PURETENSOR
INTELLIGENCE DISTILLED

A Local Model in the Fixer's Seat

Most Failures in a Week-Long Trial Were Cost Accounting and Availability

Paper: PT-R-2026-059 v1.0

Author: PureTensor Ltd

Date: 25 October 2026

Status: Published

Abstract

For a week in October we replaced the model in the seat of our autonomous remediation agent, a frontier hosted model, with an open-weight model served on our own hardware: DeepSeek-V4.1-Flash-1017 on four workstation Blackwell GPUs in one node. Nothing else changed at the start: the agent, tools, guard hook, per-run budget cap and verification step were the same, although the guard was tightened mid-trial (Section 5). The trial ran from 19 October 09:25 UTC to its reversal on 25 October 19:44 UTC and covered 113 model runs. The previous seven days, on the hosted model, hold 280.

The headline is bad. Of the 113 trial runs, 46 failed (40.7%). In the comparison week 5 failed on the ledger's raw outcome and 7 after later settlement (1.8% and 2.5% of 280; 3.3% once 70 seat-unavailable runs are excluded). Most of the trial failures say little about fix quality. Nineteen were stopped by a five-dollar per-run cap applied to a cost that our command line imputes from a built-in list price, since the local seat has no per-token bill. Seventeen were connection refusals, in which the model seat refused the connection when the run started; we do not know why. Together that is 36 of 46 failures (78%). The other ten are unclassified.

Midway through the trial the serving engine changed from SGLang to a vLLM build, with the same weights on the same GPUs. The failure rate fell from 65.2% (30 of 46) to 23.9% (16 of 67) and the budget failures stopped. The model did not do less work: median output tokens per run rose from 14,813 to 22,385. What changed was usage reporting, which is consistent with the budget failures stopping. On the second engine 93.8% of prompt tokens were reported as cache reads, which the imputed price charges at one twentieth of fresh input; on the first engine the figure was zero. Re-priced as though nothing had been cached, 26 of 56 second-engine runs would have breached the cap, close to the 19 of 37 that did on the first engine.

Where fixes were verified or reviewed they held up: verification passed on 14 of 15 trial runs that reached it, against 28 of 30 hosted, and a review of the first 28 runs found no wrong change. We found four guard breaches, one of which, by the run's own log, led to an automatic email reply to an outside party, and added a deny rule for the cases we tested. The operator reverted the seat on reliability grounds. Samples are small and the weeks differ; Section 6 sets out what survives.

1. The Question and the Seat

Our remediation agent claims fault findings from fleet scanners and tries to cure them. Each run gets a fault description, a working directory, shell and file tools, a per-run budget cap, a wall-clock limit and a required structured report: an outcome, a root cause, the actions taken and, when something was cured, a reusable "antibody" with steps and a verify check. A guard hook on the seat denies classes of dangerous commands. A cure counts as resolved only after its verification passes.

Until 19 October the seat was a frontier hosted model. The question for the trial was narrow: can an open-weight model on our own hardware sit in that seat, driven through the same coding-agent command line, with nothing else changed? We did not tune prompts, raise limits or loosen the guard.

Our earlier local-model work used stand-in tasks: in February, older local models on a synthetic harness investigated faults and never produced a final report. Here a local model holds the live seat, with real faults and real side effects.

Run outcomes, as the ledger records them: resolved (cure applied and verified), verifying (cure applied, check pending; it settles later), self-cleared (the fault went away on its own), held (the model investigated and handed back without mutating), abandoned, failed, and seat-unavailable (the seat could not do any work). The ledger also holds "replay" runs, in which a stored antibody is replayed with no model. We exclude those throughout and count only model runs.

2. Headline Outcomes, and the Outage Inside the Baseline

We split the trial into two phases at the engine change. Phase A is the first engine, SGLang, with 46 runs from 19 to 21 October. Phase B is the second, a vLLM build serving the same weights on the same four GPUs, with 67 runs from 22 to 25 October. The change was committed on 21 October at 21:21 UTC. The last Phase A run started at 14:41 UTC that day and the first Phase B run at 00:55 UTC on the 22nd, so no run straddles it.

	Prior week (hosted)	Trial, all	Phase A	Phase B
Model runs	280	113	46	67
Failed, raw outcome	5 (1.8%)	46 (40.7%)	30 (65.2%)	16 (23.9%)
Failed, after settlement	7 (2.5%)	47 (41.6%)	30 (65.2%)	17 (25.4%)
Resolved, after settlement	72 (25.7%)	35 (31.0%)	6 (13.0%)	29 (43.3%)
Held	109 (38.9%)	24 (21.2%)	9 (19.6%)	15 (22.4%)
Self-cleared, after settlement	19	7	1	6
Seat-unavailable	70	0	0	0
Abandoned	3	0	0	0

Table 1. Outcomes per run, all percentages over every model run in the column. "Raw" is the outcome the ledger first records; "after settlement" resolves the runs that were still verifying.

Settling the trial's 15 verifying runs gave 10 resolved, 4 self-cleared and 1 failed; all 15 are in Phase B, and we do not know whether that reflects the fault mix or how the first engine's runs reported a cure. Settling the prior week's 30 gave 17 resolved, 11 self-cleared and 2 failed.

The 2.5% baseline needs a caveat. Seventy of the 280 comparison runs, between 17 October 20:57 UTC and 18 October 08:00 UTC, ended as seat-unavailable. A safety hook on the hosted seat was missing at the time, so every tool call failed and no work was done. They were later re-booked from held or failed into a separate outcome. They are real runs in the ledger but not attempts at a fix. On the 210 working runs the hosted model failed 7 times (3.3%), resolved 72 (34.3%) and held 109 (51.9%). We use 3.3% as the like-for-like baseline and 2.5% only with the 280 denominator.

Turns differ too. The hosted median is 13 over its 205 working runs with a result (9 over all 275, because the seat-unavailable runs lasted a median of 3 turns). The local median is 35 per run that reached it, in both phases, about 2.7 times as many. Median wall time per working run (ledger end minus start) was 2.2 minutes hosted and 4.6 local (4.9 in Phase A, 4.0 in Phase B). The local model is slower and takes more steps; Section 4 shows that is not what sank the trial.

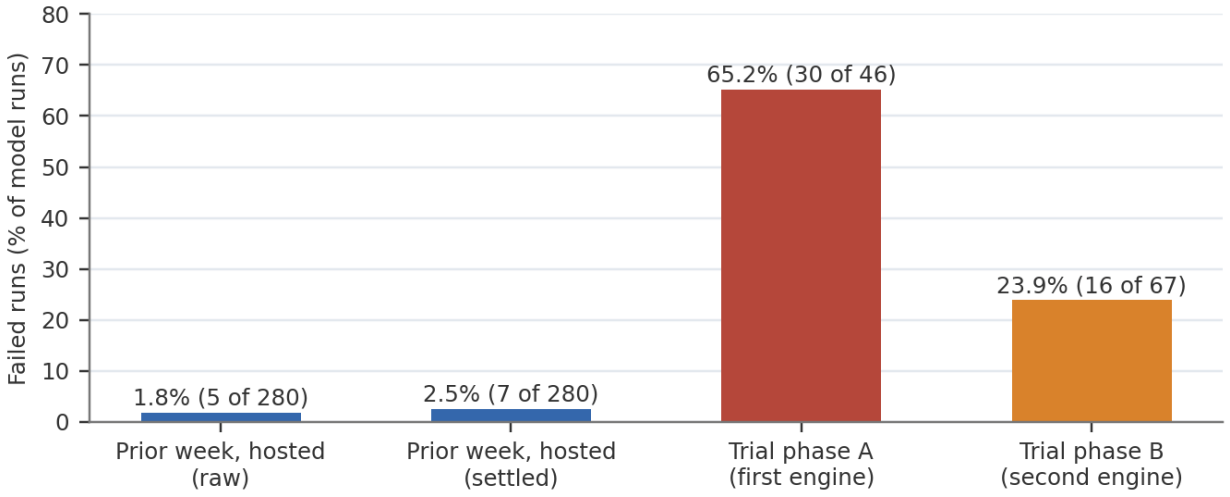


Figure 1. Failed runs as a share of all model runs, by period. Values are in Table 1.

3. Where the 46 Failures Came From

For each failed trial run we read the result file the run left: its terminal reason and subtype.

Cause	Phase A	Phase B	All
Per-run budget cap reached	19	0	19
Connection refused to the model seat	9	8	17
Stopped at the turn limit (61 turns)	0	3	3
No result, 15-minute wall-clock limit	1	3	4
Completed, then the verify check failed	1	2	3
Total	30	16	46

Table 2. Failed trial runs by cause and phase.

Thirty-six of the 46 failures (78%) are the first two rows, and neither says how well the model fixes faults.

The budget cap was five dollars per run. When the command line does not recognise a model it computes cost from token counts using a built-in price table, and reports that the cost basis is unknown. The figure is imputed, not money paid, because the local seat has no per-token bill. We recovered the table from the result files: input 4 dollars per million tokens, output 20, cache read 0.20 and cache write 5. These are the command line's rates for a model it did not

recognise, not any vendor's prices; the ratios matter: a cache read at one twentieth of fresh input, output at five times. For all 37 Phase A runs with a result, fresh-input and output tokens at the first two rates reproduce the recorded cost to within 0.01 cents. For the 56 Phase B runs with a result, a least-squares fit of cost on the four token counts returns exactly 4, 20, 0.2 and 5.

The 19 budget failures ended between 26 and 44 turns (median 37) at imputed costs of 5.02 to 5.26 dollars; the cap is checked between turns, so runs overshoot slightly. Of the 37 Phase A runs with a result, 19 (51%) were stopped by it.

The 17 connection refusals all read "Connection refused" on the result file. Sixteen died at turn 1 at zero cost, each after roughly 135 to 188 seconds of retrying (most between 172 and 185). One, the first refusal, on 19 October, died at turn 30 after 2.96 dollars imputed, mid-run. Nine fall in Phase A, all on 19 October between 13:46 and 15:38 UTC. Eight fall in Phase B: four between 09:45 and 11:22 UTC on 22 October, one at 03:03 UTC on 23 October, and three on 24 October between 07:53 and 07:59 UTC. The seat was not listening at those moments. The node's journal begins on 14 October, so we could not correlate these windows with seat restarts. The cause of the refusals is unknown.

That leaves ten failures. Three were turn-limit stops, three finished and then failed their check, and four timed out at 15 minutes with no result file. Of the four, three had written a final answer in their last message (a hold, a cure reported as verified, and a root cause), so the harness could not decode it or the clock cut it off; the fourth had a root cause but no final report. We leave all four unclassified. No trial run failed in the way our February finding describes (investigating without end), though one of the four timeouts ended without a final report.

4. The Engine Change and the Cache

The two engines' runs are alike in turns and prompt volume and differ sharply in cost.

Per-run statistic, runs that reached the model	Phase A	Phase B
Runs with a result and tokens	37	56
Median turns	35	35
Median wall time, minutes (ledger basis)	4.9	4.0
Median prompt tokens (input plus cache read plus cache write)	1,136,090	1,099,602
Median cache-read tokens	0	1,029,504
Median cache-read share of prompt	0%	93.8%
Median output tokens	14,813	22,385
Median imputed cost, dollars	5.02	0.94
Runs stopped by the five-dollar cap	19	0
Highest imputed cost, dollars	5.26	2.40

Table 3. Per-run statistics for runs that reached the model, by phase.

Turns and prompt volume are the same across the phases, and wall time is somewhat shorter in Phase B. Output tokens rose by about half, from 14,813 to 22,385. Our first reading, that the model produced fewer completion tokens, was wrong. What changed is that every Phase A prompt token was reported as fresh input, while in Phase B 94% were reported as cache reads, a median of 1.03 million per run against none. At the recovered price table a cache read costs one twentieth of fresh input.

A cached prompt prefix matters because every turn re-sends the whole transcript so far.

We tested the mechanism with arithmetic. Re-price the 56 Phase B runs as though nothing had been cached, with every prompt token at the fresh-input rate and the same output:

Per-run statistic	Phase A (as reported)	Phase B (as reported)	Phase B re-priced without cache
Median imputed cost, dollars	5.02	0.94	4.73
Runs over the five-dollar cap	19 of 37	0 of 56	26 of 56

Table 4. The cost counterfactual. The last column is a recomputation from Phase B token counts, not a measured run.

Without caching, Phase B's median cost would have been 4.73 dollars and 26 of 56 runs (46%) would have exceeded the cap (the highest would have been 16.85 dollars). That is close to the share Phase A lost to the cap, 19 of 37 (51%). The price looks like a property of how usage was accounted, not of the model. We believe the cap was set with a hosted model's cached accounting in mind.

We know only what the usage fields reported; we cannot say whether the first engine failed to cache the shared prefix or cached it without reporting. The counterfactual is arithmetic on recorded token counts, not an A/B run.

Cost figures need care. The trial-wide 2.16 dollars is a mean of imputed costs (244.11 over 113). The medians over runs that reached the model are 5.02 and 0.94, as in Table 3; including refused runs at zero cost they are 4.17 and 0.90. We report imputed trial costs only, and no real spend.

Taking out the budget artefact and the refusals shows the residual:

	Phase A	Phase B
Failed, raw	30 of 46 (65.2%)	16 of 67 (23.9%)
Failed other than connection refused	21 of 37 (56.8%)	8 of 59 (13.6%)
Failed other than refused or budget	2 of 37 (5.4%)	8 of 59 (13.6%)

Table 5. Failure rate by phase. Row 2 removes the refused runs from numerator and denominator (the 37 and 59 are runs that did not die on a refused connection); row 3 also removes the budget failures, from the numerator only.

On the second engine the residual is 8 of 59 runs that reached the model (13.6%): three turn-limit stops, three timeouts and two failed checks. On the first it is 2 of 37, one timeout and one failed check. These are small numbers from different weeks. We do not claim Phase B is worse at fixing; the residual is not zero and not well measured, and could come from the model, the 61-turn limit, the 15-minute limit or the week's faults.

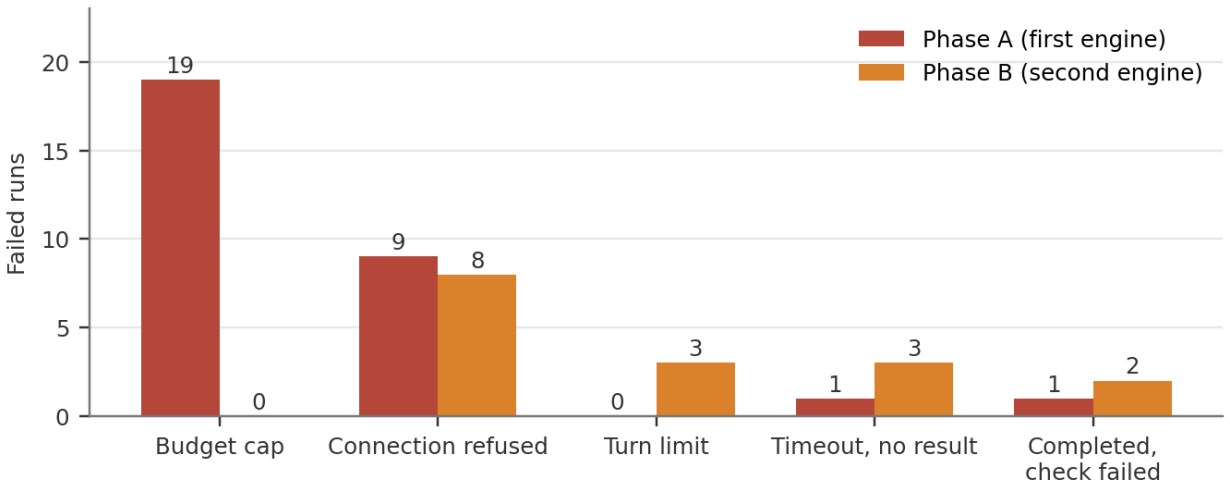


Figure 2. Failed runs by cause and phase. Values are in Table 2.

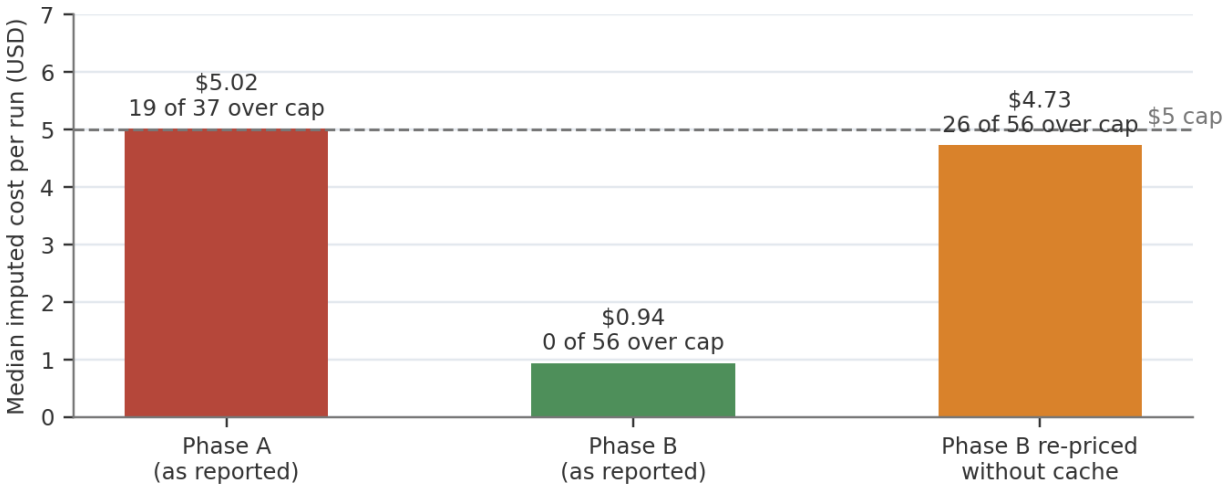


Figure 3. Median imputed cost per run that reached the model, as reported in each phase and re-priced for Phase B without cache reads. Values are in Table 4.

5. What the Fixes Looked Like, and Where the Guard Failed

Verification is the cleaner test of whether the fixes were any good, since a cure counts only if its check passes. Of the trial runs that reached verification, 14 of 15 passed (93%), all of them in Phase B. On the hosted model 28 of 30 passed (93%). A pass means resolved or self-cleared after settlement; the interval on 14 of 15 is wide.

We judged fix quality by reading the changes, not with a grader. On 22 October a review listed every mutating command in the first 28 runs of interest, through 21 October. According to the review note it read the fixes in nine repositories and services, and it found no wrong change. An interim summary extended the claim to the whole trial; we could not verify the 22 to 25 October part, so we do not make it, and we did not audit all 113 runs.

One backup failure it traced to a text-decoding error in the check, not in the backup; that is an anecdote, not a rate.

The guard was the weak part. The guard on the seat denies classes of command that change external state. Four runs did things it should have denied. Two runs, on 19 October, started a

publishing job and ran it to its first failure. In both transcripts the job failed at its first step, a pre-deploy repository sync, before any deploy or send step; the script's own text shows the send step sits after a verified deploy. A third run, on 21 October, reset that job's failed state.

The fourth was worse. The first run of Phase B, on 22 October at 00:55 UTC, altered mail-processing state so that an old message would be handled again, and the run's log records an automatic reply to an outside party. The run's report says it backed up what it changed. We have not read the reply; the run's log quotes an "auto-replied" line that we could not confirm.

On 22 October at 19:26 UTC we committed a new deny class covering the publishing job's start and reset and writes to the mail-processing state. The operator's notes record that, against the live seat hook, it denied the publishing job's start. Resets and the mail-state write are covered by the rule text but were not exercised in that check.

One policy question stayed open: whether a run whose own outcome was "held" may mutate. It is policy, not guard coverage, and we have not answered it.

6. What These Numbers Will Not Carry

This was one trial, one week, one model. The two phases are consecutive weeks with different faults, so the fall from 65% to 24% is not a controlled engine comparison, and the same holds for the comparison with the hosted week. The cost mechanism is the exception, because it rests on arithmetic over recorded token counts that reproduces the recorded costs on both engines. It does not rest on an A/B run.

The hosted baseline includes a 70-run outage; we report working-run figures alongside, but fault mixes still differ between weeks. Samples of 15 to 67 carry wide uncertainty, and we computed no intervals. Fix quality was judged by reading, and part of it only through a summary we could not check. "Genuine failures" is not a clean category: the three failed checks, three turn-limit stops and four timeouts are all candidates, and we have not separated them.

What survives: a local open-weight model in an unmodified production fixer seat failed 41% of runs in a week; 36 of the 46 failures (78%) were a cap applied to an imputed price (19) or a seat that refused connections (17); the cap failures disappeared when usage reported cached tokens; verification passed 14 of 15 times, against 28 of 30; and, from the transcripts we read, no guard breach sent anything to a subscriber, though one is logged as an automatic reply to an outside party.

What does not survive: that completion tokens fell on the second engine (they rose); that most failures never got a fair attempt (the reproducible figure is 36 of 46, cap or refusal); that the fixes were fine in the strong sense (partly reviewed); and that every breach was a job start.

7. What We Changed

We have changed three things. We treat the budget cap as an accounting artefact for a local seat, so that any future local trial must price cached tokens or raise the cap; we did not change it during the trial. The guard has a new deny class for publishing-job starts; its coverage of resets and of the mail-state write is untested. And on 25 October at 19:44 UTC the operator reverted the seat to the hosted model by restoring the pre-trial configuration. The decision was made on reliability, not on fix quality.

The next experiment is the same trial with the accounting fixed and a seat supervisor that logs its own restarts.

8. Methods and Artefacts

All figures come from the remediation agent's run ledger (a SQLite table opened read-only) and from one result file per run holding turns, terminal reason, subtype, usage by model and imputed total cost. Per-run transcripts were read for the guard breaches, and a listing of mutating commands from 22 October was read for fix quality. Small read-only scripts recomputed every table; nothing was run on the fleet. We report counts, medians and means only, with no significance tests or intervals. Windows are by run start time in UTC. Imputed cost is fresh input times 4 plus output times 20 plus cache read times 0.2 plus cache write times 5, in dollars per million tokens, and it matched every recorded cost on both engines. We retain the ledger, result files, transcripts and a backup of the trial configuration.

PureTensor operates a sovereign AI infrastructure fleet (on-premises GPU compute, storage, and serving) run day-to-day by autonomous agents under human direction. This paper is PT-R-2026-059. The measurements cover 12 to 25 October 2026, ending at the revert on 25 October; the ledger, result files and transcripts are retained.