Are We Ready For An Agent-Native Memory System?

Wei Zhou
Shanghai Jiao Tong University
weizhoudb@sjtu.edu.cn

Xuanhe Zhou*
Shanghai Jiao Tong University
zhouxuanhe@sjtu.edu.cn

Shaokun Han
Shanghai Jiao Tong University
areedd0@sjtu.edu.cn

Hongming Xu
Shanghai Jiao Tong University
muzhihai@sjtu.edu.cn

Guoliang Li
Tsinghua University
liguoliang@tsinghua.edu.cn

Zhiyu Li
MemTensor (Shanghai) Technology Co., Ltd
lizy@memtensor.cn

Feiyu Xiong
MemTensor (Shanghai) Technology Co., Ltd
xiongfy@memtensor.cn

Fan Wu
Shanghai Jiao Tong University
fwu@cs.sjtu.edu.cn

Abstract

Memory for large language model (LLM) agents has rapidly evolved from simple retrieval-augmented mechanisms into a data management system that supports persistent information storage, retrieval, update, consolidation, and dynamic lifecycle governance throughout agent execution. Despite this evolution, existing evaluations still benchmark agent memory mainly through end-to-end task success metrics (e.g., F1, BLEU), while treating the underlying system as a monolithic black box. As a result, critical system-level concerns, including operational costs, architectural trade-offs across memory modules, and robustness under dynamic knowledge updates, remain insufficiently explored.

In this paper, we present a systematic experimental study of agent memory from a data management perspective. We propose an analytical framework that decomposes agent memory into four core modules: memory representation and storage, extraction, retrieval and routing, and maintenance. Under this framework, we evaluate 12 representative memory systems and two reference baselines across five benchmark workloads spanning 11 datasets. Our extensive end-to-end evaluation shows that no single architecture dominates across all scenarios; instead, effectiveness depends heavily on how well the memory structure aligns with the workload bottleneck. Furthermore, through fine-grained ablation studies, we quantify their individual effects on representation fidelity, retrieval precision, update correctness, and long-horizon stability. Finally, we reveal cost-performance trade-offs under realistic workloads, showing localized maintenance is more cost-efficient than global reorganization. Based on these findings, we identify promising directions towards building truly agent-native memory systems. The code is publicly available at *https://github.com/OpenDataBox/MemoryData*.

1 Introduction

The rapid evolution of Large Language Model (LLM) agents has sparked a large body of exciting research and industrial efforts in building agent memory, i.e., the data management system of the LLM agent that supports long-horizon stateful execution and personalized interaction [9, 13, 17, 19, 23, 24, 28].

As shown in Figure 1, existing agent memory systems span a diverse set of architectural designs. (1) Stream-and-Reflection Memory System (e.g., MemoryBank [39]) maintains experiences as timestamped memory streams and periodically summarizes them into higher-level reflections that are written back into the stream;

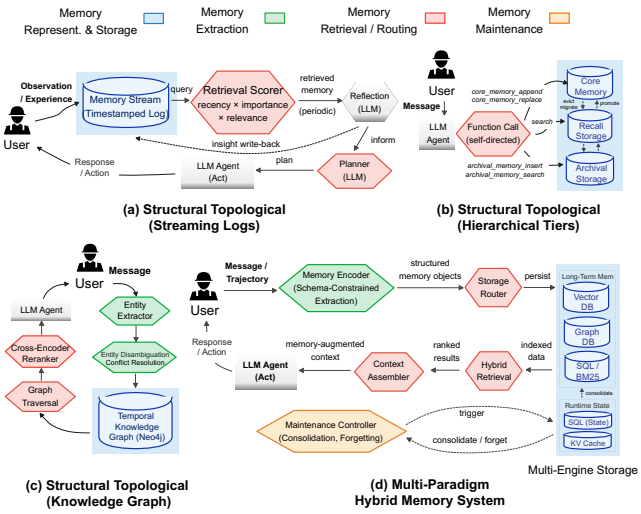

Figure 1: Typical Execution Workflows of Agent Memory.

(2) Hierarchical Tiered Memory System (e.g., MemGPT [25]) organizes memory into multiple levels with different capacities and access properties, separating core memory from archival storage with explicit movement (e.g., eviction and promotion) across tiers; (3) Knowledge Graph Memory System (e.g., Mem0^g [5], Zep [26]) represents entities, relations, and their temporal evolution in structured forms (e.g., temporal knowledge graphs), often incorporating entity disambiguation and conflict resolution; (4) Composite Hybrid Memory System (e.g., A-MEM [33]) routes schema-aware memory objects across multiple storage substrates, explicitly separating runtime state (e.g., KV caches) from long-term storage (e.g., vector, graph, keyword indexes), managed by dedicated maintenance modules. However, this rapid proliferation has also led to a highly fragmented landscape that lacks systematic evaluation from a data management perspective, raising a natural question: *Are we ready for an agent-native memory system?*

In this paper, we revisit this question for agent memory. In particular, we focus on *system-level memory* over textual, structured, and even parametric representations [5, 25, 26, 33], a fundamental infrastructure component for modern autonomous agents. We focus on memory-centric systems, rather than task-specific agent frameworks where memory is an auxiliary module [1, 40]. It is the persistent data management system that maintains information

*Xuanhe Zhou is the corresponding author.

beyond a single inference step (e.g., historical interactions, environmental observations, and intermediate tool executions) decoupled from the LLMs' parametric weights and volatile context windows. Agent frameworks rely on these external memory systems (e.g., Mem0 [5], Letta [25], Zep [26], and A-MEM [33]) to actively write, update, index, and route relevant context back into the reasoning loop. The capability of a long-horizon agent largely depends on the reliability and efficiency of this memory layer. An agent adopting a poorly designed memory architecture can suffer from factual contradictions, catastrophic forgetting, or unacceptable latencies during continuous execution [6, 38].

Recent benchmarks [20, 22, 29, 31] have evaluated agent memory and shown that external memory can improve agent performance on tasks requiring factual recall and long-context understanding. However, these evaluations are largely rooted in natural language processing and have multiple limitations when treating agent memory as a data management system (see Section 2). First, they fail to evaluate many representative memory architectures (e.g., systems such as MemoChat, MemTree, and LightMem have not been included in prior evaluations) under unified workloads, making principled cross-system comparisons difficult. Efforts from the database community [32] limit their scope to a few chatbot-centric datasets (e.g., LoCoMo and LongMemEval only), neglecting complex agentic execution scenarios. Second, existing benchmarks predominantly rely on single-sided, end-to-end task success metrics (e.g., F1 and BLEU scores) rather than a comprehensive evaluation suite. They fail to explicitly isolate and measure multi-dimensional performance indicators such as evidence-level retrieval fidelity, dynamic update robustness under conflicting knowledge, and long-horizon stability. Third, they rarely measure key operational costs from a systems perspective, such as index construction time and query latency, which are critical for production deployments. Last, they treat memory systems as monolithic black boxes rather than decomposing them into fundamental data management modules for isolated, fine-grained analysis.

We overcome these limitations and conduct comprehensive experiments and analyses from a data management perspective. The contributions are as follows.

(1) Technology Decomposition and Taxonomy (Section 3). We decompose existing agent memory systems into four core components: (i) memory representation and storage, (ii) memory extraction, (iii) memory retrieval and routing, and (iv) memory maintenance. For each component, we further establish a structured taxonomy[1] by categorizing existing approaches according to their underlying design principles, enabling principled comparisons.

(2) Overall End-to-End Performance Evaluation (Section 4). We conduct end-to-end evaluations under a unified and fair testbed (e.g., unified time-overhead traces)[2] across five distinct benchmark workloads encompassing 11 datasets. Our study includes 12 representative memory systems, each embodying different combinations of representation, storage, routing, and maintenance strategies. We evaluate their performance from five perspectives: task effectiveness (RQ1), retrieval fidelity (RQ2), dynamic update robustness (RQ3), long-horizon stability (RQ4), and operational cost (RQ5).

(3) Fine-Grained Technical Component Evaluation (Section 5). Leveraging our four-module framework, we conduct controlled and fine-grained experiments on representative strategies within each technique component. By systematically generating controlled variants that modify one module at a time, we quantify their performance trade-offs and assess their individual impacts on representation fidelity, routing precision, and update correctness.

(4) Insightful Findings. Based on our experimental results and in-depth analysis, we distill a set of insightful findings regarding the cost–performance trade-offs of agent memory systems:

❶ Are Memory Systems Effective Across Different Agent Request Workloads? No single memory architecture dominates all scenarios. Composite hybrid systems lead on conversational QA, while graph-based methods excel in single-hop factual recall but struggle with temporal reasoning. Moreover, effective memory systems remain robust across LLM backbone variants because they externalize evidence localization before answer generation.

❷ How Accurately Do Memory Systems Retrieve Stored Evidence? Explicit query planning and balanced hybrid search maximize contextual relevance. However, retrieval accuracy degrades significantly as the temporal distance between the evidence and the query increases, exposing limitations of similarity-based retrieval.

❸ Are Memory Systems Robust Under Dynamic Updates? Graph-based methods handle knowledge updates most reliably, whereas popular fact-extraction plugins and append-only stores struggle with targeted overwrites. Systems lacking lifecycle management return stale facts, leading to "hallucinations of the past".

❹ Do Memory Systems Remain Stable Over Long Horizons? Many append-only memory stores suffer from catastrophic degradation as evidence becomes more distant. For time-dependent queries, raw long-context retrieval still outperforms most memory-backed approaches, indicating that standard semantic consolidation often destroys crucial chronological cues.

❺ What Are the Operational Costs of Agent Memory? Highly structured systems incur orders-of-magnitude higher index construction time and query latency than lightweight stores, yet do not consistently deliver proportional accuracy gains.

❻ When Do Individual Memory Components Go Wrong? Each layer of abstraction (e.g., compression, summarization, and fact extraction) progressively discards information. Furthermore, fine-grained LLM-based extraction can yield modest precision gains but substantially degrade multi-hop reasoning. Finally, conservative memory consolidation serves as the best default maintenance strategy, whereas delayed flushing creates a deceptive trade-off between surface-level coverage and actual answerability.

2 Preliminaries

To support the discussion in the rest of this paper, we first clarify the scope of *agent memory* from a data management perspective. Although recent studies have examined memory from viewpoints such as cognitive taxonomy, agent architecture, and graph-based organization [6, 10, 30, 32, 34, 36], the underlying concept is still often treated primarily as an algorithmic component of the LLM or agent pipeline [6, 10, 36]. In contrast, we study agent memory as a standalone data management object and system infrastructure, with explicit attention to how it is represented, stored, retrieved,

[1] https://github.com/OpenDataBox/awesome-agent-memory
[2] https://github.com/OpenDataBox/MemoryData

updated, and maintained under real agent workloads. Under this view, we introduce a set of definitions below.

Memory Types. For an LLM agent, a wide variety of information is produced and may need to be memorized, including dialogue history, tool execution logs, distilled facts, and user preferences [36]. Following established cognitive frameworks, memory can be broadly organized along two axes [10, 30]. (1) Along the *temporal* axis, short-term memory holds the volatile state of an ongoing session, while long-term memory persists across sessions. (2) Along the *functional* axis, long-term memory is further divided into information such as concrete past events (episodic memory), abstracted factual knowledge (semantic memory) [10, 32], reusable action strategies (procedural memory), and user preferences.

Agent Memory. We define the *agent memory* $\mathcal{M}$ as the persistent data management object [25, 32] that maintains this cumulative state beyond a single inference step and makes it accessible to the agent during future reasoning and action [10, 36].

Agent Memory System. To operationalize agent memory $\mathcal{M}$, a robust infrastructure is required [10, 25]. As shown in Table 1, from a data systems perspective [14, 32], we formalize the *agent memory system* as a tuple of four modules: $\mathcal{M}_{sys} = \langle \mathcal{R}, \mathcal{S}, \mathcal{Q}, \mathcal{U} \rangle$, where each module governs a distinct phase of the memory lifecycle.

• (1) Memory Representation and Storage $\mathcal{R}$: A mapping that defines the logical and physical memory format with a data model of two facets: (a) logical representation, spanning simple primitives (discrete tokens, continuous vectors) to complex topologies (knowledge graphs, trees, and composites); and (b) physical storage, utilizing transient registers, specialized single-engine databases, or multi-engine backends for persistence and indexing.

• (2) Memory Extraction $\mathcal{S}$: A mechanism governing how heterogeneous input streams (e.g., multi-turn dialogues, tool logs) are transformed into logical memory primitives via pipelines such as raw sequence concatenation, schema-free semantic extraction, or schema-constrained structured extraction.

• (3) Memory Retrieval and Routing $\mathcal{Q}$: A function that dynamically identifies relevant memory subsets based on a query context, utilizing specific routing algorithms to traverse indices. Mechanisms span native attention-based retrieval, semantic K-nearest neighbor search, topological subgraph traversal, autonomous agentic routing via LLM planning, and multi-stage hybrid execution.

• (4) Memory Maintenance $\mathcal{U}$: Policies governing the dynamic lifecycle of memory entries, decomposed into three sub-operations: *(a) Conflict Resolution and Versioning* handles contradictions via multi-versioning, invalidation, or precedence rules; *(b) Capacity Management* enforces bounded growth through constraint-based hard eviction (e.g., FIFO, token limits) or score-based priority eviction (e.g., temporal decay); and *(c) Semantic Consolidation* utilizes the LLM to merge redundant assertions into dense summaries or execute CRUD operations via tool-calling interfaces.

Distinction from RAG and Context Engineering. Retrieval-Augmented Generation (RAG) [8, 13] typically operates as a stateless, read-only retrieval primitive: given a query, it fetches relevant passages from a static corpus to augment a single generation step. Context engineering [2] is the broader practice of curating the finite LLM context window at each inference turn (e.g., dynamically selecting prompts, tool descriptions, and retrieved facts) to mitigate context rot [2]. In contrast, an agent memory system (1) is a persistent and updatable infrastructure for managing agent-specific state over time and (2) governs the full long-term memory lifecycle, including memory representation, storage, retrieval, and maintenance, rather than merely packing the current context window.

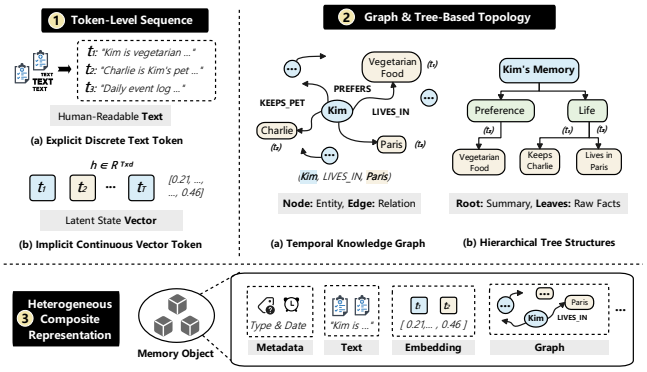

Figure 2: Memory Representation Methods.

Distinction from Traditional Database Workloads. Agent memory workloads differ substantially from conventional database OLTP / OLAP workloads [17, 25]. First, memory access is often *semantic* rather than purely predicate-based [4, 11]. Queries are commonly expressed through natural language, partial context, or latent intent, and therefore rely on approximate matching, query rewriting, or LLM-guided retrieval rather than only exact logical predicates over rigid schemas. Second, memory contents evolve under *continuous and potentially conflicting observations*. Unlike conventional transactional settings, where updates typically overwrite tuples under a predefined schema and consistency model, agent memory must accommodate uncertain, partial, and sometimes contradictory information collected across time, tools, and environments [10, 38]. Third, agent memory workloads are *highly heterogeneous* in both access pattern and granularity. A single workload may combine long-context synthesis, episodic recall, structured fact lookup, temporal reasoning, and streaming updates. As a result, practical systems often require hybrid execution strategies that combine semantic retrieval, structured filtering, and topology-aware traversal within one memory architecture [25, 32]. These properties distinguish agent memory from traditional databases, motivating dedicated abstractions and evaluation methodologies.

3 Method Overview

In this section, we carefully analyze existing agent memory systems across the four components in Section 2 and establish a unified taxonomy that summarizes representative component methods.

3.1 Memory Representation and Storage

This module consists of two components: (1) logical representation, which defines the structural encoding and organization exposed to the agent system, directly dictating capacity, accessibility, and trade-offs in expressiveness, retrieval granularity, and downstream reasoning compatibility; and (2) physical storage, which designates the persistence and indexing structures, such as volatile in-context registers, dense vector engines, or topological graph databases.

Table 1: Taxonomy and Characteristics of Agent Memory Systems.

Category	Method	Memory Representation & Storage: Representation	Memory Representation & Storage: Storage	Memory Extraction	Memory Retrieval & Query Routing	Memory Maintenance
Sequential Context	**MemoChat** [18]	❶ Token-Level Sequence (Structured JSON Memos)	❶ Transient In-Context Registers	❸ Schema-Constrained Extraction (LLM Topic Segmentation)	❹ Autonomous Agentic Routing (LLM Topic Selection)	❸ LLM-Driven Semantic Consolidation (Turn-Triggered)
	Mem0 [5]	❶ Token-Level Sequence (Discrete Facts)	❷ Specialized Single-Engine (Vector DB)	❷ Schema-Free Extraction	❷ Semantic-Based Retrieval	❸ LLM-Driven Semantic Consolidation (Tool-Calling)
	MEM1 [41]	❶ Token-Level Sequence	❶ Transient In-Context Registers	❶ Raw Sequence Concatenation	❶ Native Attention-Based Retrieval	❷ Capacity-Driven Physical Eviction
	MemAgent [35]	❶ Token-Level Sequence	❶ Transient In-Context Registers	❶ Raw Sequence Concatenation (Recursive Summaries)	❶ Native Attention-Based Retrieval	❷ Capacity-Driven Physical Eviction (RL Overwrite)
Structural Topological	**MemTree** [27]	❷ Graph & Tree-Based Topology (Hierarchical Tree)	❷ Specialized Single-Engine (Vector DB)	❷ Schema-Free Extraction (Top-Down Embedding)	❷ Semantic-Based Retrieval (Collapsed Tree)	❸ LLM-Driven Semantic Consolidation (Recursive Aggregation)
	Zep [26]	❷ Graph & Tree-Based Topology (Temporal KG)	❷ Specialized Single-Engine (Graph DB)	❸ Schema-Constrained Extraction (Triplets)	❺ Multi-Stage Hybrid Execution (Dense + BM25 + BFS)	❶ Timestamp-Based Multi-Versioning (Logical Invalidation)
	Mem0^g [5]	❷ Graph & Tree-Based Topology (Labeled Graph)	❸ Heterogeneous Multi-Engine (Vector + Graph DB)	❸ Schema-Constrained Extraction (Entity-Relation)	❸ Topological Subgraph Traversal	❶ Timestamp-Based Multi-Versioning
	Cognee [21]	❷ Graph & Tree-Based Topology (Entity-Relation Triplets)	❸ Heterogeneous Multi-Engine (Graph + Vector + Relational DB)	❸ Schema-Constrained Extraction (ECL Pipeline via Pydantic)	❸ Topological Subgraph Traversal (Dense-Seeded Triplet Extraction)	❶ Timestamp-Based Multi-Versioning (Hash-Based Deduplication)
Multi-Paradigm Hybrid	**LightMem** [7]	❸ Heterogeneous Composite (Tripartite Schema)	❷ Specialized Single-Engine (Relational DB)	❷ Schema-Free Extraction (Entropy-Gated)	❷ Semantic-Based Retrieval	❶ Timestamp-Based Multi-Versioning (Append-Only Logs)
	SimpleMem [16]	❸ Heterogeneous Composite	❸ Heterogeneous Multi-Engine (Vector DB + BM25 + SQL)	❸ Schema-Constrained Extraction	❹ Autonomous Agentic Routing (Query Expansion)	❸ LLM-Driven Semantic Consolidation (On-the-Fly Synthesis)
	MemOS [15]	❸ Heterogeneous Composite (MemCube)	❸ Heterogeneous Multi-Engine (Vector + Graph DB)	❸ Schema-Constrained Extraction (Semantic Parser)	❺ Multi-Stage Hybrid Execution (Boolean + Semantic)	❶ Timestamp-Based Multi-Versioning (Differential Writes)
	MemoryOS [12]	❸ Heterogeneous Composite (Segment-Page)	❸ Heterogeneous Multi-Engine (Keyword Index + Vector DB)	❸ Schema-Constrained Extraction	❺ Multi-Stage Hybrid Execution (Hierarchical Routing)	❷ Capacity-Driven Physical Eviction (Heat-Based Eviction)
	A-MEM [33]	❸ Heterogeneous Composite (Atomic Notes)	❸ Heterogeneous Multi-Engine (Vector + Graph DB)	❸ Schema-Constrained Extraction (JSON Attributes)	❸ Topological Subgraph Traversal	❸ LLM-Driven Semantic Consolidation (Mutation & Pruning)
	Letta [25]	❸ Heterogeneous Composite (Context Tiers)	❷ Specialized Single-Engine (Relational DB)	❸ Schema-Constrained Extraction	❹ Autonomous Agentic Routing (Function Calling)	❷ Capacity-Driven Physical Eviction (Queue Flush)

3.1.1 ***Logical Representation***. As displayed in Figure 2, this component acts as a bridge between raw data and the execution environment by organizing memory into clear models, such as graphs, or vector spaces. It determines how efficiently a system can search, combine, and use historical context for complex tasks.

❶ Token-Level Sequence Representation. This category models memory as flat, one-dimensional sequences lacking explicit structural abstractions (e.g., graphs or hierarchies). Memory is represented either as discrete, human-readable natural language tokens or as implicit, continuous latent vector tokens (e.g., fact embeddings, hidden states, or KV-cache tensors).

► *Explicit Discrete Text Token.* This category models memory as human-readable strings or independent factual statements. For instance, Mem0 isolates memory into discrete natural language facts extracted directly from interaction history. Similarly, MemoChat structures multi-turn dialogues into discrete JSON blocks (topics, summaries, raw turns) to maintain topical coherence within a plain-text paradigm. While these systems externalize their plain-text memory, others retain it within the active processing window: MemAgent restricts its internal belief state to a strictly bounded text sequence (e.g., 1024 tokens), and MEM1 encapsulates internal-state summaries within specialized boundary tags (e.g., <IS>).

► *Implicit Continuous Vector Token.* Departing from readable text tokens, this sub-category encodes memory as continuous vectors, which may be materialized either as external embeddings attached to facts and summaries for semantic retrieval, or as model-side latent states such as compressed internal states and attention caches. For example, Mem0 represents extracted facts as dense semantic embeddings and MemoRAG utilizes specifically initialized weight matrices to compress raw inputs into high-dimensional Key-Value (KV) cache tensors. Although these vector-token representations reduce explicit tokenization burdens and integrate naturally with retrieval or inference pipelines, they sacrifice structural interpretability and are difficult to manipulate via fine-grained operations, such as predicate-level filtering or targeted updates to encoded facts.

❷ Graph and Tree-Based Topological Representation. This category abstracts memory into structured graph and tree topologies with interconnected nodes and edges, allowing conversational entities, high-level concepts, and their temporal or semantic relationships to be explicitly modeled and computationally traversed.

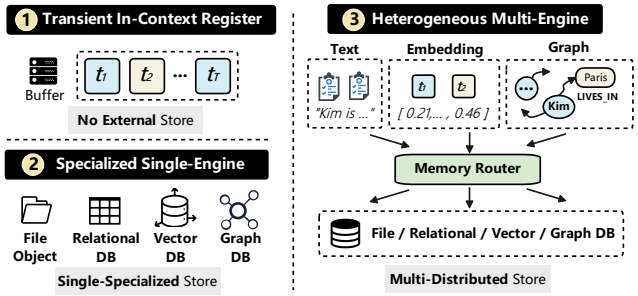

Figure 3: Memory Storage Methods.

► *Temporal Knowledge Graphs.*This sub-category models memory using graph topologies to map entities and their interconnections, natively supporting temporal reasoning and conflict detection. For example, Zep partitions memory into formally defined, temporally-aware knowledge graphs (e.g., episode, entity, and community subgraphs). Similarly, Mem0^g formalizes memory as a directed labeled graph, where vertices represent entities and edges encapsulate relationship triplets (e.g., "LIVES_IN"). To aid temporal reasoning, entity nodes are enriched with structural metadata like semantic types, dense embeddings, and creation timestamps.

► *Hierarchical Tree Structures.* This sub-category organizes knowledge into recursive, hierarchical structures, preserving highly granular observations at terminal leaves and broad semantic abstractions at ancestor nodes. For example, MemTree models memory as a dynamic, directed tree schema. Each node is structured as a tuple containing textual content, a dense embedding, topological pointers, and a depth scalar. Within this topology, deep leaf nodes retain isolated facts (e.g., a player scoring), while ancestor nodes provide high-level conceptual summaries (e.g., the match result), with a specialized root node serving as the definitive entry point.

❸ Heterogeneous Composite Representation. This category moves past simple token sequences and standard graphs by packaging memory into complex, multi-part data containers. These architectures directly combine unstructured text with highly structured metadata (e.g., timestamps, categorical labels, vector embeddings, and network links) to form a single functional unit. For example,

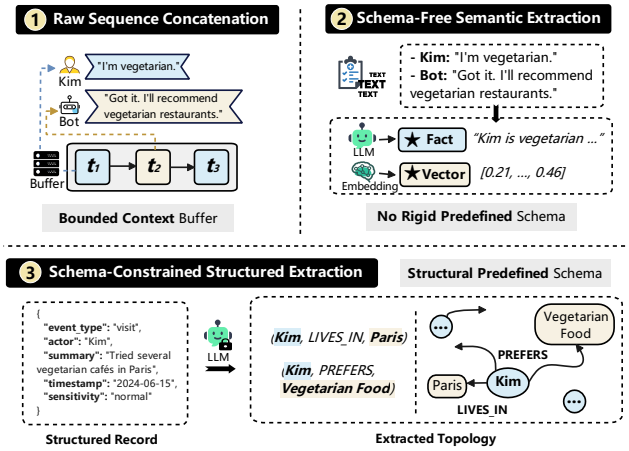

Figure 4: Memory Extraction Methods.

MemOS proposes the MemCube, a unified data object that organizes memory into three distinct payloads (plain-text, activation, and parametric memory) alongside structured details (e.g., ID tags).

3.1.2 ***Physical Storage and Indexing.*** As shown in Figure 3, this component manages how data is physically stored and accessed, relying on systems like in-memory caches, files, vector engines, or databases. It sets the actual capacity limits and determines the speed, throughput, and overall scalability of memory operations.

❶ **Transient In-Context Register.** To eliminate disk I/O and external traversal latency, this category retains memory exclusively within the active hardware state (e.g., dynamic context windows or KV caches). MemoChat avoids dedicated external memory engines and keeps structured JSON-style memos within the LLM context input during its memorization-retrieval-response loop, while MemAgent directly stores summary tokens as Key-Value (KV) cache tensors via dense positional embeddings.

❷ **Specialized Single-Engine Storage.** This category physically warehouses formulated units within a standalone, homogeneous backend strictly tailored to the memory's logical structure. Depending on the ingestion paradigm, architectures deploy specific backend topologies: (1) *Dense Vector Databases* are utilized to project data into continuous high-dimensional spaces; Mem0 and MemTree use centralized vector stores, Letta leverages PostgreSQL with the `pgvector` extension; (2) *Graph Databases* are deployed to enforce topological constraints; both Zep and Mem0^g execute predefined Cypher queries to physically persist logical graph components into Neo4j; (3) *Relational SQL Engines* are used to serialize structural and temporal schemas. LightMem incrementally appends factual streams to preserve global relational states; (4) *File or Object Stores* preserve raw interaction artifacts (e.g., conversation histories or tool-execution logs) as files or object blobs.

❸ **Heterogeneous Multi-Engine Storage.** This category dynamically constructs multiple index typologies or distributes data across heterogeneous backends (e.g., pairing a dense vector store with a topological graph database). SimpleMem ingests memory into LanceDB with an IVF-PQ mechanism that concurrently maintains dense embeddings, sparse BM25 indices, and SQL predicates. MemoryOS relies on a hybrid index fusing dense cosine similarity with discrete Jaccard similarity. Conversely, MemOS delegates serialized payloads to highly specialized independent backends, fusing Vector and Graph databases via a standardized memory adapter interface.

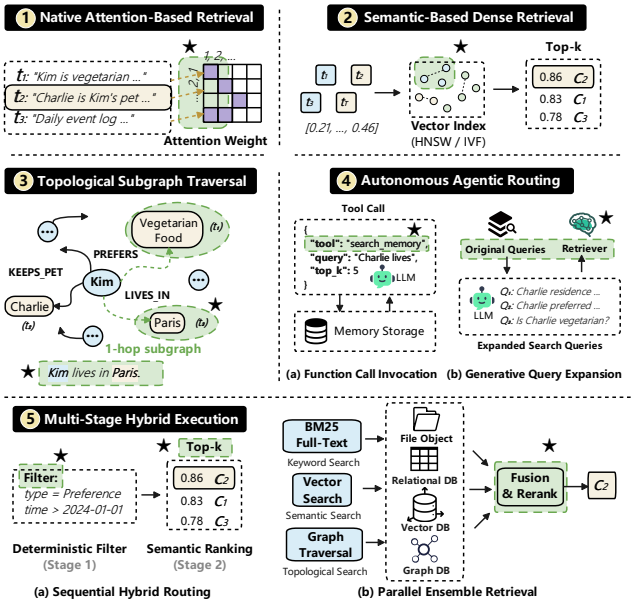

Figure 5: Memory Retrieval Methods.

3.2 Memory Extraction

Memory extraction concerns how raw interaction traces are computationally processed. It covers both the extraction pipeline, how language models extract, summarize, or parse unstructured text into logical structures. As shown in Figure 4, it defines how the agent memory system transforms heterogeneous input streams (e.g., multi-turn dialogues, and tool execution logs) into logical memory primitives prior to physical persistence.

❶ **Raw Sequence Concatenation.** To minimize computational overhead, this category bypasses explicit extraction prompts, formulating memory directly as raw token concatenations or transient state summaries (e.g., appending recent dialogue turns directly into a prompt buffer). Systems such as MEM1 and MemAgent retain their newly formulated structures exclusively within the active computational state without secondary parsing.

❷ **Schema-Free Semantic Extraction.** This category systematically distills raw, unstructured inputs into independent, high-value informational units, representing them either as explicit free-form texts or as compressed, continuous latent vectors. By isolating core knowledge from broader conversational context, it ensures precise and granular retrieval. For example, Mem0 actively parses interactions to extract and store discrete, standalone factual statements (e.g., "User is vegetarian and dairy-free").

❸ **Schema-Constrained Structured Extraction.** This category prompts the LLM to parse raw inputs and synchronously populate a rigidly predefined structural schema, producing strictly typed data rather than free-form text. The constrained output takes the form of either topological entity-relation triplets for graph insertion or multi-modal relational payloads for hybrid storage, depending on the target backend. Zep and Mem0^g extract typed directed relational edges (e.g., `LIVES_IN`, `WORKS_AT`) conforming to predefined graph

schemas, with Zep additionally applying a reflection-inspired verification step to suppress hallucinated triplets. MemoChat populates predefined structural fields to ensure data predictability by leveraging LLMs to segment conversations into strict JSON schemas.

3.3 Memory Retrieval and Query Routing

Memory retrieval and query routing determine how the agent memory system dynamically identifies and extracts relevant historical context to inform the overarching agent's current reasoning state. As shown in Figure 5, this module encompasses the complete query execution spectrum, defining the operational algorithms, predicate evaluations, and agentic workflows utilized to traverse indices.

❶ **Native Attention-Based Retrieval.** To bypass external database I/O, this category uses the transformer's native computational graph as the sole retrieval engine, relying entirely on self-attention mechanisms to implicitly weight and route information (e.g., scanning dialogue tokens directly within the KV cache). MEM1 performs implicit retrieval via self-attention over the current sequence, utilizing a two-dimensional attention mask to preserve causal consistency. MemAgent implements routing by concatenating blocks directly into the prompt template, enabling standard attention-based decoding without external cross-encoder reranking.

❷ **Semantic-Based Dense Retrieval.** Operating over continuous latent spaces, this category maps query tensors against uniform vector indices to extract localized spatial neighbors (e.g., executing a standard K-Nearest Neighbors (KNN) search). Mem0 calculates vector embeddings for incoming queries to execute a dense similarity search, fetching a constrained subset of facts. LightMem utilizes efficient cosine-similarity distance calculations over dense embeddings, bypassing computationally expensive iterative reranking. MemTree implements a collapsed-tree architecture that mathematically flattens its hierarchy, broadcasting inbound vectors to compute global cosine-similarity distributions across all candidates.

❸ **Topological Subgraph Traversal.** Departing from continuous vector spaces, this category retrieves information by traversing explicit relationship edges to extract semantic clusters structurally grounded in knowledge graphs (e.g., hopping from a `User` node to a linked `Preference` node). Mem0^g deploys an entity-centric heuristic to recursively traverse local subgraphs synchronously with semantic triplet evaluations. A-MEM identifies candidate anchors via dense K-Nearest Neighbor selection, then executes localized graph traversal to access topologically adjacent memory nodes explicitly linked within the same conceptual cluster.

❹ **Autonomous Agentic Routing.** Rather than executing deterministic database scans, this category delegates retrieval to the LLM itself, thereby functioning as an active, autonomous query planner. It generates tool-call invocations or drafts implicit search criteria.

► ***Function Call Invocation.*** This sub-category bridges the LLM with external storage by generating explicit function call commands to directly execute predefined database operations (e.g., outputting a valid JSON payload to trigger an external database API). For example, Letta orchestrates self-directed memory retrieval where the LLM evaluates its active context to explicitly generate localized function calls (e.g., emitting an `archival_storage.search()` command to extract targeted historical logs).

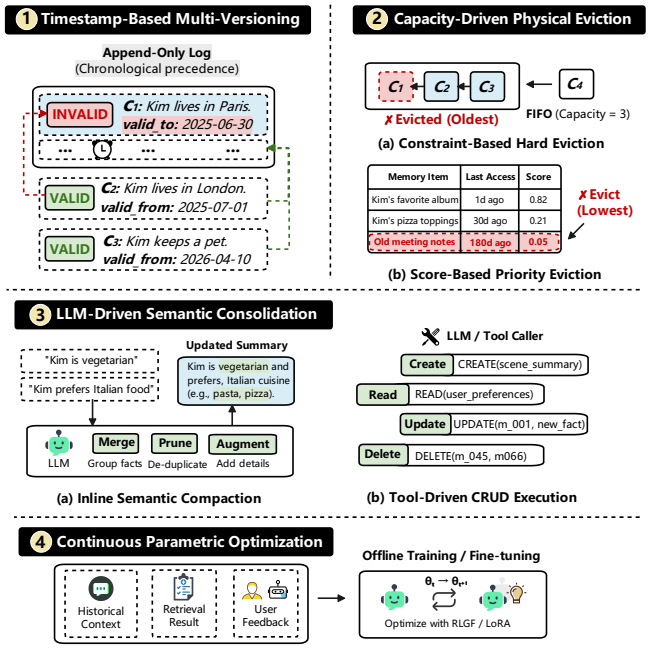

Figure 6: Memory Maintenance Methods.

► ***Generative Query Expansion.*** Unlike rigid function calling, this approach uses natural language generation to synthesize intermediate clues or decompose complex intents before mapping them to the index (e.g., rewriting vague prompts into descriptive search strings). SimpleMem uses an Intent-Aware Retrieval Planning module where the LLM dissects queries, calculates adaptive search depths, and synthesizes optimized query variants.

❺ **Multi-Stage Hybrid Execution.** To overcome the recall limitations of single-paradigm searches, this category executes multi-engine query pipelines orchestrating multi-dimensional candidate generation followed by downstream reranking frameworks.

► ***Sequential Hybrid Routing.*** This sub-category chains retrieval paradigms into a strictly ordered pipeline, systematically pruning the search space with deterministic predicates before executing fine-grained semantic extraction (e.g., applying strict SQL date filters before computationally expensive vector searches). MemoryOS executes a federated routing strategy featuring coarse-grained predicate evaluation followed by fine-grained semantic ranking strictly within isolated segments. It algebraically fuses rule-based structural Boolean filtering with dense semantic similarity routing.

► ***Parallel Ensemble Retrieval.*** In contrast to sequential filtering, this approach maximizes initial recall by simultaneously dispatching queries to multiple distinct indexing algorithms, followed by a late-stage fusion and reranking phase to optimize the aggregated pool (e.g., concurrently fetching candidates via BM25 and dense vector search, then cross-encoding the results). Zep executes simultaneous cosine semantic scans, Okapi BM25 full-text searches, and topological BFS, subsequently optimizing precision via RRF, MMR, and computationally intensive cross-encoder models.

3.4 Memory Maintenance

Memory maintenance concerns how memory is updated, maintained, compressed, forgotten, and eventually removed over time.

As shown in Figure 6, it captures the dynamic behavior of memory after it has been created, including how new information is incorporated, how outdated or conflicting content is revised, and how the system controls memory growth under limited resources.

❶ **Timestamp-Based Multi-Versioning.** Rather than executing physical row deletions, this category preserves historical continuity by utilizing timestamp metadata and append-only logs to logically deprecate expired facts. Operating via explicit metadata mutations, Zep and Mem0^g avoid physical deletion by marking obsolete or conflicting relationships as logically invalid using validity flags and timestamps. Taking an append-only approach, LightMem incrementally inserts timestamped factual streams, while SimpleMem resolves contradictions through strict chronological precedence using ISO-8601 timestamps. Synthesizing these techniques, MemOS leverages a structured Update API to execute differential writes, seamlessly updating provenance IDs to generate multi-version chains.

❷ **Capacity-Driven Physical Eviction.** In contrast to timestamp-based multi-versioning, this category manages unbounded memory growth by physically dropping or unconditionally overwriting data. It executes this physical pruning through either strict deterministic constraints or dynamically calculated eviction scores.

► ***Constraint-Based Hard Eviction.*** This sub-category enforces rigid execution bounds by utilizing deterministic rules—such as strict FIFO queues, fixed sequence boundaries, or hard token limits—to unconditionally evict older states. Executing structural overwrites, MemAgent implements a programmatic scheduling algorithm that unconditionally replaces older memory sequences with newly synthesized summary blocks at every fixed segment boundary. Enforcing hard capacity limits, MEM1 operates through a system-enforced truncation mechanism that executes an automated FIFO pruning protocol to evict older tags once active context thresholds are breached. Operating via threshold flushes, Letta strictly handles buffer capacities via an OS-inspired queue manager; when the token count breaches a terminal limit, it forces a flush sequence to evict older messages into secondary recall storage.

► ***Score-Based Priority Eviction.*** Rather than relying on static capacity limits, this sub-category dynamically forces the physical obsolescence of data by continuously calculating temporal decay or access-frequency scores. Quantifying access frequency, MemoryOS measures segment vitality via a scalar Heat score that balances retrieval frequency against exponential temporal decay, executing priority evictions that physically target the lowest-heat segments.

❸ **LLM-Driven Semantic Consolidation.** Operating as a cognitive governor, this category leverages the LLM to dynamically resolve logical conflicts and abstract redundant observations into dense summaries prior to query or persistence phases.

► ***Inline Semantic Compaction.*** During the active write phase, this sub-category dynamically evaluates and consolidates newly ingested data against existing memory nodes, systematically merging redundant assertions prior to database transaction commitment (e.g., compressing three similar dialogue turns into one dense summary node). SimpleMem executes online semantic synthesis on-the-fly, systematically merging structurally similar assertions into singular dense abstractions prior to database transaction commitment. MemTree utilizes a core scheduling operation that recursively triggers a semantic summarization prompt across all parent nodes to dynamically fuse historical states with novel payloads.

► ***Tool-Driven CRUD Execution.*** In contrast to automated fusion, this sub-category operationalizes maintenance through discrete, programmed state-mutations guided explicitly by LLM-driven tool interfaces that issue explicit Create, Read, Update, or Delete (CRUD) commands. Mem0 operationalizes its dynamic maintenance strictly through structured LLM tool-calling interfaces encompassing discrete programmed state-mutations such as UPDATE, and DELETE.

❹ **Continuous Parametric Optimization.** Completely decoupling state updates from online inference latency, this category executes heavy neural optimizations as asynchronous background processes, modifying the actual model parameters rather than the external database schema (e.g., running continuous fine-tuning on overnight batches). For example, MemoRAG leaves active inference tokens strictly static and read-only, optimizing extraction quality exclusively during an offline training phase via a Reinforcement Learning with Generation Feedback (RLGF) algorithmic framework.

4 End-to-End Assessment

In this section, we conduct a systematic evaluation of agent memory systems across five research questions. Across five distinct benchmark workloads and 11 datasets, we assess 12 representative memory systems against baselines to characterize their performance. Specifically, the five research questions are as follows.

4.1 Overall Effectiveness (RQ1)

Experimental Setting. For *"Do different agent memory systems successfully improve end-to-end task performance across workloads?"*, we evaluate 12 representative memory systems and two reference baselines (*Long Context* and *Embedding RAG*) on the three end-to-end workloads to assess whether memory improves task success beyond the underlying LLM. Specifically, we use: (1) *LoCoMo* [20]: a long-conversation QA benchmark that tests episodic, temporal, and open-domain memory over multi-turn interactions, and report the unweighted mean of category-level *Exact Match (EM)* and *Answer F1* on the four-category queries; (2) *LongMemEval* [31]: a multi-session long-memory benchmark that evaluates whether systems can reconnect facts across sessions and reason over temporally distributed evidence, and report *Substring EM*, *ROUGE-L F1*, *ROUGE-L Recall*, and *GPT-5.4-based LLM Judge Accuracy* from *MemoryAgentBench* [22]; and (3) *DB-Bench*: evaluates whether memory supports procedural execution across database operations from *LifelongAgentBench* [37], and report *Exact Match (EM)* and *Task Success Rate*.

O1-(Cross-Workload Effectiveness): No single memory system dominates all workloads, but methods that preserve task-critical evidence through structure-guided filtering remain the most competitive overall. As shown in Figure 7, the leading systems shift across workloads: (1) Structure-aware systems lead *LongMemEval*, where Zep reaches *48.0 LLM Judge Accuracy* and Cognee attains *35.3 ROUGE-L F1*; (2) Hybrid filtering is strongest on *LoCoMo* exactness, where MemOS reaches *11.5 Exact Match (EM)*; and (3) Trace-preserving memories remain strongest on *DB-Bench*, where Long Context achieves *48.20 EM* and MemoChat reaches *55.40 Task Success Rate*. However, among methods with full

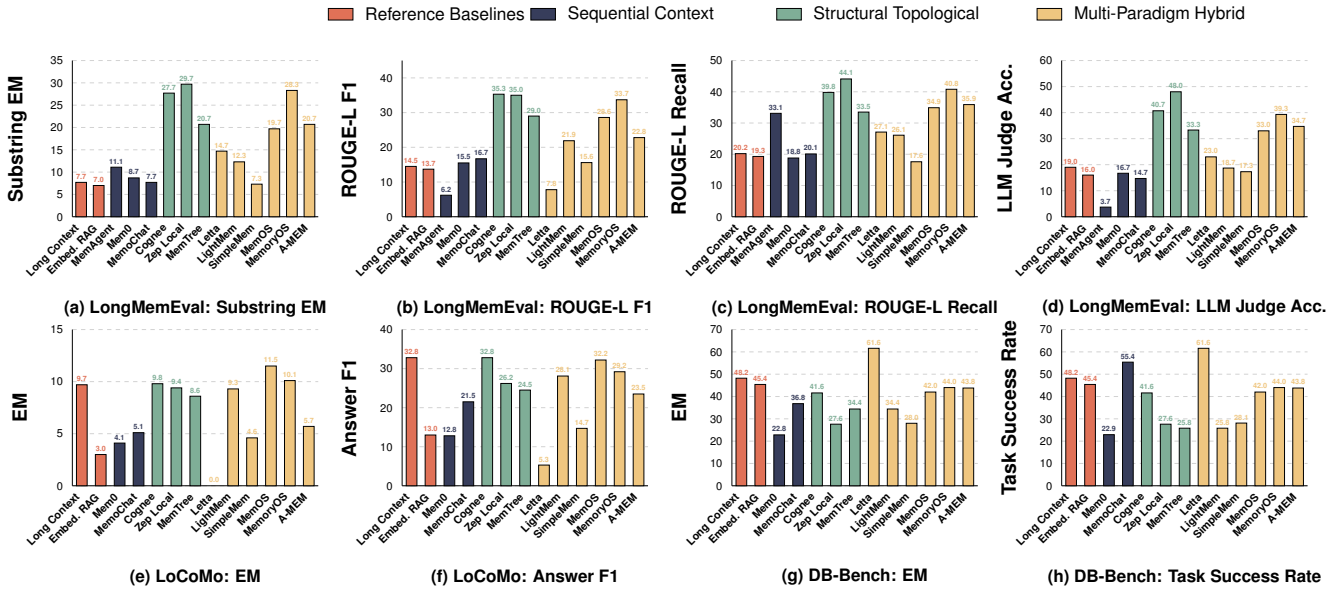

Figure 7: Effectiveness of Memory Systems over *LoCoMo*, *MemoryAgentBench (LongMemEval)*, *LifeLongAgentBench (DB-Bench)*.

workload coverage, MemoryOS and MemOS remain closest to the frontier overall, suggesting that robustness comes not from a single universal memory form, but from preserving the right evidence at the right level of abstraction before final matching. In particular, (1) Temporal or graph-organized memory is most useful for cross-session aggregation and event-order reasoning (e.g., scattered personal facts in *LongMemEval*); (2) Summary-first or coarse-to-fine routing is useful for exact grounding in long but semantically coherent dialogues (e.g., recovering a specific date or personal detail in *LoCoMo*); and (3) Trace-preserving memory is necessary when correctness depends on intermediate state changes and operation order (e.g., dependent `UPDATE` and `INSERT` operations in *DB-Bench*).

O2-(Beyond Exact Match): *EM* remains informative for tasks with canonical, directly grounded outputs, but it becomes insufficient when correctness depends on paraphrastic synthesis or executable success. As shown in Figure 7, *Exact Match (EM)* is still a meaningful signal on *LoCoMo*, where many questions target short grounded facts, as reflected by MemOS achieving the best *Exact Match (EM)*. On *LongMemEval*, however, the stronger systems are more clearly separated once semantic equivalence is considered through *ROUGE-L* and *LLM Judge Accuracy*, indicating that cross-session reasoning often yields correct answers that do not share a single canonical surface form. On *DB-Bench*, the limitation is even clearer: Long Context achieves the best *Exact Match (EM)*, but MemoChat attains a substantially higher *Task Success Rate*, showing that exact output matching does not fully capture whether memory supports successful execution. These results suggest that *Exact Match (EM)* is most appropriate when answers are short, canonical, and locally verifiable (e.g., a venue name, or object attribute in *LoCoMo*), but should be complemented once tasks require cross-session synthesis, or end-task state validation (e.g., composing a semantically correct answer from multiple sessions in *LongMemEval* or reaching the correct table state in *DB-Bench*).

Finding 1. ***(Workload-Aligned Memory).*** *RQ1 suggests that strong agent memory is not defined by a single universal representation, but by how well it supports the dominant workload bottleneck: (1) for dispersed cross-session reasoning, relation- and time-aware retrieval is most effective, as in Zep and Cognee; (2) for long but semantically coherent dialogue, coarse-to-fine filtering improves exact grounding, as in MemOS and MemoryOS; and (3) for stateful execution, preserving interaction traces is more critical than exact lexical matching alone, as in Long Context.*

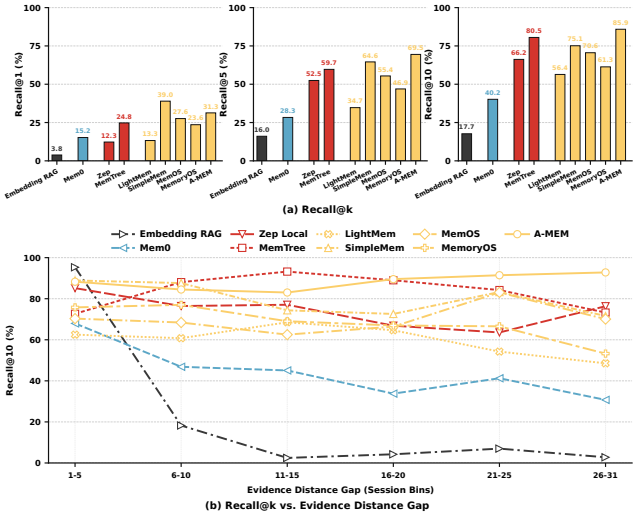

Figure 8: Retrieval Results of Memory Systems over LoCoMo.

4.2 Memory Retrieval Fidelity (RQ2)

Experimental Setting. For *"How accurately can a memory system surface the stored evidence required by a query?"*, we evaluate eight representative memory systems to assess evidence-level retrieval fidelity independently of downstream answer generation. Specifically, we use *LoCoMo* [20], which provides source-level gold evidence for queries with diverse evidence distances. We report: (1) *Recall@K*, where a hit requires the top-k retrieved source-id groups to contain the annotated gold evidence, and (2) *Recall@10* over six *evidence distance gap* bins (1–5 to 26–31), defined by the

Table 2: Robustness over Memory Update Settings.

Method	LoCoMo		LongMemEval			
	Temporal		Knowledge Update		Temporal Reasoning	
	Exact Match	Answer F1	Substring EM	ROUGE-L F1	Substring EM	ROUGE-L F1
Long Context	8.1	26.9	20.0	18.0	12.0	24.0
Embedding RAG	1.6	7.9	20.0	17.8	10.7	22.7
Mem0	3.2	6.0	15.6	17.1	10.7	22.4
MemoChat	2.4	15.4	8.9	12.9	10.7	25.3
Cognee	4.0	**28.1**	37.8	34.0	**18.7**	**35.8**
Zep	4.8	18.1	**44.4**	**36.8**	13.3	30.5
MemTree	5.6	18.6	31.1	30.6	8.0	29.9
Letta (MemGPT)	0.0	7.1	17.8	5.7	12.0	8.8
LightMem	4.0	20.1	15.6	20.2	12.0	28.6
SimpleMem	4.4	8.1	6.7	7.4	8.0	22.6
MemOS	**8.9**	28.0	28.9	30.5	12.0	31.1
MemoryOS	3.2	22.7	35.6	32.2	16.0	31.6
A-MEM	4.8	17.7	26.7	22.8	8.0	22.5

session distance between the query's final session and the earliest supporting evidence, to measure long-range retrieval accuracy.

O1-(Structured Evidence Expansion): Retrieval fidelity depends less on surfacing one relevant memory early than on preserving an explicitly organized memory structure that can gather complete and temporally distant evidence. As shown in Figure 8, the results show a clear difference between early-hit precision and overall evidence completeness: *SimpleMem* achieves the highest *Recall@1* (39.0), but *A-MEM* and *MemTree* become clearly stronger at larger retrieval budgets, reaching 69.5/85.9 and 59.7/80.5 on *Recall@5*/*@10*, respectively, while also remaining much more stable as the *evidence distance gap* increases; by contrast, the flat *Embedding RAG* baseline drops sharply after the shortest-gap bin. This pattern suggests that strong memory retrieval is not mainly a top-1 ranking problem, but an evidence-completion problem in which the required support may be old, scattered, or spread across multiple turns (e.g., personal details mentioned in different sessions or dated events referenced much later). More specifically, the results point to three different retrieval behaviors: (1) compression-oriented memory is effective for surfacing one highly relevant item early (e.g., a single salient personal detail or recent conversational fact); (2) linked or hierarchical memory organization is more effective for gathering complementary supporting evidence across the ranked results (e.g., combining pet names or dated events mentioned in different sessions); and (3) flat dense retrieval remains competitive mainly when the needed evidence is still close to the current context (e.g., recent conversational facts). For queries that require scattered or temporally distant support, the most competitive systems are therefore those that organize memory as a structured evidence space rather than a flat similarity cache.

Finding 2. ***(Evidence-Centric Memory Organization).*** *RQ2 shows that retrieval quality depends more on how a system organizes evidence for later reconstruction than on how well it ranks one relevant memory first. Specifically, (1) early localization and evidence assembly should be treated as separate design targets; (2) explicit structure, such as links or hierarchy, is most valuable when supporting evidence is scattered or temporally distant, as in A-MEM and MemTree; and (3) flat similarity search is mainly effective for short-range access.*

4.3 Memory Evolution Robustness (RQ3)

Experimental Setting. For *"Can agent memory systems reliably incorporate revised facts, preserve the correct temporal state after updates, and remain robust across answer backbones?"*, we conduct two

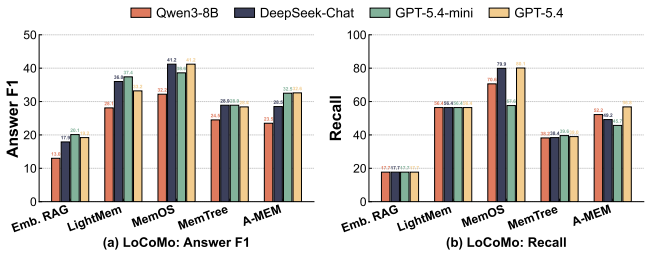

Figure 9: Ablation of LLM Backbones.

experiments: *(1) Update Robustness Comparison*, which evaluates whether systems can absorb fact revisions and answer temporally grounded queries after updates; and *(2) Backbone Robustness Ablation*, which tests whether this behavior remains stable when only the LLM backbone changes. In *(1) Update Robustness Comparison*, Table 2 compares 11 representative memory systems on *Knowledge Update* and *Temporal Reasoning* from *LongMemEval* [31], and *Temporal* from *LoCoMo* [20]. The two *LongMemEval* [31] slices are reported with *Substring EM* and *ROUGE-L F1*, while the *LoCoMo* [20] slice is reported with *Exact Match (EM)* and *Answer F1*; In *(2) Backbone Robustness Ablation*, Figure 9 evaluates 6 representative memory settings under 4 LLM backbones on the *LoCoMo* [20].

O4-(Temporal State Externalization): No single memory system dominates all update-oriented slices, but methods that preserve temporally valid evidence through structured organization remain the most competitive overall. As shown in Table 2, the leading systems shift across slices: (1) Graph- or relation-organized memory is strongest on direct fact revision, where Zep leads *Knowledge Update* with 44.4 *Substring EM* and 36.8 *ROUGE-L F1*; (2) Relationally organized retrieval is strongest on temporally dispersed evidence, where Cognee leads *Temporal Reasoning* with 18.7 *Substring EM* and 35.8 *ROUGE-L F1*; and (3) Hybrid filtered memory is strongest on exact latest-state grounding, where MemOS attains the highest *LoCoMo Exact Match (EM)* at 8.9 while Cognee attains the highest *Answer F1* at 28.1. However, among methods with full slice coverage, Cognee, MemOS, and MemoryOS remain closest to the frontier overall, indicating that robustness comes not from a single universal memory form, but from preserving the right temporal evidence at the right structure level. In particular, (1) temporal or graph-organized memory is most useful for revised personal facts and dated events (e.g., aggregating scattered updates to preferences, purchases, or past activities in *LongMemEval*); (2) hybrid or coarse-to-fine filtering is most useful when correctness depends on the currently valid state (e.g., recovering the latest date, attribute, or event order from long but semantically coherent dialogues in *LoCoMo*); and (3) flat context accumulation or dense similarity alone is weakest when stale mentions must be separated from updated ones (e.g., distinguishing an earlier personal detail from its later correction after repeated mentions over time).

O5-(Backbone Robustness): Backbone variation changes absolute answer quality more than it changes which memory pipeline remains effective, indicating that stable update behavior is determined primarily before final generation. As shown in Figure 9, *Answer F1* generally rises under stronger generators, yet the overall ordering changes only modestly: (1) MemOS remains the strongest memory-based configuration at 32.2, 41.2, 38.6, and 41.2; and (2) the only notable reversal is local, where A-MEM overtakes MemTree under GPT-5.4-mini and GPT-5.4. This stability implies that stronger backbones mostly improve answer realization after relevant evidence has already been localized, rather

than compensating for weak temporal grounding. For example, for date-grounded latest-state queries in the current *LoCoMo* temporal outputs, MemOS remains correct across all four backbones, whereas Embedding RAG remains incorrect. More concretely, methods with stronger external organization yield a more stable evidence set for different LLMs, whereas methods that rely more heavily on LLM-side synthesis exhibit greater cross-backbone movement.

Finding 3. ***(Temporal Update Fidelity).*** *RQ3 suggests that reliable post-update behavior is a pipeline-level design problem rather than a pure model-capacity problem. In particular, (1)* revisability *should be built into the memory representation so later facts can be bound to the same entity or event rather than appended as undifferentiated text, as in Zep and Cognee; (2)* query-time selectivity *should match the workload bottleneck, using filtered or hybrid routing when the task requires the currently valid state, as in MemOS and MemoryOS; and (3)* LLM scaling *is most valuable only after grounding has succeeded, so stronger backbones should refine answer expression rather than serve as the primary mechanism for resolving stale or conflicting memories.*

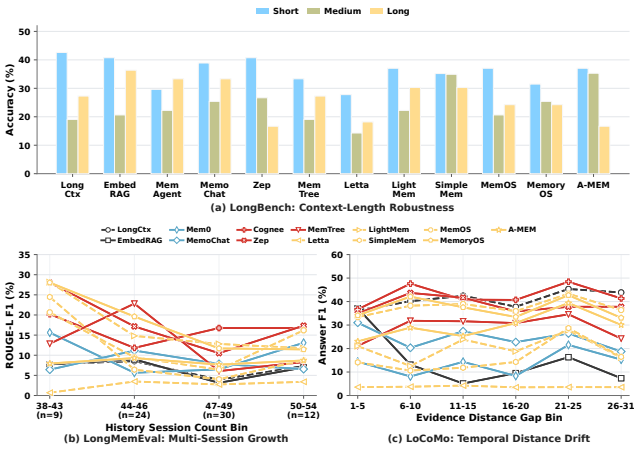

Figure 10: (a) Context-length robustness on *LongBench*; (b) Session-history growth on *LongMemEval*; (c) Temporal evidence-distance drift on *LoCoMo*.

4.4 Long Horizon Memory Stability (RQ4)

Experimental Setting. For *"How stable are agent memory systems as the effective memory horizon increases, either through longer contexts or more distant supporting evidence?"*, we evaluate 12 representative memory systems across 3 benchmarks to assess robustness to increasing context length and temporal distance. Specifically, we use: (1) *LongBench* [3], which evaluates controlled long-context difficulty in question answering, reported with *Accuracy* over *Short*, *Medium*, and *Long* context-length buckets to measure context-length robustness; (2) *LongMemEval* [31], which evaluates multi-session memory as the amount of prior interaction grows, reported with *ROUGE-L F1* over bins of historical session count to measure multi-session stability; and (3) *LoCoMo* [20], which evaluates memory drift when supporting evidence lies back in the conversation, reported with *Answer F1* over bins of evidence-distance gap between the final session and the earliest supporting-evidence.

O6-(Long-Horizon Evidence Preservation): Memory remains more stable at longer horizons when evidence is organized through explicit relational links or hierarchical consolidation, rather than left as flat text for direct matching. As shown in Figure 10, in *LongBench*, *SimpleMem* stays nearly unchanged from the Short to Medium buckets (35.2 to 34.9 *Accuracy*), whereas *Long Context* drops from 42.6 to 19.0, indicating that larger prompts alone do not sustain answer quality once long inputs accumulate distractors. In *LoCoMo*, the contrast is sharper: *Embedding RAG* falls from 37.1 to 7.4 *Answer F1* as the evidence gap widens, while graph- or consolidated-memory systems such as *Cognee*, *MemOS*, and *MemoryOS* remain substantially higher across the same bins; *LongMemEval* shows the same advantage for methods that preserve cross-session structure over longer histories. It indicates that the main difficulty at longer horizons is not memory volume, but whether the representation keeps distant facts connected to the abstractions needed for answering. More specifically, graph- or temporally organized memory preserves entity–event–time relations for distant facts (e.g., recovering a repeated personal event many sessions earlier), while hierarchical or summary-first organization preserves session-level structure (e.g., first locating the relevant session before resolving a specific local detail) so the LLM can narrow attention before final generation. Pure long-context prompting and flat dense memory provide neither form of support, and therefore degrade more sharply as the effective horizon grows.

Finding 4. ***(Horizon-Structured Memory).*** *RQ4 indicates that, as the effective memory horizon grows, the main challenge shifts from storing more history to choosing the right abstraction over it:* (1) Multi-view filtering *helps when long inputs contain many distractors, as in SimpleMem;* (2) Relation-aware indexing *helps when supporting facts are separated by many turns or sessions, as in Cognee and Zep; and* (3) Coarse-to-fine summarization *helps when the system must first identify the relevant session before resolving a local detail, as in MemOS and MemoryOS.*

4.5 Memory Operation Cost (RQ5)

Experimental Setting. For *"What is the operational cost of each memory system in terms of utility–latency trade-off and cross-workload latency footprint?"*, we evaluate 8 representative memory systems using the unified time-overhead traces recorded by our runner. We quantify two aspects: (1) *Utility–latency trade-off*, measured by *Avg. Operation Latency/Query* and *Normalized Utility*; and (2) *Cross-workload latency footprint*, measured by *Outlier-Filtered Avg. Total Latency/Query*. For (1), *Avg. Operation Latency/Query* is computed as memory construction time plus query time and interpreted as amortized per-query cost for systems with cumulative or bursty writes, while *Normalized Utility* is the mean of six min–max normalized answer-quality metrics from the current *LoCoMo* [20] and *LongMemEval* [31] runs. For (2), we report *Outlier-Filtered Avg. Total Latency/Query* on three benchmark.

O7-(Localized Maintenance): The most cost-efficient memory mechanisms are those that localize maintenance to a bounded subset of memory state, whereas mechanisms that

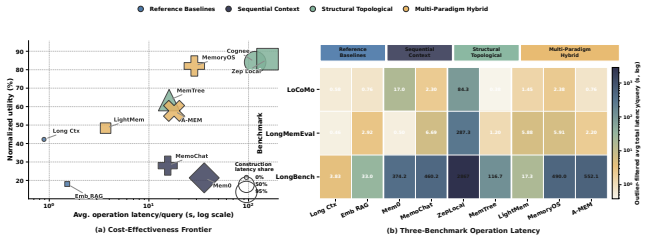

Figure 11: Operation Cost of Memory Systems.

repeatedly reorganize a large global state are the least efficient. As shown in Figure 11, (1) Among memory-augmented systems, LightMem and MemTree occupy the strongest efficiency frontier, with LightMem reaching 48.3 *Normalized Utility* at 3.67 s *Avg. Operation Latency/Query* and MemTree reaching 63.5 at 15.9 s; both are clearly more efficient than MemoChat (28.0 at 15.4 s), Mem0 (21.4 at 35.9 s), and A-MEM (57.7 at 17.9 s); (2) Higher-utility structured systems move markedly to the expensive side: MemoryOS reaches 82.0 *Normalized Utility* only at 28.6 s, while Cognee and Zep exceed 84 utility only after 116.5 s and 155.1 s; (3) The workload-specific latency view sharpens the same separation on *LongBench*: LightMem remains at 17.3 s and MemTree at 116.7 s, whereas Mem0, MemoChat, MemoryOS, and A-MEM rise to 374.2, 460.2, 490.0, and 552.1 s, respectively. It indicates that operational efficiency is governed less by whether a system uses structure than by how widely each write propagates through that structure. Specifically, (1) segmented compression and bounded hybrid retrieval keep LightMem close to the low-cost regime; (2) path-local tree aggregation allows MemTree to preserve substantially more utility without global refresh; and (3) graph-wide consolidation, multi-store synchronization, or repeated whole-memory rewriting yield stronger organization but impose the heaviest operational cost as memory grows.

Finding 5. ***(Operational Scaling Rule).*** *RQ5 shows that efficiency is governed by maintenance scope rather than structure alone. (1) Localized update and search yield the strongest cost–utility balance, as in LightMem and MemTree; (2) Richer organization helps only when its upkeep avoids broad recomputation; otherwise overhead offsets its gains, as in Cognee and MemoryOS; (3) Under long-context workloads, whole-memory coordination becomes the dominant cost driver.*

5 Fine-Grained Component Comparison

To understand the root causes behind end-to-end performance differences, we decompose agent memory systems into four fundamental modules. By systematically generating controlled variants that modify one module at a time, we evaluate the contribution of each module to overall system performance.

5.1 Memory Representation and Storage (M1)

Experimental Setting. For *"How do memory abstraction level and structural organization affect factual fidelity and downstream reasoning effectiveness?"*, we evaluate three representation-focused variants: (1) *LightMem* [7] compares *User-Only Raw*, which stores verbatim user utterances, *User-Only Summary*, which rewrites each session into an LLM-generated abstractive summary, and *User-Only Compressed*, which removes filler and redundant tokens while preserving the original phrasing and factual content; (2) *MemTree* [27] compares a shallow *Flat-biased Setting* with a *Deeper Tree Setting* to examine how hierarchical text organization affects memory fidelity. We assess the trade-off between fine-grained fact preservation and multi-step reasoning using *LoCoMo* for compositional reasoning and *LongMemEval* for multi-session factual retrieval, reporting exactness- and overlap-based metrics (e.g., *EM* and *ROUGE-L F1*).

Table 3: Ablation of Representation and Storage Mechanisms.

Method Variant		LoCoMo		LongMemEval	
		EM	Ans. F1	Substr. EM	ROUGE-L F1
LightMem	User-Only Raw	**24.2**	**38.9**	**26.0**	**31.4**
	User-Only Summary	8.5	15.6	11.7	17.4
	User-Only Compressed	23.6	38.6	10.7	19.1
MemTree	Flat-biased	18.2	30.7	23.0	29.9
	Deeper Tree	**18.7**	**31.2**	**23.3**	**30.9**
Mem0	Default	**3.2**	6.2	**9.3**	**16.5**
	Graph Store	3.0	**6.5**	8.3	15.9

O8-(Content Fidelity): Retaining the original conversational content is more important than increasing abstraction or hierarchy for sustaining both factual recall and reasoning quality. Table 3 shows that *LightMem User-Only Raw* achieves the best result on all four metrics, while *User-Only Compressed* remains close on *LoCoMo* (*Ans. F1*: 38.6 vs. 38.9; *EM*: 23.6 vs. 24.2) but drops sharply on *LongMemEval* (*Substring EM*: 10.7 vs. 26.0), *User-Only Summary* is substantially weaker on both benchmarks, and the deeper *MemTree* setting provides only modest gains over the flat setting. It indicates that the main performance boundary is the amount of recoverable evidence the representation preserves, rather than whether it applies stronger abstraction or a deeper structure. In particular, (1) Raw text is most effective when the task requires recovering exact session-level details (e.g., recalling a title such as *"Nu, pogodi!"*); (2) Light compression can still support compositional reasoning when the main meaning is preserved, but becomes unreliable for exact detail matching (e.g., relating two earlier events while missing the precise date or name); and (3) Deeper hierarchy can improve organization, but cannot restore information removed during representation (e.g., a parent node helps navigate related sessions, but not recover omitted details).

Finding 6. ***(Representation Granularity).*** *M1 shows that preserving usable evidence matters more than making memory more compact or more structured. (1)* High-retention forms *best support exact detail recovery, as in LightMem User-Only Raw; (2)* Light compression *can preserve reasoning, but weakens exact matching, as in LightMem User-Only Compressed; (3)* Hierarchy *mainly improves access, but cannot restore removed content, as reflected by the MemTree (Deeper Tree) variant.*

5.2 Memory Extraction (M2)

Experimental Setting. For *"How do write-time extraction choices affect factual fidelity and downstream reasoning effectiveness?"*, we compare extraction-related variants in three groups: *(1) MemoChat* [18], which contrasts *Heuristic Topic* and *LLM Topic* segmentation; *(2) MemOS* [15], which contrasts *Fast Memorize* and *Fine*

Table 4: Ablation of Memory Extraction Strategies.

Method Variant		LoCoMo		LongMemEval	
		EM	Ans. F1	Substr. EM	ROUGE-L F1
MemoChat	Heuristic Topic	**23.0**	33.5	**10.7**	**18.6**
	LLM Topic	22.5	**34.4**	7.3	15.9
MemOS	Fast Memorize	**25.5**	**40.8**	20.7	26.1
	Fine Memorize	2.5	5.0	**22.3**	**30.2**
LightMem	User-Only Raw	24.2	38.9	**26.0**	**31.4**
	Hybrid Raw	**25.5**	**39.7**	25.3	**31.4**

Memorize on the same `tree_text` backend; and *(3) LightMem* [7], which contrasts *User-Only Raw* and *Hybrid Raw* by extracting raw memory from user turns only or from both user and assistant turns. We evaluate *LongMemEval* for multi-session factual retrieval fidelity and *LoCoMo* for downstream multi-step reasoning, reporting exactness- and overlap-based measures (e.g., *EM* and *ROUGE-L F1*).

O9-(Coverage-Preserving Extraction): Coverage-preserving write-time extraction provides the most stable balance between factual retrieval and downstream reasoning. As shown in Table 4, *MemoChat Heuristic Topic* improves LongMemEval over *LLM Topic* (10.7 vs. 7.3 *Substr. EM*; 18.6 vs. 15.9 *ROUGE-L F1*) while keeping LoCoMo nearly unchanged (23.0/33.5 vs. 22.5/34.4 *EM*/*Ans. F1*), *MemOS Fast Memorize* far exceeds *Fine Memorize* on LoCoMo (25.5 vs. 2.5 *EM*; 40.8 vs. 5.0 *Ans. F1*) despite lower LongMemEval scores (20.7 vs. 22.3 *Substr. EM*; 26.1 vs. 30.2 *ROUGE-L F1*), and *LightMem Hybrid Raw* slightly improves LoCoMo over *User-Only Raw* (25.5 vs. 24.2 *EM*; 39.7 vs. 38.9 *Ans. F1*) with nearly unchanged LongMemEval results (25.3 vs. 26.0 *Substr. EM*; both 31.4 *ROUGE-L F1*). These results suggest that broader, less selective extraction better preserves the context needed for downstream answerability, even when more selective extraction yields modest gains on lexical factual retrieval. In particular, (1) conservative topic grouping is less likely to split a sustained thread or isolate a brief aside (e.g., a one-off hobby mention); (2) lighter memorization is more likely to retain details that later need to be combined for reasoning; and (3) including both user and assistant turns can preserve clarifying cues that user-only extraction may miss (e.g., a date or refined phrasing).

Finding 7. ***(Late Filtering Principle).*** *M2 suggests that memory extraction should preserve context at write time rather than aggressively filter details: (1) Coarser segmentation helps thread-spanning questions by keeping related cues together; (2) Limited rewriting supports compositional reasoning by retaining details that matter only when combined later; (3) Storing both user and assistant turns helps clarification-heavy dialogues by preserving refined formulations for later access.*

5.3 Memory Retrieval and Routing (M3)

Experimental Setting. For *"How do retrieval fusion and reasoning-mediated routing affect retrieval relevance and provenance-sensitive precision?"*, we compare variants in two groups: (1) *A-MEM* under *Hybrid-Balanced*, which uses a moderate dense–sparse fusion, versus *Hybrid Sparse-Leaning*, which increases the sparse contribution; and (2) *SimpleMem* under *No Planning*, which retrieves directly, versus *Planning Only*, which adds an explicit planning step, and

Table 5: Ablation of Retrieval and Routing Mechanisms.

Method Variant		LoCoMo		LongMemEval	
		Ans. F1	Recall	Substr. EM	ROUGE-L F1
A-MEM	Hybrid-Balanced	**24.6**	49.9	**27.5**	**25.9**
	Hybrid Sparse-Leaning	23.0	44.3	24.3	22.8
SimpleMem	No Planning	18.7	86.4	17.0	22.9
	Planning Only	**20.7**	**90.6**	**21.7**	**27.9**
	Planning + Reflect	20.0	88.6	21.3	26.1

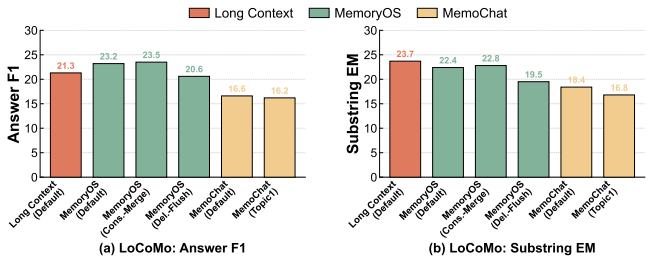

Figure 12: Ablation of Maintenance Strategies.

Planning + Reflect, which further introduces a lightweight reflection stage. We evaluate on *LongMemEval* to measure scattered-history retrieval relevance, and on *LoCoMo* to assess provenance-sensitive memory access and supporting-memory identification, reporting overlap-based measures (e.g., *Substr. EM*, *ROUGE-L F1*).

O10-(Planning and Fusion): Explicit planning and balanced retrieval fusion provide the strongest improvement in retrieval effectiveness. As shown in Table 5, *A-MEM* achieves its best performance with *Hybrid-Balanced*, reaching 24.6 *Ans. F1* and 27.5 *Substr. EM*, compared with 23.0 and 24.3 under *Hybrid Sparse-Leaning*, while *SimpleMem* achieves its best performance with *Planning Only*, reaching 20.7 *Ans. F1*, 90.6 *Strict Rec.*, 21.7 *Substr. EM*, and 27.9 *ROUGE-L F1*, above both *No Planning* and *Planning + Reflect*. These results indicate that stronger retrieval and routing performance comes from adding useful structure, rather than from simply increasing sparse matching or extra reasoning steps. In particular, (1) moderate fusion appears more effective than sparse-leaning fusion for preserving both answer quality and relevance (e.g., semantically related but lexically varied facts); (2) explicit planning consistently improves over direct retrieval (e.g., multi-constraint memory queries); and (3) adding reflection on top of planning does not yield further gains, suggesting that extra deliberation may weaken rather than improve routing decisions.

Finding 8. ***(Retrieval Strategy Guidance).*** *M3 indicates that retrieval quality improves most from targeted structure rather than added complexity: (1) moderate hybrid fusion is preferable when evidence is semantically related but lexically diverse; (2) lightweight planning is effective for constrained memory lookup; and (3) once a route is already specified, extra reflection brings limited benefit and mainly adds overhead.*

5.4 Memory Maintenance (M4)

Experimental Setting. For *"How do consolidation aggressiveness, flush timing, and summary granularity affect update correctness and long-horizon memory consistency?"*, we compare maintenance-relevant variants in two groups: *(1) MemoChat* under default multi-topic consolidation versus *Topic1*, which forces each window into a

single-topic summary; and *(2) MemoryOS* under default immediate consolidation versus *Delayed-Flush*, which enlarges the short-term buffer before backend writes, and *Conservative-Merge*, which raises the topic-similarity threshold for stricter assimilation. We evaluate on *LoCoMo* to assess whether these maintenance choices preserve updated facts and coherent memory use over extended contexts.

O11-(Conservative Consolidation): Conservative consolidation is more effective than delayed flushing or overly coarse summarization for maintaining answer-relevant memory. As shown in Figure 12, the stricter-merge variant, *MemoryOS (Conservative-Merge)*, improves over default *MemoryOS* from 23.2 to 23.5 in *Ans. F1* and from 22.4 to 22.8 in *Substr. EM*, whereas delaying flushes lowers the same system to 20.6/19.5, and forcing single-topic summaries in *MemoChat* also underperforms its default setting at 16.2/16.8 versus 16.6/18.4; *Long Context* remains highest on *Substr. EM* at 23.7. It indicates that maintenance is most effective when it selectively consolidates evidence without either leaving it unresolved or compressing it too aggressively, while raw context still better preserves exact phrasing. In particular, (1) conservative merging can retain related details for later recomposition (e.g., dispersed hobby mentions); (2) delayed flushing leaves more evidence unresolved before retrieval (e.g., activity split across turns).

Finding 9. ***(Maintenance Design Principle).*** *M4 suggests that memory maintenance works best under a balanced update regime: (1) conservative integration preserves cross-turn linkages for long-horizon reasoning; (2) delayed flushing leaves recent evidence fragmented at query time; and (3) overly coarse summarization obscures sparse but useful cues.*

6 Conclusion

We present a comprehensive review of existing agent memory systems from a data management perspective. We conduct thorough end-to-end performance evaluation of typical agent memory systems and explore their suitable application scenarios. Additionally, we delve into the impact of the individual building blocks by constructing multiple memory module variants, thereby identifying the most effective methods for representation, extraction, routing, and maintenance, as well as the most influential factors governing operational costs and long-horizon stability. Finally, we summarize the findings and present guidance for users in selecting suitable memory architectures, alongside outlining promising research directions. We will also release the testbed and evaluation framework.

References

[1] Claude Code. *(Anthropic).* https://www.claude.com/product/claude-code

[2] Anthropic Engineering. 2025. Effective context engineering for AI agents. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents.

[3] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. In *ACL (1)*. Association for Computational Linguistics, 3119–3137.

[4] Liana Caminal et al. 2025. Filtered Vector Search: State-of-the-art and Research Opportunities. In *Proceedings of the VLDB Endowment*, Vol. 18. 5488–5491.

[5] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. *arXiv preprint arXiv:2504.19413* (2025).

[6] Pengfei Du. 2026. Memory for Autonomous LLM Agents: Mechanisms, Evaluation, and Emerging Frontiers. *arXiv preprint arXiv:2603.07670* (2026).

[7] Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. 2025. LightMem: Lightweight and Efficient Memory-Augmented Generation. *CoRR* abs/2510.18866 (2025). arXiv:2510.18866 doi:10.48550/ARXIV.2510.18866

[8] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey. *CoRR* abs/2312.10997 (2023).

[9] Google. 2025. Memory – Agent Development Kit (ADK). https://google.github.io/adk-docs/sessions/memory/.

[10] Yifan Hu, Siyin Liu, Yifei Yue, Guoqiang Zhang, Benyou Liu, Fengbin Zhu, Jingkuan Lin, et al. 2025. Memory in the Age of AI Agents. *arXiv preprint arXiv:2512.13564* (2025).

[11] Guozhang Kang, Zhenying Ge, Jie Hu, Xinyuan Zhang, Li Wang, and Jianfeng Zhan. 2025. BigVectorBench: Heterogeneous Data Embedding and Compound Queries are Essential in Evaluating Vector Databases. *Proceedings of the VLDB Endowment* 18, 6 (2025), 1536–1549.

[12] Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory OS of AI Agent. In *EMNLP*. Association for Computational Linguistics, 25961–25970.

[13] Arijit Khan, Yuyu Luo, Wenjie Zhang, Mingjie Zhou, and Xiaofang Zhou. 2025. Retrieval-augmented Generation (RAG): What is There for Data Management Researchers? *ACM SIGMOD Record* 54, 4 (2025).

[14] Guoliang Li, Xuanhe Zhou, and Xinyang Zhao. 2024. LLM for Data Management. *Proc. VLDB Endow.* 17, 12 (2024), 4213–4216.

[15] Zhiyu Li, Shichao Song, Chenyang Xi, Hanyu Wang, Chen Tang, Simin Niu, Ding Chen, Jiawei Yang, Chunyu Li, Qingchen Yu, Jihao Zhao, Yezhaohui Wang, Peng Liu, Zehao Lin, Pengyuan Wang, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhen Tao, Junpeng Ren, Huayi Lai, Hao Wu, Bo Tang, Zhenren Wang, Zhaoxin Fan, Ningyu Zhang, Linfeng Zhang, Junchi Yan, Mingchuan Yang, Tong Xu, Wei Xu, Huajun Chen, Haofeng Wang, Hongkang Yang, Wentao Zhang, Zhi-Qin John Xu, Siheng Chen, and Feiyu Xiong. 2025. MemOS: A Memory OS for AI System. *CoRR* abs/2507.03724 (2025). arXiv:2507.03724 doi:10.48550/ARXIV.2507.03724

[16] Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. 2026. SimpleMem: Efficient Lifelong Memory for LLM Agents. *CoRR* abs/2601.02553 (2026).

[17] Shu Liu, Soujanya Ponnapalli, Shreya Shankar, Sepanta Zeighami, Alan Zhu, Shubham Agarwal, Ruiqi Chen, Samion Suwito, Shuo Yuan, Ion Stoica, Matei Zaharia, Alvin Cheung, Natacha Crooks, Joseph E. Gonzalez, and Aditya G. Parameswaran. 2026. Supporting Our AI Overlords: Redesigning Data Systems to be Agent-First. In *Proceedings of the 16th Annual Conference on Innovative Data Systems Research (CIDR)*.

[18] Junru Lu, Siyu An, Mingbao Lin, Gabriele Pergola, Yulan He, Di Yin, Xing Sun, and Yunsheng Wu. 2023. MemoChat: Tuning LLMs to Use Memos for Consistent Long-Range Open-Domain Conversation. *CoRR* abs/2308.08239 (2023). arXiv:2308.08239 doi:10.48550/ARXIV.2308.08239

[19] Yuyu Luo, Guoliang Li, Ju Fan, and Nan Tang. 2026. Data Agents: Levels, State of the Art, and Open Problems. *arXiv preprint arXiv:2602.04261* (2026). SIGMOD 2026 Tutorial.

[20] Adyasha Maharana, Dong-Ho Lee, Sergey Turishcheva, Kezhen Nham, Golnaz Jandaghi, Jay Pujara, and Xiang Ren. 2024. Evaluating Very Long-Term Conversational Memory of LLM Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

[21] Vasilije Markovic, Lazar Obradovic, László Hajdu, and Jovan Pavlovic. 2025. Optimizing the Interface Between Knowledge Graphs and LLMs for Complex Reasoning. *CoRR* abs/2505.24478 (2025).

[22] MemoryAgentBench Team. 2026. Evaluating Memory in LLM Agents via Incremental Multi-Turn Interactions. In *Fourteenth International Conference on Learning Representations (ICLR)*.

[23] Microsoft. 2025. Introducing Copilot Memory: A More Productive and Personalized AI. https://techcommunity.microsoft.com/blog/microsoft365copilotblog/introducing-copilot-memory.

[24] OpenAI. 2026. Context Engineering for Personalization – State Management with Long-Term Memory Notes using OpenAI Agents SDK. https://developers.openai.com/cookbook/examples/agents_sdk/context_personalization/.

[25] Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560* (2023).

[26] Preston Rasmussen, Pavel Paliychuk, Travis Beauvais, and Jesse Ryan. 2025. Zep: A Temporal Knowledge Graph Architecture for Agent Memory. *arXiv preprint arXiv:2501.13956* (2025).

[27] Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. 2025. From Isolated Conversations to Hierarchical Schemas: Dynamic Tree Memory Representation for LLMs. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. https://openreview.net/forum?id=moXtEmCleY

[28] Harmanpreet Singh, Nikhil Verma, Yixiao Wang, Manasa Bharadwaj, Homa Fashandi, Kevin Ferreira, and Chul Lee. 2024. Personal Large Language Model Agents: A Case Study on Tailored Travel Planning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Association for Computational Linguistics, Miami, Florida, US, 486–514. doi:10.18653/v1/2024.emnlp-industry.37

[29] Haoran Tan, Zeyu Zhang, Chen Ma, Xu Chen, Quanyu Dai, and Zhenhua Dong. 2025. MemBench: Towards More Comprehensive Evaluation on the Memory of LLM-based Agents. In *ACL (Findings) (Findings of ACL)*. Association for Computational Linguistics, 19336–19352.

[30] Zhiwei Tang et al. 2026. LLM Agent Memory: A Survey from a Unified Representation. *arXiv preprint arXiv:2603.0359* (2026).

[31] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, and Kai-Wei Chang. 2024. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. *arXiv preprint arXiv:2410.10813* (2024).

[32] Yanchen Wu, Tenghui Lin, Yingli Zhou, Fangyuan Zhang, Qintian Guo, Xun Zhou, Sibo Wang, Xilin Liu, Yuchi Ma, and Yixiang Fang. 2026. Memory in the LLM Era: Modular Architectures and Strategies in a Unified Framework. *Proceedings of the VLDB Endowment* (2026).

[33] Wujiang Xu et al. 2025. A-MEM: Agentic Memory for LLM Agents. *arXiv preprint arXiv:2502.12110* (2025).

[34] Chao Yang, Chuan Zhou, Yanghua Xiao, Shuai Dong, Liang Zhuang, et al. 2026. Graph-based Agent Memory: Taxonomy, Techniques, and Applications. *arXiv preprint arXiv:2602.05665* (2026).

[35] Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. 2025. MemAgent: Reshaping Long-Context LLM with Multi-Conv RL-based Memory Agent. *CoRR* abs/2507.02259 (2025).

[36] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025. A Survey on the Memory Mechanism of Large Language Model based Agents. *ACM Transactions on Information Systems* (2025).

[37] Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, Zhong-Zhi Li, Yingying Zhang, Le Song, and Qianli Ma. 2025. LifelongAgentBench: Evaluating LLM Agents as Lifelong Learners. *CoRR* abs/2505.11942 (2025).

[38] Junhao Zheng, Chengming Shi, Xidi Cai, Qiuke Li, Duzhen Zhang, Chenxing Li, Dong Yu, and Qianli Ma. 2025. Lifelong Learning of Large Language Model based Agents: A Roadmap. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2025).

[39] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. MemoryBank: Enhancing Large Language Models with Long-Term Memory. In *AAAI*. AAAI Press, 19724–19731.

[40] Wei Zhou, Xuanhe Zhou, Qikang He, Guoliang Li, Bingsheng He, Quanqing Xu, and Fan Wu. 2026. Automating Database-Native Function Code Synthesis with LLMs. *Proc. ACM Manag. Data* 3, 4 (2026), 141:1–141:26.

[41] Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. 2025. MEM1: Learning to Synergize Memory and Reasoning for Efficient Long-Horizon Agents. *CoRR* abs/2506.15841 (2025).