PURETENSOR
INTELLIGENCE DISTILLED

107 Sessions, Zero Completions

A hosted remediation tier that was never wired to live incidents, and a replay test that joined the wrong ro

Paper: PT-R-2026-045 v1.0

Author: PureTensor Ltd

Date: 29 July 2026

Status: Published

Abstract

On 26 May 2026 we commissioned a hosted deliberative remediation tier: a managed-agents service, then in public beta, driven by a small bridge process on our cluster, sitting above our deterministic playbooks and our incident triage service. On 28 and 29 July we retired it. Its session ledger, as recorded in our internal retirement note (the ledger itself could not be re-read), read 107 sessions, 98 final statuses of "stream error", 7 sessions killed by cost caps, 2 archived at bridge start-up, and none that reached the normal ending. The ledger total was 25.05 dollars, under 0.30 dollars per session. Our first account of this, written at retirement, was that 36 real incidents failed on the first day and that a replay of historical incidents agreed with the incumbent classifier only 30.4% of the time.

Re-reading the raw material, most of that account does not hold. The 36 commissioning-day sessions were one smoke test, ten synthetic incidents and 24 replays of historical incidents (35 sessions; one is unexplained). We found no mechanism by which a live incident could reach the tier, and no evidence that one did: the bridge listened on an escalation subject that nothing in the triage service published to. Over the same weeks the triage service detected 7,632 incidents. As far as we can establish, the tier was never wired to live incidents, though a manual publish of a real incident cannot be excluded.

The zero is also narrower than it looked. At least 90 recipe rows were written across the 107 sessions (quality unrated, July provenance inferred); the ledger label "stream error" hid that. What no session ever recorded was a normal end, and we never established why. The 30.4% agreement figure is invalid: the replay's join between recipes and source incidents was done by time window, with up to five sessions open at once, so 23 rows were mapped onto six distinct incidents, and the cost total repeated six session costs. By signature comparison, only 5 of 23 rows name the incident they were joined to.

Our own faults account for what we can locate: missing wiring, a missing tool definition, a prompt and shadow mode that together made capture impossible on the first burn, a wrong field name, and a ledger that recorded one coarse failure string. The cause of the stream errors is not located, and we assign it to nobody. The lesson we recorded is to count outcomes, not attempts.

1. A Tier Above the Playbooks

We run a deterministic tier of signed remediation playbooks and, above it, an incident triage service that classifies fleet incidents. In May we added a third tier for incidents the first two do not handle. It was a hosted managed-agents service, about seven weeks into its public beta, with one hosted agent session per escalated incident. The session was to triage, diagnose, and propose or run a remediation under operator gating, then write a recipe for reuse. Tools ran in a sandbox pod on our own hardware while the hosted service owned the agent loop.

The design document set a four-week validation plan on a validation budget of roughly 200 dollars. The first two weeks were shadow mode, in which every mutating action is automatically denied and the agent narrates what it would have done; the next two were live and narrow. The shadow exit criterion was at least 80% agreement on action class with the incumbent over at least 20 sessions, and the cost threshold was a mean of at most 5 dollars per session.

In June we described this tier publicly as connected and idle. That was accurate about traffic and silent about outcomes. The question here is what 107 sessions actually showed.

2. What We Built and What We Could Still Read

The bridge was a single-replica service on one always-on compute node. It consumed one message-bus subject, created one hosted session per message, republished the session's event stream onto our bus, answered permission gates (all auto-denied in shadow mode), handled one custom client-side tool, and recorded every session in a local database file (identifiers, times, final status, tokens, cost). The service and the bridge both capped active sessions at 5. Cost caps at retirement were 5 dollars hard and 3 dollars soft per day, a 1 dollar per-session forecast cap, and 197 dollars in total.

Two load generators existed. One published synthetic incidents from templates for eight shapes (pod out-of-memory, failed mount, stuck scrub and others) against a sandbox namespace that did not contain the named objects. The other replayed 24 historical incidents drawn from our triage database, at 8 second spacing, tagged as a replay.

The bridge ledger and its retirement-day copy no longer exist, so the ledger figures below are the retirement record's reading of it: internally consistent but not re-queried. What we re-read independently was the bridge source, the commit history, the design and redesign documents, the replay report, and two retained tables, the recipe table and the triage incident table.

3. The Ledger: Many Attempts, No Normal Ending

Period	Sessions	Cost (dollars)	What it was
26 May	36	9.63	commissioning day
27 May to 9 July	0	0	nothing reached the bridge (44 days)
10 July	28	5.14	synthetic load
12 July	20	5.04	synthetic load
14 July	23	5.24	synthetic load; last write to the ledger
Total	107	25.05	

Table 1. Sessions and ledger cost by period, from the retirement record. The retirement note counts the gap as 45 days; 27 May to 9 July is 44.

The sums hold (107 sessions, 25.05 dollars), and the mean of 0.234 dollars per session (derived) kept far under the cost threshold.

Final status	Sessions
stream error	98
killed (a cost cap tripped mid-session)	7
reconciled and archived at bridge start-up	2
idle (the normal end)	0

Table 2. Final status of the 107 sessions. All 107 were flagged shadow mode. Per the code, a cost cap is the only path to the killed status.

The calendar-day totals for the three synthetic days exceeded the 5 dollar cap (which the bridge applies over a rolling 24 hours) by 0.14, 0.04 and 0.24 dollars, and seven sessions were killed. On 26 May the cap had been raised to 15 dollars for the replay. A cap check can only trip after a usage event arrives, so a small overshoot is expected.

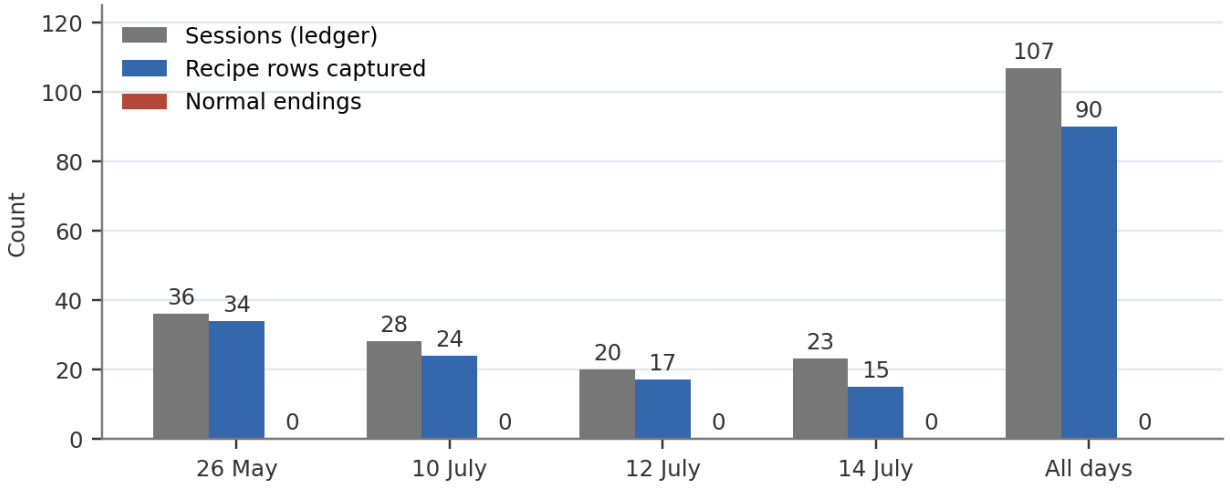


Figure 1. Sessions, recipe rows captured and normal endings per period. The ledger shows no normal endings in any period, while the recipe table shows captures on every active day. Values are in Table 3.

4. What "Stream Error" Hid

In the bridge at the retirement commit, the loop over the event stream ends in one of two ways. If it exits normally, because the stream ends or a terminal event is seen, the session is written as idle. If any exception escapes the loop body, the session is written as stream error, the traceback goes to the container log only, and the ledger keeps just the status string. The loop body does five things per event: republish to the bus, add usage, check cost caps, handle the custom tool, and run the permission-gate handler. An exception in any of the five produces the same label.

So 98 sessions share a label that covers every failure the loop body can produce, and the only copy of the exception text was the bridge pod's container log, which a restart on 26 July erased. The cause of the 98 stream errors was never established, and the retirement record says so.

One hypothesis from the code: the gate handler reads an idle event's stop reason as a key-value object, and another shape would raise and be recorded as a stream error. We did not reproduce this and make no claim that it was the cause.

The redesign (Section 6) defined a working session as one that ends with exactly one call to the capture tool. The retained recipe table shows how many rows that produced.

Day	Sessions (ledger)	Recipe rows (quarantine stamp or replay tag)	Normal endings
26 May	36	34	0
10 July	28	24	0
12 July	20	17	0
14 July	23	15	0
All days	107	90	0

Table 3. Sessions against recipe rows captured. July rows are identified by inference, not by a stored session identifier; unstamped rows created on the same days are excluded.

The July rows are the only rows in the whole table carrying a quarantine stamp, every one was created inside a burn window, and they carry the agent's recipe structure. That is circumstantial provenance only. Rows are not sessions: a repeated signature updates one row, and several July rows show a quarantine stamp days after creation, consistent with repeat captures or with another process re-stamping. Rows created on those July days without a stamp, which came from ordinary operation, are excluded from the counts. The ratios are therefore row ratios, not session counts. The redesign's capture gate asks that 95% of sessions end with a capture call. The replay produced 23 rows against 24 published messages (95.8%), commissioning day as a whole 34 of 36 (94.4%), and July 56 rows from 71 sessions (78.9%). Because the row-to-session attribution for the replay is unreliable (Section 7), we cannot say the gate was met.

So the capture path wrote at least 90 rows. What no session recorded was the completion state. An earlier working claim, that the success path was never reachable, is true only if success means the ledger's idle label.

5. No Route for a Live Incident

The retirement note says that on 26 May the bridge consumed 36 sessions off the live incident subject, that about 34 carried genuine triage signatures, and that all 36 failed. The raw evidence supports neither of the first two claims.

The bridge subscribes to a subject that exists only for escalations. The code never subscribes to the live subject. The design lists wiring the triage dispatcher to publish recipe misses on the escalation subject as work for a later phase. In the triage service's history up to 29 July there is no commit that publishes on that subject; the work was queued after the replay and, on that evidence, never done. A manual publish of a real incident cannot be excluded, since the ledger's incident identifiers are gone.

The recipe table fits this. It holds 34 rows created on 26 May between 09:50 and 12:37 UTC: one smoke test at 09:50, ten synthetic incidents in a 95 second span from 11:44 to 11:45, and 23 replay rows from 12:34 to 12:37, all prefixed as replays. That is 35 sessions against the ledger's 36; one is unexplained. The genuine-looking signatures the retirement note cites, a stuck-scrub shape and a certificate-expiry shape, are the agent's own synthetic captures from 11:45 UTC.

The 44-day gap has the same explanation. The retirement note reads it as incidents stopping; they did not stop. The triage service detected 7,632 incidents between 27 May and 29 July, and none had a path to the bridge. Row count, pod health, bus connectivity and last-write time are all satisfied by a system that fails every time; the one column that grouped by outcome showed 0 of 107 at retirement.

6. The First Burn: Three Faults, None Architectural

The first synthetic burn, before the capture-first redesign, had a stated spend of 30 dollars and produced zero recipes. The redesign document names three faults, "none architectural".

The agent definition carried only the built-in toolset. The capture service was deployed, but nothing pointed the agent at it, and the hosted service could not reach services inside our cluster without a tunnel feature that our design document records as then a research preview. The fix was a client-executed custom tool: the agent emits a tool-use event, the bridge runs the capture and posts the result back. Our first attempt at that post returned HTTP 400 because we used the generic tool-use identifier where the custom tool's own was wanted; that was our error.

Shadow mode combined with a prompt saying "only capture if success was observable" made capture impossible by construction. Every gated action was denied, so nothing was ever observable. In the redesign's words, the agent "reasoned beautifully and produced nothing".

The synthetic incidents named objects that did not exist in the sandbox, so even a live-execution run would have had no ground truth.

The redesign set four gates for the next 30 to 50 sessions: capture rate of at least 95%, median wall clock under 60 seconds, mean cost per capture of at most 0.30 dollars, and an operator rating of at least 3.5 out of 5. The first was not demonstrably met (Section 4). We recorded no wall-clock measurement. The third gate's reported value, 0.3088 dollars, is unreliable (Section 7). The fourth was never rated: the replay report has 92 unchecked verdict boxes and none checked.

The 26 May cost figures cannot be reconciled: the redesign says 30 dollars went on the first burn, a session report quotes a hosted console reading of 11.50 dollars (against a lagged dashboard figure of 1.25), and the ledger records 9.63 dollars for the whole day. We report only the 25.05 dollar ledger total, as ledger spend, not lifetime spend.

7. The Replay That Joined the Wrong Rows

The internal session report recorded a replay comparison: 24 historical incidents replayed, 23 recipes captured, "action-class agreement with the incumbent triage classes: 30.4% (7 of 23)", a mean cost of 0.3088 dollars per session, a total of 7.10 dollars, and "23 of 23 matched cleanly" to their source incidents after a backfill. Re-reading the report file and the retained tables, none of those figures describes what it claims to.

The report has 23 per-incident sections, but they name only six distinct source incidents, repeated 9, 8, 2, 2, 1 and 1 times. A 24-incident sample should name about 23. The retained recipe table agrees: its 23 replay rows carry a stored source-incident tag with the same distribution.

Source incident (anonymised)	Rows joined to it	Session cost shown (dollars)
A	9	0.2824
B	8	0.3213
C	2	0.3921
D	2	0.3253
E	1	0.1889
F	1	0.3665
Sum of rows	23	7.1022 as shown; 1.8765 over the six distinct values

Table 4. Replay rows per source incident as the report listed them, with the cost cell attached to each. The report states a total of 7.1024.

The recipe text does not fit the listed incident for most rows. Nine rows sit under one incident recording a failure of the triage service's own backend on a backup-freshness probe, while their recipes concern a stale volume path, a scheduled node power-down and a stuck backup host. Comparing each row's own signature with the signature of the incident it was assigned, only 5 of 23 rows name the same incident (a conservative reading; two further rows concern the same topic). The rest are mis-attributed or cannot be checked from the signature alone.

The cause is in the session report's own account. The capture step had dropped the source-incident tag from each recipe, so the tag was backfilled by matching a recipe's creation time to the start-to-finish window of a ledger session. The replay published one message every 8 seconds and the cap was five open sessions, so overlap was likely; session durations were not retained. Time-window matching is ambiguous under that overlap. "Matched cleanly" meant every row matched something.

The same join corrupted cost: the report looked up cost by incident identifier, so one session's cost was repeated for every recipe joined to that incident. The 23 cost cells hold six distinct values. The 7.10 dollar total and the 0.3088 mean are sums over repeated values; the six distinct costs sum to 1.8765 dollars. Neither figure measures 23 sessions.

The agreement figure fails for two further reasons. The agent's action class was inferred by a heuristic that reads only two keys from each step, and many steps keep their command under other keys, so a recipe containing a mutating image change was labelled "describe only" by default. Eleven of 23 rows carry that default label. The incumbent labels in the same rows were 11 "page operator", 8 "restart pod", 2 "ssh diagnose", 1 "describe only" and 1 "rollout restart".

So the 7 of 23 compares a crude label for a recipe with the incumbent's label for an incident that, in up to 18 of 23 rows, is not the one the recipe was written for. On the five rows where recipe and incident correspond by signature, the labels agree in 3 of 5. That is too few to support any claim (only 3 of the report's 7 match marks sit on correctly joined rows); we give it only to show that 30.4% cannot be read as 70% disagreement.

The session report read the figure as possibly a sign that the agent was reasoning past the incumbent. That rested on one hand-checked row, an image-pull case where a restart cannot fix a missing tag: one good anecdote, and the operator verdict step was never completed.

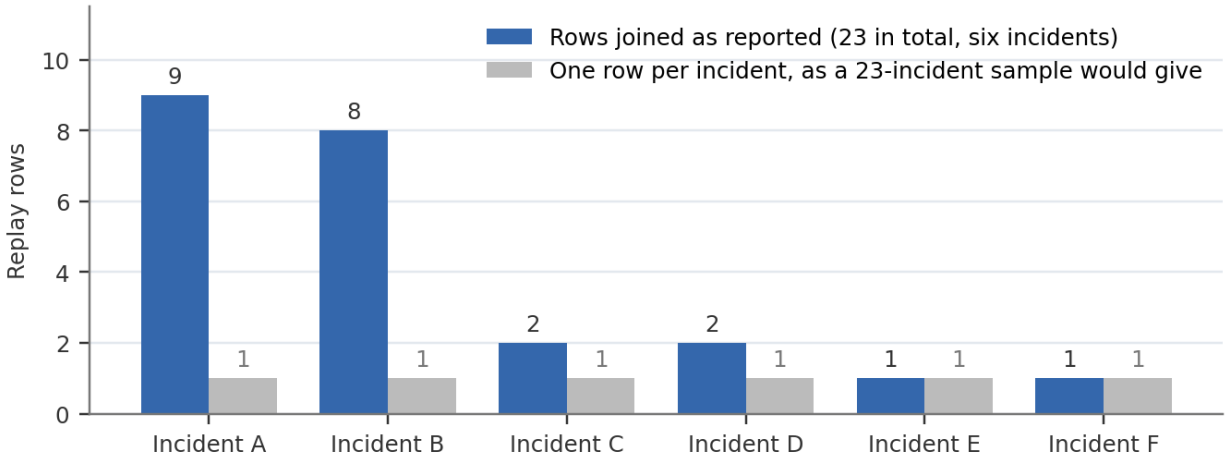


Figure 2. Replay rows joined to each source incident as reported (blue), against one row per incident that a 23-incident sample would give (grey). Six incidents absorbed all 23 rows. Values are in Table 4.

8. What Was Ours

Four faults were in our integration and one is unexplained: no producer connected the live incident stream to the bridge; the first agent definition lacked the capture tool, with shadow

mode and the prompt making capture impossible; the custom-tool result post used the wrong field name; and the ledger kept one coarse failure string while the exception text lived in a container log that a restart erased. The unexplained item is why no session ever reached the normal end. The platform was in public beta; this says nothing about the service in general.

9. What These Numbers Will Not Carry

All ledger figures come from the retirement record; we could not query the ledger or its copy. The recipe and incident tables are the independent check, and July provenance in them is inferred from the quarantine stamp, burn-window timestamps and recipe structure, not from a stored session identifier.

This was one deployment, one configuration, one hosted model and one agent definition, with no repeated runs, no control arm, and nothing compared with the incumbent on live work. Counts are small (n between 5 and 107); all ratios are descriptive.

Four claims survive. A ledger of 107 sessions with no normal endings is a fact within the limits above. We found no route by which a live incident reached the tier. At least 90 recipe rows were written. The replay agreement figure is invalid. Several claims do not survive: that 36 real incidents failed, that the success path was unreachable, that incidents stopped, and the 7.10 dollar replay total with its 0.3088 mean. We cannot say whether the hosted agent's recommendations are good. Whether any capture rate holds against live incidents is open.

10. What We Changed

On 28 July the bridge and load generator were scaled to zero, and on 29 July the capture service and a dashboard followed. The dashboard had polled the hosted service's usage interface about 1,440 times a day, charting a tier that no longer ran. We left in place, as the operator's call because removal is irreversible, the evidence volume, the namespace and its secrets, and the hosted agent definitions, which we did not see charge while idle.

The revival rule we recorded: restoring replicas would restore a system that is 0 for 107, so any revival starts by reproducing one session that reaches the normal end. The lesson we recorded is to count outcomes, not attempts.

11. Methods and Artefacts

We read the bridge source, commit history, and the design, redesign and retirement documents, and queried two retained tables read-only: the recipe table and the triage incident table (including the count of 7,632 incidents detected between 27 May and 29 July). The replay report was re-counted by script (match marks 7 yes, 16 no; cost cells; class labels; verdict boxes 0 checked, 92 unchecked). Statistics are counts and ratios only. Not retained: the bridge ledger and its copy, the hosted service's session transcripts, and container logs from before 26 July. The recipe table, the incident table, the replay report and the source history are retained.

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-2026-045. The tier was built and replayed on 26 May 2026, loaded synthetically on 10, 12 and 14 July, and retired on 28 and 29 July 2026; the recipe and incident tables, the replay report and the source history are retained, while the session ledger is not.