lines. The probe ran on the 1.5B model only, so it bounds that model's verbatim recall and says little about frontier models.

Budgets. Every attempt is capped at six turns. The turn cap does not appear to bind: in all five arms with tool logging, failing attempts used fewer tool calls on average than successful ones (Opus 5.0 against 8.5), and no failure reached the successes' maximum. A 200-second wall-clock limit was added partway through the project, so it applies to all four hint-free API arms and the DeepSeek with-hints arm, but not to the Opus, Sonnet and Gemini with-hints arms or the qwen arms. It abandoned 6 attempts in the reported Sonnet hint-free arm, 4 in Gemini's, 6 in Opus's, 5 in DeepSeek's with-hints arm and none in DeepSeek's hint-free arm. Abandonments score as failures, so this asymmetry biases the Sonnet and Gemini hint effects upward and DeepSeek's downward. For Sonnet the bias does not change the estimate: a second hint-free run with no abandonments gives the same 93.0%. For Opus the agent had already written a correct repair in five of the six abandoned attempts, so we report its arm by replaying the deterministic oracle over the saved artifacts (98.1%, against 95.1% as scored live).

Single scaffold, few samples. One agent scaffold, three samples per arm in most cases, and one or two for Opus. Opus and DeepSeek have no pass@k beyond k=1.

10 CONCLUSION

A repair task can pass the standard validity check while handing the agent its fix, while its oracle ignores most faults other than the shipped one, and while contributing nothing to separating the models it is meant to rank. Each of these showed up in a suite built with more validation than most, and each needed its own measurement. The leakage audit, the mutation audit and the paired discrimination analysis are cheap: the audits need no model calls, and the analysis reuses sweeps a benchmark already runs. The obvious next step is to run them on HWE-Bench's 417 tasks, whose inputs include real bug reports and whose oracles are project regression suites written for purposes other than adequacy. We do not know what they would find there.

REPRODUCIBILITY STATEMENT

The anonymised supplementary archive contains the full suite (with-hints and hint-free variants), the harness, every instrument as a standalone script, and the committed JSON result for every number in this paper. Section 3 and Appendix E describe the protocol and the provenance of each arm;

Appendix C gives the mutation operators; Appendix B lists harness defects and their regression tests. Three make targets reproduce the work: one runs the unit tests without a toolchain, one re-verifies the validity invariants with Icarus Verilog 13.0, Verilator 5.048 and Yosys 0.65, and one regenerates every table and figure from committed results. All audits and the 1.5B-model arms run at no API cost.

ETHICS STATEMENT

This work involves no human subjects or personal data. The external benchmarks we audit are public, and we report their oracle weaknesses as measurements of design choices, alongside the same weaknesses in our own suite.

AI USE STATEMENT

In this work, we used generative AI tools to generate the benchmark tasks, to write and run code, to analyse results, and to draft the text of this paper. An AI coding assistant, Claude (Claude Sonnet 4.6 for the initial version of the project, Claude Opus 5 thereafter), co-wrote all 81 tasks, including their buggy artifacts, testbenches, specifications and reference fixes, and the original difficulty labels, which we found carried no signal and do not use. It also co-wrote the harness, the audit and analysis scripts and the task-authoring tooling under the authors' direction, ran the experiment sweeps, and kept the lab notebook from which every reported number is traced; all but the first of the repository's 162 commits carry its co-author trailer. It drafted the manuscript prose from the authors' outline and evidence files. Because Claude Opus 5 and Claude Sonnet 5 are also evaluated models, we note this overlap in §9. The LLMs listed in §3 are the subjects of the evaluation, and their outputs are the data. The authors chose the research questions, made the framing and inclusion decisions, and decided which claims to retract. Every task was checked mechanically by the invariants in §3 and the audits in §4–6, the code is covered by an automated test suite, every number in the paper was checked against its committed result file, and the authors reviewed the full text. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

REFERENCES

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Fan Cui, Hongyuan Hou, Zizhang Luo, Chenyun Yin, and Yun Liang. HWE-Bench: Benchmarking LLM agents on real-world hardware bug repair tasks. *arXiv preprint arXiv:2604.14709*, 2026.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

René Just, Darioush Jalali, and Michael D. Ernst. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *International Symposium on Software Testing and Analysis (ISSTA)*, 2014a.

René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. Are mutants a valid substitute for real faults in software testing? In *ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 2014b.

Derrick Lin, James Koppel, Angela Chen, and Armando Solar-Lezama. QuixBugs: A multi-lingual program repair benchmark set based on the Quixey challenge. In *SPLASH Companion*, 2017.

Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. VerilogEval: Evaluating large language models for Verilog code generation. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023.

Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. RTLLM: An open-source benchmark for design RTL generation with large language model. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024.

OpenAI. Introducing SWE-bench Verified. `https://openai.com/index/introducing-swe-bench-verified/`, 2024.

YunDa Tsai, Mingjie Liu, and Haoxing Ren. RTLFixer: Automatically fixing RTL syntax errors with large language models. In *Design Automation Conference (DAC)*, 2024.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. LiveBench: A challenging, contamination-free LLM benchmark. *arXiv preprint arXiv:2406.19314*, 2024.

Andreas Zeller and Ralf Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, 28(2):183–200, 2002.

A DEFECTS FOUND IN THE HAND-BUILT SUITE

Table 3: Defect classes among the 23 defective tasks of 52 in the pre-validation suite (22 of 42 toy, 1 of 10 design-derived). Several tasks had more than one defect, so counts overlap. Every task listed had its testbench, artifact, toolchain flag or reference fix changed in the repository history, except the one excluded non-bug.

Defect class	Tasks
Not a bug (artifact already correct)	3
Testbench or artifact does not compile	6
Expected values no implementation can produce	4
Inverted oracle (passes only a design lacking the tested feature)	1
Weak oracle (cannot detect the task's own bug)	3
Evaluation race at time zero	7
Reset or stimulus race	3
Mis-timed sampling	1
Undocumented second bug in the artifact	3

The inverted oracle was in a two-stage pipeline with a stall input. Its checks required stage two to advance during a stall and then receive the value presented during the stall. We ran three designs against it: one that followed the specification, one that stalled only stage one, and one that ignored the stall signal. Only the last passed.

B HARNESS DEFECTS

Each defect below was found by inspecting agent artifacts, not by a summary number, and each has a regression test.

1. **Patch drift.** `write_patch` applied hunks at their claimed offsets without checking context. Fixing it moved Sonnet 5 from 0.397 to 0.825 pass@1 on the 63-task suite of the time (with hints).
2. **Silent no-op patches.** Malformed diffs applied zero hunks and reported one file changed.
3. **Argument rejection.** Tool dispatch raised on undeclared optional arguments, so `read_file` with a line range failed and the agent never saw the code.
4. **Cost accounting.** Per-call token logs were never summed, so every run reported zero cost.
5. **No HTTP timeout.** One request stalled for 9.5 hours. Requests now time out at 180 seconds and progress is checkpointed per sample.
6. **Rate limits scored as failures.** Rate-limited calls were recorded as failed tasks rather than retried.
7. **Crash recovery that crashed.** The checkpoint code added for item 5 referenced an unimported name, and a 2h41m DeepSeek run died at its first checkpoint with nothing saved. The linter flagged it, but had been failing on 47 older errors for weeks, so nobody looked. All 47 are cleared.
8. **Billing failure scored as model failure.** Described in §4.

Three further defects would have invalidated results rather than lowered them: reference fixes were readable by agents through `list_files`; delta debugging minimised some failures into empty module shells that failed only because outputs were undriven (fixed by requiring identical testbench output before and after reduction); and deduplication used Python's per-process salted hash, so it was not reproducible across runs.

C MUTATION OPERATORS

Every single-site application of the following is generated on each reference implementation. Comparisons using <= are mutated only inside parenthesised conditions, never as nonblocking assignments.

- **Operator**: $+ \leftrightarrow -$; $== \leftrightarrow !=$; $\&\& \leftrightarrow ||$; $< \leftrightarrow >$; $\geq \rightarrow >$ and $\geq \rightarrow <$; $\hat{} \rightarrow \&$, $\& \rightarrow |$, $| \rightarrow \hat{}$; $\ll \leftrightarrow \gg$.
- **Sized literal**: perturbation of the value of a sized constant such as `8'h71`.
- **Bit index**: shifting a single-bit select, for example `[31]` to `[30]`.
- **Clock edge**: `posedge` $\leftrightarrow$ `negedge`, and active-low reset polarity inversion.

Table 4: Mutation score by class (numeric version of Figure 1).

Mutation class	RTLLM	Fail2Bench	VerilogEval
Operator	62.4%	79.9%	88.8%
Sized literal	65.1%	64.6%	90.8%
Bit index	54.9%	41.5%	97.6%
Clock edge	22.6%	19.1%	96.9%
All mutants	52.9%	61.6%	91.8%
Oracle design	directed	directed	differential
Tasks scored	39 / 50	78 / 81	124 / 156

Adapting the external suites. VerilogEval's testbenches reference a signal inside `$dumpvars` before declaring it, which Icarus Verilog 13 rejects; we remove only the waveform-dump block, which cannot affect the mismatch counters that decide the verdict. Tasks whose unmutated reference does not compile are excluded, never scored. RTLLM testbenches print verdicts in three formats. Our first pass recognised only two and reported 31.2%, because caught mutants were scored as survivors; all three formats are now enumerated from the repository and pinned by a test.

Two-defect tasks. We attempted to derive harder tasks by adding a second, independent mutation to each artifact. Only 14 of 81 tasks yielded a second defect that the existing oracle detects, which is the same weakness seen from another direction.

D TAXONOMY

Table 5: Controlled taxonomy over the 81 evaluated tasks. The original free-text labels had 35 bug-class and 19 category strings, with the two axes mixed.

Bug class	n	Category	n
missing-logic	12	arithmetic	10
width-or-sign	7	control-logic	8
wrong-operator	6	bus-protocol	7
off-by-one	6	datapath	7
protocol-violation	4	encoding	7
bit-selection	4	sequential-control	5
priority-error	3	memory	4
timing-structure	3	arbitration	2
wrong-constant	3	clocking	2
state-machine	2		
reset-error	2		

E ARMS, PROVENANCE AND ADDITIONAL RESULTS

Arms. Every reported arm is over the same frozen 81 tasks. Hint-free: Opus s=2 (recovered from saved artifacts; no cost record), Sonnet s=3 ($3.48), DeepSeek s=3 ($1.80), Gemini s=3 ($2.27), qwen2.5-coder-1.5B s=3 (local). With hints: Opus s=1, Sonnet s=5, DeepSeek, Gemini and qwen s=3. A recovered arm is less preferred than a clean live run because it includes tasks the live run would have abandoned at the wall-clock limit; we flag it wherever it appears. Two further runs were quarantined and are not reported.

Rule-based baseline. A pattern-matching repair agent with no model calls solves 3 of 81 tasks (3.7%).

Pass@k with hints. Sonnet with the answer stated, five samples: pass@1 0.956 [0.911, 0.990], pass@3 0.970 [0.933, 1.000], pass@5 0.975 [0.938, 1.000]. These are with-hints numbers and are not comparable to Table 2.

Contamination probe. The model is shown the first 40% of a reference implementation and asked to continue; we score exact line recall on the held-out remainder. Of 81 designs, 57 were long enough to score. Mean recall was 18.2%, five designs reached 50% (all with 9 to 18 held-out lines), and the longest run of consecutive verbatim lines anywhere was five.

Retracted findings. For transparency, claims we reported internally and withdrew after further measurement: a significant terse-oracle effect (§6); a threefold increase in discriminative power from withholding the answer (§7); partition-level hint effects; a 22.4-point hint effect for Gemini, which came from an incomplete arm; a 69.2% mutation score that excluded clock-edge mutants; a gap between toy and design-derived tasks; and the predictive value of LLM-assigned difficulty labels. None appears in any table, and a test in the artifact enforces that.