How to Audit a Benchmark

Eleven audit passes' worth of sins, ordered by how cheaply they're caught.

June Kim
Independent Researcher
june@june.kim
ORCID 0009-0005-3153-9396

July 10, 2026

Over the past few months I audited ten benchmarks in eleven audit passes: SWE-bench Verified (as a runner), DeepSWE and its revision, SWE-bench Pro, ProgramBench, τ-bench's contamination story, Terminal-Bench, MirrorCode, FrontierCode, Frontier-Bench, and SlopCodeBench. They broke in different places, and DeepSWE, re-audited after its revision, mostly held. This post compresses the lenses into a checklist a stranger can run.

Disclosure: I applied for a role at Epoch AI, which co-produced MirrorCode, one of the audits drawn on here. An interest disclosure, not a funding one; it changes none of the receipts, which is the point of check 4 below.

audit	broke at	finding
SWE-bench Verified, as runner	the run itself	test-edit exploit caught; 44 unrunnable tasks scored zero
DeepSWE	gold	4 of 113 answer keys fail their own verifiers
DeepSWE v1.1	mostly held	determinacy floor 2.7%; verdict receipts still unretrievable
SWE-bench Pro	spec	proven floor of 15% underdetermined
ProgramBench	oracle	21 recall-only witnesses; 29 self-capturing goldens
τ-bench	decay	memorized answer passes 0% regenerated, leaked query 100%
Terminal-Bench	frame	83 of 83 pass after off-task destruction
MirrorCode	claim	title sells autonomous whole-program builds; metric scores scoped reimplementation from a live oracle, 17/25 targets contaminated
FrontierCode	claim	sold as mergeability; of 59 coded closures, the patch decided 3
Frontier-Bench	frame	agent container torn down before grading; 9 of 9 pass after off-task destruction
SlopCodeBench	claim	static code-shape drift sold as extension robustness; iteration confounded with expanding scope

A benchmark is a measurement instrument, and an instrument makes a contract. Above every clause sits the claim: the capability the headline advertises has to be the one the metric measures (the claim clause), or the number is precise about the wrong thing. The instruction pins the target (the spec clause). The grader checks the target (the oracle clause). The grader guards everything the task never named (the frame clause:

a run that completes the task by wrecking unrelated state must not pass). The answer key passes its own test (the gold clause). The headline number means what the leaderboard says it means (the score clause). And the number survives its own publication (the decay clause).

Each clause can be broken, each break is a distinct sin, and each sin has a check. The checks below are ordered by cost, and the order carries the lesson: every severe defect in these audits surfaced before any model ran. Cost here means compute and cash; several of the free checks still cost an afternoon of reading.

The clauses are not independent, and that is the one thing this list got wrong for a while. A mitigation for one clause can be a regression in another. Frontier-Bench runs every task with the verifier in its own container, which kills a real attack where agent-produced code double-forks a daemon and overwrites the reward file after the tests write it. It buys that by tearing down the agent's container before the verifier starts, so the state a frame check would read no longer exists. Sound oracle-integrity fix, straight subtraction from the frame. Neither the rubric nor the review pipeline registers the trade, because nothing scores a clause against its neighbours. When a maker hardens one clause, ask which other clause paid for it.

#	check	cost	sin	exemplar
1	read the claim against the construct	free	claim	title sells reconstruction, metric scores recall
2	read the paper against the repo	free	wrong rulebook	a week and $1,000 on the wrong task
3	ask how tasks were selected	free	selection by failure	0% floor read as a frontier
4	recompute the headline	free	score	footer says 113, mean divides by 111
5	retrieve one receipt	free	unfalsifiable verdicts	`has_model_patch: true`, every path 404
6	run the answer key	~$1	gold	4/113, 3/731, 6/89 golds fail
7	read what the assertions pin	cheap	oracle	21 recall witnesses, 29 self-capturing goldens
8	probe what pins the graded value	cheap	spec	15% proven underdetermined
9	mutate the gold and regrade	cheap	frame	`rm -rf .git` still scores 1
10	ask what survives publication	reading	decay	memorized 0%, leaked query 100%
11	audit the run	varies	run	restored tests bought 46 points

1. Read the claim against the construct

Free, and it catches the deepest sin, the one every check below can pass while the benchmark still misleads: the title advertises a capability the metric does not measure. Read the headline, then read what the grader rewards, and see whether they name the same thing. ProgramBench sold reconstruction from behavior and scored recall of published algorithms. MirrorCode sells "the largest software project AI can complete on its own" and scores scoped reimplementation from a live reference oracle, two thirds of its targets showing memorization. Neither number is miscomputed; each is precise about something narrower than the title names. That is a construct-validity gap, not a bug: the grader is sound on what it checks, and what it checks is not the claim. The tell is an equivocation you can quote, a headline in one register beside a methods section in another, and it survives the mechanical checks below because they audit the instrument, not the advertisement.

2. Read the paper against the repo

Free, and it catches the sin of the wrong rulebook. The scoring rules often live in the paper while the runnable code implements something looser. On SWE-bench Pro I lost a week and about a thousand dollars solving the wrong task because the held-out-test rule was on pages 4 through 9 of the paper and nowhere in the repo. A surprising number is a stop sign, and the first thing to check when you see one is whether you and the authors are playing the same game.

3. Ask how the tasks were selected

Free, because the selection procedure is in the paper's construction section, and it catches a sin that inflates difficulty and defect rates in the same stroke. A benchmark that discards every problem current models solve reliably has selected on failure, and failure enriches for broken tasks, because an underdetermined spec, an unsolvable subtask, and a mis-keyed grader all present identically as "no model passes." A near-zero headline is then ambiguous between hard and defective, and nothing downstream separates them. ProgramBench's zero-percent floor across nine models read as a capability frontier until the recall witnesses showed it was a gate. Selection by failure also anchors the benchmark to one model generation, so the difficulty claim expires with the models it was screened against.

The check: was any task demonstrated solvable by something other than its own author? A human baseline is the strongest answer, and humans-passing-but-models-failing is the highest-signal result a benchmark can produce. Without one, solvability rests on nothing beyond the authors' belief.

Selection cuts both ways, so check the other direction too. A benchmark that selects away from the cases it cannot grade fairly, the codecs and hashes no source-blind solver reconstructs, has done real work: MirrorCode left two such targets where ProgramBench had twenty-one, and that is a credit to record, not a defect to find. And when the headline is a human comparison, "a task that would take a human weeks," ask whether that number was measured or believed. MirrorCode's was believed, four contributors' estimates with a sevenfold spread and no completed baseline. You can often falsify it from public record: the targets were open-source programs, and their git histories, commits and contributors and calendar span, anchor the real human labor without asking anyone.

4. Recompute the headline

Free, and it catches score sins, the family where the number stops meaning what the board says. Divide the shipped per-task results by both plausible denominators and see which one the headline used. DeepSWE's footer announced 113 tasks while the headline mean divided by 111. Count the exclusions and check they're disclosed: DeepSWE excluded 122 trials and mentioned 73.

Then check the statistics against what the data can carry. Confidence intervals that treat forty trials of one task as forty independent samples are narrower than the truth. A directional claim resting on one run over ten tasks has no variance behind it. Wall-clock reported as a metric measures the fleet's load as much as the model. A board that ranks model-and-effort pairs while presenting itself as ranking models is comparing configurations, and the reader can't tell. None of this needs a model or a rig. It needs division.

5. Retrieve one receipt

Free, and it tests whether the benchmark is falsifiable at the verdict level. Pick one scored trial and try to reach the artifact behind it. On DeepSWE, `has_model_patch: true` was a flag, and every path to the patch itself returned 404. A verdict whose receipt cannot be retrieved is a promise, and a benchmark built of promises is not an instrument anyone can check, including its authors.

While you're there, look for the conflicts-of-interest statement. None of a producer's commercial relationships are conflicts when the artifact is marketing; each of them is a conflict when the artifact is asked to be science.

6. Run the answer key

Costs about a dollar, and it has caught a defect in every benchmark I've pointed it at. Apply each task's own reference solution and run the task's own grader. That's the whole check. DeepSWE: 4 of 113 golds failed their own verifiers. SWE-bench Pro: 3 of 731. Terminal-Bench 2.1: 6 of 89.

An answer key asserted correct but never run against its own test is a confabulation, a plausible artifact nobody checked, and a task whose gold cannot pass cannot anchor any verdict built on it. A panel of annotators signing off on the reference solutions does not substitute, because attestation is reading and the defect only shows under execution; every failing gold above shipped from a benchmark whose authors believed it correct. This is the highest-yield dollar in benchmark auditing, it requires no model, and it doubles as the pre-ship check the makers should have run.

When golds fail, quarantine rather than adjudicate. Some failures are your harness's fault (my first DeepSWE run failed all 113 tasks at once, which is never 113 independent defects; it was one missing Docker plugin on my side). Exclude the failures from your denominator, report them as unresolved, and let the maintainers sort artifact from defect. Quarantine what you never attempted too, and label it as such: a task you skipped

because your host lacked the disk or the architecture is not a task whose gold failed, and collapsing the two moves your own limits onto the maker's ledger.

Before running it, check whether they already do. Frontier-Bench instructs every runner to execute the oracle five times before trusting a result and validates it in CI on every task PR, which is the first time this check has come back empty by policy rather than by luck. That is the outcome the check exists to produce, and a maker who has internalized it deserves the credit in writing.

7. Read what the assertions pin

Cheap, and it catches oracle sins, where the grader checks the wrong thing. Open the tests and classify every exact-value assertion by one question: how would a solver without the reference obtain this value? ProgramBench sorted into three bins. Discoverable values come from the materials at hand. Brute-searchable values live in a finite space too large for the budget. Recall-only values are the output of a hash, a cipher, a codec, something no observation reproduces, and one such value behind a conjunctive metric puts the whole task beyond the stated construct. Twenty-one ProgramBench programs carried a verified recall witness, which is why a zero-percent resolve rate across nine models measured recall rather than the reconstruction the paper claimed.

The same read catches goldens that encode one implementation's incidentals where the contract left the output free, and the eeriest specimen in the family, the self-capturing golden, a graded test that writes its own answer key when the reference file is missing. ProgramBench had at least 29. A test must not author its own oracle.

Then ask a question one step earlier: where do the verifier's inputs come from? On a benchmark that grades in a separate container, some of them arrive from the run and some are baked into the verifier's own image, and the tests look identical either way. In Frontier-Bench's `cargo-flight-dispatch` the verifier image copies its own pristine `/app/data/`, then executes the agent's code against that copy. The verdict is a function of the artifact and a restored world rather than the world the run actually left. Damage to those inputs is not merely unnoticed; it is undone before grading. Call it environment reconstitution, and treat it as the self-capturing golden's structural cousin: in both, the grader supplies the conditions of its own success. Its census is cheap, since the reconstituted paths are visible in the verifier's Dockerfile.

8. Probe what pins the graded value

Cheap and mostly mechanical, and it catches spec sins, where passing means recovering the author's unstated choice rather than solving the stated problem. The determinacy audit runs in two tiers. The mechanical spine: a model proposes what to grep for, the exact file and string a from-codebase solver would need to find, and you run the grep yourself; when the graded constant appears nowhere outside the gold patch and the hidden test, no reading of the materials produces it, and the verdict rests on the grep, so the tier carries no model judgment. That alone proved 11.4% of SWE-bench Pro underdetermined.

The judgment tier: one model family constructs two faithful readings of the prose with verbatim spans, an independent family tries to refute the split, and only survivors count. Together, a proven floor of 15% on Pro; the same instrument put DeepSWE v1.1's floor at about 2.7%, which is what a tight benchmark looks like.

The classes are worth memorizing because they recur everywhere. Airtight: the graded constant exists only in the gold and the test. Misdetermined: the codebase determines one value and the test grades another. Plural: the codebase or the prose licenses two readings and the test pins one silently. Claim underdetermination only on positive evidence, never on a failure to find a convention, and report a floor rather than a rate.

9. Mutate the gold and regrade

Cheap, model-free, and it catches the frame sin, the one clause almost no benchmark writes down: the grader must guard what the task never named. Take the reference solution, append one careless accident, `rm -rf .git` or a deleted file the solution never touched, and re-run the official grader. On Terminal-Bench 2.1, 40 of 83 gold-passing tasks still scored 1 after a careless deletion inside their own workspace, and all 83 passed after off-task user assets were wiped, an outcome final-state grading entails. The reward ordering that falls out is the sin in one row: a destructive completion scores 1, a safe failure scores 0, and the outcome users fear most outranks the harmless one.

Instrument the accident, not just the verdict. An accident that leaves no evidence it occurred is indistinguishable from one that silently no-opped, and both look like a pass. Have the mutation write a witness recording the state before and after, carry it out through the harness's own artifact path, and void any verdict whose witness is incomplete. Two of my mutations ran against tasks whose shipped `solve.sh` had no trailing newline, so the appended accident concatenated onto the last command and aborted the solution; without a witness those read as the grader catching damage that never happened.

There are two grades of this sin, and the newer one is worse. On Terminal-Bench the frame was unasserted: the state sat in the container and no test looked at it, which is why 83 runs were needed to establish it. On a benchmark that tears down the agent's container before grading, the frame is unavailable: no test can reach it however well written, so the property follows from the interface and a handful of runs demonstrate rather than sample it. When the frame is unavailable, the check itself has to move. The mutation still goes in the same place, but the observation has to happen before teardown, through whatever pre-teardown hook the harness offers.

Then look for the exception and invert it. Terminal-Bench had exactly one task with a hand-written frame check, and it was the one task where the frame collapsed to a single `git diff`. Frontier-Bench has exactly four tasks that carry the frame across the boundary, all git-backed with a pinned base, all adopted to cut artifact bloat rather than for validity. Checks appear where they are cheap, not where they are needed, and the motivation need not be correctness at all. So find the task that does it right, work out what made it affordable there, and the complement is your coverage gap. In Frontier-Bench that complement is the 36 of 74 tasks declaring a single artifact path, which have no bloat to motivate the fix and no less capacity to wreck their environment.

The frame check generalizes to any benchmark that grades environment state, and the fix generalizes too: manifest the world at agent handoff, diff the final state against it, and gate the delta on the reference solution's own footprint. The oracle the benchmark already ships is the frame it never wrote. Since separate-verifier isolation is where harnesses are heading for good security reasons, this is a harness-architecture finding more than a benchmark one, and the capture belongs in the runner where every dataset on it inherits the fix.

10. Ask what survives publication

A reading of the design, no run required, and it catches decay sins. Once instances reach the public web, a high score is ambiguous between getting better at the task and getting better at remembering the answer, and regeneration only re-prices that ambiguity when the scored target co-varies with the regenerated world. If the correct answer is a fixed point of the whole grader family, regeneration does nothing, and memorizing it survives. So ask three questions. What exactly is scored, the state or the artifact? Would the scored value change on a reseeded instance? And against which adversary does the claim hold: a verbatim replayer, a memorizer with cheap transport, or a model fine-tuned on the generator itself?

The concrete case that separates the questions: on regenerated τ-bench worlds, a memorized final answer passes zero percent of instances while a leaked state-general query passes every one, because the query is what the grader scores and the query never changes. So state the conclusion scoped to an adversary, since "resists verbatim replay" and "resists fixed-answer memorization" are different claims, and a benchmark that cannot say which one its number makes hasn't priced its own decay.

11. If a score is the claim, audit the run

Everything above audits the instrument. When someone reports a number, the run is a second instrument with its own sins, and I committed most of them myself before learning to name them. Oracle leakage through the prompt: restoring held-out tests bought 46 points on a 50-instance sample, half the benchmark's apparent difficulty. Oracle leakage through the capture: grade a raw diff and you grade the agent's tampering with the tests.

The gate mistaken for the verdict: an agent's internal green light is a stop signal allowed to lie, and only the official grader in a fresh container decides. Infra laundering: re-runs granted after seeing which losses you'd like to excuse are a re-roll lever, so predeclare the fault classes with invariants before any re-dispatch. And your own denominator: unrunnable instances stay in it at zero, and losses get committed next to wins.

The auditor's own contract

The checks above are worthless without discipline on the reporting side, and the discipline compresses to a few rules that recur across the series.

Audit yourself first. Every campaign in this series began with a false positive of my own: a missing Docker plugin masquerading as 113 defects, my determinacy tool certifying test-fixture strings as authorial constants, a sentinel bug that silently voided 35 runs. The instrument you trust least should be yours.

Make the validity gate symmetric. When I added the witness check above, I first used it only to void suspicious findings, leaving suspicious exonerations to stand. That is backwards in the direction that flatters the auditor, and it hid a bug for two more tasks: a broken splice was scoring as the grader catching damage. A gate that fires on one verdict and not its opposite is not a gate, it is a thumb. Void both directions or neither, and publish the failing runs so a reader can tell which.

Report floors, never rates. A defect you can exhibit is a lower bound; the tier you couldn't adjudicate stays out of the count, and the number only grows with more search.

Make every claim a rerun. The warrant is the re-derivable receipt, the pinned image, the committed diff, the grader's own reward file, never the model's say-so and never yours. An audit is a rerun, so it has to be re-runnable to be an audit.

Bound the claim. "Three things hold, and no more" beats a diagnosis you can't support; cause-finding is the maintainer's job, and an auditor who adjudicates beyond the receipts is spending credibility the receipts didn't earn.

Give right of reply. Send the findings to the authors before or as you publish, link their response when it comes, and file the actionable part where they work, as an issue or a PR. The audits in this series that produced fixes are the ones that arrived as fixes.

And build the instrument so it can come back empty. An audit that always finds sins is a press release. The determinacy floor of 2.7% on DeepSWE v1.1 and the 26 Terminal-Bench tasks that caught the deletion are as much the method working as the failures are, because a check that cannot exonerate cannot convict.

Name the cure, not only the disease. When the finding is that a number measures the wrong thing, say what number would measure the right one, and the maker has something concrete to argue with instead of a diagnosis to wave off. For MirrorCode the cure is a preregistered profile, whole-task success and time-to-success against program size, in place of one collapsed percentage. Hold the proposed metric to the same standard as the audit: I floated a tidy speed-and-accuracy pair first, and it took a round of review to see the accuracy interval was leaning on a test-independence assumption the suites do not meet.

File the sin upstream

Two published metaevaluations already catalogue benchmark failure, and an audit should end by reconciling with them. McGregor and coauthors maintain BenchRisk, 57 failure modes with 196 mitigations scored across 26 benchmarks, plus a public registry that accepts new modes through issue templates. Reuel and coauthors maintain BetterBench, 46 best practices assessed across a benchmark's lifecycle. Both catalogues grew on chatbot benchmarks, where the pipeline runs prompt to inference to judged output. That pipeline has no answer key, no executable grader, and no environment, so most of the contract's clauses fall outside it; the BenchRisk paper defers agentic benchmarks to future work for this reason.

The reconciliation is one row per clause:

clause	nearest BenchRisk mode	status
claim	47: benchmark does not measure a property linked to the user task	adjacent; the mode faults the reader's inference, while the quotable equivocation in the title is unregistered
spec	none	hidden tests grading unstated choices are unregistered
oracle	25: ground truth placed within the system chain	adjacent; the mode covers SUT developers cheating, while recall-only witnesses and self-capturing goldens are unregistered
frame	none	filed from this checklist as BenchRisk#8
gold	none	answer keys failing their own graders are unregistered

clause	nearest BenchRisk mode	status
score	34–36, 57: sample size, uncertainty propagation and presentation	partial; miscomputed denominators and undisclosed exclusions are unregistered
decay	4, 21, 23, 42, 44, 46, 49, 50	covered; cite the mode numbers

So the checklist ends with a routing rule. When an audit surfaces a sin the registry already carries, cite the mode number, because the shared name is what lets maintainers and other auditors connect the finding to its siblings on other benchmarks. When it surfaces a sin the registry lacks, file it through the new-failure-mode template with the receipt and the cure attached, one filing at a time. The registry is where a finding outlives its benchmark: the verdict covered one artifact, the check covers every later one, and the filed mode is how the next auditor inherits both.

The bill

Every sin above was found with the benchmark's own shipped artifacts: its golds, its graders, its JSON, its tests. Nothing required privileged access, a big model, or a budget beyond a few dollars and the patience to read. That is the uncomfortable part for makers and the encouraging part for everyone else. The contract clauses are checkable at publication time, by the authors, for less than the cost of one leaderboard submission, and the cheapest one, run the answer key, would have caught a shipped defect in three of the benchmarks named here before anyone reported a score.