

When Agent Memory Becomes Stale: Provenance, Revocation, and Decision-Time Validation

Veera Ravindra Divi
Independent Researcher
Austin, TX
ravi3divi@gmail.com

Abstract

An agent can retrieve an accurate historical fact and still make an incorrect present-day decision. A supplier preference may have changed, a source document may have been revoked, or a summary may combine evidence governed by different permissions. This white paper treats persistent agent memory as a mutable evidence system. It synthesizes recent memory benchmarks and established provenance and authorization work, then proposes a decision-time eligibility protocol that checks scope, lineage, validity, permissions, and conflicts before memory influences a consequential action. A synthetic procurement example exposes failure modes that recall accuracy alone misses. A reproducible evaluation design separates legitimate completion from stale evidence use, revocation exposure, and operational cost. The protocol is an engineering proposal, not an implemented system or an experimentally validated defense. Its value is a concrete contract and falsifiable test plan for teams deploying agents across changing enterprise state.

The failure is valid recall of invalid evidence

Consider a procurement assistant that remembers a supplier as approved. The remembered statement accurately reflects last week's interaction. Today the supplier has been suspended. A similarity search returns the old statement, the model explains its choice coherently, and a purchase is submitted. Neither retrieval precision nor fluent reasoning establishes that the evidence remained eligible at the action boundary. The engineering question is whether memory is sufficiently current, authorized, and complete for this decision.

LongMemEval already evaluates knowledge updates, temporal reasoning, and abstention alongside other conversational memory abilities [1]. MemoryArena connects memory to interdependent multi-session actions [2]. EvoMemBench separates retention from revision and examines both knowledge-oriented and execution-oriented settings [3]. These are substantial advances; this paper does not claim that existing evaluation ignores change. It asks a narrower deployment question: how should an application enforce evidence eligibility when sources, permissions, and derived memories change on different schedules?

This question fits a distributed-systems treatment because persistent agent memory has a write path, derived views, asynchronous indexing, caches, and readers. Omri and colleagues explicitly characterize memory construction, retrieval, and generation, including the tradeoff between serving stale memory and waiting for construction [4]. The proposal here is a contract between these components, not a replacement for a memory algorithm. It can surround lexical retrieval, vector search, linked notes, or compressed summaries.

What the literature establishes, and what remains a design choice

The reviewed sources were selected for direct relevance to mutable memory, decision-time access, and dependency tracking, using primary papers and standards. This is a focused narrative synthesis, not a systematic review or a measured ranking of citation velocity. Contemporary benchmarks make memory revision and action dependence timely research directions; they do not establish that the specific protocol proposed below improves deployed systems.

MemoryArena carries memory across subtasks so earlier interactions can inform later actions [2]. EvoMemBench evaluates different forms of memory evolution and reports that a useful memory form depends on the task setting [3]. LongMemEval provides an earlier basis for testing whether assistants update and abstain appropriately [1]. Together these works motivate evaluating changing evidence in context, rather than assuming that higher static recall transfers to every workflow.

A-Mem constructs structured notes and links, with updates to existing memories as new information arrives [5]. Mem0 extracts and consolidates salient information and includes a graph-memory variant [6]. These mechanisms illustrate why the unit requiring validation may be a derived representation rather than a source document. A claim about a source can survive inside a summary even after its original index entry is removed. This is a possible failure

in systems that omit dependency invalidation, not an allegation that either cited implementation necessarily has that defect.

Two recent preprints sharpen the distinction between retrieving evidence and using it appropriately. MemCalib evaluates over-use and under-use of propositions supplied in composite memory blocks [9]. Its response-level influence judgments do not establish authorization or freshness. Wang and colleagues instead evaluate source-grounded evidence completion under a token budget, separating candidate granularity from graph-conditioned retrieval effects [10]. Their observable execution relations are not a complete causal-dependency oracle. The proposed protocol addresses a complementary enforcement question: whether a retrieved, source-supported claim remains eligible for this principal, purpose, and execution state. It does not claim to replace either memory-use calibration or evidence-completion evaluation.

W3C PROV-DM supplies established concepts for entities, activities, responsible agents, and derivation [7]. Zanzibar demonstrates authorization decisions that respect causal ordering amid access-list and object changes [8]. Neither source is an agent-memory solution. They offer useful ingredients: retain derivation information and make consistency requirements explicit. The remaining design choices include which dependencies invalidate a memory, how conflicting evidence is resolved, and what happens when a current policy check cannot complete.

Define eligibility independently of relevance

A memory item should carry a stable identifier; subject and tenant scope; original source identifiers and immutable versions; event time and ingestion time; applicable validity interval; transformation identifier; dependency identifiers; policy classification; and a lifecycle state. Useful lifecycle states are candidate, eligible, superseded, quarantined, and revoked. The retrieval index may return any candidate internally, but only eligible content should enter the agent's authorized decision context. Eligibility is decision-specific: a stored lifecycle label is only a preliminary hint and never substitutes for evaluating the current decision predicates.

For a decision d , principal u , time t , and memory m , define $\text{Eligible}(m,d,u,t)$ as the conjunction of five predicates: `ScopeMatch`, `LineageValid`, `TimeValid`, `AuthorizedNow`, and `ConflictResolved`. This is a specification, not a learned confidence score. `ScopeMatch` checks tenant, subject identity, and intended purpose. `LineageValid` checks the dependency versions required by the claim. `TimeValid` evaluates the applicable temporal rule. `AuthorizedNow` checks current permissions for this principal and use. `ConflictResolved` requires the application's explicit conflict policy to return an admissible claim.

Relevance remains a separate ranking signal applied to eligible candidates. A highly relevant item can be ineligible; an eligible item can be irrelevant. This separation prevents cosine similarity, model confidence, or a recent ingestion timestamp from acting as an accidental permission grant. It also makes failure reasons observable: an item can be rejected because a dependency was revoked rather than because the model judged its text unhelpful.

Event time and ingestion time must remain distinct. A delayed import of a three-month-old policy is newly ingested but not newly effective. Likewise, an authorized historical question may legitimately require a superseded version. The decision contract therefore specifies whether it asks for current operational state or a historical state at an explicit time. Historical eligibility must still respect current access rules; a past permission does not automatically authorize disclosure today.

Table 1. Eligibility checks proposed for decision-time memory use.

Check	Question	Failure behavior
Scope	Correct subject, tenant, purpose?	Exclude from context
Lineage	Required source versions valid?	Quarantine or rebuild
Time	Valid for this decision time?	Use historical mode only if requested
Authorization	Current principal permitted?	Deny exposure
Conflict	Authority rule resolves claims?	Defer or escalate

A decision-time validation protocol

Step 1: create an evidence envelope during ingestion. Preserve source versions and extraction provenance before summarization discards detail. For a derived claim, record the dependencies necessary to support that claim. A summary that mixes restricted and unrestricted facts should be split into claims or assigned the restrictive combined policy. If dependency granularity is unknown, the conservative default is to invalidate the whole summary when a material input changes. This sacrifices availability but avoids silently asserting unsupported independence.

Step 2: publish changes through a versioned control path. A source update, permission revocation, or correction emits an event that names affected source versions and an authoritative policy version or revocation epoch. A reverse dependency index marks dependent memories dirty. Index deletion is an optimization; the validation service must remain capable of rejecting an old candidate even while a vector index or cache still contains it. Event processing needs retries, deduplication, and reconciliation scans because a dropped invalidation event can otherwise become a persistent exception.

Step 3: retrieve within permitted scope. Apply tenant and coarse authorization filters before content is exposed to a model or reranker. Then validate candidate metadata against a sufficiently current authoritative state. A policy-version comparison is meaningful only within its defined authority and ordering domain; unrelated service counters cannot be compared as if they formed a global clock. Multi-authority decisions need a vector of versions or an explicitly coordinated authorization service.

Step 4: resolve evidence conflicts. Define authority rules per claim class. An authenticated approval registry may override a conversational preference for supplier eligibility, while a user's current instruction may govern presentation preferences. Newer does not always mean more authoritative. If equally authoritative sources disagree, return an unresolved conflict rather than merging them into a confident statement. A model may explain the discrepancy, but it should not invent the policy that settles it.

Step 5: bind the decision to its evidence. Record an evidence manifest containing accepted claim identifiers, source versions, policy versions, conflict outcomes, and the decision's intended action. Before a consequential external effect, revalidate that manifest at the execution gateway. Where supported, perform the check and effect against the same transactional snapshot or use conditional writes. If the tool cannot bind authorization and mutation atomically, state the residual time-of-check/time-of-use window and require compensating controls or human approval for actions that exceed the application's risk tolerance.

Step 6: make uncertainty operational. An unavailable validator yields a visible defer, restricted fallback, or escalation, according to the action class. It must not silently reuse a previously authorized result for a higher-risk action. Read-only historical answers may have a bounded fallback policy; purchases or disclosures may require current validation. Log the reason and reserve sufficient deadline budget for validation and escalation. Otherwise a system can appear safe only because it abandons nearly every difficult task.

A manifest should describe the minimal sufficient evidence for the action, rather than every item retrieved during planning. Otherwise a harmless update to an irrelevant source can invalidate an entire workflow. Preserve the broader trace for auditing, but distinguish supporting, rejected, and contextual claims. Sign or otherwise integrity-protect manifests crossing service boundaries, constrain their audience and action scope, and give them explicit expiry semantics. A replayed manifest must not authorize a different amount, recipient, tenant, or tool invocation. Token integrity alone does not establish freshness; the gateway still needs the required authoritative-state check.

Figure 1 separates memory retrieval from eligibility enforcement and action authorization. The invalidation path reaches derived claims, while the execution gateway checks the evidence manifest again before an effect.

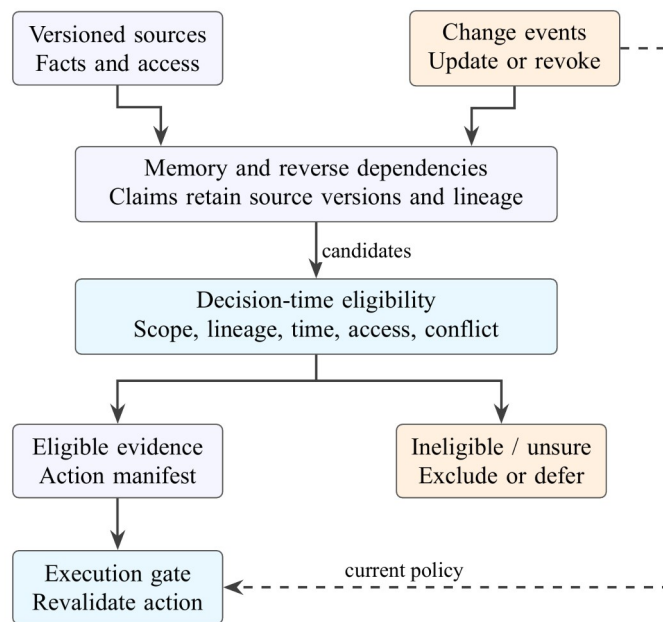


Figure 1. Memory eligibility as an enforcement path. Source and policy changes can invalidate derived claims even while an index retains them. The final execution gate must bind validation to the action or expose the residual race.

A worked procurement trace

The following sequence is synthetic and illustrates semantics rather than measured performance. At event A, supplier S is approved under registry version 17, and a user preference names S as the preferred vendor. The agent writes memory M1 and derives summary M2 from M1 and a delivery estimate. At event B, the registry publishes version 18 suspending S. The vector index has not yet incorporated the change. At event C, a buyer asks the agent to place an order.

A similarity-only system can return M2 because the summary remains semantically relevant. The proposed protocol checks M2's dependency on approval version 17, sees that the operational eligibility claim is no longer current, and prevents that claim from authorizing the order. It may still use the historical preference as a preference, provided that the buyer can access it, while searching for an approved alternative. Treating preference and eligibility as different claim types matters: invalidating one need not erase the other.

Now suppose the buyer loses access to the project between retrieval and execution. An ingestion-time permission tag cannot settle that change. The execution gateway must revalidate the principal and action. If the order service supports a conditional authorization token tied to current policy state, it can reject an outdated manifest. Without such enforcement, the architecture provides detection and a bounded race window rather than a guarantee against unauthorized effects.

A second variant revokes the delivery-estimate document but leaves supplier approval intact. If M2 lacks claim-level lineage, the system cannot establish which sentence remains supported, so it quarantines M2 and falls back to the original authorized approval record. A third variant asks why S was selected last week. The answer may cite version 17 as historical evidence while clearly stating that it cannot support a new order. These variants expose why deletion, temporal reasoning, and authorization are separate obligations.

An evaluation that cannot win by refusing everything

The primary hypothesis is that decision-time validation reduces operational use of ineligible evidence while preserving useful completion under an explicit cost and latency budget. This is falsifiable: validation may add excessive delay, suppress valid evidence, or fail when dependencies are incomplete. The evaluation should compare implementations under identical models, prompts, retrieval budgets, source histories, and workload arrival patterns. No such experiment has been conducted for this paper.

Construct a versioned testbed with synthetic tenants, principals, source documents, and an independent authoritative state machine. Include unchanged controls; factual corrections; future-effective updates; retroactive

corrections; permission revocations; conflicting authorities; stale caches; delayed invalidation; derived summaries; and changes between retrieval and execution. Cross source-change timing with action type and memory representation. Keep development scenarios separate from held-out event sequences, entities, and templates to reduce tuning to the test generator.

Compare five baselines: full authorized history at decision time; similarity retrieval with ingestion-time permissions; time-to-live expiration; source-version validation without lineage; and full lineage plus execution-boundary validation. The full-history baseline must obey the same access rules and context budget; it is an operational comparator, not an oracle. The independent state machine supplies ground truth. Add ablations that remove conflict resolution, reverse dependencies, and the final gateway check. Publish model snapshots, index settings, policy semantics, and the exact change-event schedule.

Report legitimate completion as correctly completed tasks divided by all tasks that are feasible and authorized under the ground-truth state. Report stale-action incidence as actions causally relying on invalid current-state claims divided by action opportunities containing such claims. Report unauthorized exposure as trials in which disallowed content reaches any unauthorized component divided by trials containing a relevant revocation or access restriction. Explicitly enumerate components, including rerankers and model prompts; a final-answer refusal does not undo an earlier disclosure.

Probe enforcement after revocation and measure the interval from authoritative effective time to the last observed prohibited use, with the observation window and probe resolution disclosed. If no prohibited use occurs, report none observed at that resolution rather than asserting instantaneous enforcement. If violations persist at the horizon, report a right-censored lower bound rather than a completed enforcement delay. Count falsely suppressed eligible claims over all ground-truth eligible claim opportunities. Report p50, p95, and p99 end-to-end task latency, construction backlog, validation calls, tokens, and storage overhead. Show both per-task and per-tenant distributions. A low exposure rate coupled with near-zero legitimate completion is an unusable operating point, not evidence of a successful defense.

Repeat stochastic runs with declared seeds and confidence intervals clustered at the independent task or event-sequence level. Paired comparisons should reuse the same histories and change schedules. Vary invalidation lag, dependency fan-out, authority-service availability, and change frequency. Report the Pareto frontier of completion, exposure, and cost rather than collapsing them into one arbitrary score. Preserve every failed and timed-out run, and test whether conclusions persist across at least two model families before making cross-model claims.

Causal reliance needs an operational definition before measurement. In a synthetic action environment, tag each action precondition with the authoritative claim identifiers it requires, then compare the executed action and evidence manifest with those preconditions. For free-form answers, separate citation correctness from actual behavioral dependence: an attached citation is not proof that the model used it. Counterfactual reruns that remove or replace an invalid claim can provide supporting evidence, but stochastic model variation limits causal certainty. Report the stricter machine-checkable action metric separately from any human- or model-judged attribution score.

To isolate lineage from retrieval organization, compare the same candidate texts and scorer with validation enabled and disabled, and separately vary flat versus source-aligned candidates. This follows the attribution concern in execution-provenance retrieval [10]. Add a memory-influence diagnostic that distinguishes ignoring needed eligible evidence from overusing ineligible or irrelevant claims, motivated by MemCalib [9]. Neither diagnostic replaces ground-truth action and exposure scoring; good retrieval coverage and well-calibrated prose can coexist with an unauthorized tool effect.

Operating limits and deployment choices

The protocol's strongest property is conditional: if all supporting dependencies are recorded, authoritative validation is current, every content path enforces the same policy, and execution is atomically bound to that state, an ineligible memory item cannot authorize the action through this path. Each condition is substantial. Missing lineage, bypass tools, untracked copies, or a race at an external service breaks the argument. This is an invariant of the specified control flow, not a proof that model-generated text is correct.

Provenance establishes origin and derivation, not truth. A perfectly tracked source can be wrong or malicious. The application still needs source authentication, input sanitization, and independent checks for high-impact assertions. Authorization establishes permission, not epistemic reliability. Conversely, revoking access does not necessarily

mean a fact is false. Keep truth status, access status, and temporal applicability in separate fields so operational policy does not corrupt historical analysis.

Revocation of use is also different from physical erasure. The proposed gate can stop future authorized-path retrieval while copies remain in logs, backups, caches, or model-provider systems. An actual erasure requirement needs a separate retention and deletion design, including derived artifacts and audit exceptions. Persistent memory should avoid unnecessary secrets and minimize logged content. Evaluation artifacts should use synthetic identities and revocations unless real-data permissions explicitly permit broader release.

Adoption can begin with read-only shadow validation: compare decisions with an authoritative state without exposing blocked content to production users. Next, enforce current-source checks for selected high-impact claim classes, then add lineage and gateway binding where integration supports them. Record false suppressions and operator overrides. Release decisions should depend on measured failure rates and utility constraints, not on a universal threshold invented for this paper. The remaining implementation work includes authority adapters, dependency maintenance, secure manifests, and workload-specific conflict policies.

The cost of dependency tracking can itself change the preferred memory representation. A compressed note may reduce prompt length while increasing the work needed to reconstruct support after a correction. A coarse dependency graph is cheaper to maintain but invalidates more content; fine-grained dependencies preserve more utility but can omit a subtle supporting relation. The evaluation should therefore sweep dependency granularity, report invalidation fan-out, and count the fraction of decisions that fall back to source retrieval. This makes provenance an explicit systems tradeoff rather than an assumed free annotation.

Conclusion

Persistent memory is useful only when its evidence remains appropriate for the decision being made. Recent benchmarks broaden the evaluation of memory beyond static recall, while distributed-systems practice supplies tools for provenance and consistent authorization. This paper combines those ideas into an explicit eligibility contract, a decision-time validation protocol, and a test plan that makes stale use, revocation exposure, and legitimate completion independently visible. The practical next step is to implement and challenge that contract under delayed updates, incomplete lineage, and external execution races. Until those tests are performed, the protocol remains a concrete engineering proposal with stated assumptions, not a validated reliability claim.

Acknowledgment

Prepared as an independent research white paper. The views expressed do not represent an employer. Proposed mechanisms require implementation and empirical validation.

References

- [1] D. Wu, H. Wang, W. Yu, Y. Zhang, K.-W. Chang, and D. Yu, "LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory," arXiv:2410.10813v1, Oct. 14, 2024. <https://arxiv.org/abs/2410.10813v1>
- [2] Z. He et al., "MemoryArena: Benchmarking Agent Memory in Interdependent Multi-Session Agentic Tasks," arXiv:2602.16313v1, Feb. 18, 2026. <https://arxiv.org/abs/2602.16313v1>
- [3] Y. Wang et al., "EvoMemBench: Benchmarking Agent Memory from a Self-Evolving Perspective," arXiv:2605.18421v1, May 18, 2026. <https://arxiv.org/abs/2605.18421v1>
- [4] Y. Omri et al., "Agent Memory: Characterization and System Implications of Stateful Long-Horizon Workloads," arXiv:2606.06448v1, Jun. 4, 2026. <https://arxiv.org/abs/2606.06448v1>
- [5] W. Xu, Z. Liang, K. Mei, H. Gao, J. Tan, and Y. Zhang, "A-Mem: Agentic Memory for LLM Agents," arXiv:2502.12110v1, Feb. 17, 2025. <https://arxiv.org/abs/2502.12110v1>
- [6] P. Chhikara, D. Khant, S. Aryan, T. Singh, and D. Yadav, "Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory," arXiv:2504.19413v1, Apr. 28, 2025. <https://arxiv.org/abs/2504.19413v1>
- [7] L. Moreau and P. Missier, eds., "PROV-DM: The PROV Data Model," W3C Recommendation, Apr. 30, 2013. <https://www.w3.org/TR/2013/REC-prov-dm-20130430/>
- [8] R. Pang et al., "Zanzibar: Google's Consistent, Global Authorization System," in USENIX ATC, 2019, pp. 33-46. <https://www.usenix.org/conference/atc19/presentation/pang>
- [9] R. Cao et al., "MemCalib: Benchmarking and Optimizing Memory Use in LLM Agents," arXiv:2609.24259v3, Oct. 8, 2026, preprint. <https://arxiv.org/abs/2609.24259v3>
- [10] Y. Wang, J. Ju, J. Zhang, Z. Sun, Y. Duan, M. Zheng, and T. Cai, "When Does Execution Provenance Help Agent Memory Retrieval?," arXiv:2609.25913v3, Oct. 25, 2026, preprint. <https://arxiv.org/abs/2609.25913v3>