# The Agent Memory Staleness Problem

*Persistence ≠ Truth: why your agent confidently remembers things that are no longer true, and the operator's framework for when to distrust it.*

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

~16 min read

**Tags** · AI Agents · Memory · Reliability · Production Systems

AI agent memory systems have solved persistence: they reliably store and retrieve what an agent learns. They have not solved truth-maintenance, knowing when a confidently-stored fact has quietly become false. This paper argues that staleness is a distinct reliability failure mode, separate from forgetting, knowledge cutoff, and RAG freshness, and that a stale fact retrieved with full confidence is worse than no memory at all. Surveying the current memory stack (Mem0, Zep/Graphiti, Letta, and others), it shows that every shipped fix is reactive, waiting for a contradiction to arrive, while a fact that goes silently false in the world stays marked valid forever. It then offers what the academic framing omits: an operator's framework, built from running a memory-persisting multi-agent system in production, comprising a decay-rate taxonomy (facts have radically different half-lives, so a single time-to-live is the original sin) and a re-verification responsibility model, reduced to five controls an operator can implement this week.

## The argument, in moves

1. **We solved remembering.** The last two years produced a real memory stack for agents: Mem0′s extraction-and-consolidation pipeline [4], Letta (formerly MemGPT) paging memory like an operating system [6], Zep's Graphiti storing memory as a temporal knowledge graph [5], A-MEM linking notes Zettelkasten-style. Each is a serious answer to *how does an agent retain and recall information across a long run.*
2. **We did not solve truth-maintenance.** Those systems optimize formation, extraction, consolidation, ranking, and retrieval. None has a first-class mechanism for noticing that a stored fact, true when written, has since become false in the world. The fact is not retracted; it sits in the store, ranked by relevance, served with the same confidence as a fact that is still true.
3. **This is a distinct failure mode**, not a relabeling. It is not catastrophic forgetting (loss of *correct* knowledge), not the model's knowledge cutoff (which lives in the weights and is patched by retrieval) [14], not RAG index freshness (which lives at the document layer). It is the decay of a *persisted belief* the agent wrote down and trusts, a context-memory conflict in the Knowledge Conflicts taxonomy [2].
4. **It is pervasive, and it is hard.** DyKnow found outdatedness a critical problem across the state of the art, and that leading knowledge-editing algorithms could not reliably reduce outdated answers [1].

5. **The fixes that exist are reactive.** Zep invalidates an edge and Mem0 issues a DELETE, but only when a contradicting message arrives [5] [4]. A fact that goes quietly false, with nothing announcing the change, stays marked valid forever.
6. **The operator's question is unanswered.** Which stored memories should I distrust, how fast does each kind go stale, and who re-verifies them before they are served? That framework is this paper's contribution.

The one-liner: the next reliability cliff is not that your agent forgets; it is that it remembers, perfectly, something that is no longer true, and tells you so without a flicker of doubt.

## The strongest evidence: the benchmarks already measure it

What makes the thesis hard to dismiss is that the failure is already quantified, from every angle, by benchmarks that were not even built to argue this point. TimeQA's best model reaches **46%** on time-evolving facts where humans reach **87%** [7]. LongMemEval measures a roughly **30%** accuracy drop for commercial assistants maintaining information across sustained interactions, and one of its named abilities is "knowledge updates" [8]. SituatedQA finds that about **one in six** information-seeking questions has an answer that changes with time or place, and that models trained on past data lose about fifteen points answering present-day questions even with an updated evidence corpus [9]. DyKnow tested **24** open and commercial LLMs against Wikidata and concluded outdatedness is critical across the field [1]. RealTime QA and FreshLLMs make the same point on the model-knowledge side [11] [10].

These are not edge cases dredged up to support a thesis. They are the standard time-sensitivity benchmarks, and they agree: the moment a fact can change, both models and the stores that feed them degrade sharply, and a stale fact retrieved confidently is worse than no memory at all, because the agent does not hesitate. It acts.

## The brain of the field, on itself

A useful distinction comes from temporal databases and shows up, not by accident, in the most advanced agent-memory system to adopt it. Zep's Graphiti tracks two independent time axes on every fact: *transaction time* (when the system learned a fact, and when it retired it) and *valid time* (the span during which the fact was actually true in the world) [5]. The gap between those two axes is exactly where staleness lives. A store that records only transaction time, which is most of them, knows when it wrote a fact down. It does not know whether that fact is still true.

The cleanest evidence the field has under-attended to this is its own literature. Read the major surveys of LLM-agent memory and the vocabulary is consistent: writing, management, reading; formation, consolidation, retrieval; summarization, recall, reflection. The language of staleness, *obsolete*, *expired*, *invalidate*, *superseded*, *freshness*, barely registers against it. We deliberately do not claim a tally of "N formation papers versus zero staleness papers," because no survey tabulates it that way and we will not invent a number. The honest framing is the one the texts support: the language of remembering outnumbers the language of distrusting a memory by a wide margin, and the specific notion of a stored belief that silently goes false is, in the agent-memory literature, close to unspoken.

Close to, not entirely. STALE, published in May 2026, asks in its title whether LLM agents can know when their memories are no longer valid, observes that current benchmarks primarily measure static fact retrieval while overlooking the ability to revise stored beliefs, and names the case we care about most: *Implicit Conflict*, where a later observation invalidates an earlier memory without explicitly negating it [3]. We cite it head-on. But STALE is an *evaluation*, a benchmark that measures whether a model can *detect* implicit conflict when shown the evidence. That is upstream of the operator's problem. Knowing that models are mediocre at spotting an implicit contradiction does not tell the person running an agent which of ten thousand stored

facts to re-check tonight, how often, or at whose expense. The diagnosis exists. The operating manual does not.

## The supporting evidence: where every shipped fix stops

It is worth walking the current toolbox and marking precisely where each stops, because every one of these is a real, shipped technique and several are excellent.

| Technique | Where it stops | Exemplar |
|---|---|---|
| Contradiction-on-write | Reactive: requires the contradicting input to arrive; never fires on a silent change | Mem0 DELETE, Zep edge invalidation |
| Bi-temporal validity | Can represent expiry, does not detect or schedule it; waits to be told | Zep / Graphiti |
| Self-editing + background consolidation | Aimed at tidiness, not truth; reorganizing a false fact does not make it true | MemGPT / Letta sleep-time |
| Knowledge editing | Downstream: assumes you already know which fact is wrong | ROME, MEMIT |
| Time-sensitivity benchmarks | Diagnose the problem; do not operate the store | DyKnow, TimeQA, STALE |

Contradiction-on-write (Mem0′s DELETE, Zep's edge invalidation) is the right floor, but fundamentally reactive: the hardest stale facts are the ones the world changed without telling your agent [4] [5]. Bi-temporal validity (Zep/Graphiti) is, in our view, the most important primitive in the space, because it is the data model that *can* represent staleness at all, but representing it is not detecting it [5]. Letta's self-editing and asynchronous "sleep-time" pass is the closest structural hook for what we propose, yet its stated purpose is tidiness, not truth [6]. Knowledge editing (ROME, MEMIT) applies a correction once you already know a fact is wrong, and DyKnow's finding that editing fails to reduce outdatedness in practice underlines that you cannot edit your way out of a problem you cannot first detect [15] [1]. Everything we have is either reactive, structural without a trigger, or corrective without detection. Nobody is proactively asking, of a memory sitting quietly in the store, *is this still true, and how would I know.*

It is worth being concrete about the boundary. When Anthropic writes about "stale" content in its context-management guidance, it means stale *tool calls and results* cleared from the context window as it approaches a token limit, a budget concern about the working set, not a truth concern about the store [13]. Its persistent memory tool is described as create-read-update-delete on files in a dedicated directory; it is CRUD on files, with no notion of whether a file's contents are still true. We are not singling out Anthropic; this is the norm, and it is the clearest illustration that the truth layer is simply absent from the tools people deploy.

## A taxonomy of decay

Here is the observation the academic framing misses and the operator cannot avoid: facts do not decay at the same rate, and the spread is enormous. Treating "memory" as one thing with one freshness policy is the original sin. Consider the memories a real assistant accumulates about one person.

| Decay class | Examples | Half-life |
|---|---|---|
| Effectively permanent | Date of birth, the city someone grew up in, the fact that they have a sister. Re-verifying these is waste. | decades |

| | | |
|---|---|---|
| Slow | Home address, employer, job title, the car they drive. Consequential when wrong: an agent that books travel to a stale home address fails expensively. | years |
| Medium | Stated preferences, current projects, who they are working with, their goals this quarter. The dangerous middle: stable enough to trust, changes often enough to betray you. | months |
| Fast | Mood, what they are doing today, the open thread in the current conversation. Acting on yesterday's state is a visible, immediate failure. | hours |
| Volatile | Prices, availability, the contents of a shared document, anything with an authoritative external source. Arguably should not be memorized at all; they should be re-fetched. | minutes |

This is why a single time-to-live is the wrong instrument. Set the TTL aggressive enough to catch the volatile facts and you burn re-verification budget churning birthdays; set it slack enough to leave permanent facts alone and every price you cached is a lie within the hour. The research that takes this seriously arrives at the same conclusion from the formal side: HALO had to model a *half-life* per fact precisely because outdated facts result from exceeding an expiration date that is not uniform across facts [12]. The practical artifact is a mapping from *fact class* to *re-verification policy*, assigned on write: permanent facts are never re-checked; slow facts are re-checked on a long cadence or when high-stakes actions depend on them; volatile facts are not stored as truth at all but as a cache with a hard expiry and a source to re-fetch from. The point is not the exact buckets, which every team will tune, but the principle: **staleness policy is a property of the fact, assigned at write time, not a global setting.** No memory system we surveyed asks the writer that question.

## Who re-verifies, and how

A taxonomy of decay is inert without an owner. The second half of the framework is a responsibility model: a clear answer to who or what re-checks a memory, and how, before it is served as truth. Three mechanisms, in increasing order of how much they close the gap.

**Provenance as a prerequisite.** You cannot re-verify a fact you cannot trace. Every memory should carry where it came from: the source episode, message, document, or tool result. It is the thinnest, most underbuilt part of the stack, the cheapest to add, and nothing else works without it. A fact with no source should be trusted least of all.

**Proactive re-grounding, budgeted by value.** The operator's move is to go looking before being told. A background pass walks the store and re-checks facts that are both past their half-life and high-value, where value is retrieval frequency times the stakes of the actions that depend on them. This is Letta's sleep-time slot, repointed from tidiness to truth [6], and the same idea as the search-augmented refresh that FreshLLMs applies to model knowledge, applied instead to the persisted store [10]. A fact retrieved a hundred times a week and last verified six months ago is the first thing the pass should re-ground; a birthday verified once is the last.

**Distrust as a first-class state.** The most important change is the cheapest. Introduce a third state: present but past its confidence horizon. A stale-by-policy memory is not silently served and not silently deleted; it is surfaced, flagged, re-grounded on demand, or escalated. Making decayed memory visibly decayed is most of the fix, because the failure we are preventing is not that the agent lacks a fact. It is that the agent serves a decayed fact with the full confidence of a fresh one. None of these three requires the protocol's

cooperation or a new model; they are operator-side controls, implementable on top of any of the memory systems named above.

## From the operator's chair

We promised this was written from inside a running system, so here is the scar tissue, including a piece of it produced in the act of writing. We run Agent OS, a multi-agent system with a persistent memory layer that has accumulated for months. The symptoms in this paper are not hypotheticals; they are our incident log.

The memory index itself bloated past the point of trust. The human-readable memory file the system loads each session grew to fifty-four kilobytes and over two hundred and seventy lines, and the harness now warns, on load, that it is too large and that entries should be moved out. A memory store that has to warn you it can no longer be reasoned about in one pass is a store accumulating faster than it is being validated. Persistence without truth-maintenance does not fail loudly. It silts up.

We had already, before writing this, been forced into one of the controls above by the failure it prevents. The system's recall instructions now carry an explicit rule: a recalled memory reflects what was true when it was written, and if it names a file, a function, or a configuration flag, that reference must be re-verified against the live system *before* it is acted on. That rule did not come from theory. It came from agents confidently acting on remembered file paths and flags that had since moved or been renamed. It is, in miniature, exactly the distrust-as-a-state control: a class of memory (code references) marked low-half-life, with a mandatory re-grounding step before use.

And then, while researching this very paper, the thesis demonstrated itself on its author. A long-standing entry in our memory said, in effect, *the house writing style uses no em-dashes.* We nearly wrote the entire paper to that rule. Before doing so, we sent an agent to measure the actual published papers it was supposed to match. They use the em-dash heavily, dozens of times each. The memory had been true once, for a different surface and a different voice, and had quietly gone false as the body of work evolved. It was confidently stored, retrieved exactly when relevant, and wrong. The only thing that caught it was a re-grounding step against the source of truth, run because the stakes of getting the house voice wrong were high enough to justify the budget. That is this paper's entire argument, compressed into a punctuation decision: *persistence is not truth, and the only defense is to go and look.*

## What to do Monday

The framework reduces to five controls an operator can implement this week, on top of whatever memory system is already running. They are ordered by leverage.

1. **Stamp every memory, on write, with two things: a `last_verified` timestamp and a fact class.** The timestamp is the transaction-time anchor; the class assigns a half-life. This single change converts an undifferentiated store into one that can reason about its own freshness, and it costs the least while unlocking the rest.
2. **Run a proactive re-verification pass, budgeted by value.** Background work that re-grounds facts which are both past their half-life and high-retrieval or high-stakes. Reuse the sleep-time/consolidation slot you may already have; point it at truth, not tidiness.
3. **Keep contradiction-detection on write as the floor, not the ceiling.** Mem0′s DELETE and Zep's invalidation are correct and you should have them. Just do not mistake reacting to an announced contradiction for detecting a silent one.
4. **Track provenance on every fact.** Source episode, message, document, or tool. A fact with no source cannot be re-grounded and should be trusted least. This is the cheapest gap to close and the prerequisite for control 2.

5. **Make stale memory a visible state, not a silent answer.** Past its confidence horizon, a memory is flagged, re-grounded on demand, or surfaced for review, never served with the confidence of a fresh fact. The goal is to prevent the confident lie, which is the failure that actually costs you.

The weighting matters: prefer proactive re-grounding over reactive contradiction, and prefer visible distrust over silent service. The reactive techniques catch the contradictions that arrive. The damage is done by the ones that do not. The reason this is becoming urgent now, rather than a permanent footnote, is duration: an agent that runs for months accumulates a body of beliefs that drifts out of sync with the world at a rate proportional to how much it has stored. The field solved remembering because remembering was the obvious problem. Persistence was the easy half. Truth is the half that is still open, and it belongs, for now, to whoever is willing to go and look.

## Frequently asked questions

**Is this just the model's knowledge-cutoff problem?**
No. Knowledge cutoff lives in the model's weights and is addressed by retrieval or web search. Staleness here lives in the agent's persisted memory, the facts it wrote down and trusts. A perfectly current model will still confidently serve a stale fact from its own store.

**Doesn't RAG solve this by re-fetching?**
For facts that have an authoritative external source, yes, and those arguably should be re-fetched rather than memorized. RAG does not help for derived memories, like inferred preferences, relationships, or summarized state, which have no single document to re-read. Those are exactly the memories that go stale silently.

**Don't TTLs solve it?**
A single TTL cannot, because facts decay at radically different rates. A TTL tight enough for prices churns through permanent facts; one slack enough for birthdays leaves prices stale for hours. The fix is a half-life per fact class, assigned at write time, not a global constant.

**Isn't this the same as cache invalidation?**
Cache invalidation assumes a canonical source of truth to invalidate against. The hardest agent memories, inferred preferences and conversational state, have no oracle to check, which is why both Zep and Mem0 use an LLM to detect contradiction rather than a key-based eviction. It is harder than cache invalidation, not equivalent to it.

**Won't bigger context windows make memory staleness moot?**
No. A larger window changes how much you can hold, not whether a held fact is true. As Anthropic notes, context windows of all sizes remain subject to pollution and relevance problems. A stale fact in a larger window is still a stale fact.

## References


1 Mousavi, S., Alghisi, S. & Riccardi, G. (2024). "DyKnow: Dynamically Verifying Time-Sensitive Factual Knowledge in LLMs." arXiv:2404.08700 (EMNLP 2024 Findings). https://arxiv.org/abs/2404.08700

2 Xu, R. et al. (2024). "Knowledge Conflicts for LLMs: A Survey." arXiv:2403.08319 (EMNLP 2024). https://arxiv.org/abs/2403.08319

3 Chao, Bai, Sheng, Li & Sun (2026). "STALE: Can LLM Agents Know When Their Memories Are No Longer Valid?" arXiv:2605.06527 (May 2026). Existence and framing verified; findings not independently assessed. https://arxiv.org/abs/2605.06527

4 Chhikara, Khant, Aryan, Singh & Yadav (2025). "Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory." arXiv:2504.19413. https://arxiv.org/abs/2504.19413

5 Rasmussen, Paliychuk, Beauvais, Ryan & Chalef (2025). "Zep: A Temporal Knowledge Graph Architecture for Agent Memory." arXiv:2501.13956. https://arxiv.org/abs/2501.13956

6 Packer, C. et al. (2023). “MemGPT: Towards LLMs as Operating Systems.” arXiv:2310.08560. (Letta.) https://arxiv.org/abs/2310.08560

7 Chen, W., Wang, X. & Wang, W. Y. (2021). “A Dataset for Answering Time-Sensitive Questions (TimeQA).” arXiv:2108.06314 (NeurIPS 2021). https://arxiv.org/abs/2108.06314

8 Wu, D. et al. (2024). “LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory.” arXiv:2410.10813 (ICLR 2025). https://arxiv.org/abs/2410.10813

9 Zhang, M. J. Q. & Choi, E. (2021). “SituatedQA: Incorporating Extra-Linguistic Contexts into QA.” arXiv:2109.06157 (EMNLP 2021). https://arxiv.org/abs/2109.06157

10 Vu, T. et al. (2023). “FreshLLMs: Refreshing Large Language Models with Search Engine Augmentation (FreshQA).” arXiv:2310.03214. https://arxiv.org/abs/2310.03214

11 Kasai, J. et al. (2022). “RealTime QA: What’s the Answer Right Now?” arXiv:2207.13332 (NeurIPS 2022). https://arxiv.org/abs/2207.13332

12 Ding, Z. et al. (2025). “HALO: Half-Life-Based Outdated Fact Filtering in Temporal Knowledge Graphs.” arXiv:2505.07509. https://arxiv.org/abs/2505.07509

13 Anthropic (2025). “Effective context engineering for AI agents” and “Managing context on the Claude Developer Platform.” anthropic.com. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents

14 Huyen, C. (2025). “Agents.” huyenchip.com, 2025-01-07. (Knowledge-cutoff framing.) https://huyenchip.com/2025/01/07/agents.html

15 Meng, K. et al. (2022). “ROME: Locating and Editing Factual Associations in GPT.” arXiv:2202.05262. / “MEMIT.” arXiv:2210.07229. https://arxiv.org/abs/2202.05262