Memory in LLM Systems: Design Principles for Experience, Storage, and Recall

Jonathan Politzki[1]

[1]Jean Technologies, Inc.

Abstract

As large language models transition from stateless inference engines to persistent, agentic systems, the absence of robust memory mechanisms has emerged as a primary architectural bottleneck. Current approaches—retrieval-augmented generation, knowledge graphs, and expanding context windows—were largely retrofitted onto systems designed without memory as a first-class concern. This paper proposes a first-principles framework for reasoning about AI memory architecture. We begin by providing a formal functional definition of AI memory, distinguishing it from adjacent concepts such as retrieval-augmented generation and context engineering. We introduce the $\mathsf{E} \to \mathsf{S} \to \mathsf{R}$ (Experience, Storage, Recall) pipeline as a unifying operational framework that models memory as an active computational process rather than a passive data store. From the physical and economic constraints of transformer-based inference, we derive three design principles: the Generalization Limitation, which establishes memory as compensation for bounded model reasoning; the Memory Frontier, which formalizes the fundamental tradeoff between partitioning efficiency and semantic fragmentation; and the Routing Constraint, which governs the organization of partitioned context. Through the lens of these principles, we evaluate the dominant technical architectures—vector databases, knowledge graphs, hybrid systems, long-context inference, parametric fine-tuning, and emerging agentic memory swarms—and argue that the field is undergoing a transition from passive retrieval to active, autonomous memory processes. We conclude with an analysis of open problems including evaluation methodology, sleep-time compute, cross-application context portability, and dynamic self-organizing memory topologies.

1 Introduction

A companion paper in this series establishes that large language models, through training on vast corpora of human-generated text, implicitly encode structured representations of human psychology—personality traits, values, communication patterns, and cognitive styles [Politzki, 2026a]. If this thesis is correct, it raises an immediate practical question: how can systems *operationalize* that implicit understanding? The technology forks into two paths. *Context-first* approaches store and retrieve raw context about a user to maintain continuity across interactions—this is the domain of AI memory. *Representation-first* approaches compress the model's understanding of a person into mathematical embeddings for tasks such as matching and recommendation. This paper examines the context-first path; companion papers in this series address the representation-first path through cross-space alignment theory and contrastive embedding methods [Politzki, 2026b,c,d].

The dominant paradigm of modern artificial intelligence is the large language model (LLM), a neural network trained via next-token prediction on vast corpora of text. During training, these models encode extensive parametric knowledge into their weights, functioning as probabilistic databases of compressed linguistic and factual patterns [Chollet, 2019]. At inference time, however, the standard interaction is entirely stateless: each API call begins with an empty

slate, and the model's behavior is determined solely by the tokens present in the current context window. Without specific, relevant context injected into the prompt, the model defaults to the mean of its training distribution—a generic, undifferentiated response that reflects no awareness of the user, the task history, or the accumulated knowledge of prior interactions.

This fundamental statelessness creates three interrelated limitations that constrain the deployment of LLMs in persistent, personalized, and agentic settings:

1. **Statelessness.** Each inference call is independent. There is no native mechanism for an LLM to carry forward information from one interaction to the next. Any appearance of continuity must be explicitly engineered through external systems.

2. **Training cutoff.** The parametric knowledge encoded during training becomes stale immediately upon deployment. Events, facts, and user-specific information that postdate the training cutoff are inaccessible unless injected at inference time.

3. **Context window degradation.** Although context windows have expanded dramatically—from thousands to millions of tokens—the performance of attention-based retrieval degrades as context length increases. Empirical evidence demonstrates that models struggle to attend to information positioned in the middle of long contexts [Liu et al., 2024], and more recent work reveals systematic failures on tasks requiring precise retrieval from sequences exceeding fifty thousand tokens [Hsieh et al., 2024].

The response to these limitations has been a rapid proliferation of external memory architectures. What began as the simple concatenation of conversation histories into prompts has evolved through several generations: web search integration, retrieval-augmented generation (RAG) via vector databases [Lewis et al., 2020], knowledge graph construction [Edge et al., 2025], operating-system-inspired virtual memory management [Packer et al., 2023], and most recently, autonomous agentic memory systems that treat recall as an active reasoning process rather than a database query.

This proliferation, however, has proceeded largely without a unifying theoretical framework. Practitioners select architectures based on benchmarks, vendor recommendations, or familiarity, often without a systematic understanding of the fundamental tradeoffs that govern memory system design. The comprehensive survey by Hu et al. [2025], spanning seventy-seven pages and cataloguing over one hundred systems, provides an invaluable empirical map of the landscape. What remains lacking is a *principled framework*—a small set of design principles derived from first principles that explain *why* architectures are structured as they are and *how* to reason about the tradeoffs between them.

This paper provides such a framework. Our contributions are as follows:

1. **A formal definition of AI memory** that distinguishes it from adjacent concepts—retrieval-augmented generation, context engineering, and parametric knowledge—and establishes memory as a functional, active, experiential process (Section 2).

2. **The $\mathsf{E} \to \mathsf{S} \to \mathsf{R}$ pipeline** (Experience, Storage, Recall), a unifying operational framework that models memory as a three-stage computational pipeline optimized for downstream task performance (Section 3).

3. **Three design principles**—the Generalization Limitation, the Memory Frontier, and the Routing Constraint—derived from the physical and economic constraints of transformer-based inference, which collectively govern the design space of memory architectures (Section 4).

4. **A comparative architectural analysis** that evaluates vector, graph, hybrid, long-context, parametric, and agentic memory systems through the unified lens of these principles (Section 5).

5. **An analysis of the memory stack** as a layered abstraction hierarchy, and a forward-looking examination of open problems including evaluation methodology, sleep-time compute, and cross-application context portability (Sections 6–7).

We position this work as complementary to the exhaustive empirical catalogue of Hu et al. [2025]. Where that survey documents *what exists*, we analyze *how to think about it*. Our aim is to provide practitioners and researchers with a principled vocabulary and analytical framework for navigating the rapidly evolving landscape of AI memory.

The paper proceeds as follows. Section 2 establishes our formal definition of AI memory and its taxonomy. Section 3 introduces the $\mathsf{E} \to \mathsf{S} \to \mathsf{R}$ pipeline. Section 4 derives the three design principles. Section 5 evaluates technical architectures through these principles. Section 6 describes the memory stack abstraction. Section 7 examines open problems and future directions. Section 8 concludes.

2 Defining AI Memory

Despite the rapid proliferation of memory-augmented AI systems, the field lacks a consensus definition of what constitutes AI memory. The term is used loosely to describe everything from chat history concatenation to knowledge graph construction to model fine-tuning. This conceptual ambiguity impedes systematic comparison and principled design. We begin by establishing a formal functional definition, a taxonomy of memory types, and a precise delineation of the boundaries between memory and adjacent concepts.

2.1 A Functional Definition

We define AI memory as follows:

> **Definition 1.** *AI Memory refers to the mechanisms by which AI systems encode, organize, and recall information from experience to improve task performance.*

This definition is deliberately functional rather than architectural. It does not prescribe vector databases, knowledge graphs, or any particular implementation. Instead, it specifies three essential properties that any system must satisfy to qualify as a memory system:

1. **Functional.** The system must demonstrably improve task performance. Storage without downstream utility is archival, not memory. A memory system is evaluated not by how much it retains, but by the degree to which retained information enhances the quality of the model's outputs on subsequent tasks.

2. **Active.** The system must perform non-trivial processing—encoding, organization, or synthesis—beyond raw persistence. Writing conversation logs to a file does not constitute memory; extracting salient facts, resolving contradictions, and organizing information for future retrieval does.

3. **Experiential.** The information must derive from the system's own interactions and operational context, not from external corpora that the system has not directly processed. This distinguishes memory from web search (accessing information the system has not experienced) and from parametric knowledge (information absorbed during training but not consciously encoded).

The experiential criterion warrants further elaboration. Consider a customer support agent that, during a conversation, learns that a particular user prefers email communication over phone calls. This preference, extracted from direct interaction, constitutes a memory. By

contrast, if the same agent queries a web search engine to find the user's publicly listed contact preference, the retrieved information does not qualify as memory under our definition—it was not derived from the system's own experience. The distinction is not semantic pedantry; it has architectural implications. Experiential information requires encoding and consolidation pipelines that external retrieval does not.

2.2 Memory Taxonomy

We categorize AI memory along two orthogonal dimensions: *temporal scope* and *cognitive type*.

2.2.1 Temporal Scope

The most fundamental distinction is between short-term and long-term memory, summarized in Table 1.

Table 1: Temporal scope taxonomy of AI memory.

Property	**Short-Term Memory**	**Long-Term Memory**
Scope	Single session or conversation	Cross-session, persistent
Storage medium	Context window (in-context)	External database or store
Capacity	Bounded by context window	Theoretically unbounded
Latency	Zero (already in context)	Non-zero (requires retrieval)
Degradation	Lost in the middle effect	Fragmentation risk
Example	Current conversation history	User preference database

Short-term memory corresponds to the information present in the current context window. It is immediate, high-fidelity, and requires no retrieval mechanism, but is bounded by the context window size and subject to the degradation patterns documented by Liu et al. [2024]. Long-term memory persists across sessions in external storage, is theoretically unbounded in capacity, but requires explicit retrieval mechanisms that introduce latency and the risk of information loss through fragmentation.

2.2.2 Cognitive Types

Adapting terminology from cognitive psychology to the specific mechanisms of artificial intelligence, we identify four functional types of memory:

1. **Semantic Memory** stores timeless facts and general knowledge—for example, "the user's preferred programming language is Python" or "the company's fiscal year ends in March." The primary architectural challenges for semantic memory are version control, contradiction resolution, and entity updating as facts evolve over time. Semantic memory must support not only insertion but also modification and deletion.

2. **Episodic Memory** records specific interactions, events, and experiences anchored in time—for example, "during the meeting on February 15th, the team decided to postpone the product launch." The architectural challenges are temporal referencing, handling of recurring events, chronological ordering, and managing the cumulative volume of sequential interaction logs. Episodic memory captures the *what happened* of the system's experience.

3. **Procedural Memory** encodes workflows, operational procedures, and learned behavioral patterns—for example, "when deploying to production, always run the integration test suite before merging" or "this user prefers concise responses without code comments." Procedural knowledge is inherently fragile: encoding it too rigidly makes agents brittle in the face of

changing requirements, while encoding it too loosely permits deviation from critical protocols. Procedural memory captures the *how to act* of the system's operational knowledge.

4. **Associative Memory** represents the web of connections between disparate memories that enables higher-order inference. For example, from a collection of episodic memories about a user ordering diverse cuisines, visiting art galleries, and traveling to unusual destinations, an associative memory system might infer a high degree of "openness to experience" without this trait ever being explicitly stated. We propose associative memory as a distinct fourth type, complementing the standard semantic-episodic-procedural trichotomy from cognitive science. The capacity to derive implicit understanding from the juxtaposition of isolated data points is precisely the capability that distinguishes sophisticated memory systems from simple key-value stores.

It is important to note that memories are frequently hybrid. A single interaction may generate a semantic fact ("the user has two children"), an episodic record ("mentioned children during the March 3rd call"), a procedural update ("ask about children's ages when discussing scheduling"), and an associative link (connecting this information to previously stored family-related preferences). Strict categorical assignment is often brittle; effective memory systems must accommodate the inherently multi-faceted nature of experiential information.

2.2.3 Context Dimensions

Beyond temporal scope and cognitive type, memories vary along several additional dimensions that influence architectural decisions:

- **Modality**: text, images, audio, structured data, or multimodal combinations.
- **Structure**: a spectrum from fully structured (database records, JSON objects) to fully unstructured (free-form conversation transcripts). Structured representations enable precise querying but impose rigid schemas; unstructured representations preserve nuance but complicate retrieval.
- **Source and provenance**: individual-level memories versus enterprise-level shared knowledge; private information versus public records. The provenance of a memory determines access control policies, sharing permissions, and the degree of confidence assigned during recall.

2.3 Relationship to Adjacent Concepts

To sharpen the definition of AI memory, it is necessary to distinguish it from three frequently conflated concepts: LLM memory, retrieval-augmented generation, and context engineering.

Agent Memory vs. LLM Memory. The distinction between an LLM and an agent built on an LLM is central to understanding memory. An LLM possesses parametric memory—knowledge encoded in its weights during training—but no mechanism for updating this knowledge at inference time. An agent, by contrast, is an LLM augmented with tools, persistent state, and the capacity for autonomous action. Agent memory encompasses the full lifecycle of experiential information: formation, evolution, retrieval, and application. LLM memory is a subset, referring only to the parametric knowledge frozen at training time. As Hu et al. [2025] observe, the relationship is one of containment: LLM memory is a component within the broader agent memory system.

Memory vs. RAG. Retrieval-augmented generation [Lewis et al., 2020] is a specific retrieval pattern, not a memory system. RAG addresses the *recall* stage of memory—given a query, retrieve relevant documents—but does not encompass the formation, consolidation, evolution, or organization of memories. A memory system may use RAG as one component of its

recall mechanism, but RAG alone does not constitute memory. A vector database of chunked documents, queried via cosine similarity, is a retrieval system. A memory system additionally determines what is worth storing, how to organize it, when to update or delete it, and how to synthesize retrieved information into contextually appropriate responses.

Memory vs. Context Engineering. Context engineering [Mei et al., 2025, Anthropic, 2025] is the broader discipline of optimizing the information payload—instructions, tool schemas, retrieved documents, and state—injected into the model's context window to maximize inference quality. Memory is one component of the context engineering stack. Context engineering encompasses prompt design, tool configuration, system instruction optimization, and dynamic context assembly, of which memory retrieval is a single (albeit critical) element. The distinction is one of scope: context engineering manages the entire input to the model; memory manages the persistent, experiential knowledge that informs that input.

Table 2 summarizes these distinctions.

Table 2: Comparison of AI memory with adjacent concepts.

Concept	**Scope**	**Primary Function**	**Persistence**
LLM Memory	Parametric knowledge	Recall from training data	Static (frozen)
RAG	Retrieval mechanism	Query-time document retrieval	External corpus
Context Engineering	Full input optimization	Maximize inference quality	Transient
AI Memory	Full experiential lifecycle	Encode, organize, and recall experience	Persistent, evolving

3 The Experience-Storage-Recall Pipeline

Having defined AI memory and its taxonomy, we now introduce the operational framework that governs its implementation. We propose the $\mathsf{E} \to \mathsf{S} \to \mathsf{R}$ pipeline—Experience, Storage, Recall—as a unifying model for understanding how memory systems function. While Hu et al. [2025] describe analogous dynamics as *Formation*, *Evolution*, and *Retrieval*, our formulation explicitly models memory as a continuous computational pipeline optimized for downstream task performance, rather than a static database architecture with discrete lifecycle phases.

3.1 Overview

The central thesis of the $\mathsf{E} \to \mathsf{S} \to \mathsf{R}$ framework is that memory is not a noun—a store of facts—but a verb: an active, multi-stage computational process. The goal of the pipeline is to optimize the quality of Recall (R) for a defined Task ($\mathcal{T}$). Each stage performs distinct transformations on information, and the design decisions at each stage propagate downstream to determine the system's overall effectiveness.

The pipeline admits two broad pathway architectures for integration with the primary language model:

1. **Parallel architecture.** A background listener process operates alongside the primary LLM, continuously monitoring the interaction stream and asynchronously processing information through the Experience and Storage stages. Memory operations occur in parallel with inference, introducing no latency to the user-facing interaction. The Anthropic Memory SDK exemplifies this approach [Anthropic, 2025].

2. **Unified architecture.** The LLM itself invokes memory operations as tool calls—for example, via the Model Context Protocol (MCP). In this architecture, the LLM decides when to store and when to recall, treating memory as an explicit reasoning step within its agentic loop. MemGPT [Packer et al., 2023] pioneered this approach by modeling memory management as an operating-system-level concern within the agent's own reasoning.

Neither architecture is universally superior. The parallel approach minimizes inference latency and offloads memory management from the primary model's cognitive budget. The unified approach grants the model explicit control over what to remember and when, enabling more deliberate memory formation. In practice, hybrid designs that combine background processing with explicit memory tool calls are increasingly common.

3.2 Experience: The Input Layer

The Experience stage is the point of contact between the raw data stream—conversations, documents, tool outputs, environmental observations—and the memory system. It determines *what is worth remembering*.

In naive implementations, the Experience stage is passive: every token of every interaction is forwarded to Storage without filtering. This approach is computationally wasteful, introduces noise into the memory store, and degrades retrieval quality by expanding the search space with irrelevant entries. Empirically, the majority of tokens in a typical conversational interaction are ephemeral—greetings, acknowledgments, filler phrases, meta-commentary about the conversation itself—and carry no informational content worthy of long-term retention.

Effective Experience implementations operate as active filtering layers with the following properties:

- **Signal extraction.** The system identifies and extracts salient facts, preferences, decisions, and commitments from the raw interaction stream, discarding conversational noise.
- **De-duplication.** Information already present in Storage is identified and suppressed to prevent redundant entries that inflate the search space.
- **Classification.** Extracted information is classified by cognitive type (semantic, episodic, procedural, or associative) and tagged with relevant metadata (timestamps, source attribution, confidence scores).
- **Non-blocking execution.** In well-designed architectures, the Experience stage operates as an asynchronous background process, imposing no latency on the primary inference path. The interaction stream is processed in parallel, and extracted memories are written to Storage independently of the user-facing response generation.

The design of the Experience stage has outsized influence on overall system quality. Aggressive filtering risks discarding information that proves relevant in unforeseen future contexts. Permissive filtering risks the accumulation of noise that degrades retrieval precision. The optimal filtering threshold is task-dependent and constitutes an active area of research.

3.3 Storage: The Organization Layer

The Storage stage is responsible for organizing memories for future utility. It must not merely persist information; it must actively curate and maintain the memory topology over time.

The key design questions at the Storage stage are:

1. **Contradiction resolution.** When new information contradicts an existing memory ("the user previously preferred Python but now states a preference for Rust"), should the old

memory be updated, versioned, or deleted? Temporal ordering, source reliability, and recency all factor into this decision.

2. **Standalone vs. modification.** Should a new piece of information be stored as an independent memory or as a modification to an existing entry? The answer depends on whether the new information supersedes, supplements, or contextualizes the existing record.

3. **Entity relationships.** How should the system represent and maintain relationships between entities? A mention of "the user's daughter" should be linked to any existing entity record for that individual, enabling multi-hop retrieval.

4. **Granularity.** At what level of detail should memories be stored? Excessively granular storage (individual token-level records) inflates the search space; excessively coarse storage (paragraph-level summaries) loses important details. The optimal granularity depends on the expected recall patterns.

A critical concept for the Storage stage is **sleep-time compute**—background processing that reorganizes, consolidates, and optimizes the memory store during periods of inactivity. The term, introduced by Packer et al. [2023] in the context of memory management for LLM agents, draws a deliberate analogy to human memory consolidation during sleep. During sleep-time compute, the system performs operations that are too expensive or disruptive to execute during active interaction:

- **Consolidation**: merging redundant memories into unified, authoritative records.
- **Compaction**: summarizing verbose episodic logs into concise semantic facts.
- **Pruning**: removing obsolete, contradicted, or low-utility memories to maintain a lean search space.
- **Re-indexing**: updating embeddings, graph edges, and routing metadata to reflect the current state of the memory store.
- **Distillation**: extracting higher-order associative patterns from collections of lower-level memories.

Sleep-time compute transforms Storage from a static archive into a living, self-maintaining knowledge base. Systems that neglect this dimension accumulate "memory debt"—a growing body of stale, redundant, and contradictory entries that progressively degrades recall quality.

3.4 Recall: The Synthesis Layer

The Recall stage is the ultimate determinant of memory system value. It is the mechanism by which stored information is retrieved, synthesized, and presented to the primary model in a form that maximizes task performance. Critically, Recall is not merely retrieval; it is an act of *creative synthesis* that must assemble contextually relevant information from potentially disparate memory entries and present it in a coherent, actionable format.

We identify three levels of recall depth, ordered by increasing computational cost and synthesis capability:

1. **Static context injection** (`list_memory`). The simplest recall mechanism dumps a fixed set of memories—the most recent N entries, or all entries matching a predefined category—into the model's context window at the start of each interaction. This approach requires zero intelligence at recall time: no query formulation, no relevance scoring, no synthesis. It is fast, deterministic, and effective when the total memory volume is small enough to fit within the context budget. It fails when the memory store grows beyond this threshold, as indiscriminate injection wastes context capacity on irrelevant information.

2. **Semantic retrieval** (`search_memory`). Query-based retrieval using embedding similarity, keyword matching, graph traversal, or some combination thereof. The model (or the orchestrating system) formulates a query based on the current conversational context, and the retrieval system returns the top-k most relevant memories. This is the dominant paradigm in RAG-based systems [Lewis et al., 2020]. It introduces a dependency on query quality: the recall system can only return information that the query successfully addresses, and important context that is relevant but not directly queried will be missed.

3. **Intelligent query generation** (`deep_memory`). The most sophisticated recall mechanism employs an autonomous reasoning process that asks not "what matches this query?" but "what do I need to know to perform this task well?" Rather than executing a single retrieval pass, the system performs multi-hop, iterative retrieval: an initial query returns preliminary results, which inform the formulation of secondary queries, which in turn may trigger tertiary explorations. This agentic recall process is substantially more expensive in terms of latency and compute, but it is capable of discovering non-obvious connections, synthesizing information across memory types, and assembling context that no single query could have retrieved.

The fundamental measure of a memory system is not how much it remembers, but how effectively it mobilizes stored information to improve downstream task performance. Formally, if we denote the task performance metric as $\mathcal{T}$ and the recall function as R, then the optimization objective of the memory system is:

$$\max_{\mathsf{R}} \ \mathcal{T}\big(\mathsf{R}(\mathsf{S}(\mathsf{E}(x)))\big) \tag{1}$$

where x denotes the raw experiential data stream, E is the Experience function, S is the Storage function, and R is the Recall function. The composition $\mathsf{R} \circ \mathsf{S} \circ \mathsf{E}$ constitutes the full $\mathsf{E} \to \mathsf{S} \to \mathsf{R}$ pipeline. Each stage is a design variable; the objective is end-to-end task performance.

4 Three Design Principles

We now derive three design principles that govern the architecture of AI memory systems. These principles emerge from the physical, computational, and economic constraints of transformer-based inference, and they collectively define the design space within which all memory architectures operate.

4.1 The Generalization Limitation: Memory as Compensation for Bounded Reasoning

4.1.1 Premise

Modern large language models exhibit remarkable in-context learning capabilities but fundamentally limited generalization relative to human cognition. In the framework of Chollet [2019], intelligence is defined as skill-acquisition efficiency over a distribution of tasks, and generalization is the capacity to apply learned abstractions to novel situations with minimal data. While LLMs excel at pattern interpolation within their training distribution, they struggle with the kind of adaptive, compositional reasoning that humans perform effortlessly when confronting novel problems with sparse data.

4.1.2 Formalization

Let $\mathcal{T}$ denote task performance, $\mathcal{G}$ the generalization capability of the model, $\mathcal{D}$ the data (context) available at inference time, and $\mathcal{C}$ the complexity of the task. We model task performance as a function of these three variables:

$$\mathcal{T} = f(\mathcal{G}, \mathcal{D}, \mathcal{C}) \tag{2}$$

When $\mathcal{C}$ is high and $\mathcal{D}$ is low, the system must rely on $\mathcal{G}$ to bridge the gap between available evidence and the correct response. Human cognition excels in this regime—inferring latent structure from sparse observations, transferring abstractions across domains, reasoning counterfactually about unseen scenarios. LLMs, by contrast, degrade rapidly when required to operate outside the support of their training distribution.

The critical observation is that at inference time, $\mathcal{G}$ is effectively fixed. We cannot improve the frozen model's generalization capability during a conversation. The complexity $\mathcal{C}$ is determined by the problem and is likewise not a design variable. The *only* lever available to the system designer is $\mathcal{D}$: the quality and relevance of the data provided in context.

4.1.3 The Principle

> **Principle 1 (The Generalization Limitation).** *Because the generalization capability $\mathcal{G}$ of a deployed model is fixed and insufficient for high-complexity, low-data regimes, memory systems exist to maximize the quality and relevance of injected data $\mathcal{D}$, thereby compensating for bounded generalization by constraining the solution space.*

This principle provides the foundational justification for the existence of memory systems. It also explains an empirical observation that has become a practitioner heuristic: *a less capable model with excellent context frequently outperforms a more capable model with poor context.* Formally, if $\mathcal{G}_1 < \mathcal{G}_2$ but $\mathcal{D}_1 \gg \mathcal{D}_2$, then $f(\mathcal{G}_1, \mathcal{D}_1, \mathcal{C}) > f(\mathcal{G}_2, \mathcal{D}_2, \mathcal{C})$ for a wide range of practical tasks. The dominance of data quality over model capability in determining task performance echoes the Bitter Lesson of Sutton [2019]: general methods that leverage computation and data consistently outperform methods that rely on clever, human-engineered priors.

Memory, in this framing, is the mechanism by which $\mathcal{D}$ is maximized. The memory system's role is to ensure that every relevant piece of experiential information is available to the model at the moment it is needed, compensating for the model's inability to derive the correct answer from first principles alone.

4.2 The Memory Frontier: The Partitioning-Fragmentation Tradeoff

4.2.1 The Ideal and Its Impossibility

In an ideal theoretical scenario, memory recall would require no partitioning at all. The model would ingest an individual's entire experiential history—every conversation, every document, every preference ever expressed—into a single context window, and the attention mechanism would jointly attend to all of it, identifying relevant connections across the full span of available information.

This ideal is computationally intractable. The self-attention mechanism of the standard transformer architecture [Vaswani et al., 2017] computes pairwise attention scores between all tokens in the input sequence, yielding a computational complexity of $O(N^2)$ in both time and memory, where N is the sequence length in tokens. For a user with one million tokens of accumulated experiential data, a single forward pass would require approximately 10^{12} floating-point operations for the attention computation alone—a cost that scales quadratically with each additional token of context.

Beyond the raw computational cost, empirical evidence demonstrates that retrieval accuracy degrades as context length increases. Liu et al. [2024] show that models systematically fail to attend to information positioned in the middle of long contexts, a phenomenon termed the "lost in the middle" effect. Hsieh et al. [2024] further demonstrate that on tasks requiring

precise retrieval from noisy long contexts, performance collapses at context lengths beyond approximately fifty thousand tokens. The degradation is not merely a matter of cost; it is a fundamental limitation of the attention mechanism's capacity to maintain uniform retrieval quality across long sequences.

4.2.2 The Original Sin of Memory: Partitioning

To circumvent the quadratic wall, memory systems commit what we term the *original sin of memory*: they partition the context. Rather than passing all N tokens through a single attention computation, the system divides them into P partitions of approximately N/P tokens each and processes each partition independently (or retrieves only the most relevant partitions for inclusion in the context window).

This operation yields immediate computational benefits. If only $k \ll P$ partitions are retrieved for a given query, the effective context length drops from N to kN/P, and the attention cost drops from $O(N^2)$ to $O((kN/P)^2)$—a reduction that can be several orders of magnitude. Vector databases, which chunk documents into fixed-size segments and index them independently, represent the most aggressive form of this partitioning strategy.

However, partitioning introduces a fundamental cost: **semantic fragmentation**. When contiguous text is divided into independent chunks, the logical adjacency, causal relationships, and contextual dependencies between chunks are severed. Information that is meaningful only in conjunction—a pronoun and its antecedent, a conclusion and its premises, a decision and its rationale—may be split across partitions and rendered individually unintelligible. If the retrieval mechanism fails to recover all relevant partitions, the context presented to the model is not merely incomplete; it is *incoherent*.

4.2.3 The Frontier

The *Memory Frontier* formalizes the fundamental tradeoff between partitioning efficiency and semantic fragmentation. We define the total system cost $\mathcal{L}$ as the sum of two opposing terms:

$$\mathcal{L}(P) = \underbrace{\mathcal{L}_{\text{inference}}(P)}_{\text{decreasing in } P} + \underbrace{\mathcal{L}_{\text{fragmentation}}(P)}_{\text{increasing in } P} \tag{3}$$

where $\mathcal{L}_{\text{inference}}(P)$ captures the computational cost of inference (which decreases as partitioning increases, since less context is processed per query) and $\mathcal{L}_{\text{fragmentation}}(P)$ captures the cost of information loss due to fragmentation (which increases with more aggressive partitioning). The optimal operating point P^* satisfies:

$$P^* = \arg\min_P \; \mathcal{L}_{\text{inference}}(P) + \mathcal{L}_{\text{fragmentation}}(P) \tag{4}$$

This optimal point represents what we term **Minimum Viable Partitioning**: segment the data aggressively enough to enable efficient retrieval, but keep partitions large enough to preserve the logical connections within and across related memories.

4.2.4 Fast Memory and Slow Memory

The *Memory Frontier* induces a spectrum of architectural positions, which we characterize as *fast memory* and *slow memory*:

- **Fast Memory** (high partitioning): SQL databases, key-value stores, vector RAG systems. These architectures aggressively partition data into small, independently indexed units. Retrieval is fast—often sub-millisecond—and scales linearly with query volume. However, each partition is semantically impoverished, and the system relies entirely on the retrieval mechanism to reassemble coherent context from fragments. Fragmentation risk is maximal.

- **Slow Memory** (low partitioning): agentic reasoning over large context blocks, long-context inference, multi-step synthesis. These architectures maintain large, contiguous blocks of information, preserving logical structure and enabling the model to synthesize understanding from extended passages. Retrieval requires more compute and time, but the quality of the retrieved context is substantially higher. Fragmentation risk is minimal, but latency and cost are maximal.

No architecture can simultaneously achieve the speed of fast memory and the coherence of slow memory. The *Memory Frontier* is, in the language of multi-objective optimization, a Pareto frontier: improvements along one dimension necessarily entail costs along the other.

> **Principle 2 (The Memory Frontier).** *The partitioning of context into independently indexed units eliminates the quadratic attention cost but introduces semantic fragmentation. The Memory Frontier represents the Pareto-optimal tradeoff between inference efficiency and recall coherence. Every memory architecture occupies a position on this frontier, and there exists a Minimum Viable Partitioning that optimizes the joint cost for a given task distribution.*

4.3 The Routing Constraint: Organization Determines Recall Quality

4.3.1 The Problem

Given that some degree of partitioning is necessary (Principle 2), the question becomes: how should context be distributed across partitions? This is not a trivial question. Poorly organized partitions—where semantically related information is scattered across multiple, unrelated groups—amplify the fragmentation cost, because the retrieval mechanism must successfully query multiple disparate partitions to assemble a coherent context.

4.3.2 Formalization

Consider a memory store partitioned into P groups $\{G_1, G_2, \ldots, G_P\}$. Let $|G_i|$ denote the token volume of partition i. An efficient routing scheme satisfies two properties:

1. **Balance**: the token volume is approximately uniformly distributed across partitions, *i.e.*, $|G_i| \approx N/P$ for all i. This ensures that no single partition becomes a bottleneck for search and that retrieval load is distributed evenly.
2. **Coherence**: semantically related memories are co-located within the same partition or within a small number of adjacent partitions. This minimizes the number of partitions that must be retrieved to assemble a coherent context for any given query, thereby reducing fragmentation cost.

The tension between balance and coherence is the core challenge of the Routing Constraint. Perfectly balanced partitions (equal token volume) may distribute related information across many groups; perfectly coherent partitions (all related information co-located) may result in extreme imbalance, with some partitions containing millions of tokens and others nearly empty.

4.3.3 The Principle

> **Principle 3 (The Routing Constraint).** *Partitioning lowers the search space but introduces fragmentation. To minimize fragmentation cost, the system must organize partitions to maintain logical coherence across related memories while distributing search load approximately uniformly. The quality of memory organization determines the quality of recall.*

In practice, the Routing Constraint is addressed through organizational ontologies—predefined or learned schemas that group memories by topic, entity, time period, or cognitive type. A customer support agent might maintain separate partitions for product knowledge, customer history, and operational procedures. A personal assistant might partition by domain (work, family, health, finance) and sub-domain within each.

The design of these ontologies is often more art than science. The optimal partitioning depends on the distribution of queries the system will encounter, which may not be known in advance and may shift over time. This motivates research into dynamic, self-organizing partitioning schemes that adapt their topology based on observed query patterns—a direction we discuss further in Section 7.5.

5 Technical Architectures Through the Lens of Design Principles

We now evaluate the dominant technical architectures for AI memory, analyzing each through the unified lens of the three design principles established in Section 4.

5.1 Vector/Embedding-Based Systems (RAG)

Retrieval-augmented generation [Lewis et al., 2020] was the first widely adopted solution for augmenting LLMs with external knowledge and remains the most prevalent architecture in production deployments. The approach is straightforward: documents or memory entries are chunked into fixed-size segments, each segment is embedded into a dense vector representation, and at query time, the system retrieves the k segments whose embeddings are most similar to the query embedding.

Position on the Memory Frontier. RAG occupies the extreme right of the frontier: maximal partitioning. Each chunk is an independent unit, indexed solely by its embedding vector. There is no structural relationship between chunks; adjacency, causality, and co-reference are discarded at indexing time.

Strengths. RAG excels at semantic memory retrieval—locating specific facts that are lexically or semantically related to the query. Retrieval latency is low (typically sub-100ms for approximate nearest-neighbor search), the architecture scales gracefully to large corpora, and the infrastructure ecosystem is mature. For tasks that reduce to "find the document that contains the answer," RAG is highly effective.

Limitations. The aggressive partitioning that makes RAG fast also makes it fragile. Semantic fragmentation is maximal: related information split across chunks may not be co-retrieved, and the model receives a set of isolated fragments rather than a coherent narrative. RAG retrieves based on mathematical similarity in embedding space, not logical utility; information that is relevant but phrased differently from the query may be missed entirely. Furthermore, RAG provides no mechanism for memory evolution—updating, merging, or deleting memories—and treats the memory store as a static, append-only corpus.

Through the three principles. From the perspective of the Generalization Limitation, RAG provides data ($\mathcal{D}$) to compensate for limited generalization ($\mathcal{G}$), but the quality of that data is constrained by the quality of the query and the embedding model's ability to capture semantic nuance. From the perspective of the Memory Frontier, RAG accepts maximal fragmentation in exchange for minimal latency. From the perspective of the Routing Constraint, standard RAG imposes no organizational structure beyond the embedding space itself, relying entirely on vector similarity to route queries to relevant content.

5.2 Graph-Based Memory

Knowledge graph systems represent memories as entities (nodes) connected by typed relationships (edges). A memory entry such as "the user prefers Python for data analysis" is decomposed into entities (*user*, *Python*, *data analysis*) and relationships (*prefers*, *used for*), creating a structured, queryable representation.

Position on the Memory Frontier. Graph-based systems occupy a moderate position on the frontier. Individual memories are partitioned into discrete entity-relationship triples, but the graph structure preserves relational connections between them. This enables multi-hop traversal—following chains of relationships to discover information that is indirectly but meaningfully related to the query.

Strengths. Knowledge graphs directly address the fragmentation problem by encoding relationships as first-class objects. They excel in bounded, well-structured domains where the entity and relationship types are known in advance—enterprise knowledge management, inventory systems, organizational hierarchies. Multi-hop reasoning ("find all projects managed by people who report to the VP of Engineering") is natural and efficient in graph architectures.

Limitations. The rigidity that makes knowledge graphs powerful in structured domains makes them brittle in unstructured ones. Open-ended conversations do not conform to predefined ontologies; forcing free-form dialogue into entity-relationship triples requires aggressive interpretation that frequently loses nuance, misattributes relationships, or creates spurious connections. The extraction pipeline that converts raw text into graph structures is itself a source of error, and errors compound as the graph grows. Furthermore, knowledge graphs struggle with ambiguity, context-dependence, and the kind of soft, associative connections that characterize much of human experience.

Through the three principles. Graph-based systems provide structured data ($\mathcal{D}$) with explicit relational context, partially compensating for limited model generalization by making logical connections explicit rather than requiring the model to infer them. On the Memory Frontier, they sacrifice some retrieval speed (graph traversal is slower than vector lookup) for reduced fragmentation. The graph structure itself serves as a routing mechanism, addressing the Routing Constraint by organizing information around entities and their relationships.

5.3 Hybrid Systems

Recognizing that no single architecture optimally serves all memory types and query patterns, hybrid systems combine multiple retrieval mechanisms.

5.3.1 GraphRAG

GraphRAG [Edge et al., 2025] combines vector-based retrieval with knowledge graph traversal. Documents are simultaneously indexed as vector embeddings and decomposed into graph structures. At query time, the system performs both vector similarity search and graph traversal, merging the results to assemble context.

In principle, GraphRAG inherits the strengths of both approaches: the broad recall of vector search and the relational coherence of graph traversal. In practice, the architecture often inherits the *weaknesses* of both: the fragmentation of vector indexing and the brittleness of graph extraction, compounded by the complexity of merging heterogeneous retrieval results. GraphRAG is most effective when the graph relationships are narrow and well-defined enough to provide genuine routing value, rather than adding noise to the retrieval process.

5.3.2 Agentic RAG

Agentic RAG interposes an LLM reasoning layer between raw retrieval and final context injection. The architecture operates in three stages: (1) a broad, over-inclusive retrieval pass

gathers candidate memories from the vector store or graph; (2) a lightweight, fast reasoning model evaluates the retrieved candidates, filters irrelevant results, resolves contradictions, and synthesizes a coherent context; (3) the synthesized context is injected into the primary model's prompt.

This approach represents a critical insight: the fragmentation introduced by partitioning can be partially repaired by applying compute at recall time. The reasoning model serves as a "context sanitizer" that reassembles coherence from fragments. For memory stores exceeding approximately fifty thousand tokens per entity—where long-context inference becomes prohibitively expensive—agentic RAG offers an attractive position on the Memory Frontier: moderate partitioning with computational repair of fragmentation.

5.4 Long-Context as Memory

The rapid expansion of context windows—from 4,096 tokens in early GPT-3 to over one million tokens in recent models—has prompted the question of whether long-context inference can serve as a memory system by eliminating the need for partitioning altogether.

Position on the Memory Frontier. Long-context inference occupies the extreme left of the frontier: zero partitioning, zero fragmentation. The entire memory store is included in the context window, and the attention mechanism jointly attends to all of it.

Strengths. For one-shot analytical tasks—summarizing a large document, answering questions about a specific corpus—long-context inference is powerful. The model has access to all relevant information simultaneously, and the full-attention mechanism can identify connections across arbitrarily distant passages.

Limitations. For persistent memory, long-context inference is economically ruinous. The $O(N^2)$ attention cost makes it prohibitively expensive to process a million-token context on every interaction. Furthermore, the "lost in the middle" degradation [Liu et al., 2024] means that retrieval quality is not uniform across the context: information at the beginning and end of the window is attended to more reliably than information in the middle. For a memory system that accumulates information over weeks or months, the practical limitations of long-context inference make it unsuitable as a standalone architecture.

Useful pattern. Long-context inference is most effective as a component of a hybrid architecture. Background sleep-time compute jobs can process the full memory store using long-context inference, generating comprehensive summaries or analytical reports that are then cached and injected as compact context at the start of subsequent conversations. This pattern combines the synthesis capability of long-context inference with the efficiency of pre-computed context injection.

5.5 Fine-Tuning for Memory

Fine-tuning encodes information directly into the model's parameters, creating *parametric memory* that requires no retrieval mechanism at inference time.

Position on the Memory Frontier. Fine-tuning occupies a unique position outside the partitioning-fragmentation axis entirely. Information is not partitioned into external chunks but compressed into the model's weights. There is no retrieval step, no fragmentation, and no latency—the information is "always in context" by virtue of being part of the model itself.

Strengths. Fine-tuning excels at procedural memory: teaching the model *how to behave* rather than *what to know*. Company coding conventions, communication style preferences, domain-specific terminology, and workflow patterns are well-suited to parametric encoding. Once learned, these behaviors require no explicit recall and impose no context window overhead.

Limitations. Fine-tuning is the most expensive and least flexible form of memory. Each update requires a training run; information cannot be selectively updated, deleted, or versioned; and the risk of catastrophic forgetting—where new training overwrites previously learned

patterns—makes fine-tuning unsuitable for rapidly evolving semantic or episodic memory. Fine-tuning teaches the model how to *speak* (style, conventions, behavioral patterns) but not what to *know* (specific facts, user preferences, interaction histories). Conflating these two functions is a common architectural error.

Through the three principles. Fine-tuning directly modifies $\mathcal{G}$ rather than $\mathcal{D}$, making it the only architecture that addresses the Generalization Limitation by improving the model itself rather than augmenting its context. However, the improvement is narrow and brittle. The Memory Frontier does not apply, since there is no partitioning. The Routing Constraint does not apply, since there is no retrieval.

5.6 Agentic Memory Systems

The most recent and most ambitious architectural paradigm treats memory not as a passive store that responds to queries but as an active, autonomous system that performs intelligent reconnaissance on behalf of the primary model.

5.6.1 Deep Memory Swarms

We describe a three-phase architecture for agentic memory that we term *Deep Memory Swarms*:

1. **Scout phase.** A coordinator dispatches dozens of lightweight, fast model agents ("scouts") across the memory store. Each scout is assigned a partition, a data source, or a specific aspect of the query. Scouts operate in parallel, performing focused retrieval within their assigned domains.

2. **Iterative discovery phase.** When a scout identifies a potentially important finding, it can trigger secondary searches—following a reference to a related entity, expanding a time range, or exploring an adjacent topic. This iterative, self-directed exploration enables the system to discover non-obvious connections that no single query could have surfaced.

3. **Synthesis phase.** The findings from all scouts are aggregated and passed to a more capable reasoning model (the "synthesizer"), which filters redundant results, resolves contradictions, identifies the most task-relevant information, and assembles a coherent context payload for the primary model.

Position on the Memory Frontier. Agentic memory systems occupy a flexible position on the frontier. They can operate with relatively low partitioning, since the scout agents are capable of processing larger context blocks than a simple vector lookup. The iterative discovery phase actively repairs fragmentation by identifying and bridging connections between related but separated memories.

Through the three principles. Agentic memory systems maximize data quality ($\mathcal{D}$) through intelligent, multi-step retrieval, directly addressing the Generalization Limitation. They navigate the Memory Frontier dynamically, adjusting the effective level of partitioning based on the complexity of the query. And they implement sophisticated routing through the scouts' domain-specific assignments and the synthesizer's cross-domain integration.

5.6.2 Architectural Patterns

Agentic memory systems admit several organizational patterns:

- **Single agent**: one model manages all memory operations sequentially.
- **Network**: multiple agents communicate peer-to-peer, sharing findings.

- **Supervisor**: a central coordinator delegates tasks and aggregates results.
- **Hierarchical**: multiple levels of supervisors manage increasingly specialized sub-agents.
- **Custom orchestration**: hand-designed pipelines that combine elements of the above patterns based on task-specific requirements.

As inference costs continue to decline, the economic viability of agentic memory is improving rapidly. The approach is currently cost-prohibitive for high-frequency, low-latency applications, but for complex, high-value tasks where recall quality is paramount, the computational investment in agentic memory is justified by the quality of the resulting context.

5.7 Comparative Analysis

Table 3 summarizes the position of each architecture across the key dimensions defined by our design principles.

Table 3: Comparative analysis of memory architectures through the lens of the three design principles.

Architecture	Partitioning	Fragmentation	Latency	Cost	Best Suited For
Vector RAG	Very High	High	Low	Low	Fact retrieval, semantic search
Knowledge Graph	Moderate	Low	Moderate	Moderate	Bounded, structured domains
GraphRAG	High	Moderate	High	High	Relational + semantic queries
Agentic RAG	Moderate	Low	High	High	Large entity stores (>50K tokens)
Long-Context	None	None	Very High	Very High	One-shot document analysis
Fine-Tuning	N/A	N/A	None	Very High[†]	Procedural memory, style
Agentic Memory	Low–Moderate	Low	Very High	Very High	Complex multi-hop reasoning

[†]Upfront training cost; zero marginal inference cost.

Several patterns emerge from this comparison. First, there is a clear inverse relationship between retrieval latency and recall coherence: architectures that minimize latency (vector RAG) maximize fragmentation, while architectures that minimize fragmentation (agentic memory, long-context) maximize latency. Second, no single architecture dominates across all dimensions. This confirms the Pareto nature of the Memory Frontier: the optimal architecture is task-dependent, determined by the specific requirements for latency, cost, and recall quality.

Third, and most importantly for the trajectory of the field, the architectures are converging toward a common design pattern: *use compute to repair fragmentation*. Agentic RAG uses a reasoning model to sanitize retrieved fragments. Agentic memory uses scout agents to bridge connections across partitions. Even long-context inference, when used as a sleep-time compute mechanism for generating summaries, applies compute to reduce the effective context that must be processed at interaction time. The common thread is the recognition that partitioning is a necessary evil whose costs can be partially offset by intelligent, compute-intensive post-processing.

6 The Memory Stack

Memory is not a single tool or library; it is a multi-layered stack within the broader AI infrastructure. Understanding this stack is essential for architectural decision-making, as the choice of abstraction level determines the tradeoff between development velocity and system control.

6.1 Three Layers

We identify three layers of the memory stack, ordered by increasing abstraction:

1. **Foundational databases.** The lowest layer consists of raw storage and retrieval infrastructure: vector databases (Pinecone, Weaviate, Qdrant, ChromaDB), graph databases (Neo4j, Amazon Neptune), and traditional relational or document stores. These systems provide no memory-specific logic; they offer indexing, querying, and persistence primitives upon which memory systems are built. The developer is responsible for implementing the entire $\mathsf{E} \to \mathsf{S} \to \mathsf{R}$ pipeline—extraction, schema design, conflict resolution, retrieval logic—from scratch.

2. **Memory frameworks.** The middle layer provides opinionated abstractions over foundational databases. Frameworks such as Mem0 [Chhikara et al., 2025] and Zep [Rasmussen et al., 2025] implement the core $\mathsf{E} \to \mathsf{S} \to \mathsf{R}$ pipeline as a reusable library: automatic extraction of memories from conversations, storage with conflict resolution and deduplication, and retrieval via semantic search or graph traversal. The developer configures the framework's behavior (memory types, extraction rules, retrieval strategies) but does not implement the pipeline from scratch.

3. **Opinionated memory products.** The highest layer consists of end-to-end memory solutions that embed specific design philosophies and require minimal configuration. These products make strong assumptions about memory organization, retrieval strategies, and integration patterns, offering a "batteries included" experience at the cost of reduced flexibility. The Anthropic Memory SDK and the memory systems embedded in consumer-facing AI assistants exemplify this layer.

6.2 The Abstraction Tradeoff

The memory stack exhibits the standard abstraction tradeoff observed in all software engineering:

- **Higher abstraction** (memory products and frameworks) enables rapid development and reduces engineering burden, but constrains the system to the design assumptions embedded in the abstraction. If those assumptions do not match the application's requirements, the developer must either work around the abstraction or abandon it.
- **Lower abstraction** (foundational databases) provides full control over every design decision—partitioning granularity, embedding models, conflict resolution policies, retrieval algorithms—but requires substantially more engineering effort and domain expertise.

6.3 Horizontal and Vertical Dimensions

Orthogonal to the abstraction axis, memory systems vary along a horizontal-vertical dimension:

- **Horizontal (general-purpose)** systems are designed to serve any application domain. They make minimal assumptions about the structure of memories, the distribution of queries, or the nature of the task. Vector databases and general-purpose memory frameworks fall into this category.

- **Vertical (domain-specific)** systems are optimized for a particular application domain—healthcare, legal, customer support, personal assistance—and embed domain-specific ontologies, extraction rules, and retrieval strategies. They sacrifice generality for performance within their target domain.

There is no universal optimal position on either axis. A startup building a personal AI assistant may benefit from a high-abstraction, horizontal memory product that enables rapid prototyping. An enterprise deploying a domain-specific agent with strict compliance requirements may require a low-abstraction, vertical solution built on foundational databases with custom $\mathsf{E} \to \mathsf{S} \to \mathsf{R}$ logic. The choice is determined by the intersection of the application's requirements, the team's engineering capacity, and the maturity of available abstractions.

7 Open Problems and Future Directions

The field of AI memory is in a period of rapid evolution. Several fundamental problems remain unresolved, and the trajectory of the field points toward a set of emerging research directions that we examine in this section.

7.1 The Agentic Transition

The dominant narrative in the evolution of AI memory is the transition from passive retrieval to active, autonomous memory processes. RAG, the first widely adopted memory architecture, was a *transitional technology*: a solution retrofitted onto stateless LLMs before the tools existed to build memory as a first-class system concern. RAG was the obvious initial approach because the necessary infrastructure—embedding models, vector databases, approximate nearest-neighbor search—already existed for information retrieval applications.

However, as the tooling for agentic systems matures—structured output, function calling, tool use, and orchestration frameworks—the limitations of passive retrieval become increasingly apparent. A vector similarity search cannot reason about *what information is needed*; it can only retrieve information that is similar to the query it is given. The transition to agentic memory flips the paradigm: rather than the user (or orchestrating system) formulating queries for the memory store, the memory system itself reasons about what information would be most useful for the current task and autonomously retrieves, synthesizes, and presents it.

This transition is already visible in the design philosophy of recent systems. The Anthropic Memory SDK treats memory as a background process that runs in parallel with inference, continuously extracting and organizing information without explicit user intervention [Anthropic, 2025]. Letta (formerly MemGPT) models memory as an operating-system-level concern that the agent manages through explicit tool calls [Packer et al., 2023]. The generative agents of Park et al. [2023] demonstrated that autonomous memory retrieval and reflection can produce emergent social behaviors in simulated environments.

The agentic transition represents a fundamental shift in the locus of intelligence in the memory pipeline. In passive systems, intelligence resides in the query formulation (typically performed by the orchestrating system or the user). In agentic systems, intelligence resides in the memory system itself, which determines not only *what* to recall but *when* and *how much* to recall.

7.2 Evaluation Challenges

The evaluation of memory systems remains an unsolved problem. Current benchmarks, such as LoCoMo [Hu et al., 2025], evaluate memory systems on retrieval accuracy: given a query, can the system retrieve the correct memory? This framing is insufficient for two reasons.

First, retrieval accuracy is a proxy metric, not an end-to-end performance measure. A memory system that retrieves the correct documents but presents them in a format that confuses the primary model has high retrieval accuracy but low functional value. Conversely, a system that retrieves slightly less relevant documents but synthesizes them into a coherent, actionable context may achieve superior task performance despite lower retrieval accuracy. The measure of a memory system should be downstream task performance—the degree to which memory improves the quality of the model's outputs—not intermediate retrieval precision.

Second, existing benchmarks are susceptible to overfitting. When a standard benchmark becomes the primary evaluation criterion, system designers optimize for that specific benchmark, often at the expense of general capability. Rasmussen et al. [2025] have noted that benchmark performance can be inflated through architectural choices that are well-suited to the benchmark's specific question types but do not generalize to real-world memory workloads.

Developing robust evaluation methodologies for AI memory is an open research challenge. We identify several desiderata for a principled evaluation framework:

- **End-to-end task evaluation**: measure downstream task performance, not intermediate retrieval metrics.
- **Temporal dynamics**: evaluate performance over extended interaction histories that span days, weeks, or months.
- **Contradiction handling**: test the system's ability to resolve conflicting information across time.
- **Negative recall**: evaluate the system's ability to *not* retrieve irrelevant information, which can be as important as retrieving relevant information.
- **Generalization**: assess whether the system performs consistently across diverse domains, user populations, and interaction patterns.

7.3 Sleep-Time Compute and Background Processing

The concept of sleep-time compute, introduced in Section 3.3, represents a nascent but critical research direction. The analogy to human memory consolidation during sleep is more than metaphorical: both systems face the same fundamental challenge of converting high-volume, noisy experiential data into organized, retrievable knowledge.

In human memory consolidation, sleep serves several functions: transferring episodic memories from the hippocampus to neocortical long-term storage, strengthening important memories while weakening trivial ones, extracting abstract patterns from specific experiences, and integrating new information with existing knowledge structures [Bengio, 2025]. Artificial memory systems require analogous processes: consolidating redundant entries, pruning obsolete records, extracting higher-order patterns, and updating organizational schemas.

The key architectural challenge is determining *when* and *how aggressively* to perform background processing. Conservative approaches run periodic batch jobs that perform basic deduplication and compaction. Aggressive approaches continuously reorganize the memory topology, re-embedding entries, updating graph structures, and generating new associative connections. The optimal approach depends on the rate of memory accumulation, the stability of the information domain, and the latency tolerance of the application.

Sleep-time compute also opens the possibility of *anticipatory recall*: background processes that pre-compute likely context needs based on scheduled events, historical interaction patterns, or environmental signals. A personal assistant that knows a user has a meeting with a particular client in an hour could pre-assemble relevant context during idle time, reducing recall latency when the interaction begins.

7.4 Cross-Application Context

Perhaps the most consequential open problem in AI memory is the challenge of cross-application context. Currently, every application that implements memory does so in isolation. A user's personal assistant, email client, project management tool, and health application each maintain independent memory stores with no mechanism for sharing information across application boundaries.

This isolation creates a fundamental limitation: no single application has access to the user's full experiential context. The personal assistant does not know about the health concerns discussed with the medical AI. The project management tool does not know about the family commitments that affect the user's availability. Each application operates with a partial, fragmented view of the user's identity.

The resolution of this fragmentation is what we term the **cross-application context problem**: the challenge of creating a portable, privacy-respecting mechanism for sharing experiential context across application boundaries. This is, in our assessment, the most valuable unsolved problem in AI memory, because its resolution would enable qualitative improvements in personalization and task performance that are unachievable within siloed architectures.

Several approaches to cross-application context have been proposed:

- **Centralized context stores**: a single, trusted repository that aggregates memories from all applications and provides a unified retrieval interface. This approach maximizes data availability but raises significant privacy and trust concerns—the central store operator has access to the user's entire digital identity.
- **Federated architectures**: each application maintains its own memory store, but a federated protocol enables cross-application queries without centralizing data. This preserves data sovereignty but introduces latency and complexity.
- **Portable latent representations**: rather than sharing raw memories, applications share compressed, domain-invariant representations of user identity. This approach, explored in the context of general personal embeddings [Politzki, 2024], promises privacy-preserving portability but requires advances in representation learning to ensure that compressed representations retain sufficient information for cross-domain utility.

The cross-application context problem is not purely technical; it is also a business and governance challenge. The application that owns the user's context layer possesses an extraordinary competitive advantage, as every other application must integrate with it to access the user's full identity. The resolution of this problem will be shaped as much by market dynamics, regulatory frameworks, and user trust as by technical architecture.

7.5 Dynamic Partitioning

The Routing Constraint (Section 4.3) highlights the importance of memory organization for recall quality. Current systems rely on hand-designed ontologies—predefined categories and schemas into which memories are classified at storage time. These static ontologies are brittle: they reflect the designer's assumptions about query patterns, which may not match reality, and they cannot adapt as the distribution of memories and queries evolves over time.

Dynamic partitioning refers to self-organizing memory topologies that adapt their structure based on observed usage patterns. Rather than classifying memories into predefined categories, a dynamic partitioning system monitors which memories are frequently co-retrieved, which queries consistently fail to find relevant information, and which partitions are disproportionately large or small. Based on these observations, it reorganizes the memory topology: splitting overloaded partitions, merging underutilized ones, and re-routing memories that are more frequently accessed in contexts different from their original classification.

The technical challenges are substantial. Dynamic partitioning requires a feedback loop from the Recall stage back to the Storage stage—information about recall quality must inform storage organization. This creates a potential instability: aggressive reorganization can disrupt existing retrieval patterns, while conservative reorganization may fail to adapt quickly enough to changing query distributions. Developing stable, convergent algorithms for dynamic memory topology optimization is an open research problem with significant practical implications.

8 Conclusion

Memory is not retrieval-augmented generation. It is not a vector database, a knowledge graph, or an expanding context window. It is a complex, active computational pipeline that filters experiences, organizes them for future utility, continuously reorganizes and consolidates them, and synthesizes them into task-relevant context at recall time.

This paper has proposed a principled framework for reasoning about AI memory architecture. We began with a formal functional definition that establishes memory as an active, experiential process oriented toward improving task performance. We introduced the $\mathsf{E} \to \mathsf{S} \to \mathsf{R}$ pipeline—Experience, Storage, Recall—as a unifying operational model that captures the full lifecycle of experiential information, from initial filtering through persistent organization to task-optimized synthesis.

From the physical and economic constraints of transformer-based inference, we derived three design principles that govern the architecture of memory systems. The Generalization Limitation establishes that memory exists to compensate for the bounded reasoning capacity of deployed models, providing data to constrain the solution space where generalization fails. The Memory Frontier formalizes the fundamental tradeoff between the efficiency gains of context partitioning and the information loss of semantic fragmentation, defining a Pareto frontier along which all architectures must operate. The Routing Constraint specifies that the quality of memory organization—the distribution and coherence of partitioned context—determines the quality of recall.

Through the lens of these principles, we evaluated the dominant technical architectures and found that the field is converging on a common insight: the costs of partitioning can be partially offset by applying intelligence at recall time. This observation drives the transition from passive retrieval (querying a database) to active memory (deploying autonomous agents that reason about what information is needed and synthesize coherent context from fragmented stores).

The open problems we have identified—evaluation methodology, sleep-time compute, cross-application context portability, and dynamic self-organizing memory topologies—represent the next frontier of research. As AI agents become more sophisticated and autonomous, their memory architectures must evolve correspondingly, from simple databases that respond to queries to intelligent systems that actively manage, organize, and mobilize experiential knowledge. The design principles proposed in this paper provide a framework for navigating this evolution.

Finally, we note that this paper addresses one branch of a fundamental fork in how AI systems can operationalize their implicit understanding of humans [Politzki, 2026a]. The *context-first* approach examined here—storing and retrieving experiential context to maintain continuity—is complementary to the *representation-first* approach, which compresses that understanding into dense mathematical embeddings suitable for matching, recommendation, and cross-domain transfer. Companion papers in this series examine the representation-first path: the geometric alignment of representations across training objectives [Politzki, 2026b], cross-domain prefix alignment via shared Matryoshka embeddings [Politzki, 2026c], and the synthesis toward universal human embeddings [Politzki, 2026d]. Context is the oxygen that feeds representations—the memory systems examined here generate the rich, persistent context from which representation-first systems can extract structured understanding. The two approaches

are not competitors but complements, and a complete account of AI personalization requires both.

References

Anthropic. Building effective agents and the art of context engineering. Technical report, Anthropic, 2025.

Bengio, Y. The minds of modern AI. *Keynote address, NeurIPS*, 2025.

Chhikara, P., Khare, A., Thakkar, D., and Patel, T. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Chollet, F. On the Measure of Intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., and Larson, J. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2025.

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., and Ginsburg, B. NoLiMa: Long-context evaluation beyond literal matching. *arXiv preprint arXiv:2411.11522*, 2024.

Hu, Y., Liu, S., Yue, Y., et al. Memory in the age of AI agents: A survey. *arXiv preprint arXiv:2512.13564*, 2025.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Mei, K., Zhang, Z., Li, X., Liu, Y., and Wang, M. Context engineering for AI agents: A survey. *arXiv preprint arXiv:2502.09164*, 2025.

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., and Gonzalez, J. E. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2023.

Politzki, J. General personal embeddings: A case for universal human representations. Substack, 2024.

Politzki, J. On the implicit encoding of human psychology in LLM representations. Working paper, Jean Technologies, Inc., 2026.

Politzki, J. Geometric alignment across representation spaces: A survey of measurement, methods, and convergence. Working paper, Jean Technologies, Inc., 2026.

Politzki, J. Shared Matryoshka embeddings: Cross-domain prefix alignment for person representations. Working paper, Jean Technologies, Inc., 2026.

Politzki, J. Towards universal human embeddings: Cross-domain alignment of person representations. Working paper, Jean Technologies, Inc., 2026.

Rasmussen, D., Shipper, P., and Gallagher, T. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Sutton, R. S. The bitter lesson. Blog post, http://www.incompleteideas.net/IncIdeas/BitterLesson.html, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008, 2017.