AI Memory Architecture for Large Language Models

From Context Windows to Persistent Intelligence: A Comprehensive Technical Survey

Anjan Goswami

General Manager, SmartInfer.com

January 2026

Abstract: Memory is the foundation enabling AI systems to retain, recall, and leverage information across interactions. This comprehensive survey examines memory architectures for large language models through the lens of the 3D-8Q taxonomy proposed by Wu et al. [1], which classifies memory along three dimensions: object (personal vs. system), form (parametric vs. non-parametric), and time (short-term vs. long-term). We analyze major architectural paradigms including virtual context management (MemGPT/Letta) [2], neural long-term memory (Titans) [3], retrieval-augmented generation [4], and graph-based memory systems [5]. The survey covers KV cache optimization techniques achieving 96%+ memory utilization through PagedAttention [6] and FlashAttention [7], production systems like Mem0 demonstrating 26% accuracy gains [8], and emerging neural memory modules from Meta [9] that add 128B parameters without proportional compute. We examine GPU memory hierarchies from registers to HBM3e, NVIDIA's open-source infrastructure including TensorRT-LLM and Dynamo, and disaggregated serving architectures processing 100+ billion tokens daily [10]. The analysis reveals that learned memory modules can outperform retrieval-based approaches for long-context reasoning while the convergence of neural memory with hardware-optimized KV management represents the next frontier in LLM serving infrastructure.

Contents

1. Introduction

The race to give large language models persistent, human-like memory is fundamentally reshaping how AI systems learn, remember, and reason. Memory architectures have evolved from simple context windows to sophisticated multi-tier systems that combine neural parameters, external databases, and graph structures—mirroring human cognitive memory more closely than ever before. This transformation enables AI agents to maintain coherent identities across sessions, accumulate knowledge over time, and reason over million-token contexts, marking a paradigm shift from stateless inference to truly persistent intelligence.

Memory, in both human cognition and artificial intelligence, refers to the process of encoding, storing, and retrieving information [1]. For humans, this allows retention of experiences, knowledge, skills, and facts over time. In the era of large language models, memory refers to the ability of an AI system to retain, recall, and use information from past interactions to improve future responses. The development of sophisticated memory architectures has become critical as applications demand increasingly complex reasoning over extended contexts.

Contemporary LLMs face fundamental memory limitations that constrain their practical utility. Despite dramatic increases in context length—from 2,048 tokens in early GPT models to over 200,000 tokens in Claude 3 and Gemini 1.5—these windows remain finite. A typical business document corpus easily exceeds these limits, forcing applications to either truncate information or implement external retrieval mechanisms. The self-attention mechanism at the heart of transformers computes pairwise relationships between all tokens, resulting in $O(n^2)$ complexity for both computation and memory. While the KV cache reduces per-token generation cost, the fundamental scaling challenge remains for long contexts. Furthermore, standard LLM deployments treat each conversation independently, with no persistence between sessions, requiring users to repeatedly provide context while systems cannot learn from accumulated interactions without explicit fine-tuning.

This survey provides a comprehensive analysis of memory architectures for LLMs, covering foundational taxonomies, architectural paradigms, optimization techniques, and production systems. We synthesize research from academic institutions, major technology companies, and open-source communities to provide actionable insights for researchers and practitioners building memory-augmented AI systems. The key contributions include a detailed analysis of the 3D-8Q memory taxonomy [1] with mappings to production systems, comprehensive coverage of architectural paradigms from MemGPT [2] to Titans [3], technical analysis of KV cache optimization spanning PagedAttention [6] and FlashAttention [7], and identification of open problems and future research directions through 2026.

2. Foundational Taxonomy and Theoretical Framework

The theoretical foundation for AI memory draws from decades of cognitive science research, particularly the Atkinson-Shiffrin Multi-Store Model [11], which segments human memory into distinct stores with different characteristics. Understanding these parallels provides crucial design insights for AI memory systems and reveals both the potential and limitations of current approaches.

2.1 Human-AI Memory Parallels

The Atkinson-Shiffrin model identifies three primary memory stores: sensory register, short-term store (working memory), and long-term store. Each component finds direct analogs in modern LLM architectures, though with important differences in implementation and constraints.

In humans, sensory memory briefly buffers incoming perceptual data—visual, auditory, and haptic information—before processing, typically lasting only 200-500 milliseconds for iconic (visual) memory and 3-4 seconds for echoic (auditory) memory. Unattended information decays without reaching conscious awareness. In LLMs, the analog is input tokenization and embedding: the initial conversion of text, images, or audio into machine-processable representations. The tokenizer segments raw input into discrete tokens, which are then mapped to dense vector representations through embedding layers. Like human sensory memory, tokens outside the context window or attention span are effectively discarded without influencing model outputs.

Human working memory, governed by the "central executive" in Baddeley's model, maintains and manipulates information for immediate cognitive tasks. Its capacity is famously limited to approximately seven plus or minus two items according to Miller's Law, though modern estimates suggest four plus or minus one for unrelated items. The transformer's attention mechanism [12] functions as the AI analog, orchestrating which information receives processing priority. The attention computation dynamically weights the relevance of all context tokens to each query position, and the KV cache serves as an episodic buffer, storing key-value pairs from previous tokens to enable efficient autoregressive generation. Critically, working memory capacity constraints create similar bottlenecks in both humans and LLMs. While LLM context windows are orders of magnitude larger than human working memory—over 100,000 tokens versus four to seven items—both systems must employ strategies for managing information beyond immediate capacity: chunking, compression, and selective attention in humans; retrieval, summarization, and memory management in LLMs.

Human explicit (declarative) memory divides into episodic memory, which stores personal experiences and events, and semantic memory, which contains factual knowledge about the world. Both are consciously accessible and can be verbally described. AI episodic memory stores user-specific interactions, preferences, and conversational history. Systems like ChatGPT Memory [13], MemoryBank [14], and Mem0 [8] maintain records of past exchanges to enable personalization across sessions. The key difference from human episodic memory is persistence: human memories naturally decay following the Ebbinghaus forgetting curve, while AI systems require explicit decay mechanisms to achieve similar behavior. AI semantic memory encodes factual knowledge within model parameters through training. The billions of parameters in an LLM encode compressed representations of training data, enabling recall of facts, relationships, and patterns without explicit storage. Unlike human semantic memory, which is fallible and reconstructive, parametric knowledge is deterministic given fixed weights—though it cannot be easily updated without retraining.

Human implicit memory encompasses skills, habits, and conditioned responses that operate below conscious awareness, with procedural memory specifically handling "how to" knowledge such as riding a bicycle, typing, or playing an instrument. In AI systems, implicit and procedural memory manifests as learned task execution patterns. Voyager's skill library [15] stores refined procedures for Minecraft tasks, building a repertoire of reusable action sequences. ReAct's thought-action-observation loops [16] encode conditioned responses to environmental states, while Reflexion [17] accumulates successful reasoning patterns through self-reflection. These systems demonstrate that procedural knowledge can be separated from the base model and accumulated through experience.

2.2 The 3D-8Q Memory Taxonomy

Wu et al. [1] introduced the Three-Dimensional, Eight-Quadrant (3D-8Q) Memory Taxonomy, establishing the first comprehensive classification framework for LLM memory systems. This taxonomy provides essential conceptual clarity for understanding the diverse landscape of memory architectures and guides the design of new systems.

The taxonomy spans three orthogonal dimensions, each capturing a fundamental design choice. The **Object Dimension** distinguishes between personal memory, which serves user-facing applications by remembering preferences, maintaining conversation continuity, and enabling personalization, and system memory, which supports internal model capabilities through reasoning traces, skill libraries, and knowledge bases that enhance task performance without direct user visibility. The **Form Dimension** separates parametric memory, which encodes information within model weights either through training or dynamic weight updates, from non-parametric memory, which stores information in external structures such as databases, files, and graphs that the model queries at inference time. This distinction has profound implications for update mechanisms, scalability, and interpretability. The **Time Dimension** differentiates short-term memory, which maintains coherence within a single session or task through context windows or working memory buffers, from long-term memory, which persists across sessions to enable accumulated learning and cross-conversation continuity.

Table 1: Three Dimensions of AI Memory Classification

Dimension	Axis A	Axis B	Distinguishing Factor
Object	Personal Memory	System Memory	User-facing personalization vs. internal reasoning enhancement
Form	Parametric	Non-parametric	Encoded in model weights vs. stored in external databases
Time	Short-term	Long-term	Session-level coherence vs. cross-session persistence

The intersection of these three binary dimensions creates eight distinct quadrants, each representing a fundamentally different memory paradigm with distinct use cases and representative systems. Quadrant I encompasses personal, non-parametric, short-term memory—essentially working memory supporting multi-turn dialogue coherence. Every major LLM chatbot operates here by default, including ChatGPT, Claude, and Gemini, which load conversation history as context for each response. The technical implementation involves role-content formatted dialogue encoded and truncated when context limits are exceeded.

Quadrant II represents personal, non-parametric, long-term memory: episodic memory enabling personalization across sessions. This quadrant contains the most active research, including memory-RAG systems such as Mem0 [8], MemoryScope, and LangGraph Memory; commercial implementations including ChatGPT Memory [13] and Apple Intelligence Personal Context [18]; and specialized frameworks like MemoryBank [14] and A-MEM [19]. Memory construction in this quadrant involves four stages: construction (storage), management (reflection and reorganization), retrieval (semantic search), and usage (personalized generation).

Quadrant III covers personal, parametric, short-term memory: cached working memory for acceleration. Systems like Anthropic's Contextual Retrieval [20] and OpenAI's Prompt Cache [21] pre-store frequently requested personal data in parametric caches, reducing API costs and improving response latency for multi-turn dialogues. Quadrant IV addresses personal, parametric, long-term memory through personalized fine-tuning. Character-LLM enables embodying specific personas through fine-tuning on biographical data, while AI-Native Memory compresses and evolves personal memory within model parameters. The challenge here is that fine-tuning requires substantial computational resources, limiting scalability for per-user customization.

Quadrant V encompasses system, non-parametric, short-term memory: reasoning working memory storing intermediate outputs during complex problem-solving. ReAct's thought-action-observation loops [16], Chain-of-Thought prompting [22], and Reflexion's self-improvement cycles [17] all operate in this quadrant, maintaining working state for single tasks without persisting across sessions. Quadrant VI covers system, non-parametric, long-term memory: procedural memory capturing historical experience for skill accumulation. Buffer of Thoughts [23] refines reasoning chains into reusable templates, Voyager [15] builds skill libraries from environmental feedback, and ExpeL [24] learns from both successes and failures through comparative analysis.

Quadrant VII addresses system, parametric, short-term memory: KV cache management for computational efficiency. This represents the most mature optimization area, with vLLM's PagedAttention [6], H2O's heavy-hitter eviction [25], StreamingLLM's attention sinks [26], and FlashAttention's memory-efficient computation [7] all contributing to longer contexts within fixed memory budgets. Finally, Quadrant VIII represents system, parametric, long-term memory: foundational knowledge encoded in parameters. This frontier of learned memory includes the Memorizing Transformer [27] with kNN-augmented attention, WISE's dual parametric memory for lifelong editing [28], Titans' neural memory modules with test-time learning [3], and Meta's Memory Layers at Scale [9] adding 128 billion parameters for factual knowledge.

3. Architectural Paradigms

This section examines the major architectural approaches to LLM memory in depth, covering context window mechanisms, virtual context management, neural long-term memory, retrieval-augmented generation, and graph-based systems. Each paradigm offers distinct trade-offs between capacity, latency, accuracy, and implementation complexity.

3.1 Context Window as Associative Memory

The transformer attention mechanism [12] operates as an associative memory block where key-value associations are stored and retrieved through pairwise similarity computation. Understanding this mechanism is essential for appreciating both its power and limitations as a memory system.

During inference, the attention mechanism computes a weighted combination of values based on query-key similarities. For each query position i, the output is computed as:

$$y_i = \sum_j \frac{\exp(Q_i \cdot K_j/\sqrt{d})}{\sum_l \exp(Q_i \cdot K_l/\sqrt{d})} \cdot V_j \tag{1}$$

where Q_i is the query vector for position i, K_j and V_j are key and value vectors for position j, and d is the key dimension. This formulation can be interpreted as soft retrieval over a memory bank: queries select relevant keys through similarity, and corresponding values are retrieved and aggregated. The softmax normalization ensures that attention weights sum to one, creating a probability distribution over memory locations.

The KV cache stores intermediate Key and Value tensors across all layers and attention heads, avoiding redundant computation during autoregressive generation. For a model with L layers, h attention heads, sequence length t, and key/value dimension d_k, the cache requires memory proportional to $2 \times L \times h \times t \times d_k \times$ precision_bytes. For a 70-billion parameter model with 80 layers, 64 heads, 128,000 token context, and FP16 precision, this amounts to approximately 42 gigabytes—approaching or exceeding the model weights themselves for long contexts.

The quadratic scaling challenge emerges from self-attention's fundamental structure. Computing the attention matrix QK^T requires $O(n^2 \cdot d)$ operations for a sequence of length n, storing the matrix requires $O(n^2)$ memory, and the softmax normalization and value aggregation add further $O(n^2)$ operations. While KV caching transforms per-step generation complexity from $O(n^2)$ to $O(n)$ by reusing previously computed keys and values, the initial prefill phase remains quadratic. For a 128,000 token context, prefill computes approximately 16 billion attention pairs per layer—a substantial computational burden that motivates the memory-efficient algorithms discussed in Section 6.

3.2 Virtual Context Management: The MemGPT Paradigm

Packer et al. [2] introduced MemGPT by drawing direct inspiration from operating system virtual memory management. Just as OS virtual memory creates an illusion of unlimited RAM through intelligent paging between fast memory and slow storage, MemGPT creates an illusion of unlimited context through hierarchical memory tiers and intelligent paging. This analogy proves remarkably apt: both systems must balance access speed against capacity, implement policies for what to keep in fast storage, and provide seamless access abstractions that hide underlying complexity.

The MemGPT architecture organizes memory into two tiers with distinct characteristics. The main context functions as RAM: a fixed-size working context that fits within the LLM's context window. This tier contains the system prompt defining agent behavior, core memory blocks that are always present (including a persona block describing agent identity and capabilities, and a human block containing user information and preferences), recent conversation turns, and temporary task-specific working context. The main context is explicitly bounded by the model's context window—for example, 8,000 tokens for GPT-4—and when this limit approaches, the system must "page out" less critical information.

The external context functions as disk storage, comprising unlimited external storage accessed through explicit retrieval. Recall memory stores complete conversation history in a database with pagination support, allowing the agent to search and retrieve specific past exchanges. Archival memory provides unlimited long-term storage implemented via vector databases such as Chroma or pgvector, supporting semantic search for retrieving relevant historical information based on meaning rather than exact keyword matching.

The key innovation of MemGPT is enabling the LLM to actively manage its own context through function calls. Rather than relying on external orchestration, the agent decides when and how to modify its memory using tools like `core_memory_append` to add information to core memory blocks, `core_memory_replace` to update existing core memory, `archival_memory_insert` to store information for long-term retrieval, and `archival_memory_search` to retrieve relevant archived information. This self-editing capability enables emergent behaviors: the agent can proactively store important information, update its understanding of the user over time, and retrieve relevant context without explicit prompting from users or external systems.

MemGPT also introduces a heartbeat mechanism for proactive, multi-step behavior. When the agent's function call includes a request for heartbeat, the system immediately re-invokes the agent without waiting for user input. This enables autonomous reasoning across multiple steps, proactive memory management through background organization and consolidation, and continuous task execution spanning multiple function calls. The heartbeat creates an inner loop where the agent can perform multiple memory operations before generating a user-facing response.

Performance evaluations demonstrate that MemGPT's document question-answering performance remains stable regardless of document length, as relevant sections are retrieved rather than loaded entirely into context. Multi-session chat experiments show agents maintaining coherent personalities and user relationships across conversations. Perhaps most significantly, MemGPT operates 10 to 30 times cheaper and 6 to 13 times faster than iterative retrieval methods like IRCoT, which require multiple LLM calls for retrieval and reasoning. The Letta platform, which evolved from the original MemGPT research, extends this architecture with sleep-time compute—background processing during idle periods to consolidate memories, update summaries, and reorganize information for efficient future retrieval.

3.3 Neural Long-Term Memory: The Titans Architecture

Behrouz et al. [3] from Google Research introduced Titans, representing a paradigm shift where memory is not stored in external databases but learned within neural network parameters that update during inference. This approach treats the memory module's parameters as the memory itself, enabling test-time memorization that adapts to each input sequence.

The core insight of Titans is that memory formation should be driven by surprise: events that violate expectations should create stronger memories, mirroring findings from cognitive neuroscience about how humans form memories. The memory update mechanism implements this insight through a formulation where the memory state M_t at time t is updated according to:

$$M_t = (1 - \alpha_t) \cdot M_{t-1} + S_t \tag{2}$$

$$S_t = \eta_t \cdot S_{t-1} - \theta_t \cdot \nabla \ell(M_{t-1}; x_t) \tag{3}$$

Here, S_t represents the surprise metric combining past surprise (a momentum term controlled by η_t) and momentary surprise (the current gradient scaled by θ_t). The forgetting gate α_t controls how quickly old memories decay. The loss function $\ell(M; x) = \|M(k_t) - v_t\|_2^2$ measures how well the current memory predicts the association between keys and values in the input. When the current input strongly violates the memory's predictions—producing high loss and high gradient—the surprise signal is large, and the input is strongly encoded. Predictable inputs generate small gradients and weak encoding, naturally implementing a saliency-based memory formation policy.

Titans offers three architectural variants for integrating the neural memory module with attention. The Memory as Context (MAC) variant segments sequences into chunks and retrieves from long-term memory as a context prefix. For each window of tokens, the system retrieves relevant memory, concatenates it with the window, applies full causal attention within this augmented context, and then updates memory with information from the window. MAC provides explicit memory retrieval while maintaining full attention quality within windows.

The Memory as Gate (MAG) variant processes short-term and long-term memory in parallel. Sliding window attention computes a short-term representation y, while the neural memory module independently computes a long-term representation m. These are combined through learned gating: $o = y \otimes m$. The gating mechanism learns to balance local context from attention with global memory, enabling information flow across arbitrary distances without the quadratic cost of full attention.

The Memory as Layer (MAL) variant processes sequentially, with the memory layer compressing context before attention operates. This creates a bottleneck where memory must compress relevant information for downstream processing, encouraging the memory to learn efficient representations of long-range dependencies.

Beyond the learnable memory that updates at test time, Titans includes a persistent memory module consisting of learnable, input-independent parameters that store task-related meta-knowledge. Unlike the dynamic memory, persistent memory is fixed during inference (trained but not updated at test time) and stores task instructions, format knowledge, and common patterns. This component redistributes attention away from "attention sink" tokens—initial tokens that absorb disproportionate attention due to softmax normalization artifacts—and serves a role analogous to human procedural knowledge that remains stable across episodes.

Performance evaluations reveal remarkable capabilities. Titans processes contexts exceeding two million tokens with stable performance, dramatically exceeding the practical limits of standard transformers. On the BABILong benchmark [29], Titans models with only 170 to 760 million parameters outperform GPT-4 and Llama 3.1-70B, demonstrating that learned memory can be far more parameter-efficient than brute-force context extension. On needle-in-haystack retrieval tasks, Titans achieves 98 to 99 percent accuracy at 16,000 tokens compared to less than 30 percent for Mamba2. Experiments also show that memory depth matters critically: architectures with two or more memory layers significantly outperform single-layer variants, suggesting that hierarchical memory processing captures important structure.

3.4 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) [4] combines information retrieval with generative models, representing the most widely deployed approach to extending LLM memory beyond the context window. Rather than attempting to fit all relevant information into context or encode it in parameters, RAG systems maintain external knowledge bases that are queried at inference time to provide relevant context for generation.

The standard RAG pipeline follows four stages. During indexing, documents are processed for retrieval by splitting them into manageable chunks (typically 256 to 1,024 tokens), converting chunks to dense vectors via encoder models like BGE, E5, or OpenAI embeddings, indexing these embeddings in vector databases such as FAISS, Pinecone, or Weaviate, and storing metadata for citation and filtering. At retrieval time, given a query, the system finds relevant documents by embedding the query using the same encoder, computing similarity (typically cosine distance or dot product) against indexed embeddings, and returning the top-k most similar chunks. The augmentation stage incorporates retrieved context by formatting chunks with source attribution and inserting them into the prompt, typically before the query. Finally, generation produces a response using the augmented context, synthesizing information from retrieved documents, generating citations to sources, and handling any conflicting information across sources.

Retrieval methods fall into three categories with complementary strengths. Dense retrieval uses transformer encoders to map text to continuous vector spaces, capturing semantic similarity and handling synonyms effectively. However, it requires embedding storage, may retrieve "semantic lookalikes" whose surface similarity masks important differences (the embeddings of "I like fishing" and "I don't like fishing" remain dangerously close), and struggles with exact match requirements. Popular dense retrieval models include BGE-large, E5-mistral, and OpenAI's text-embedding-3.

Sparse retrieval using BM25 or TF-IDF relies on term frequency statistics rather than learned representations. This approach is fast, interpretable, and memory-efficient, with excellent performance for exact keyword matching. However, it lacks semantic understanding and suffers from vocabulary mismatch problems when queries use different terms than documents. Implementation typically uses inverted indexes through systems like Elasticsearch or Lucene.

Hybrid retrieval combines both approaches, typically through score fusion: $\text{score} = \alpha \times \text{dense\_score} + (1 - \alpha) \times \text{sparse\_score}$ with α usually between 0.5 and 0.7. An alternative is Reciprocal Rank Fusion, which combines rankings without requiring score normalization. Research from IBM demonstrates that three-way retrieval combining BM25, dense embeddings, and sparse learned vectors achieves 10 to 30 percent precision improvements over single-method approaches.

The choice of chunking strategy fundamentally affects retrieval quality. Fixed-size chunking simply splits by character or token count with overlap (typically 10 to 20 percent to prevent boundary information loss), offering speed but potentially breaking semantic boundaries. Recursive character chunking uses hierarchical splits with separators, trying paragraph breaks first, then sentences, then fixed-size splits, preserving document structure better. Semantic chunking groups content by embedding similarity, computing sentence embeddings, merging adjacent sentences with high similarity, and splitting when similarity drops below a threshold. This requires higher compute but produces better semantic coherence. The most sophisticated approach uses LLM-based chunking, prompting a language model to identify logical boundaries with document structure understanding—the highest accuracy but also highest cost.

Despite widespread adoption, RAG faces fundamental limitations. The retriever bottleneck means that if relevant information is not retrieved, the LLM cannot use it regardless of its capabilities. Standard RAG struggles with multi-hop reasoning when answers require synthesizing information across multiple documents. Retrieval adds 50 to 200 milliseconds of latency per query. Irrelevant retrieved content can pollute the context and degrade generation quality. And embedding drift means query and document embeddings may not align well, particularly for domain-specific content.

3.5 Graph-Based Memory Systems

Graph-based approaches address RAG's multi-hop reasoning limitations by explicitly modeling relationships between entities. Rather than treating documents as isolated chunks, these systems construct knowledge graphs that capture how entities relate to one another, enabling retrieval that follows relationship paths.

Gutiérrez et al. [5] introduced HippoRAG, drawing from hippocampal indexing theory in neuroscience. The hippocampus does not store memories directly but maintains an index linking cortical representations—HippoRAG applies this principle to LLM memory by separating the index (a knowledge graph) from the content (document passages).

The HippoRAG architecture comprises three components. An artificial neocortex, implemented as an LLM, handles language processing and knowledge extraction. A parahippocampal region, implemented as an embedding model, performs entity detection and synonymy linking. An artificial hippocampus, implemented as an open knowledge graph, stores entity-relation-entity triples that index into the document corpus.

During offline indexing, the system extracts named entities via NER, generates knowledge graph triples capturing relationships between entities, integrates these triples into a schema-less open knowledge graph, and creates entity embeddings for detecting when different surface forms refer to the same entity. During online retrieval, the system extracts entities from the query, links query entities to knowledge graph nodes (handling synonymy via embedding similarity), runs Personalized PageRank (PPR) from these seed nodes to identify related entities, and retrieves passages associated with high-scoring entities.

The critical innovation is that PPR propagates relevance through entity relationships. If the query mentions Entity A, and Entity A is connected to Entity B in the graph, both A and B (and their associated passages) receive relevance scores—even if the query never explicitly mentions B. This enables single retrieval steps to achieve multi-hop reasoning that would require multiple iterations with standard RAG. Experimental results show up to 20 percent improvement over state-of-the-art RAG on multi-hop question answering while being 10 to 30 times cheaper and 6 to 13 times faster than iterative approaches that require multiple LLM calls.

Mem0's graph extension [8] takes a different approach, representing user-specific memory as a directed labeled graph where nodes represent entities with types, embeddings, and metadata, while edges encode relationships as labeled triplets such as "works_at" or "prefers." The hybrid datastore combines vector databases for semantic similarity search with graph backends like Neo4j, Memgraph, or Neptune for relational traversal. Query processing first uses vector search to narrow candidates based on semantic similarity, then employs graph traversal to return related context following entity relationships, and finally merges and ranks results for context assembly. This approach captures both the fuzzy matching needed for natural language queries and the precise relational structure needed for reasoning about entity relationships.

4. Neural Memory as Alternative to Retrieval

A significant emerging trend is the use of neural memory modules—learned parameters that store and retrieve information without external databases. Rather than querying vector stores or knowledge graphs, these systems encode knowledge directly in neural network weights that can be accessed through forward passes. This section examines key systems and compares learned memory against retrieval-based approaches.

4.1 Memory Layers at Scale

Meta's Memory Layers at Scale [9] demonstrates that trainable key-value lookup mechanisms can add massive parameter counts for factual knowledge without proportional compute increases. The core insight is that factual recall—looking up who invented something, when an event occurred, or what property an entity has—differs fundamentally from reasoning and should be handled by specialized memory rather than general-purpose attention.

Memory layers are inserted between transformer blocks and operate through a lookup mechanism. The input is projected to a query vector, which then retrieves from a large key-value memory through similarity matching. Retrieved values are aggregated and projected back to the model's hidden dimension. The critical challenge is efficiency: naive implementation would require comparing against all N keys, resulting in $O(N)$ operations that negate the benefits of separating memory from compute.

Product-key quantization addresses this challenge by decomposing keys into products of smaller sub-keys. Instead of N full keys, the system maintains two sets of $\sqrt{N}$ sub-keys each. Retrieval first finds the top candidates in each sub-key set, then combines them to identify the best full keys, requiring only $O(2\sqrt{N})$ comparisons—sublinear in memory size. Custom CUDA kernels achieve remarkable throughput: 3 terabytes per second of memory bandwidth for memory access, 7.8 times faster than baseline PyTorch implementations, enabling 128 billion memory parameters without proportional FLOPs increase.

The performance improvements are substantial. A 1.3 billion parameter model with 64 million memory keys achieves 168 percent improvement on Natural Questions, jumping from 7.76 to 20.78 percent accuracy, and approaches Llama2-7B performance with 10 times fewer FLOPs. An 8 billion parameter model trained on just 1 trillion tokens matches models trained on 15 trillion tokens for factual recall tasks. The key finding is that factual knowledge can be "outsourced" to memory layers, allowing the core transformer to focus compute on reasoning and generation rather than fact storage.

4.2 MemoryLLM and Self-Updatable Models

MemoryLLM [30] takes a different approach, embedding a fixed-size memory pool within the transformer's latent space and enabling true self-update of model parameters during inference without backpropagation. The system combines a 7 billion parameter Llama2 base model (which can be frozen or fine-tuned) with approximately 1 billion parameters dedicated to memory, organized as 30 blocks of 256 tokens of memory vectors.

Memory operations occur through a controller that mediates between the base model's hidden states and the memory pool. Reading uses cross-attention from hidden states to memory vectors, retrieving relevant stored information to augment processing. Writing uses a gated update mechanism that modifies memory vectors based on input content, with gates controlling how much new information to incorporate versus how much existing memory to retain. Exponential decay of old memories prevents unbounded growth and naturally implements forgetting of stale information.

Unlike traditional models that require gradient-based training for updates, MemoryLLM updates memory through forward passes alone. New information enters as input tokens, the memory controller computes update signals based on input content and current memory state, and memory vectors are modified in place without backpropagation. This enables real-time knowledge injection as new facts become available, user-specific personalization without per-user fine-tuning, and continuous learning from interactions.

A critical finding from evaluation is that the system shows no degradation after approximately one million memory updates. The exponential decay mechanism ensures that old, unreinforced memories gracefully fade while frequently accessed memories remain strong, memory capacity stays bounded, and base model capabilities are preserved without catastrophic forgetting. The 2025 extension M+ [31] combines a co-trained retriever with latent memory, extending effective context to 160,000 tokens—eight times the base capacity—while maintaining the self-updatable property, demonstrating that retrieval and learned memory are complementary rather than competing approaches.

4.3 When to Choose Each Approach

The choice between retrieval-augmented generation and learned memory modules depends on workload characteristics, with emerging hybrid approaches combining both. Learned memory modules excel when the same facts are queried repeatedly (amortizing training cost over many queries), when long-context reasoning is required (where RAG struggles with multi-hop dependencies), when latency is critical (eliminating retrieval overhead), when reasoning must be deeply integrated with memory (allowing memory to influence attention patterns), and when context requirements exceed retriever limits (with learned memory demonstrated at 2 million+ tokens).

RAG remains preferable when information changes rapidly (requiring no retraining for updates), when the corpus is vast (exceeding what can fit in parameters, such as web-scale knowledge), when source citation is required (providing natural attribution), when training budget is constrained (requiring only embedding rather than full training), and when compliance or audit needs demand verifiable retrieved sources.

Table 2: Comparison of RAG and Learned Memory Approaches

Dimension	RAG	Learned Memory
Latency	Variable: retrieval adds 50–200ms	Consistent: single forward pass
Factual accuracy	Dependent on retriever quality	Up to 168% improvement demonstrated
Maximum context	Limited by retriever plus LLM window	Over 2 million tokens demonstrated
Update cost	Low: index new documents incrementally	High: requires training or adaptation
Multi-hop reasoning	Weak: retriever bottleneck	Strong: superior BABILong results
Interpretability	High: can cite sources directly	Lower: knowledge encoded in weights

The emerging consensus, articulated in work on memory operating systems, is that memory should be treated as a system-level resource with multiple modalities: parametric memory for learned facts, patterns, and skills; KV cache for working memory in current context; external retrieval for large-scale, dynamic knowledge bases; and graph structures for relational knowledge and entity tracking. Future systems will likely combine these approaches, using learned memory as an intermediate layer between fast parametric access and slower external retrieval.

5. Memory Operations and Lifecycle Management

Effective memory systems require sophisticated mechanisms for encoding new information, retrieving relevant knowledge, updating stored facts, and forgetting stale or irrelevant content. This section examines these operations in detail, drawing on both production systems and research advances.

5.1 Memory Encoding and Construction

Modern LLM memory systems extract memorable facts through LLM-driven processing [8]. The extraction pipeline processes conversation turns (user message paired with assistant response), assembles context from recent conversation summary and current exchange, prompts an LLM to identify salient facts, and produces structured outputs as subject-predicate-object triples or key-value pairs. Extraction criteria typically include user preferences (such as interface settings or dietary restrictions), personal facts (location, family, occupation), skills and expertise, goals and intentions, opinions and attitudes, and experiences and events.

Hierarchical summarization through Reflective Memory Management [32] creates multi-level memory structures that support both detailed retrieval and efficient overview access. Prospective reflection dynamically summarizes at multiple granularities: utterance level captures key points from individual messages, turn level summarizes dialogue acts, session level produces overall conversation summaries and outcomes, and cross-session level maintains an evolving user model and relationship context. Retrospective reflection reorganizes based on access patterns, promoting frequently accessed memories to faster tiers, linking related memories for efficient retrieval, and merging or pruning redundant memories.

Different storage formats offer distinct trade-offs for memory systems. Key-value storage provides $O(1)$ lookup complexity for simple fact retrieval but limited semantic search capability. Vector stores enable semantic similarity search with $O(\log n)$ to $O(n)$ complexity depending on the indexing algorithm but struggle with multi-hop reasoning. Knowledge graphs excel at relationship reasoning with complexity proportional to edge traversal but require more sophisticated construction. Hybrid approaches combining multiple formats provide the most flexibility but also the most complexity.

5.2 The Impossible Triangle and WISE

Wang et al. [28] identified a fundamental tension in lifelong model editing that they term the impossible triangle: three competing objectives that cannot be simultaneously achieved with naive approaches. Reliability requires that the model remember both current and previous edits after sequential editing. Locality requires that editing not influence irrelevant pretrained knowledge. Generalization requires that the model understand edits and generalize to different query forms.

Traditional approaches fail to satisfy all three objectives. Long-term memory through direct parameter editing achieves generalization (the model understands the edit deeply) but suffers poor reliability (catastrophic forgetting of previous edits) and poor locality (unrelated knowledge gets affected). Working memory through retrieval-based approaches achieves reliability (retrieved facts remain stable) and locality (base model is unchanged) but lacks generalization (only exact matches work, not paraphrases or inferences).

WISE bridges this gap through dual parametric memory that separates the main memory (pretrained parameters) from side memory (edited parameters). Knowledge sharding ensures different edit sets reside in distinct parameter subspaces through orthogonal subspace projection, preventing interference between edits. A trained router mechanism learns to classify queries and directs them to the appropriate memory. Knowledge merging periodically consolidates shards into shared memory through TIES merging that prevents parameter conflicts. Experiments across GPT, LLaMA, and Mistral architectures demonstrate that WISE maintains reliability, generalization, and locality simultaneously—navigating rather than accepting the impossible triangle.

5.3 Forgetting Mechanisms

Effective memory requires intelligent forgetting to manage capacity and reflect changing information. Zhang et al. [25] developed H2O (Heavy-Hitter Oracle) based on the observation that attention scores follow a power-law distribution: approximately 20 percent of tokens contribute most of the attention value across layers. The eviction algorithm tracks running importance scores based on attention received, periodically evicts lowest-importance tokens while keeping recent tokens regardless of importance, achieving up to 29 times throughput improvement versus DeepSpeed Zero-Inference while maintaining generation quality.

Xiao et al. [26] discovered the attention sink phenomenon: initial tokens receive disproportionate attention regardless of their semantic importance. This occurs because softmax normalization requires attention scores to sum to one, and when a query has no semantically meaningful keys to attend to, the attention mass must go somewhere—initial tokens absorb this "leftover" attention. StreamingLLM exploits this by always retaining four attention sink tokens plus a sliding window of recent tokens, enabling stable processing to over four million tokens with constant memory usage and 22 times speedup over sliding window recomputation without any fine-tuning. The limitation is that StreamingLLM does not expand true context understanding; information outside the window remains inaccessible.

MemoryBank [14] implements decay inspired by human forgetting curves, with retention computed as base strength times an exponential decay factor modulated by access count. Memories decay exponentially with time, but each access strengthens the memory and flattens the decay curve. Frequently accessed memories become effectively permanent "long-term" memories while neglected ones fade naturally, mimicking human memory consolidation where rehearsal strengthens memories.

6. KV Cache Optimization Techniques

KV cache management has become a critical optimization target as context lengths grow beyond what can be efficiently processed with naive implementations. This section examines techniques spanning memory allocation, computation algorithms, and architectural modifications.

6.1 PagedAttention and Memory Utilization

vLLM's PagedAttention [6] revolutionized KV cache management by applying operating system virtual memory concepts to attention computation. Traditional KV cache allocation pre-allocates maximum sequence length per request, cannot share memory between requests with common prefixes, and results in 60 to 80 percent memory waste due to fragmentation and over-provisioning.

PagedAttention addresses these issues by partitioning the KV cache into fixed-size blocks, typically 16 tokens corresponding to approximately 12.8 kilobytes for a 13 billion parameter model. A block table maps logical blocks (the sequence of tokens as the model sees them) to physical blocks (actual GPU memory locations), with blocks allocated on-demand as tokens are generated. Physical allocation is non-contiguous, analogous to how virtual memory pages need not be physically adjacent.

This design enables several key optimizations. On-demand allocation means new blocks are allocated only when needed, eliminating pre-allocation waste. Reference counting allows blocks to be freed when all sequences referencing them complete. Copy-on-write enables shared prefixes (common in beam search or multi-turn conversation) to use the same physical blocks until divergence. Preemption allows low-priority sequences to be swapped to CPU memory when GPU memory is exhausted.

The results are transformative for production serving. Memory utilization reaches over 96 percent compared to 40 percent with traditional allocation. Throughput improves by 2 to 24 times over HuggingFace Transformers depending on workload. Beam search memory overhead drops by 55 percent through copy-on-write sharing. The system can run 4 times larger batches in the same memory envelope. PagedAttention has become standard infrastructure, adopted in vLLM, TensorRT-LLM, HuggingFace TGI, SGLang, and LightLLM.

6.2 FlashAttention and Memory-Efficient Computation

FlashAttention [7, 33, 34] addresses the memory bottleneck in attention computation itself. Standard implementations compute QK^T to produce an $n \times n$ attention matrix in HBM, apply softmax (reading and writing the full matrix), and then multiply by V (reading the matrix again). This approach has $O(n^2)$ memory complexity, which becomes prohibitive for long sequences—a 128,000 token context at FP16 precision would require 32 gigabytes per layer just for the attention matrix.

FlashAttention tiles the computation to fit in SRAM rather than HBM. The algorithm partitions Q, K, and V into blocks that fit in the 128 to 256 kilobytes of shared memory per streaming multiprocessor. For each Q block, it iterates over K and V blocks, computing partial attention using an online softmax algorithm that maintains running statistics for numerically stable incremental computation. Results accumulate in registers, and only final outputs are written to HBM. The $n \times n$ attention matrix is never materialized; only block-sized intermediate results exist, stored in fast SRAM.

The evolution of FlashAttention tracks GPU architecture advances. FlashAttention-1 (2022) achieved 2 to 4 times speedup and memory-efficient computation, enabling 4 times longer contexts. FlashAttention-2 (2023) improved work partitioning across thread blocks, reduced non-matrix-multiply FLOPs, achieved 2 times speedup over version 1, and reached 50 to 73 percent GPU utilization. FlashAttention-3 (2024) targets Hopper architecture specifically, using WGMMA (warpgroup matrix-multiply-accumulate) instructions, hardware-accelerated TMA memory transfers, warp specialization for overlapping compute and memory access, and FP8 support reaching 1.2 petaFLOPS throughput with approximately 75 percent GPU utilization.

6.3 Quantization and Compression

Quantization reduces KV cache precision to decrease memory footprint while maintaining acceptable accuracy. KIVI [35] observed that keys and values require different quantization strategies: key cache has outliers concentrated in specific channels, requiring per-channel quantization, while value cache has no consistent outlier pattern, requiring per-token quantization. This asymmetric 2-bit quantization achieves 2.6 times peak memory reduction, enables 4 times larger batch sizes, and maintains less than 1 percent accuracy loss on benchmarks without requiring calibration data.

More aggressive approaches like KVQuant use pre-RoPE quantization (quantizing before rotary position embedding produces smoother distributions), non-uniform quantization with learned levels for outliers, and per-vector dense-and-sparse handling of outliers. These techniques enable million-token contexts on a single A100-80GB GPU for LLaMA-7B at 3-bit precision. NVIDIA's Blackwell generation introduces native FP4 support with hardware tensor cores, providing 50 percent memory reduction versus FP8 with minimal accuracy impact for KV cache.

SqueezeAttention [36] takes a different approach, observing that different layers have different importance for the KV cache. Some layers barely use the cache (information passes through unchanged), while others heavily depend on cached context. The algorithm measures layer importance via cosine similarity between hidden states before and after self-attention, categorizes layers into importance groups, and assigns different KV budgets per group. This achieves 30 to 70 percent memory reduction with up to 2.2 times throughput improvement, and the technique is orthogonal to sequence-wise compression—both can be combined.

6.4 Architectural Modifications

Fundamental attention architecture changes can reduce KV cache requirements by design. Multi-Query Attention (MQA) [37] uses a single key-value head shared across all query heads, achieving 10 to 100 times smaller KV cache and 12 times faster inference with some quality degradation. This approach is used in PaLM and Falcon-40B.

Grouped-Query Attention (GQA) [38] interpolates between standard multi-head attention and MQA by partitioning query heads into G groups, with each group sharing one key-value head. When G equals the number of heads, it reduces to standard attention; when G equals one, it becomes MQA. A key finding is that existing multi-head attention models can be uptrained to GQA with only 5 percent of original pre-training compute while achieving quality close to multi-head attention and speed close to MQA. GQA is now standard in Llama 2 and 3, Mistral, Gemma, and GPT-4.

Sliding Window Attention (SWA) restricts each token to attending only within a window of W previous tokens. Mistral 7B [39] implements this with a window of 4,096 tokens. Although each layer only sees local context, information propagates across layers: with 32 layers, the theoretical attention span reaches $4096 \times 32 = 131,072$ tokens. Combined with GQA (8 key-value heads versus 32 query heads), Mistral achieves 50 percent cache memory savings at 8,192 sequence length while matching Llama 2 13B performance with only 7 billion parameters.

State space models like Mamba [40] take an even more radical approach, replacing attention entirely with selective state space models that have $O(n)$ complexity. Input-dependent state parameters enable content-aware processing while hardware-aware parallel scan algorithms maintain training efficiency. Mamba achieves 5 times higher throughput than transformers, with Mamba-3B matching transformers twice its size. Adoption includes Codestral Mamba from Mistral, Jamba from AI21, and IBM's Granite 4.0 as a hybrid architecture.

7. GPU Memory Hierarchy

Understanding GPU memory architecture is essential for optimizing LLM inference. Modern accelerators present a multi-tiered memory system where each level offers distinct bandwidth-latency-capacity trade-offs that algorithms must navigate.

The fastest tier consists of registers, providing approximately 256 kilobytes per streaming multiprocessor on H100 GPUs with roughly 8 terabytes per second effective bandwidth and single-cycle latency. Registers hold active operands for tensor core operations and are scarce enough that kernel launch parameters must specify register budget per thread, affecting occupancy.

Shared memory and L1 cache provide 128 to 256 kilobytes per SM in a configurable split, operating at 15 to 20 terabytes per second aggregate bandwidth with approximately 30 cycle latency. This tier serves as the critical staging area for FlashAttention's tiling strategy, allowing attention computation without materializing the full attention matrix in slower memory.

The L2 cache is shared across all SMs, providing 50 megabytes on H100 and expanding to 126 megabytes on Blackwell B200. With approximately 3 terabytes per second bandwidth at 150 to 200 cycle latency, L2 is increasingly important for caching hot KV entries and frequently accessed model weights as models grow larger.

HBM (High Bandwidth Memory) represents the primary storage for model weights and KV cache. The H100 provides 80 gigabytes of HBM3 at 3.35 terabytes per second; the H200 increases this to 141 gigabytes of HBM3e at 4.8 terabytes per second (76 percent more capacity, 43 percent more bandwidth); and Blackwell B200 delivers 192 gigabytes at 8 terabytes per second. HBM uses stacked memory dies connected via silicon interposers, achieving high bandwidth through parallelism with thousands of data pins compared to hundreds for DDR.

Beyond GPU memory, CPU DRAM provides 256 to 512+ gigabytes at 32 to 900 gigabytes per second depending on interconnect (PCIe versus NVLink-C2C), serving as overflow for large contexts. NVMe SSDs provide terabytes of storage at 5 to 14 gigabytes per second for throughput-oriented batch processing.

LLM inference exhibits fundamentally different characteristics between its two phases. The prefill phase, which processes the input prompt, is compute-bound: large matrix multiplications have high arithmetic intensity of 100 to 1000 FLOPs per byte, saturating tensor cores while leaving memory bandwidth underutilized. The decode phase, which generates output tokens one at a time, is memory-bound: batch-1 operations achieve only approximately 2 FLOPs per byte, far below the H100's ridge point of approximately 298 FLOPs per byte needed to saturate compute. This dichotomy motivates disaggregated serving architectures that use separate GPU pools optimized for each phase.

CXL (Compute Express Link) is emerging as a new tier in the memory hierarchy, providing cache-coherent memory expansion over PCIe fabric with load/store semantics. CXL 3.0 delivers 128 gigabytes per second bidirectional bandwidth with 200 to 400 nanosecond latency—positioned between HBM and remote memory. Research systems like Beluga demonstrate CXL 2.0 switch-based architectures for LLM inference, achieving 89.6 percent time-to-first-token reduction versus RDMA solutions and 7.35 times throughput improvement in vLLM.

8. NVIDIA Open-Source Infrastructure

NVIDIA provides a comprehensive ecosystem of open-source tools for memory-efficient LLM serving, from training frameworks to inference engines to distributed coordination. Understanding this stack is essential for production deployment.

TensorRT-LLM [41] serves as the foundational inference engine, providing optimized kernels and sophisticated memory management released under the Apache 2.0 license. Memory is managed through three major contributors: weights (fixed based on model size, precision, and parallelism), activation tensors (pre-computed at engine build time with TensorRT's liveness-based optimization that reuses memory for non-overlapping tensors), and KV cache (by default allocated 90 percent of remaining GPU memory, configurable via parameters). Key features include paged KV cache with configurable block sizes from 8 to 128 tokens, in-flight batching for continuous request processing without padding, FP8/INT8/INT4 quantization support, and host memory offloading for overflow.

NVIDIA Dynamo [42] addresses datacenter-scale distributed inference, released under Apache 2.0 with over 5,500 GitHub stars. The KV Cache Manager (KVBM) implements cost-aware offloading across memory hierarchies from HBM through DRAM and SSD to network storage, with intelligent eviction policies balancing lookup latency against recomputation cost. The Smart Router tracks KV cache location across the cluster, calculates overlap between new requests and cached blocks, and routes requests to maximize cache hit rate, achieving 3 times improvement in time-to-first-token and 2 times reduction in average latency. NIXL (NVIDIA Inference Transfer Library) provides a unified API for high-throughput, low-latency communication across NVLink, InfiniBand, RoCE, and Ethernet, enabling efficient KV cache transfer between disaggregated prefill and decode GPUs. The GPU Resource Planner uses SLA-based planning to determine optimal prefill/decode configurations with dynamic scheduling based on real-time demand.

Megatron-LM provides memory parallelism strategies for training that scale to the largest models. Tensor parallelism splits tensors within layers across GPUs. Pipeline parallelism distributes layers across GPUs. Sequence parallelism shards LayerNorm and Dropout activations for up to 70 percent memory reduction. Context parallelism splits sequences for processing contexts beyond 32,000 tokens. NeMo builds on Megatron-Core with production-ready recipes for 16,000 to 1 million token sequence training, activation recomputation with selective checkpointing at approximately 2.7 percent FLOPs overhead, activation offloading to CPU during forward pass, and distributed optimizer with ZeRO-style state sharding. Models trained in NeMo export seamlessly to TensorRT-LLM for inference.

Transformer Engine provides mixed-precision training and inference support for Hopper, Ada, and Blackwell GPUs, with FP8/FP4 support, automatic precision selection per layer, and FlashAttention integration.

9. Production Systems and Benchmarks

This section examines deployed memory systems and evaluation frameworks, providing insight into what works at scale and where gaps remain.

9.1 Disaggregated Serving Architectures

The recognition that prefill (compute-bound) and decode (memory-bound) have fundamentally different resource requirements has driven a new generation of disaggregated serving architectures. DistServe [43], presented at OSDI 2024, separates LLM inference into distinct GPU pools with separate prefill instances generating KV cache from prompts and decode instances handling autoregressive token generation. This eliminates prefill-decode interference that causes 10 to 20 percent output latency degradation in colocated systems and enables independent parallelism strategies optimized for each phase. Results show 7.4 times more requests served within the same SLO and 12.6 times tighter SLO achievable versus vLLM baseline. By 2025, disaggregation has become the default approach for production LLM serving, integrated into vLLM, SGLang, NVIDIA Dynamo, and llm-d.

Mooncake [10] from Moonshot AI received the FAST 2025 Best Paper award for its KV-cache-centric disaggregated architecture powering the Kimi service. Operating at extraordinary scale—over 100 billion tokens processed daily across thousands of nodes—Mooncake features a disaggregated cache pool spanning CPU DRAM, SSDs, and remote RDMA storage. The Conductor scheduler routes requests based on KV cache distribution across the cluster, with hot-spot migration for frequently-accessed blocks and a custom transfer engine achieving 2.4 times faster RDMA transfers than alternatives. Production results show 75 to 115 percent more requests handled versus baseline, 525 percent throughput increase in long-context scenarios, and 2.36 times higher cache hit rate via global scheduling.

9.2 Memory-Augmented Agent Frameworks

Mem0 [8] provides a production-ready memory layer for AI agents through a two-phase pipeline. The extraction phase processes message pairs to identify salient facts using LLM-driven analysis with conversation context. The update phase compares each candidate fact to existing memories via vector similarity, with an LLM determining whether to add new memories, update existing ones, delete outdated information, or take no action. The graph extension adds entity extraction, relationship generation, conflict detection for overlapping or contradictory elements, and temporal reasoning in update resolution.

Benchmark results on LOCOMO [44] demonstrate 26 percent relative accuracy improvement over OpenAI's memory implementation, with the graph variant achieving an additional 2 percent improvement. The system delivers 91 percent lower p95 latency versus full-context baselines with over 90 percent token cost savings.

A-MEM [19] implements the Zettelkasten note-taking methodology for LLM memory, where each memory note contains content, timestamp, LLM-generated keywords, tags, context descriptions, dense embedding, and links to related memories. A link generation module uses embedding similarity to identify nearest neighbors while LLM analysis creates nuanced relationship understanding. A memory evolution module triggers updates to historical memories when new information arrives, refining contextual representations over time. Results demonstrate 85 to 93 percent token reduction compared to baselines with particularly strong multi-hop reasoning performance, doubling baseline metrics. A-MEM running Llama 3.2 1B on a single GPU outperforms MemGPT, SCM, and other baselines across six foundation models.

9.3 Evaluation Benchmarks

LOCOMO [44] from Snap Research evaluates long-context conversational memory with 10 to 50 conversations spanning up to 35 sessions, approximately 300 turns and 9,000 to 16,000 tokens each. The benchmark tests five reasoning types: single-hop, multi-hop, temporal, commonsense, and adversarial. Key findings reveal that even the best systems lag human levels by 56 percent overall, with the temporal reasoning gap reaching 73 percent.

BABILong [29], presented at NeurIPS 2024, extends the bAbI benchmark to 20 reasoning tasks scalable to 50 million tokens, with facts hidden within the PG19 book corpus as distractors. Critical findings show that LLMs utilize only 10 to 20 percent of context effectively, RAG achieves approximately 60 percent accuracy regardless of context length, performance degrades sharply with increased task complexity, and GPT-4 (128K context) shows degradation beyond 10 percent of its capacity.

10. Open Problems and Future Directions

Despite rapid progress, significant challenges remain in AI memory architecture. This section identifies key problems and emerging research directions.

Scalability to million-plus token contexts with acceptable latency remains partially solved. While Titans demonstrates 2 million+ token processing and StreamingLLM handles 4 million+ tokens for streaming scenarios, true understanding (not just processing) of such long contexts remains elusive. The accuracy-efficiency trade-off persists across compression techniques: quantization introduces noise, eviction loses potentially relevant information, summarization may miss nuances, and no lossless compression exists for semantic content.

Cross-session persistence raises fundamental questions about what to remember versus forget, how to handle contradictions that emerge over time, privacy implications of long-term storage, and verification of memory accuracy. Memory hallucination—where systems "remember" events that did not occur, retrieval errors cascade to generation, or adversarial attacks corrupt memory—presents both technical and safety challenges without current reliable solutions.

Wu et al. [1] identify six evolutionary directions for the field. The transition from unimodal to multimodal memory will require unified representations across text, images, audio, and video with cross-modal retrieval capabilities. The shift from static to streaming memory demands real-time memory formation during interaction without offline retraining. Moving from specific to comprehensive memory calls for unified architectures spanning all eight taxonomy quadrants with automatic memory type selection. The evolution from exclusive to shared memory will enable multi-agent collaboration with conflict resolution protocols and access control. Advancing from individual to collective privacy requires group-level frameworks for shared memory systems. And progressing from rule-based to automated evolution will produce self-improving memory systems with learned policies rather than hand-crafted rules.

Memory scaling laws remain poorly understood. Research [45] found that LLM fact knowledge capacity scales linearly with model size but decreases exponentially with training epochs, implying that memorizing all Wikidata facts would require 1000 billion parameters trained for 100 epochs—practically infeasible. Unlike compute scaling laws established by Kaplan and Hoffmann, no comprehensive memory-specific relationships exist connecting memory capacity, retrieval accuracy, and compute cost.

Evidence suggests learned memory and retrieval approaches are converging rather than competing. PagedAttention's block-based allocation could extend to paged neural memory modules with swappable memory pages. Compressive transformers and Titans' surprise-gated updates both point toward adaptive compression with learned salience. Hardware-aware attention like FlashAttention combined with associative retrieval suggests eventual hardware-aware learned retrieval. Google's MIRAS framework provides theoretical unification, revealing that any sequence model can be viewed as an associative memory module differing only in memory architecture, attentional bias, retention gate, and update algorithm.

11. Conclusion

Memory architecture for large language models has evolved from simple context windows to sophisticated multi-tier systems that increasingly mirror human cognitive organization. The 3D-8Q taxonomy [1] provides essential conceptual clarity, revealing that effective memory must span personal and system objects, parametric and non-parametric forms, and short-term and long-term persistence. Each of the eight quadrants represents distinct use cases with different optimal implementations, and the most capable systems will likely span multiple quadrants.

Three insights emerge as particularly significant for future development. First, the impossible triangle identified by Wang et al. is navigable rather than absolute. WISE [28] demonstrates that reliability, locality, and generalization can coexist through dual parametric memory with knowledge sharding, challenging assumptions that constrained earlier approaches to knowledge editing.

Second, forgetting mechanisms prove as important as remembering. H2O's heavy-hitter eviction [25], StreamingLLM's attention sinks [26], and Ebbinghaus-inspired decay all demonstrate that intelligent forgetting enables efficient scaling. The 20 percent of tokens that matter most can substitute for maintaining complete histories, and natural decay prevents unbounded memory growth while preserving important information.

Third, hybrid architectures consistently outperform pure approaches. Combining attention (accurate for short-term) with neural memory (efficient for long-term), dense retrieval (semantic) with sparse retrieval (exact) and graph traversal (relational), consistently outperforms single-paradigm systems. The emerging vision of memory as a system-level resource—schedulable, tiered, and differentiable—points toward architectures that blur the boundaries between what a model knows, what it remembers, and what it retrieves.

Production systems validate these research advances. Mem0 [8] demonstrates 26 percent accuracy improvement and 91 percent latency reduction. Mooncake [10] processes over 100 billion tokens daily with disaggregated architecture. Titans [3] proves that 2 million+ token contexts are achievable through test-time memorization with models 70 times smaller than competitors. The gap between research innovation and production deployment is closing rapidly.

The research trajectory points toward test-time memorization becoming standard by 2026, memory scaling laws formalized shortly after, multimodal memory integration maturing by 2027, and memory-native architectures fundamentally different from transformer extensions potentially emerging beyond 2028. The tools are open-source, the benchmarks are public, and the opportunity to build the next generation of memory-efficient LLM systems has never been more accessible.

Author's Note

This survey synthesizes research from the rapidly evolving field of AI memory architecture, drawing on work from Google Research, Meta AI, Anthropic, OpenAI, Apple, Microsoft, and leading academic institutions. The landscape continues to change quickly, with new architectures and optimizations emerging regularly. Readers are encouraged to consult the cited papers for implementation details and the latest developments.

About the Author

Anjan Goswami is General Manager at SmartInfer.com. He has over 20 years of AI leadership experience across major technology companies including Microsoft, Adobe, Salesforce, Walmart, Amazon A9, and eBay. He holds a PhD in Computer Science from UC Davis and has 10+ patents in ranking and search systems.

References

[1] Yaxiong Wu, Sheng Liang, Chen Zhang, Yichao Wang, Yongyue Zhang, Huifeng Guo, Ruiming Tang, and Yong Liu. From human memory to ai memory: A survey on memory mechanisms in the era of llms. *arXiv preprint arXiv:2504.15965*, 2025. Huawei Noah's Ark Lab.

[2] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[3] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024. Google Research.

[4] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[5] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. *arXiv preprint arXiv:2405.14831*, 2024.

[6] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[7] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[8] Prateek Chhikara et al. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[9] Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, et al. Memory layers at scale. *arXiv preprint arXiv:2412.09764*, 2024. Meta AI Research.

[10] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *arXiv preprint arXiv:2407.00079*, 2024.

[11] Richard C Atkinson and Richard M Shiffrin. Human memory: A proposed system and its control processes. *Psychology of Learning and Motivation*, 2:89–195, 1968.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[13] OpenAI. Memory and new controls for chatgpt. https://openai.com/blog/memory-and-new-controls-for-chatgpt, 2024.

[14] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38:19724–19731, 2024.

[15] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[16] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[17] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[18] Apple Inc. Introducing apple intelligence. https://www.apple.com/apple-intelligence/, 2024.

[19] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[20] Anthropic. Introducing contextual retrieval. https://www.anthropic.com/news/contextual-retrieval, 2024.

[21] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

[22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[23] Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. *arXiv preprint arXiv:2406.04271*, 2024.

[24] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. 38:19632–19642, 2024.

[25] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

[26] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[27] Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022.

[28] Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Wise: Rethinking the knowledge memory for lifelong model editing of large language models. *arXiv preprint arXiv:2405.14768*, 2024.

[29] Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *arXiv preprint arXiv:2406.10149*, 2024.

[30] Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, et al. Memoryllm: Towards self-updatable large language models. *arXiv preprint arXiv:2402.04624*, 2024.

[31] Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian McAuley, Dan Gutfreund, Rogerio Feris, and Zexue He. M+: Extending memoryllm with scalable long-term memory. *arXiv preprint arXiv:2502.00592*, 2025.

[32] Zhen Tan, Jun Yan, I Hsu, Rujun Han, Zifeng Wang, Long T Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, et al. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. *arXiv preprint arXiv:2503.08026*, 2025.

[33] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[34] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *arXiv preprint arXiv:2407.08608*, 2024.

[35] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *Proceedings of Machine Learning Research*, 2024.

[36] Zihao Wan et al. Squeezeattention: 2d management of kv-cache in llm inference via layer-wise optimal budget. *arXiv preprint arXiv:2404.04793*, 2024.

[37] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

[38] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

[39] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[40] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[41] NVIDIA Corporation. Tensorrt-llm: High-performance inference for large language models. https://github.com/NVIDIA/TensorRT-LLM, 2024.

[42] NVIDIA Corporation. Nvidia dynamo: A low-latency distributed inference framework. https://github.com/ai-dynamo/dynamo, 2024.

[43] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, 2024.

[44] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*, 2024.

[45] Xingyu Lu et al. Scaling laws for fact memorization of large language models. *Findings of EMNLP*, 2024.