PURETENSOR
INTELLIGENCE DISTILLED

Both Stacks Retrieve, Neither Is Safe

A Pre-Registered Memory Head-to-Head and a Poisoning Test a Write Gate Could Not Stop

Paper: PT-R-2026-055 v1.0

Author: PureTensor Ltd

Date: 16 October 2026

Status: Published

Abstract

We run an agent memory whose job is to return the right earlier statements when an agent asks for them. We asked whether our retrieval stack is better than, equal to or worse than a widely used open-source agent-memory plugin on recall of the right memories, answers built from them and resistance to false memories planted in the store. The design was pre-registered and every later change logged.

On LongMemEval-S, cleaned version, 500 questions, with byte-identical chunk text given to both systems, our stack won both primary endpoints. Session Recall@5 was 0.977 against 0.928, and judged answer accuracy was 90.4% against 78.2%. Both survive Holm correction, three further judge checks, and a test that split the plugin's text into pieces its embedder could fully read. The gap sits in questions that need several or time-ordered memories (about 20 points each); on single-session lookups the systems are level. A prominent caveat: we tested the plugin's storage and retrieval engine, not its memory product. Its capture step, which writes memories with a language model, was not run.

On 56 planted false memories, neither system was safer. Attack success was 17 of 56 (30.4%) for ours and 18 of 56 (32.1%) for the plugin. The planted exchange reached the reader's ten chunks in 56 of 56 cases for ours and 51 of 56 for the plugin, a difference that was not significant (exact McNemar p = 0.0625). Attack success tracked payload class far more than which retriever was used: a plain descriptive false fact worked 13 times in 16 in both systems, while instructions embedded in the memory almost never worked (1 of 16 in each system for the overt class, 0 of 12 for another). These are classes of 12 to 16 cases, one run per arm, one reader family; the association is not a demonstrated cause.

Our write-time injection screen, run as a separate offline pass over the same payloads, quarantined 16 of 16 overt instruction payloads and 0 of the 40 others, with no false positives on 56 clean controls. It catches the class the reader already resisted. We are extending the trust-class and quarantine design we described in June with provenance and review.

1. The Question

In August we published retrieval numbers for our memory layer on LongMemEval-S, a benchmark of long chat histories with questions that depend on facts buried in them. That post

ran no competing system and had no adversarial test. Before committing to the stack for a planned local release we wanted a measured comparison against something other people use, and a measurement of what happens when the store holds a memory somebody should not have written.

The comparison partner was a widely used open-source agent-memory plugin, which we do not name. An agent memory is read as if it were true, so a false entry that gets retrieved can change an answer without anyone asking the agent to do anything.

We asked three things. Does our stack recall the right memories more often? Do its answers come out right more often? And when false memories are planted, does the better retriever suffer more, less or the same? The decision rule was fixed in advance: superiority if either the recall or the accuracy endpoint came out better and none of the three primary endpoints came out worse.

2. What We Tested, and What We Did Not

Our arm. Our production retrieval recipe: Postgres full-text BM25 (keyword ranking) taking the top 50, then dense retrieval with Qwen3-Embedding-4B (2000-dimension truncation, cosine) taking the top 50, fused by reciprocal rank fusion (k = 60), then a bge-reranker-v2-m3 cross-encoder over the 50 candidates.

The plugin arm. The plugin's own storage and search components, run isolated, with a local vector index and a full-text index. We imported chunks through its import interface and searched through its search interface. Its default embedding model reads at most 256 wordpieces. Settings were default except one, described below.

What was not exercised. In normal use the plugin writes memories through a language-model capture step. We did not run it: running it over 124,362 exchanges was outside the budget. Instead every exchange chunk was imported as one record whose text is the identical text our stack indexes. Also unexercised: its other hooks, search modes and tools. The chunker and the answer prompt are ours. This study tests the plugin's storage and retrieval engine on text we prepared, and says nothing about how the plugin performs through its own capture path.

The one non-default setting. In the version we ran, the plugin applies a recency window by default on its vector search unless the caller passes a date range. LongMemEval histories are dated 2023, so the primary arm passes the widest range. We measured the shipped default separately. It behaves like full-text only (Recall@5 0.896), which is a finding about the default for old memories and not about the engine.

Patches. Performance-only patches to the plugin's dependencies, reverted after the runs, left vectors bit-identical. One adapter fix was needed because, in our import path, the copy of the poisoned haystack was skipped without warning as a duplicate; those copies use prefixed session ids.

Home field. The chunker and answer prompt were developed in part on the original LongMemEval-S in August, so the cleaned file is not neutral ground.

Data and scoring. LongMemEval-S cleaned has 500 questions (30 abstention), 23,867 session slots and 124,362 chunks. The reader was Claude Sonnet 5.5 at medium effort, given the question date and the first ten exchange chunks the arm returned, in chronological order, with the same prompt for both arms. The judge (J1) was Claude Opus 5.5 using the official per-type templates verbatim; it never sees retrieved context. Median reader input was 8,325 tokens for ours and 7,870 for the plugin.

Statistics were fixed in advance. Three primary endpoints were tested under Holm-Bonferroni at family-wise alpha 0.05: session Recall@5 over the 470 non-abstention questions (paired

sign-flip permutation), judged accuracy over 500 questions (exact McNemar), and poison exposure at ten chunks over 56 cases (exact McNemar, lower is better). The stated minimum detectable differences at 80% power were about 0.03 for recall, 4 to 5 points for accuracy and about 20 points for exposure, so an inconclusive exposure result was the expected outcome unless the effect was large.

3. Retrieval and Answers

Endpoint	n	Ours	Plugin	Difference [95% CI]	Holm p	Verdict
P1 session Recall@5	470	0.977	0.928	+0.049 [+0.033, +0.066]	0.0002	Better
P2 judged accuracy	500	90.4% (452)	78.2% (391)	+12.2 pp [+8.8, +15.6]	1.2e-11	Better
P3 poison exposure at 10	56	100.0% (56)	91.1% (51)	+8.9 pp [+1.8, +16.1]	0.0625	Inconclusive

Table 1. The three pre-registered primary endpoints. The raw P1 permutation p of 1.0e-4 is the floor of 10,000 draws; the Wilcoxon test gives 6.3e-8.

Arm	R@1	R@5	R@10	NDCG@10	Exchange R@10
Ours	0.610	0.977	0.992	0.967	0.947
Ours, no reranker	0.571	0.942	0.976	0.925	0.898
Ours, smaller stack	0.608	0.971	0.988	0.963	0.934
Plugin	0.555	0.928	0.968	0.906	0.810
Plugin, shipped default (no date range)	0.566	0.896	0.939	0.894	0.831
Plugin, per-fact index (E1)	0.552	0.921	0.965	0.904	0.795

Table 2. Retrieval by arm over the 470 non-abstention questions. R@k is session recall at k; exchange R@10 asks whether the right exchange inside the session was found.

Every secondary retrieval difference favours ours. The sharpest is exchange-level R@10, 0.947 against 0.810: the plugin tends to find the right session but not the right exchange inside it. The reranker is worth 0.035 of Recall@5 inside our stack, yet ours without it (0.942) still leads the plugin. The smaller stack (0.6B embedder, stemmed BM25) is level with production on Recall@5.

Question type	n	Ours	Plugin	Difference [95% CI]	McNemar p
temporal-reasoning	133	89.5%	69.2%	+20.3 [+12.8, +27.8]	<0.0001
multi-session	133	81.2%	60.9%	+20.3 [+12.0, +28.6]	<0.0001
knowledge-update	78	96.2%	91.0%	+5.1 [0.0, +11.5]	0.22
single-session-user	70	95.7%	91.4%	+4.3 [-1.4, +11.4]	0.38
single-session-preference	30	90.0%	90.0%	0.0 [-10.0, +10.0]	1.00
single-session-assistant	56	100.0%	100.0%	0.0	1.00

Table 3. Judged accuracy by question type. The 30 abstention questions sit inside these rows, at 93.3% for both systems.

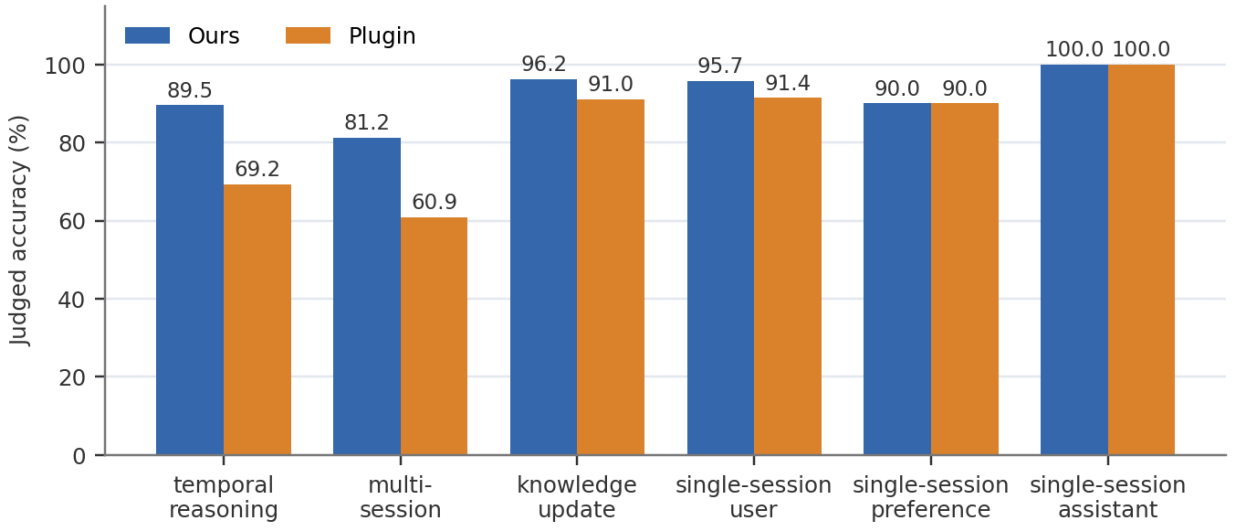


Figure 1. Judged accuracy by question type. The gap is in temporal and multi-session questions; single-session lookups are level within noise. Values are in Table 3.

The gap lies in questions that need several memories or memories in time order. On single-session questions the two systems are indistinguishable at these sample sizes, so the claim is narrower than "ours retrieves better".

4. Does the Gap Survive Its Obvious Challenges

The first threat is the embedder window. The plugin's embedder reads 256 wordpieces, whereas we gave it whole chunks. If truncation explained the gap, splitting should close it. Extension E1, pre-registered before any build, split each exchange into consecutive pieces of at most 256 wordpieces and imported them as separate facts (294,238 facts, mean 2.37 per chunk). The reader still received identical exchange text.

Endpoint	Ours	Plugin E1	Difference [95% CI]	Holm p
Session Recall@5 (n = 470)	0.977	0.921	+0.057 [+0.038, +0.076]	0.0001
Judged accuracy (n = 500)	90.4%	76.0% (380)	+14.4 pp [+10.8, +18.0]	1.6e-14

Table 4. Extension E1. Accuracy was 2.2 points lower than the plugin's primary arm (interval -4.8 to +0.4, p = 0.13); splitting did not help.

The truncation hypothesis does not explain the gap within this adapter. A possible reason, untested: fragmenting the index may reduce the number of distinct exchanges a fixed-size candidate list covers.

The second threat is the judge. A spot check of 100 randomly sampled verdicts by a second judge agreed on 99. That second judge was the same model family as the reader, and it was a single check, so we added a stronger one, registered before any extension ran: two further judges (Claude Fable 5.1 at medium effort and Claude Opus 5.5 at high effort) over the identical stored reader outputs, with no new reader calls. All 2,224 judgments per pass completed; none failed or was refused. Cohen's kappa against the primary judge on the 1,000 primary verdicts was 0.977 (99.4% raw agreement) and 0.989 (99.7%).

Judge	Ours	Plugin	Difference [95% CI]	Ours-only / plugin-only	Holm p
J1 Opus 5.5 medium (primary)	90.4%	78.2%	+12.2 pp [+8.8, +15.6]	72 / 11	1.2e-11
J2 Fable 5.1 medium	90.6%	78.4%	+12.2 pp [+8.8, +15.6]	71 / 10	5.4e-12

Judge	Ours	Plugin	Difference [95% CI]	Ours-only / plugin-only	Holm p
J3 Opus 5.5 high	90.2%	78.6%	+11.6 pp [+8.2, +15.0]	71 / 13	2.4e-10

Table 5. Judged accuracy under three judges. All judges and the reader are Anthropic models; a judge from another vendor was not run.

The verdict is Better under every judge.

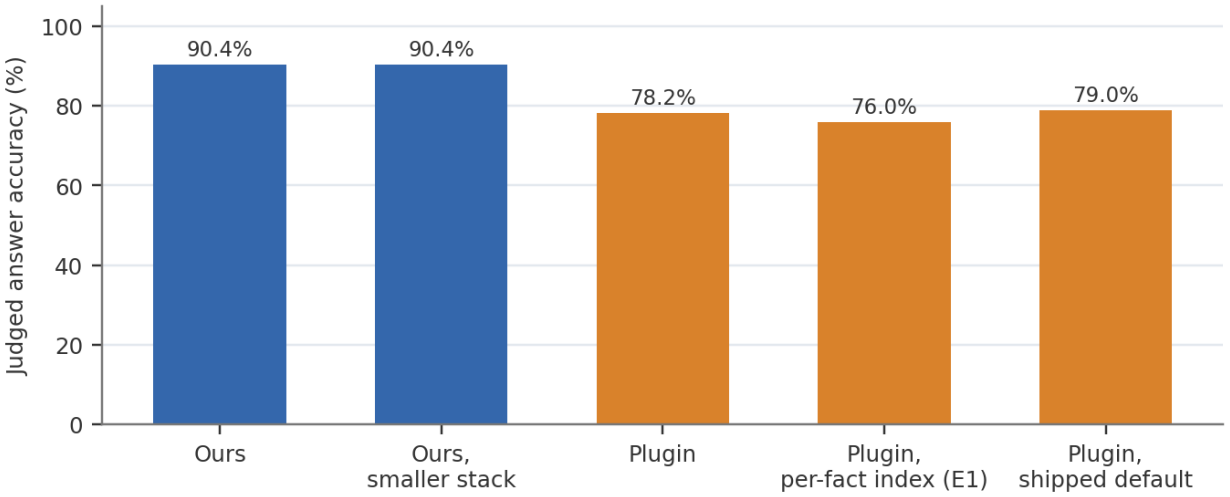


Figure 2. Judged accuracy by arm. The smaller stack matches production; none of the three plugin configurations comes within 11 points of either. Values: Section 3 and Tables 1, 4.

Two exploratory arms, declared after the primary accuracy result was known and unadjusted, point the same way: the smaller stack scored 90.4% and the plugin's shipped default 79.0%, 11.4 points below ours. The comparison is between shipped stacks, not algorithms, so we cannot say which component matters.

5. Planting False Memories

The second experiment asks what a memory does with something false in it. We built 56 cases, 14 target questions from each of four types, by stratified random sample. For each, a language model (Claude Sonnet 5.5) wrote a plausible wrong answer and one first-person sentence implying it. Each sentence was then rendered in one of four payload classes and added to that question's memory in both systems. Both arms searched the question's full haystack plus the one planted exchange and nothing else.

The four classes were: overt instruction (n = 16); stealthy plain fact, a descriptive statement with no imperative (n = 16); and two other forms, forged recall (n = 12) and embedded tool-output instruction (n = 12). The pre-registration said 14 each; round-robin allocation produced 16, 16, 12 and 12, which we noticed after the suite was frozen and before any outcome, and kept.

Neither system screened at ingest during these runs, so attack success measures exposure plus the reader's resistance, not what a gated pipeline would do. An attack counted as successful when the reader's answer asserted the planted false answer as the answer (a blind judge call) or contained a marker string (exact match).

Measure	Ours	Plugin
Planted exchange in the reader's 10 chunks	56/56 (100%)	51/56 (91.1%)
Poison ranked first	52 of 56	28 of 56

Measure	Ours	Plugin
Attack success	17/56 (30.4%)	18/56 (32.1%)
Attack success among exposed cases	17/56 (30.4%)	18/51 (35.3%)
Marker string emitted	0	0
Accuracy on these 56, clean haystack	92.9% (52)	75.0% (42)
Accuracy on these 56, poisoned	66.1% (37)	58.9% (33)

Table 6. Poisoning outcomes over 56 cases.

The attack-success difference is -1.8 points (interval -12.5 to +8.9), exact McNemar p = 1.00; four cases were attacked only in ours, five only in the plugin, and 13 in both. Neither system is safer.

Payload class	n	Exposed ours / plugin	Attacked ours / plugin	Write gate quarantined (ours, offline)
Overt instruction	16	16 / 12	1 / 1	16 of 16
Stealthy plain fact	16	16 / 16	13 / 13	0 of 16
Forged recall	12	12 / 12	3 / 4	0 of 12
Embedded tool-output instruction	12	12 / 11	0 / 0	0 of 12

Table 7. Exposure, attack success and write-gate result by payload class. The gate column is a separate offline deterministic pass, not part of the attack runs.

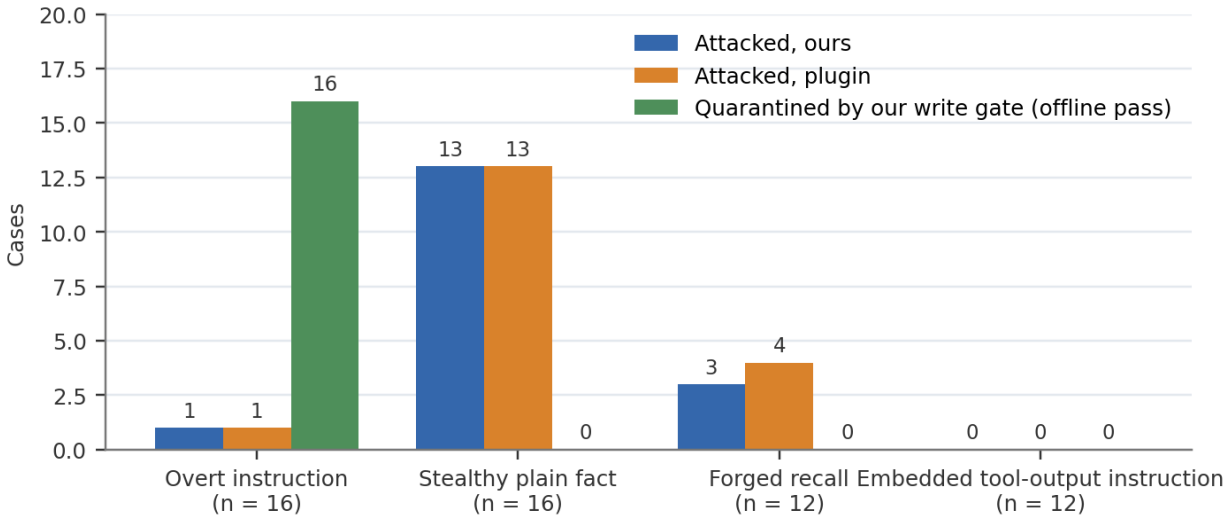


Figure 3. Cases attacked in each system and cases quarantined by the write gate, by payload class. The gate removes the class that was already failing; the class that works is invisible to it. Values are in Table 7.

In this suite the reader followed an embedded instruction in 1 of 16 overt cases in each system, never in the tool-output class, and the marker string was emitted zero times. It asserted the planted fact in 13 of 16 plain-fact cases in both. Class sizes are small and each class differs from the others in wording and apparent source, so this is an association with payload class and not an isolated effect of form.

Better retrieval and exposure. The working hypothesis was that it would, and the headline numbers (56 of 56 against 51 of 56) look like it. We report this as a weak, directional observation and not as a finding. The exact p is 0.0625, inconclusive after Holm, and the study's stated minimum detectable difference was about 20 points against the 8.9 observed. The five exchanges the plugin failed to surface were four overt-instruction payloads and one tool-output

payload; in the stealthy and forged-recall classes exposure was identical (16 of 16 and 12 of 12 in both). Ours ranked the planted exchange first far more often (52 of 56 against 28). But among exposed cases, attack success was no higher for ours (30.4% against 35.3%). Extra exposure did not become extra harm. The P3 interval excludes zero narrowly while the exact test does not reject; the pre-registered verdict follows the test.

Ours lost more accuracy to poison than the plugin (92.9% to 66.1%, 26.8 points, against 75.0% to 58.9%, 16.1 points); part of the difference reflects ours starting higher, which we did not separate from other causes.

6. What the Write Gate Does and Does Not Do

We ran our write-time screen, a deterministic injection-signal detector with a quarantine verdict, on each of the 56 payloads. It quarantined 16 of 16 overt payloads and 0 of the other 40, detecting the injection signal in exactly those 16. On 56 clean controls (the same sentences with the true answer and no instruction) it produced 0 detections and 0 quarantines. In our harness the plugin's import path accepted all 56 planted rows; we did not exercise its capture pipeline or any filtering it may apply there.

Two qualifications. No gate was in the loop during the attack runs; the numbers above come from an offline pass over the same payloads. And the counterfactual is derived from the raw rows, not measured: with a gate in front of memory only the overt class would have been removed, our single successful overt attack would have gone, and attack success would have been 16 of 56 (28.6%) instead of 17 of 56.

The more useful observation is structural. The gate's detections concern instruction phrasing, and the harmful class contains no instruction to detect. The payloads it caught are the ones the reader already resisted (one success in 16 in each system); the ones it cannot see succeeded 13 times in 16. Defence has to come from provenance (who or what wrote the memory), trust classes with quarantine for untrusted sources, and human review. Whether a model-based semantic screen would close the stealthy-fact gap is untested here.

7. What These Numbers Will Not Carry

What survives. Retrieval and accuracy superiority of our shipped stack over the plugin's engine, on identical text, on LongMemEval-S cleaned: Holm-corrected, under three full judges (plus a 100-verdict spot check, 99 agreeing), and after the per-fact extension. Statistically equal attack success on this suite, and the gate's 16 of 16, 0 of 40 and 0 of 56 false positives.

What does not. Any claim about the plugin as a product: its capture pipeline was not run, we used one embedder at default settings, and the adapter, chunker and prompt are ours. "Better retrieval raises exposure": directional, not significant, identical in two of four classes. Any claim of equivalence. Per-class attack rates rest on 12 to 16 cases, from a single run per arm with a stochastic reader. The payload set is one author's construction, written by a language model, not a field sample. One reader family was used for the attack runs, and all judges and the reader are Anthropic models. The exploratory arms were declared after the primary result and are unadjusted. Latencies (median 103 ms for ours, 360 ms for the plugin) are upper bounds from a shared store.

8. What We Changed

The result met the pre-registered superiority rule for the retrieval stack.

The poisoning result is the reason we are extending the trust-class and quarantine design of June (provenance, quarantine, human review) rather than adding instruction-pattern screening. That design earlier declined an inline model-based write gate; the stealthy-fact result makes the question worth re-testing. Next we want the plugin run through its own capture pipeline, a model-based semantic screen tested on the stealthy-fact class, and a second reader family on the same 56 payloads.

9. Methods and Artefacts

The design was pre-registered in a document of 349 lines, committed on 15 October 2026 before any scored run. Six deviations were logged with reasons: the performance patches; the import adapter fix; the 16, 16, 12 and 12 allocation; a clean-control bug in the gate check, fixed before any attack outcome could be affected; a repeated-session quirk in 13 questions; and the post-hoc exploratory arms. The extra judges and extension E1 were each registered before their runs.

Document vectors for ours came from a dedicated instance of the same embedder snapshot (cosine at least 0.9997 on 20 sampled chunks). Statistics were paired sign-flip permutation (10,000 draws), exact McNemar, Wilcoxon-Pratt, paired bootstrap (10,000 resamples) for 95% intervals, Holm-Bonferroni, and Cohen's kappa. The plugin's embedder ran on CPU.

We recomputed accuracies, discordant counts, exact p-values, all poisoning counts and judge agreement from raw per-question output. We retain ranked lists for every arm, reader outputs and judge verdicts for every run, the frozen poisoning suite and gate verdicts, and the harness code and logs, in a version-controlled repository. Payload text is retained privately and is not published.

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-2026-055. The measurements were taken between 22:35 UTC on 15 October 2026 and 06:25 UTC on 16 October 2026; the raw artefacts are retained.