Persistent Memory Systems for Autonomous AI Agents

AltLab Research Paper #001

AltLab Research

June 2026

1 Abstract

Modern large language model (LLM) agents operate almost entirely within the bounds of a finite context window. Every fact, instruction, and prior decision that does not fit inside this window is, for practical purposes, forgotten. This creates a fundamental mismatch between how these systems are built and how autonomous, long-running agents are expected to behave: an agent tasked with managing a project over months, collaborating with other agents, or learning from past mistakes cannot do so if its only memory is the text currently in front of it.

This mismatch is the *memory problem*. It manifests in three related ways. First, context windows, even when large, are bounded and expensive; naively appending all history to every prompt does not scale in latency or cost. Second, retrieval-augmented generation (RAG), the most common workaround, treats memory as a static document store rather than a dynamic, evolving record of an agent's experience, and it offers no principled mechanism for deciding what to keep, what to discard, or how memories relate to one another over time. Third, none of the dominant architectures address multi-agent settings, where several agents may need to read from and write to a common, consistent memory without conflict.

Persistent memory matters because it is the difference between a tool that answers isolated queries and an agent that

accumulates competence, context, and judgment over time. Without it, every session restarts from zero; with it, an agent can recognize recurring situations, avoid repeating errors, and build a working model of the people, projects, and environment it operates in.

This paper introduces **AltMemory**, a research architecture for persistent agent memory. AltMemory separates memory into working, episodic, semantic, and shared components, and introduces two explicit subsystems — a Memory Ranking Engine and a Forgetting Engine — that govern what enters long-term storage, how it is weighted at retrieval time, and how it decays. We present the architecture, motivate each design decision against the limitations of context-window-only and RAG-only systems, and propose a concrete experimental program for evaluating ranking, forgetting, retrieval, and multi-agent sharing. We are explicit about what AltMemory does not solve: it is a proposed architecture, not a deployed or benchmarked system, and several of its components (notably automated forgetting and multi-agent consistency) raise open problems that we discuss rather than resolve.

---

2 1. Introduction

2.1 1.1 The Limitations of Current AI Memory

Contemporary LLM-based agents are, by default, stateless between calls. Whatever an agent “knows” during a given interaction is determined entirely by what has been placed into its prompt. This design has been adequate for single-turn or short-session use cases, but it breaks down as soon as an agent is expected to operate continuously: across days, across tasks, or in coordination with other agents.

Three specific limitations recur across existing approaches.

2.2 1.2 Context Windows

Context windows have grown substantially, and some models now accept context measured in the hundreds of thousands of tokens. This growth has reduced, but not eliminated, the underlying problem. Three issues persist regardless of window size:

- **Cost and latency scale with context length.** Re-sending an agent’s entire history on every call is computationally wasteful, and the cost grows with the very accumulation of experience that persistent memory is supposed to enable.
- **Relevance degrades with volume.** Models do not attend uniformly to all tokens in a long context; information placed early in a very long prompt is frequently under-utilized relative to information placed near the end of it. A larger window does not guarantee that the right fact will be *used* at the right time.
- **A window is not a memory.** Context windows have no notion of importance, recency-weighted decay, or relationship between facts. They are a flat buffer, not a structured store. Even an arbitrarily large window would still require some process to decide what belongs in it for a given task.

2.3 1.3 Retrieval-Augmented Generation (RAG)

RAG addresses the scaling problem by storing information externally and retrieving only what appears relevant to a given query, typically via embedding similarity search over a vector database. This is a meaningful improvement over pure context-window reliance, but standard RAG implementations have limitations specific to agent memory:

- **RAG stores documents, not experience.** Most RAG pipelines are built around static or slowly-changing corpora (manuals, wikis, codebases). Agent memory is different: it is generated continuously by the agent’s own actions and must be written, not just read.
- **No principled forgetting.** Vector stores grow monotonically. Without an explicit forgetting mechanism, retrieval quality degrades over time as outdated, contradicted, or low-value memories accumulate and compete with current, useful ones at retrieval time.
- **Similarity is not the same as importance.** Embedding similarity retrieves what is topically related to a query, not what is most consequential, most recent, most reinforced, or most trustworthy. A memory that is semantically close to a query but represents a one-off, low-confidence observation may be retrieved ahead of a memory that is more central to the agent’s task.
- **No native multi-agent semantics.** Conventional RAG assumes a single reader/writer relationship between an application and its index. It does not define how concurrent agents should resolve conflicting writes, partition private versus shared knowledge, or avoid one agent’s noise polluting another’s retrieval results.

2.4 1.4 Long-Term Memory Challenges

Beyond the specific shortcomings of context windows and RAG, persistent memory for autonomous agents raises challenges

that are not yet well addressed by any widely deployed system:

- **Selective retention.** An agent that tries to remember everything will eventually retrieve worse, not better, because signal is diluted by noise. Some form of triage is required.
- **Temporal and causal structure.** Many agent tasks depend on *sequences* of events ("what happened, in what order, leading to what outcome"), which is poorly captured by similarity search over independent text chunks.
- **Consistency and contradiction.** Memories can become outdated or contradicted by later information. A persistent memory system must have some way to reconcile or supersede old facts rather than simply accumulating them.
- **Evaluation.** There is no broadly agreed-upon benchmark for "good agent memory," which makes it difficult to compare approaches or claim improvement with confidence. We return to this point in Section 7, where we propose, rather than report, a set of experiments.

AltMemory is motivated by these gaps. It does not claim to resolve them all; rather, it proposes a structured architecture within which they can be studied empirically.

---

3 2. Problem Statement

Designing persistent memory for autonomous agents requires answering four interrelated questions. We treat each as an open design problem rather than a solved one.

3.1 2.1 Which Memories Should AI Keep?

Not all information an agent encounters is worth retaining. A naive policy ("store everything") guarantees that retrieval quality will degrade as the store grows, since later retrieval must search through an increasing amount of irrelevant or redundant material. A useful retention policy must consider:

- **Task relevance** — does this information bear on goals the agent is likely to pursue again?
- **Novelty** — does this add information not already captured by an existing memory?
- **Source reliability** — was this observed directly, inferred, or reported by an untrusted party?
- **Reinforcement** — has this been corroborated by repeated observation, or is it a single, unverified instance?

We do not propose a single retention rule. Instead, Section 3 introduces a Memory Ranking Engine that scores candidate memories along several of these dimensions and a Forgetting Engine that revisits those scores over time, rather than committing to retention or deletion permanently at write time.

3.2 2.2 Which Memories Should AI Forget?

Forgetting is not simply the inverse of retention; it is a separate design problem with its own risks. Aggressive forgetting can discard information that turns out to matter later (catastrophic forgetting of the useful kind). Conservative forgetting allows stale or contradicted information to persist and compete with current information at retrieval time.

Candidate forgetting signals include:

- **Decay by recency** — older, unreferenced memories lose priority over time.
- **Supersession** — a memory is explicitly contradicted or replaced by newer information.
- **Low retrieval utility** — a memory that is rarely or never retrieved despite many opportunities is a candidate for pruning.
- **Explicit invalidation** — the agent, a user, or an external system marks a memory as incorrect.

A central open question, which we do not resolve in this paper, is whether forgetting should be implemented as hard deletion, soft decay (down-weighting without removal), or archival (moving to cold storage that is excluded from default retrieval but not destroyed). We discuss this tradeoff further in Section 5.5.

3.3 2.3 How Should Memories Be Ranked?

Given a query or task context, a memory system must decide which of potentially many relevant memories to surface, and in what order. We propose that ranking should be a function of at least:

- **Semantic similarity** to the current context (the conventional RAG signal),
- **Recency**,
- **Importance**, as estimated at write time and revised over time,
- **Access frequency**, as a proxy for demonstrated usefulness,
- **Source and confidence**, distinguishing directly observed facts from inferred or reported ones.

No single signal is sufficient on its own. Pure similarity ranking, as discussed in Section 1.3, conflates relatedness with usefulness. Pure recency ranking discards well-established, still-valid knowledge in favor of whatever happened most recently. Section 3.4 proposes a ranking engine that combines these signals; Section 7.1 proposes how that combination might be evaluated empirically rather than assumed correct.

3.4 2.4 How Can Multiple Agents Share Memory?

When more than one agent operates in a shared environment, several problems arise that do not exist in a single-agent setting:

- **Write conflicts** — two agents may produce contradictory observations about the same entity or event.
- **Privacy and scope** — not all of an agent's memory should be visible to every other agent; some memories are properly private (e.g., an agent's internal reasoning trace) while others are properly shared (e.g., a fact about a shared task).
- **Noise propagation** — without filtering, one agent's low-quality or erroneous memories can be retrieved by another agent that had no opportunity to evaluate the source.
- **Update propagation latency** — if shared memory is not synchronized promptly, agents may act on stale shared state.

We treat multi-agent memory sharing as the least mature part of the proposed architecture and flag it explicitly as an area requiring further empirical work (Section 7.4 and Section 8).

---

4 3. Proposed Architecture

We propose **AltMemory**, an architecture composed of nine interacting components. Figure 1 shows the overall structure; this section describes the role of each component and the rationale behind it.

4.1 3.1 Working Memory

Working memory is the agent's active context: the information currently available to the model for the task at hand, analogous to the context window of an LLM call. It is the only component that the underlying model interacts with directly. All other components exist to populate working memory with the most useful possible content, given finite space, and to capture what should be written out of it once the active task or turn concludes.

Working memory is deliberately treated as a *cache*, not a store of record. Nothing in working memory is assumed to persist unless explicitly written to one of the long-term stores described below.

4.2 3.2 Episodic Memory

Episodic memory holds records of specific events: what happened, when, in what context, and with what outcome. Examples include a record of a completed task, a user's stated preference expressed at a particular time, or an error the agent made and how it was corrected.

Episodic memory is structured around *traces* — time-ordered records — rather than isolated facts, because many agent tasks depend on understanding sequences (Section 1.4). This distinguishes it from semantic memory, which stores the distilled, time-independent content that episodic traces eventually produce.

4.3 3.3 Semantic Memory

Semantic memory holds generalized facts and concepts that are not tied to a specific event: stable preferences, learned relationships between entities, domain knowledge the agent has accumulated, and conclusions distilled from multiple episodic traces.

The relationship between episodic and semantic memory is intentionally similar to the distinction made in human memory research: episodic memory records "what happened," while semantic memory records "what is true," typically derived from many episodes over time. In AltMemory, this distillation is a function of the Memory Ranking Engine and is one of the more speculative aspects of the architecture, discussed further in Section 5.6.

4.4 3.4 Memory Ranking Engine

The Memory Ranking Engine is responsible for two related tasks: (1) scoring candidate memories at write time to determine their initial importance, and (2) scoring stored memories at retrieval time to determine their priority given a current query.

We propose a ranking function combining the signals identified in Section 2.3 — similarity, recency, importance, frequency of access, and source confidence — as a weighted combination rather than a single learned end-to-end score, so that each

component remains interpretable and independently tunable. We do not claim this weighting is optimal; Section 7.1 proposes how the weighting should be evaluated and adjusted empirically rather than fixed by assumption.

4.5 3.5 Forgetting Engine

The Forgetting Engine operates over episodic, semantic, and shared memory, applying the retention and decay logic discussed in Section 2.2. We propose that it run as a periodic maintenance process rather than synchronously with every write, since forgetting decisions benefit from observing a memory's usage pattern over time rather than being made once at creation.

The Forgetting Engine's output is not necessarily deletion. We propose three possible actions — decay (lower ranking weight), archive (remove from default retrieval, retain in cold storage), and delete (permanent removal) — and treat the choice between them as an empirical question rather than a fixed policy, since the cost of incorrectly deleting a memory is likely much higher than the cost of incorrectly retaining one.

4.6 3.6 Shared Agent Memory

Shared Agent Memory is a memory partition explicitly scoped for access by more than one agent. It is structurally similar to episodic and semantic memory but carries additional metadata: which agents may read or write to it, provenance (which agent produced a given memory), and conflict-resolution state when multiple agents have written contradictory information about the same entity.

We do not propose a single conflict-resolution policy in this paper. Section 2.4 and Section 7.4 frame multi-agent consistency as an open experimental question; a production system would likely need task-specific rules (e.g., last-write-wins, confidence-weighted merge, or explicit human adjudication) depending on the cost of being wrong.

4.7 3.7 Vector Storage

Vector storage holds embeddings of episodic, semantic, and shared memories, indexed to support approximate nearest-neighbor search. It is the substrate that the Retrieval Engine queries; it is not itself a decision-making component. We treat vector storage as a largely solved infrastructure problem (existing vector databases are adequate for this role) and focus the architecture's novelty on the layers that sit above it — ranking, forgetting, and retrieval logic — rather than on the index itself.

4.8 3.8 Retrieval Engine

The Retrieval Engine accepts the current working memory context as a query, performs similarity search against vector storage, and applies the Memory Ranking Engine's scoring to the candidate results before returning a final, bounded set of memories to be written into working memory. It is the component responsible for converting "what is topically related" (a property of the vector index) into "what should actually be surfaced now" (a property of ranking).

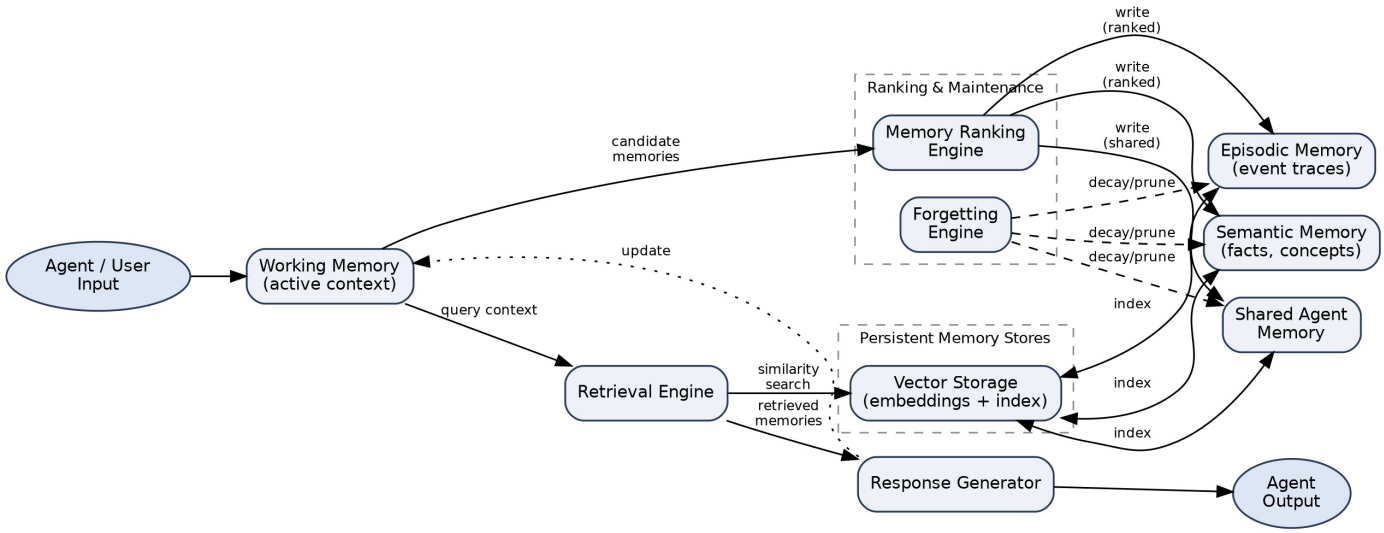
4.9 3.9 Response Generator

The Response Generator is the underlying language model call: it consumes working memory, including whatever the Retrieval Engine has surfaced, and produces the agent's output. After generation, it is responsible for proposing candidate new memories (e.g., a notable fact stated by the user, or the outcome of a task) for the Memory Ranking Engine to evaluate for storage. This closes the loop between memory and action: what the agent does and says becomes, after ranking, part of what it can later remember.

---

5 4. Architecture Diagram

Figure 1. AltMemory system architecture. Agent input flows into Working Memory, which forwards candidate memories to the Memory Ranking Engine for scoring and storage, and forwards the active context to the Retrieval Engine for similarity search against Vector Storage. The Retrieval Engine returns ranked memories to the Response Generator, which produces output and writes updates back into Working Memory. The Forgetting Engine operates asynchronously over Episodic, Semantic, and Shared Agent Memory.



AltMemory Architecture

---

6 5. Experiments

We propose four experimental programs corresponding to the open questions raised in Section 2. None of these experiments have been run; they are proposed as the validation plan for AltMemory, and we describe them at the level of design and metric rather than result.

6.1 5.1 Memory Ranking

Goal. Determine whether the multi-signal ranking function proposed in Section 3.4 retrieves more useful memories than similarity-only ranking.

Design. Construct a benchmark of agent sessions with known ground-truth “useful memory” labels per query (e.g., via human annotation of which prior facts were actually relevant to a correct response). Compare similarity-only ranking, recency-only ranking, and the proposed weighted combination, with weights tuned on a held-out split.

Metrics. Recall accuracy and precision of retrieved memories against ground-truth relevance labels; latency of the ranking step itself, since ranking is applied on top of vector search and must not dominate end-to-end response time.

6.2 5.2 Forgetting Algorithms

Goal. Evaluate whether decay-based forgetting improves retrieval quality over time relative to unbounded accumulation, and compare decay, archival, and deletion as forgetting actions.

Design. Run extended simulated agent sessions (thousands of turns) under each forgetting policy, periodically measuring retrieval quality on a fixed evaluation query set. A policy that forgets well should maintain or improve retrieval quality as the memory store grows; a policy that does not forget should show degradation.

Metrics. Recall accuracy and precision over time (to detect degradation or improvement as the store ages); context retention, defined here as the proportion of task-critical facts from early sessions that remain correctly retrievable in later sessions; latency, since unpruned stores should show increasing retrieval latency relative to pruned ones.

6.3 5.3 Semantic Retrieval

Goal. Evaluate the quality of the Retrieval Engine independent of ranking — i.e., whether the underlying vector search and query formulation surface the right candidate set before ranking is applied.

Design. Compare retrieval using raw working-memory context as the query against retrieval using a reformulated or summarized query, and compare different embedding models, on a fixed set of question-answer pairs requiring memory lookup.

Metrics. Recall accuracy and precision of the candidate set prior to ranking; latency of the search step.

6.4 5.4 Multi-Agent Shared Memory

Goal. Evaluate whether Shared Agent Memory improves multi-agent task performance relative to isolated per-agent memory, and measure the cost of conflicting writes.

Design. Construct a multi-agent task requiring information sharing (e.g., two agents jointly tracking the state of a shared project). Compare performance with no sharing, with naive shared storage (no conflict resolution), and with the conflict-aware Shared Agent Memory design proposed in Section 3.6.

Metrics. Recall accuracy of shared facts across agents; context retention of shared state across the task duration; precision, specifically the rate at which agents retrieve incorrect or superseded shared memories; latency of write propagation between agents.

6.5 5.5 Summary of Metrics

Metric	Definition	Primary Experiments
Recall Accuracy	Proportion of ground-truth relevant memories successfully retrieved	5.1, 5.2, 5.3, 5.4
Precision	Proportion of retrieved memories that are actually relevant or correct	5.1, 5.2, 5.3, 5.4
Latency	End-to-end and per-component time cost of memory operations	5.1, 5.2, 5.3, 5.4
Context Retention	Proportion of task-critical facts that remain correctly retrievable over session time	5.2, 5.4

Table 1. Summary of proposed evaluation metrics and the experiments in which each is primary.

---

7 6. Future Work

The architecture proposed here is a starting point, not an endpoint. We highlight four directions that follow naturally from its limitations.

7.1 6.1 Hierarchical Memory

The present design treats episodic, semantic, and shared memory as largely flat stores within their respective categories. A hierarchical extension would organize memory at multiple levels of abstraction — for instance, individual episodes rolling up into summarized “chapters,” which in turn inform semantic generalizations — allowing retrieval to operate at the level of granularity appropriate to a given query rather than always searching the finest-grained store.

7.2 6.2 Memory Compression

As episodic traces accumulate, storing every individual event indefinitely is unlikely to remain efficient even with forgetting in place. Memory compression — summarizing clusters of related episodes into compact representations without discarding their essential content — is a natural complement to the Forgetting Engine and may reduce the tension between retention and storage cost.

7.3 6.3 Self-Organizing Memory

The current design relies on explicit ranking and forgetting policies defined by the system designer. A longer-term direction is memory that reorganizes itself based on observed usage patterns: restructuring its own indices, adjusting its own ranking weights, or reclustering related memories, without requiring those policies to be hand-tuned. This is speculative, and we flag it as a research direction rather than a planned component.

7.4 6.4 AI-Native Operating Systems

At a broader scale, persistent memory is one instance of a more general question: what does an operating environment look like when its primary “process” is a continuously running agent rather than a human-directed application? An AI-native operating system would need to treat memory, scheduling, and resource access as first-class, agent-addressable primitives rather than as services bolted onto a conventional OS. AltMemory, in this framing, is a candidate memory subsystem for such an environment, but the broader systems question is well beyond the scope of this paper.

---

8 7. Conclusion

The memory problem in autonomous AI agents is not primarily a question of context window size. Larger windows reduce, but do not eliminate, the need for a system that decides what an agent should retain, what it should forget, how competing memories should be ranked, and how memory should be shared across multiple agents. Retrieval-augmented generation addresses the scaling half of this problem but offers no native answer to retention, forgetting, or multi-agent consistency.

We have proposed AltMemory, an architecture that separates agent memory into working, episodic, semantic, and shared components, and introduces explicit ranking and forgetting subsystems as first-class parts of the design rather than afterthoughts. We have been deliberate in stating what this paper does not establish: AltMemory is a proposed architecture, not a validated one. The ranking weights, forgetting policies, and multi-agent conflict-resolution rules described here are design proposals requiring empirical validation through the experiments outlined in Section 5. We see this as appropriate for an initial research paper from an independent lab: a clear articulation of the problem and a falsifiable architecture, rather than premature claims of a solved problem.

We expect the most difficult open questions to remain in multi-agent memory sharing and in forgetting policy, both of which involve tradeoffs (between agents' autonomy and consistency, and between storage efficiency and the risk of losing information that later proves important) that are unlikely to have a single correct answer independent of the task at hand. We intend AltLab Research Paper #002 to report empirical results from the experimental program proposed in Section 5.

---

9 Appendix A: Component Summary

Component	Function	Primary Failure Mode If Absent
Working Memory	Active context available to the model	Agent has no usable state at all
Episodic Memory	Time-ordered records of specific events	Agent cannot recall sequences or causes
Semantic Memory	Generalized, time-independent facts	Agent re-derives the same conclusions repeatedly
Memory Ranking Engine	Scores memories at write and retrieval time	Retrieval degrades to similarity-only matching
Forgetting Engine	Decays, archives, or deletes low-value memories	Store grows unbounded; retrieval quality degrades
Shared Agent Memory	Scoped cross-agent memory partition	Agents cannot coordinate on shared state
Vector Storage	Embedding index for similarity search	No efficient retrieval over large memory stores
Retrieval Engine	Converts similarity results into ranked, bounded context	Irrelevant or excessive context returned to the model
Response Generator	Produces output; proposes new candidate memories	No mechanism for new experience to enter memory

Table 2. Summary of AltMemory components and the consequence of omitting each.
