Evals for Agent Memory: You Can't Improve What You Never Score

We benchmark what a model knows. We barely score what its memory does — and an unevaluated memory is a confident liability that grows every day the agent runs.

Kymata Labs Research · An independent research institution studying how AI systems actually behave in production.

~16 min read

Tags · AI Agents · Memory · Evals · Verifiers · Reliability · Production Systems

This is not another long-context benchmark, not a leaderboard, and not a survey of memory architectures. Plenty of good work already describes how agents *form*, *store*, and *retrieve* memory. This is about the thing nobody scores: whether the memory an agent has accumulated is any *good*. Not "does the system persist data" and not "can it recall a fact when asked," but the operator's question — is retrieval surfacing the *right* memory, is a stored fact still *true*, is the write policy saving signal or noise, and is any of it being measured at all? Memory is the one subsystem of a production agent that teams ship without a test suite. We have evals for model knowledge, for tool use, for document retrieval, and for end-to-end agent tasks. Memory sits in the middle of all of it, ungraded — and an unevaluated memory is not neutral. It is a confident liability that grows every day the agent runs. Companion to **The Agent Memory Staleness Problem**; same operator, same chair.

The thesis, in moves

1. **We obsessively score what a model knows.** MMLU, GPQA, the whole leaderboard economy. Knowledge is the most-benchmarked quantity in the field.
2. **We barely score what an agent's memory does.** Academic memory benchmarks exist (LongMemEval, LOCOMO), but they measure recall on a fixed transcript. They do not measure the operational health of a memory store an agent has been writing to for three months.
3. **Memory has distinct, compounding failure modes.** Wrong retrieval, confidently-served staleness, noisy writes, runaway bloat, and failure to forget are five different bugs with five different fixes. A single "memory works / doesn't" verdict hides all of them.
4. **A benchmark is not a harness.** A score on someone else's frozen dataset tells you nothing about *your* store. What an operator needs is a set of mechanical checks they can run against their own memory, on their own data, on a schedule.
5. **The verifier that scores memory is the same artifact that improves it.** This is the move that makes the whole thing worth building. An eval, made mechanical, is a reward signal. Score the memory, and

the score is the gradient: gate the stale facts, down-rank the bad retrievals, reward the write policy that kept the signal. You do not eval *then* improve. The eval *is* the improvement loop.

The one-liner that ties this to the prior paper: persistence is not truth, and untested is not working. The staleness paper argued that a confidently-remembered fact can quietly go false. This paper argues the more basic point underneath it. You will never *catch* that false fact, or any of memory's other failures, until you decide to score memory as a first-class system. The body of work is the same operator standing in front of the same agent: paper 3 named the failure, paper 4 is how you measure for it.

Why memory needs its own evals

Start with the gap, because it is conspicuous. Scan what the people building agents are actually publishing right now and memory is almost absent from the two conversations that matter most. The runtime conversation is about moving the agent loop server-side into a sandboxed computer (Google's managed-agents API, the agent-native clouds). The eval conversation is about verifiers as the bottleneck for both evaluation and reinforcement learning. Memory appears in neither as a thing you *evaluate*. It shows up as a feature you *add*. (Kymata Labs roster signal scan, June 2026; the same gap that motivated paper 3)

That is strange, because retrieval is not knowledge and memory quality, not model IQ, is what gates a long-running agent. A frontier model with a polluted memory will confidently act on the wrong recalled fact. A weaker model with a clean, well-scored memory will not. As agents run for weeks and months rather than minutes, the reliability curve bends on the memory, and the memory is the part with no instrument on it.

Consider the asymmetry. If a model regresses on a knowledge benchmark, someone notices within a release cycle, because the benchmark runs in CI. If an agent's memory store silently fills with duplicates, drifts stale, and starts retrieving the wrong note for the wrong query, nobody notices until a user is told something wrong with total confidence. There is no red number on a dashboard, because there is no dashboard, because there is no eval.

When this matters, and when it does not

A fair objection: not every app needs a memory test suite, and pretending otherwise is how good ideas become cargo cults. So be precise about who this is for.

Memory evals earn their keep in proportion to two things: how long the agent runs and how much it writes. A stateless chatbot that retrieves nothing across turns has no memory to score; skip all of this. A retrieval-augmented app over a curated, human-maintained document set is doing document-RAG, and RAG evaluation already covers it. The case for memory evals is specifically the *self-writing, long-running* agent: the one that accumulates its own store over weeks, decides for itself what to persist, and acts on what it recalls without a human in the loop. That is the configuration where every failure mode in this paper compounds, because the agent is both the author and the consumer of a store that nobody is grading.

The blunt test: if your agent writes to its own memory and you cannot answer "how good is that memory right now" with a number, you are in scope, and the gap between your confidence in the agent and your evidence about its memory is exactly the size of the risk.

The six dimensions

The contribution of this paper is a taxonomy: the dimensions an operator should score memory on, the failure each one catches, and the mechanical metric that catches it. Six dimensions, because memory fails in at least six distinguishable ways and a single number hides them.

Dimension	The failure it catches	The mechanical metric
Retrieval correctness	Surfaces the wrong or irrelevant memory for the query.	Precision / recall@k over labeled query⟶expected-memory pairs. Gold labels must mean *useful to the task*, not lexically similar.
Truth-maintenance	Serves an outdated fact at full confidence (staleness).	Percent stale-but-served on a time-shifted probe set, checked against each fact's valid-time. Paper 3, recast as a tracked number.
Write-policy precision	Saves noise — chatter, duplicates, things that should never persist.	Precision of writes against a "should-persist" gold set drawn from real session logs. Long-run quality is set at write time.
Consolidation	Duplicates or contradicts itself instead of merging and updating.	Duplicate rate and internal-contradiction rate across the store. Append-don't-merge accumulates three versions; retrieval plays roulette.
Context efficiency	Recalled memory pollutes the context more than it helps.	Task-score delta with vs. without the recalled block — signal minus bloat. Every recalled token is budget and attention; dilution is the default at scale.
Forgetting & hygiene	Keeps what it must drop — PII past purpose, "forget me" facts, dead state.	Percent of required deletions actually honored. Here forgetting is a *required* behavior with legal and trust teeth, not a capability loss to fear.

The dimensions are not independent, and that is the real reason a single "memory score" is dangerous. They compound. A loose write policy (dimension 3) admits a near-duplicate, which becomes a consolidation failure (dimension 4), which means retrieval now has two versions to choose between and sometimes returns the wrong one (dimension 1), and if the wrong one is the older one, that is a staleness hit too (dimension 2). One root failure at write time radiates into three retrieval-time symptoms. Score only the symptom and you will keep treating retrieval while the disease is upstream. Scoring the dimensions separately is what lets you find the root instead of chasing the radiation.

These are not exotic. Each is a check you could write in an afternoon against a few hundred labeled examples. The reason they do not exist in most stacks is not difficulty. It is that nobody decided memory was a thing you grade. The grounding for several of them is already in the literature: time-sensitive evals like TimeQA were built to expose exactly the temporal misalignment of dimension two (arXiv:2108.06314); Mem0′s delete-on-contradiction and Zep's bi-temporal invalidation are consolidation policies for dimension four (arXiv:2504.19413 arXiv:2501.13956); and the catastrophic-forgetting line frames dimension six by its opposite — forgetting as a loss to avoid in training, where for an operator it is a duty to verify (arXiv:1612.00796).

The reference-dataset problem

The honest obstacle is the gold set. You cannot evaluate retrieval correctness without labeled query-to-memory pairs, you cannot score staleness without valid-time labels, and you cannot score write policy without a should-persist judgment. Memory evals need a labeled history, and most teams do not have one sitting around.

The good news is that an operator is generating the raw material constantly: the session logs. The construction is cheap and mostly mechanical. Take real sessions, and for retrieval, log which memory the agent actually used and whether the turn succeeded; that is weak-but-real relevance signal you can harden with a small hand-labeled core. For staleness, you already have timestamps; a fact written on one date and contradicted later gives you a valid-time boundary for free. For write policy, sample what got persisted and label a few hundred as should-persist or noise; that is a Saturday, not a research program. For forgetting, every deletion request is a labeled positive.

The point is that the reference dataset is a by-product of running the agent, not a separate data-collection project. The barrier was never the labels. It was the decision to keep them.

One eval, end to end

Abstraction is cheap, so here is the cheapest high-yield eval built start to finish: retrieval correctness, measured as recall@k. The goal is a single number you can watch move and a lever attached to it.

Build the gold set.
Pull a few hundred real turns from your logs where the agent retrieved memory before answering. For each, record the query the agent issued against the store and which memory items it pulled. Now add the cheap label: did the turn succeed? A succeeded turn where a specific item was used is weak evidence that item was the right one to retrieve; a failed turn where the obviously-relevant item was *not* retrieved is strong evidence of a miss. Hand-correct a core of fifty to be unambiguous gold, and let the rest ride as weak labels. This is an afternoon, and it is reusable forever.

Run the metric.
For each gold query, ask the store for its top-k items and check whether the expected item is in the returned set. Recall@k is the fraction of queries where it was. Run it at k of one, three, and ten; the spread tells you whether the right memory is being found at all (recall@10) versus being ranked to the top where it gets used (recall@1). A store that scores well at ten and badly at one has a ranking problem, not a coverage problem, and that distinction is the difference between re-embedding and re-ranking.

Read it and pull the lever.
A low recall@10 says the right memory is not being surfaced: the embedding or the index is the suspect, and the fix is re-embedding with a better model or widening the candidate set. A healthy recall@10 with a poor recall@1 says retrieval finds the memory but buries it: the fix is a re-ranker over the candidates. Either way the number told you which knob, and after you turn it you run the same hundred queries and watch the number move. That last sentence is the whole method. The eval is not a verdict; it is an instrument with a control wired to it.

Notice what this did not require: no frozen academic benchmark, no labeled corpus from a vendor, no research budget. It required your own logs and the decision to score them. Every one of the other five dimensions is the same shape with a different gold label.

Eval as reward: the loop that makes it worth doing

Here is why you build the harness rather than run a benchmark once and forget it. The verifier you write to *score* a memory dimension is the same artifact that *improves* it.

This is not a Kymata Labs idea; it is the strongest current in the eval conversation. LangChain's work on efficient verifiers for legal agents makes the case directly: verifiers are the cost bottleneck for both evaluation and RL post-training, and the same verifier that grades an output can serve as the reward model that trains the policy (LangChain, "Designing Efficient Verifiers," June 2026). Batch them on open or cheap models and the cost of running them continuously drops far enough that "score it on every change" becomes

affordable rather than aspirational. Eugene Yan's eval-rigor work points the same direction: the discipline of a mechanical, reproducible score is what lets you optimize against it at all (eugeneyan.com).

Apply it to memory and each of the six dimensions becomes a control loop. Retrieval recall@k is low: the score is the signal to re-rank the retriever. Stale-served percent is high: gate or down-weight facts past their half-life and watch the number fall. Write precision is low: tighten the write gate and re-measure. The eval is not a report card you read once. It is the reward that closes the loop.

We have a live receipt for this, shipped this week. Agent OS's build factory generates code and UI, then grades each attempt with adversarial verifiers (a model-family-diverse reviewer, a real-browser design measurement, a test runner) and feeds the critique back as the next attempt's input, looping until the verifiers pass. The verifier is the reward; the score is the gradient. We did not build that for memory, but it is the identical shape, and it is the shape this paper argues you should point at your memory store. (Implementation receipt, not a benchmark claim.)

The cost angle matters because memory evals, to be useful, have to run continuously rather than once. A memory store changes every session. A score from last month is itself stale. The only eval that catches a degrading store is one cheap enough to run on a schedule, which is exactly why the "batch verifiers on small models" result is load-bearing here and not a footnote.

Think about the cadence concretely. A knowledge benchmark can run quarterly because the model only changes on release. A memory store changes every time the agent acts, which means its quality has a much shorter half-life than any model's, and an eval cadence to match. The practical target is to run the cheap dimensions (retrieval and staleness) on every meaningful batch of writes, the way a test suite runs on every commit, and the expensive ones (context-efficiency against live tasks) nightly. That cadence is only affordable if each check is a small, batched verifier rather than a frontier-model call per item, and it only produces improvement rather than just anxiety if each run's number is wired to a knob. The discipline is the same one that made continuous integration normal for code: a fast, cheap, automatic check on every change, with a red light that stops the line. Memory has had no such line. The argument of this paper is that it needs one, and that the pieces to build it — the gold sets from your own logs, the batched verifiers, the reward wiring — are all already in reach.

Why the existing benchmarks do not close it

LongMemEval and LOCOMO are real and useful, and this paper is not a swipe at them. LongMemEval explicitly includes a knowledge-updates ability and reports large accuracy drops on long-horizon memory (arXiv:2410.10813); LOCOMO is the standard long-conversation memory benchmark that the systems papers report against (arXiv:2402.17753). Credit where due: they established that memory degrades measurably over long horizons.

But they are benchmarks, not harnesses, and the difference is the whole point. They are static: a frozen transcript, scored once. They are single-shaped: built around long-conversation recall, not the operator's six failure modes; there is no write-policy axis, no forgetting axis, no context-efficiency-against-your-task axis. They run on their data, not yours, so a good score tells you the system *can* be good, not that *your three-month-old store* is. And critically, they have no reward loop: they produce a number for a paper, not a gradient for your retriever.

The prior-art on the *problem* side is similar. STALE (arXiv:2605.06527, May 2026) argues that current benchmarks primarily measure static fact retrieval and overlook the ability to revise stored beliefs. We cite that framing head-on, the same way the staleness paper did, and extend past it: naming that benchmarks miss belief-revision is the academic flag; giving the operator a continuous, reward-wired harness across all six dimensions is the operator's contribution. (STALE existence verified; its specific results postdate our verification cutoff and are not asserted here as fact.)

From the operator's chair: Agent OS receipts

Everything above is sharper when it is something we actually ran into, not a thought experiment. A few receipts from running Agent OS, an autonomous operator with a persistent memory.

Context efficiency, the hard way.
The session-memory index file grew past the point where the harness itself warned it was bloated and only partially loading. That is dimension five with a red light on it: a memory store whose recalled block had started to cost more context than it returned in value. The fix is not "store less"; it is to *measure* the yield of recalled memory and prune what does not earn its tokens.

Truth-maintenance, built into the protocol.
The recall protocol carries an explicit instruction: a recalled memory reflects what was true when written, so verify a named file or fact still exists before acting on it. That is a hand-coded version of the staleness dimension, and the fact that it had to be hand-coded is the argument for scoring it: the discipline only holds if a number holds it accountable. The facts store carries a last-verified column for the same reason; a column is a place to *put* a freshness score, which is most of the way to *being* one.

A write-policy failure, caught only by looking.
The most honest receipt is a miss. A stored preference said this author should avoid em-dashes. The actual published papers used them heavily. The memory was confidently wrong, and it was caught not by any eval but by a human happening to look. That is dimension three (a bad write that persisted) and dimension two (a fact that went false) failing together, silently, on the author, mid-project. It is the cleanest possible argument for the harness: persistence kept the claim, and only an eyeball, not a system, caught that persistence was not truth.

Eval as reward, shipped.
And the receipt that points forward: the build factory's verifier-as-reward loop, shipped this week, is the exact mechanism this paper says to aim at memory. The pattern is proven in our own stack on code and UI. Pointing it at the memory store is the next build, not a hypothesis.

What to score Monday

Concrete controls, ordered by yield-per-effort. None of these requires a research team. Each is mechanical, and each is wired to an action, because a score with no lever attached is just anxiety.

1. **Retrieval recall@k on a small labeled set.** Hand-label a few hundred query-to-expected-memory pairs from real logs. Measure recall@k. The lever: re-rank or re-embed when it drops. Highest yield, lowest effort; start here.
2. **Stale-served percent on a time-shifted probe.** Use your own timestamps to build superseded-fact probes. Measure how often the store returns the old version. The lever: gate facts past a fact-class half-life (the staleness paper's decay-rate table is the gating policy).
3. **Write precision on a should-persist sample.** Label a sample of writes as signal or noise. Measure precision. The lever: tighten the write gate, then re-measure. This is the dimension that pays compounding dividends, because it fixes quality at the source.
4. **Duplicate and contradiction rate across the store.** A nightly scan. The lever: merge-on-write instead of append-on-write.
5. **Context-efficiency delta.** For a sample of tasks, run with and without the recalled block and diff the task score. The lever: prune memory that does not move the number.
6. **Required-deletion honored percent.** Every forget request is a test case. The lever: fix the forget path until the number is 100, because this one has legal teeth.

Run the cheap two (1 and 2) first, on a schedule, and wire each to its lever so the score becomes a gradient rather than a report. That is the whole method: measure the dimension, attach the reward, close the loop.

Close

Staleness is one failure mode of memory. This paper is about the instrument that catches staleness and the five other failures next to it. The quiet reliability cliff ahead is not that agents will forget; it is that they will remember, confidently, things that are wrong, irrelevant, duplicated, or bloated, and that no number anywhere will go red when they do.

The fix is not a better memory architecture. It is the decision to treat memory as a first-class, evaluated system, with a harness an operator runs on their own store, on a schedule, with each eval wired to the reward that improves the thing it measures. We benchmark what models know. It is time to score what their memory does, because untested is not working, and an unevaluated memory is just a confident liability waiting for a user to trust it.

Frequently asked questions

Is this just RAG evaluation?
No. RAG eval scores retrieval over a static document corpus with ground-truth answers. Memory eval scores a store the agent *wrote itself*, where the gold labels are about derived usefulness, freshness, and write quality, and where there is no document oracle to check against.

Don't long-context models make memory eval moot?
No. A bigger window changes what fits, not whether what is recalled is correct, fresh, or worth its tokens. Context pollution is a property of the content, not the window size.

Do I need all six dimensions on day one?
No. Retrieval recall@k and stale-served percent are the cheap, high-yield pair. Start there, wire each to a lever, add the others as the store grows.

Why wire evals to rewards instead of just reporting them?
Because a memory store changes every session; a one-time score goes stale itself. Only a continuous, reward-wired eval catches a degrading store, and recent verifier-cost work makes continuous scoring affordable.

References

1. **LongMemEval** — long-term memory benchmark with a knowledge-updates ability; reports large long-horizon accuracy drops. arXiv:2410.10813. https://arxiv.org/abs/2410.10813
2. **LOCOMO** — long-conversation memory benchmark; the standard reporting target for memory systems. arXiv:2402.17753. https://arxiv.org/abs/2402.17753
3. **Mem0** — agent memory system; LOCOMO results; delete-on-contradiction consolidation. arXiv:2504.19413. https://arxiv.org/abs/2504.19413
4. **Zep / Graphiti** — bi-temporal memory (valid-time and transaction-time); contradiction-driven invalidation. arXiv:2501.13956. https://arxiv.org/abs/2501.13956
5. **TimeQA** — time-sensitive QA exposing temporal misalignment; design template for time-shifted probes. arXiv:2108.06314. https://arxiv.org/abs/2108.06314
6. **STALE** — argues benchmarks measure static retrieval and overlook belief revision (May 2026; existence verified, results not asserted). arXiv:2605.06527. https://arxiv.org/abs/2605.06527
7. **Catastrophic forgetting / EWC** — forgetting as capability loss in training (contrast with required forgetting). arXiv:1612.00796. https://arxiv.org/abs/1612.00796
8. **LangChain** — "Designing Efficient Verifiers for Legal Agents" (June 2026): verifiers as the eval-and-RL-reward bottleneck; batch on open models to cut cost; verifier doubles as reward model.
9. **Eugene Yan** — eval-rigor essays; mechanical reproducible scoring as the precondition for optimization. eugeneyan.com

10 **Kymata Labs** — "The Agent Memory Staleness Problem" (paper 3); roster signal scan (June 2026); Agent OS build-factory verifier-as-reward implementation (2026-06-13).