VikingMem: A Memory Base Management System for Stateful LLM-based Applications

Jiajie Fu[*†]
Zhejiang University
China
jiajiefu@zju.edu.cn

Junwen Chen[†]
ByteDance
China
chenjunwen@bytedance.com

Mengzhao Wang[*]
Zhejiang University
China
wmzssy@zju.edu.cn

Aoxiang He
ByteDance
China
heaoxiang@bytedance.com

Maojia Sheng
ByteDance
China
shengmaojia@bytedance.com

Xiangyu Ke
Zhejiang University
China
xiangyu.ke@zju.edu.cn

Yifan Zhu
Zhejiang University
China
xtf_z@zju.edu.cn

Yunjun Gao
Zhejiang University
China
gaoyj@zju.edu.cn

ABSTRACT

Large Language Models have revolutionized interactive applications, however, their finite context windows pose a critical data management challenge for maintaining stateful, long-term interactions. Existing memory approaches, however, often rely on simplistic extraction that leads to incomplete memories or use rigid, single-use memory extract prompts tailored for one use case like Chatbots. Consequently, they lack generalizability and perform poorly on diverse downstream tasks. To bridge this gap, we introduce the Memory Base, a novel data management paradigm to *manage the persistent state of long-term interactions*. It is characterized by three core principles: *selective extraction* of high-value memories from raw information streams; inherent *statefulness and evolution*, where memory content is progressively summarized, corrected, and temporally weighted to prioritize recent interactions; and a *generalizable abstraction* paradigm designed for robust transferability across diverse applications, including education, recommendation, and agent memory.

Building on this foundation, we present VikingMem, an end-to-end Memory Base Management System implemented on the VikingDB vector engine. VikingMem materializes this paradigm through interconnected event and entity abstractions. It features event-centric memory extraction to selectively handle complex information streams, while entities are dynamically updated by events to achieve stateful evolution. Using temporal compression via a topic-wise timeline and time-weighted recall, the system can progressively produce high-level summary memories, prioritize recent items, while compressing and fading older ones. Retrieval is further optimized through multi-vector-based reranking. Extensive evaluations on long-term memory benchmarks demonstrate that VikingMem achieves state-of-the-art performance, outperforming baselines by up to 30% in memory retrieval effectiveness while maintaining low latency essential for interactive applications.

PVLDB Reference Format:
Jiajie Fu, Junwen Chen, Mengzhao Wang, Aoxiang He, Maojia Sheng, Xiangyu Ke, Yifan Zhu, and Yunjun Gao. VikingMem: A Memory Base Management System for Stateful LLM-based Applications. PVLDB, 19(12): 4344 - 4357, 2026.

[*]Work done while working with ByteDance.
[†]Both authors contributed equally to this research.

doi:10.14778/3827998.3828037

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/BytedanceFu/VikingMem.

1 INTRODUCTION

Recent advances in large language models (LLMs) have catalyzed a new generation of services — persistent assistants [1, 3, 51], autonomous agents [6], and domain-tailored chat systems [66] — that must act coherently across *many sessions* and *long time horizons*. While LLMs' context windows are expanding [17] , it remains a finite, costly, and latency-sensitive resource [48]. Simply embedding ever-longer histories into every prompt is not only impractical for performance-sensitive services but is architecturally incapable of providing structured, lifecycle-aware state management for consolidation, forgetting, or provenance [10, 44]. For instance, consider a personalized writing tutor that interacts with a single student over months, retaining past drafts, tracking recurring mistakes, and proactively suggesting targeted exercises. Midway through a long-term engagement, the assistant might repeat questions asked weeks earlier or contradict its own prior corrections. These issues do not primarily stem from limitations in the LLM's reasoning ability. Instead, they arise because long-term state is often handled in an ad hoc and fragmented manner at the application layer. While an LLM's context window is transient and bounded, persistent state is frequently spread across logs, shallow caches, and task-specific heuristics rather than maintained through a unified memory mechanism. This fragmented design can lead to broken continuity, stale or conflicting memory, reduced user trust, and substantial developer overhead for custom reconciliation logic [22, 29]. Consequently, applications are *shifting from short, stateless queries toward long-lived, stateful interaction patterns*. This shift necessitates a service-grade

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828037

memory substrate. Here, we draw inspiration from cognitive science, defining "memory" not as volatile in-memory storage, but as the persistent, evolving record of experiences that shapes future behavior. Such a substrate must be architected to persistently capture interaction traces, manage their lifecycle through consolidation and updates, enable context-aware retrieval with efficient reranking, and adhere to privacy and audit policies [24, 41].

This shift necessitates a dedicated memory management substrate, yet existing solutions fail to meet the scale and latency demands of production environments. Recent systems attempting to structure memory via Knowledge Graphs (KGs)—such as ZEP [45] and Mem0's graph integrations [7]—introduce prohibitive operational overhead. For instance, the dynamic KG construction employed by these frameworks incurs significant build-time latency, making them unusable under the high-throughput write loads typical of large-scale services[1]. Furthermore, some systems like Memobase [38] are rigidly engineered for narrow verticals—primarily conversational user profiles. This specialization creates a critical adaptability gap: a schema optimized for chat history cannot accommodate the procedural SOPs required by autonomous agents[8, 33]. Consequently, the industry lacks a unified, configurable memory infrastructure capable of serving these diverse scenarios with a single underlying paradigm.

To address these challenges, we present VikingMem, a unified memory management system architected upon three core design principles derived from our production experience (detailed in § 2.1). First, we argue that effective memory requires *selective information extraction*. Unlike static archives, VikingMem employs fine-grained segmentation to transform low-density raw streams into high-value, discrete signals, maximizing the utility of finite context windows while minimizing token costs. Second, we design the system to possess *inherent statefulness and evolution*. Rather than a static archive, VikingMem treats memory as a living lifecycle where discrete events are managed via temporal weighting to prioritize recent context, while simultaneously driving the progressive consolidation and continuous refinement of persistent entities (e.g., user profiles [33] or agent SOPs [8]). This dual mechanism ensures that both episodic history and evolving state remain coherent and up-to-date. Finally, to support diverse industrial scenarios—ranging from personalized search and education to complex autonomous agents [5, 24]—we implement a *generalizable design paradigm*. By abstracting memory into a flexible Event-Entity model, VikingMem decouples storage logic from specific business rules, allowing a single infrastructure to serve disparate downstream tasks with minimal reconfiguration.

To this end, we designed and implemented VikingMem, a cloud-native Memory Base Management System built atop the VikingDB vector engine[2] (a cloud-native vector database used in ByteDance). Architecturally, VikingMem features an intelligent segmentation engine that leverages a two phase LLM-based strategy for precise noise reduction and semantic partitioning (§ 3.1), a unified operator library supporting diverse update logic for dynamic entity evolution (§ 3.2), and a schema-driven extraction pipeline that enables efficient single-pass processing (§ 2.2.1), i.e., extracting multiple types of memory from the same input data with a single LLM call. Crucially, this architecture delivers significant efficiency gains, validated through both extensive benchmarks and large-scale production deployment at ByteDance (§ 5). Compared to naive RAG or raw-log approaches, VikingMem reduces storage costs by up to 83.2% through selective retention and lowers retrieval latency (P95) by 900ms. Furthermore, its single-pass extraction pipeline is highly token-efficient: when extracting k distinct memory types, it reduces token consumption by approximately $(k-1)\times$ compared to multi-pass baselines, where each memory type is extracted with a separate LLM call over the same input. This efficiency enables VikingMem to handle extreme-scale workloads—where a single tenant alone generates over 1 billion tokens daily—with minimal computational overhead. VikingMem has been deployed in production and is commercially available. It has supported a broad range of stateful LLM applications like social companionship, efficiency and collaboration tools and personalized education, detailed usage guidelines are in [52]. Furthermore, to foster community research and technical knowledge sharing in LLM memory systems, we have open-sourced a subset of the core capabilities of VikingMem via OpenViking[3], an open-source Context Database for AI Agents.

Contribution. Our main contributions are summarized below.

- We formally define the Memory Base, a new architectural framework for persistent state in LLM applications. We distinguish it from prior memory processing methods by establishing three core principles—selective extraction, inherent statefulness and evolution, and a generalizable design paradigm (§ 1). We provide the underlying design schemas of our event-entity model (§ 2) to serve as a foundational reference, offering guidance in this emerging field.
- We design and implement VikingMem, an end-to-end Memory Base Management System (MBMS) that realizes our proposed framework. We present its detailed design and key technical modules (§ 3), such as intelligent memory segmentation strategy, flexible operator-based update mechanism for entity evolution, configurable schema for memory extraction generalizability and other implement details.
- We demonstrate the framework's adaptability by detailing its deployment across five distinct industrial scenarios at ByteDance, ranging from autonomous agents to personalized education (§ 4).
- Through extensive experiments on public benchmarks, we demonstrate that VikingMem achieves state-of-the-art performance, we demonstrate that VikingMem achieves state-of-the-art performance, outperforming existing baselines by up to 30% in retrieval accuracy (LLM-Judge Score) maintaining the lowest P95 retrieval latency among all compared methods (§ 5).

2 SYSTEM DESIGN PRINCIPLES AND DATA MODEL

In this section, we first articulate the industrial observations and core design principles that shape VikingMem (§ 2.1). We then formally define the system's foundational data abstractions—the Event-Entity Memory Model (§ 2.2)—and finally outline the key design objectives that guide its implementation (§ 2.3).

[1] In our production environment, individual tenants can generate over 1 billion tokens of memory data daily.

[2] https://www.byteplus.com/en/product/vectordatabase

[3] https://github.com/volcengine/OpenViking

2.1 Industry Observations and Design Principles

Based on our extensive experience serving diverse enterprise clients and commercializing LLM solutions at ByteDance, we identify three critical requirements that a production-grade memory substrate must satisfy. These observations directly motivate the core capabilities of VikingMem.

Observation 1: Low Signal-to-Noise Ratio in Raw Streams. Unlike traditional RAG systems that ingest high-density, pre-processed documents [14, 25], production memory systems operate on low-density raw streams (e.g., meeting transcripts, debugging logs) where valuable information is sparse and topics are interleaved [20, 54]. Simply truncating or summarizing these streams blindly leads to "context pollution," where LLMs are distracted, or "over-squashed", by irrelevant noise [2].

Principle 1: To address this, VikingMem is built upon *selective information extraction*. The system must go beyond naive chunking and provide *fine-grained event extraction and segmentation*. Specifically, this requires a sophisticated strategy to prune irrelevant noise (e.g., greetings, fillers) and identify the precise start and end indices of semantic segments, reconstituting disjointed information into coherent "events" (§ 3.1). Furthermore, given the high token cost of raw streams, this extraction must be *computationally efficient*. The system should employ a low-cost technique that minimizes redundant processing when extracting multiple memory types from the same stream, avoiding the excessive token consumption of traditional multi-pass approaches [7].

Observation 2: Dynamic Nature of State. Real-world entities — whether a user learning a language or an agent executing a workflow—are constantly changing. Traditional memory approaches that rely on static vector storage (i.e., simple "insert-and-retrieve" RAG pipelines) fail to capture this *evolution*. They merely accumulate history without updating the underlying state, leading to contradictions and stale information.

Principle 2: Drawing inspiration from cognitive science [31, 37], we design VikingMem to possess *inherent statefulness and evolution*. It treats memory as a living lifecycle where discrete events actively drive the continuous refinement of persistent entities, mirroring human memory consolidation processes [46]. To realize this, the system must support *dynamic entity-state tracking* through a unified update mechanism. This involves a library of reusable *operators* (e.g., `LLM_MERGE`, `SUM`, `TIME_COMPRESS`) that programmatically link events to entities (§ 3.2). These operators handle both efficient updates and progressive consolidation, strategically summarizing or forgetting dated information [58] to maintain a compact state. Furthermore, to ensure long-term relevance during retrieval, the system applies dynamic weighting mechanisms (including temporal and business-specific weights) to prioritize the most pertinent context (§ 3.2).

Observation 3: Fragmentation of Memory Architectures. Different downstream applications demand fundamentally disparate memory structures and contents. For instance, emotional companions focus on capturing user personality and conversational style from dialogue history [24], whereas autonomous agents require extracting executable experiences from historical trajectories to refine their standard operating procedures (SOPs) [55]. Building siloed solutions for each scenario leads to high maintenance costs and prevents capability sharing.

Principle 3: To support these diverse scenarios, VikingMem adopts a *generalizable design paradigm*. We implement a flexible and transferable memory schema that decouples the underlying storage logic from specific business rules. By abstracting memory into a configurable *Event-Entity model* defined via a JSON-like schema (§ 2.2.1), a single infrastructure can represent disparate memory types (e.g., declarative preferences, procedural SOPs) and serve downstream tasks with minimal reconfiguration. This schema-driven design is also the key enabler for the single-pass extraction efficiency mentioned in Principle 1.

2.2 Event-Entity Primitives

The core of VikingMem's architecture is built upon two fundamental data abstractions, Events and Entities. This section defines these components and the mechanism that links them.

2.2.1 Core Components and Schemas. In this part, we will formally define the fundamental components that form the basis of our model: the Event and Entity data abstractions. We will detail their respective schemas, which specify their structure and—critically for the Entity—embed the Operator logic that links the two.

Event: An event is the foundational unit of memory, representing a discrete, timestamped, and episodic record.[4] The full event instance includes immutable metadata, such as a timestamp; these elements are omitted from Figure 1 for clarity. A detailed example of the complete event instance can be found in § 4.2. They are the unit of ingestion: compact, meaningful records selectively extracted from raw interaction streams (sessions) that capture a single, salient piece of information. The Event Definition Schema shown in Figure 1 serves as a customizable template. Importantly, an event in VikingMem is not a raw recency-based context window, but a selectively extracted, semantically coherent, schema-constrained memory unit derived from potentially interleaved raw sessions.

Entity: An entity represents a persistent, evolving state representation (e.g., a comprehensive user profile, an agent's library of Tool Usages). Unlike instantaneous, episodic events, entities integrate and consolidate information over time to form a coherent, long-term memory. The Entity Schema (right part in Figure 1) grants similar flexibility, allowing users to define an `EntityType`, `Description`, and its constituent `Properties`. The core mechanism for this stateful evolution is detailed within each property's `AggregateExpression`. This expression explicitly dictates the rule for property updates by defining the required `EventType` and `PropertyName` from an incoming event, thereby specifying exactly which events trigger the entity's evolution. Likewise, an entity is not merely a compacted status note, but a persistent state representation incrementally materialized from events through explicit aggregation expressions and operators.

Operator: An operator is an *application-defined* function that governs *how an entity's state is updated by an associated event*. Operators abstract the business logic of memory evolution. When a linked

[4] Human memory is fundamentally event-based and organized into discrete episodic records [18].

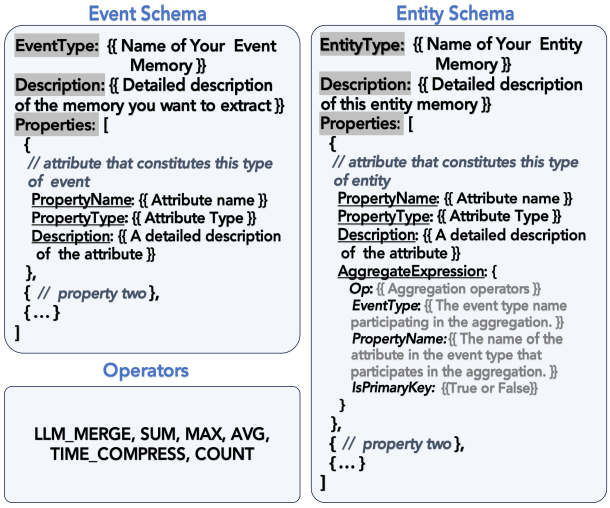

Figure 1: The definition schema of event, entity for memory extraction, and the built-in operators in VikingMem

event is ingested, the operator specified in the AggregateExpression's Op field is executed, using the designated event attribute to update the entity's related attributes. This process completes the memory's update and evolution cycle, ensuring the entity remains a coherent state. As shown in Figure 1, VikingMem provides a set of built-in operators (e.g., `SUM`, `MAX`, `AVG`, `COUNT`, `LLM_MERGE`) that can aggregate numeric data, overwrite old values, or perform complex textual merges (the details of the operators can be found in §3.2).

2.2.2 Demystifying the Event-Entity Paradigm. The rationale for this event–entity abstraction is its ability to serve as a general substrate for diverse downstream tasks. Traditional memory architectures are, in essence, built around a single notion of short-lived, event-level (or "episode") memory [7, 54]. The additional "memory types" they introduce are usually not new abstractions: they either (i) change the prompts used for event extraction, or (ii) hard-code task logic into prompts, so that each business workflow encodes its own event selection, summarization, and update rules directly in LLM prompts. As a result, *every new scenario requires another layer of prompt engineering*, while the memory system itself does not provide a stable, reusable interface. In contrast, VikingMem treats the event store as a generic log and the entity store as a family of materialized views over that log. The relationship between events and entities is expressed as a parametric query, rather than a hand-crafted prompt, as illustrated below:

$$\texttt{entity} := \texttt{SELECT OP(event.content) FROM Events}$$
$$\texttt{WHERE filters(event) GROUP BY keys(event).}$$

Here, keys defines how events are grouped into entities (e.g., at the per-user level, per user–assistant pair, or per topic), filters constrains which events are eligible (e.g., a time window), and OP is chosen from a fixed operator library (e.g., `LLM_MERGE`, `TIME_COMPRESS`, `AVG`, `SUM`). Application-specific prompts appear only at the edges of this pipeline—within event extraction and, where necessary, inside LLM-related operators—while the overall transformation pattern remains unchanged.

This design yields both abstraction and flexibility. By changing only keys, filters, and the selected operator, the same event log can materialize user profiles, SOPs, agent working memories for tools, search–recommendation–QA traces, or subject-level learning progress records, without modifying the core memory engine. Moreover, the same operators are reused across scenarios: an `LLM_MERGE` operator over "user–day" groups produces periodic entity snapshots, while the same `LLM_MERGE` operator over "user" groups realizes incremental, lifelong entity refinement. In this sense, VikingMem elevates memory design from prompt-by-prompt engineering to a small, reusable algebra over events and entities, making the system inherently reusable, programmable, and compatible with diverse downstream applications.

2.3 Design Objectives

The design of VikingMem targets a core trade-off: improving the utility of memory for downstream LLM tasks while controlling the cost of maintaining and retrieving it. We focus on two objectives. ① **Memory Effectiveness:** Given a query Q, the system retrieves or constructs a memory context M that maximizes the performance of $\mathcal{L}(Q, M)$, such as accuracy, continuity, and relevance. Effective memory bridges the gap between the LLM's bounded context window and the persistent state required for long-term interactions. ② **Operational Efficiency:** The system must reduce the cost of the memory lifecycle, including the extraction cost of identifying salient events from low-density streams and the retrieval latency of fetching, reranking, and consolidating memories into prompts.

Overall, VikingMem aims to provide high-value memory context without imposing prohibitive latency or computational overhead.

3 SYSTEM DESIGNS

In this section, we provide a detailed explanation of the VikingMem system design and its core components. As illustrated in Figure 2, the system is architected around a central pipeline comprising three key stages: Memory Extraction (§ 3.1), Memory Management (§ 3.2), and Memory Retrieval (§ 3.3). VikingMem is implemented as a memory-management layer over VikingDB, which serves as the underlying storage and retrieval substrate. Concretely, event memories are persisted as searchable records with text, metadata, and hybrid retrieval features; entity memories are maintained as separately updated state records; and the keyword graph is materialized as an auxiliary association index. In this design, VikingDB is responsible for persistent storage while VikingMem implements the memory-specific logic above it, such as schema-driven extraction, deduplication and operator-based event-to-entity updates. The Memory Extraction module processes raw data to capture salient information; the Memory Management module handles the memory lifecycle, including event-to-entity updates via operators and progressive consolidation; and the Memory Retrieval module selects relevant memories to provide context for a stateful response. In the following, we detail the core technical designs, optimizations, and motivations behind each component.

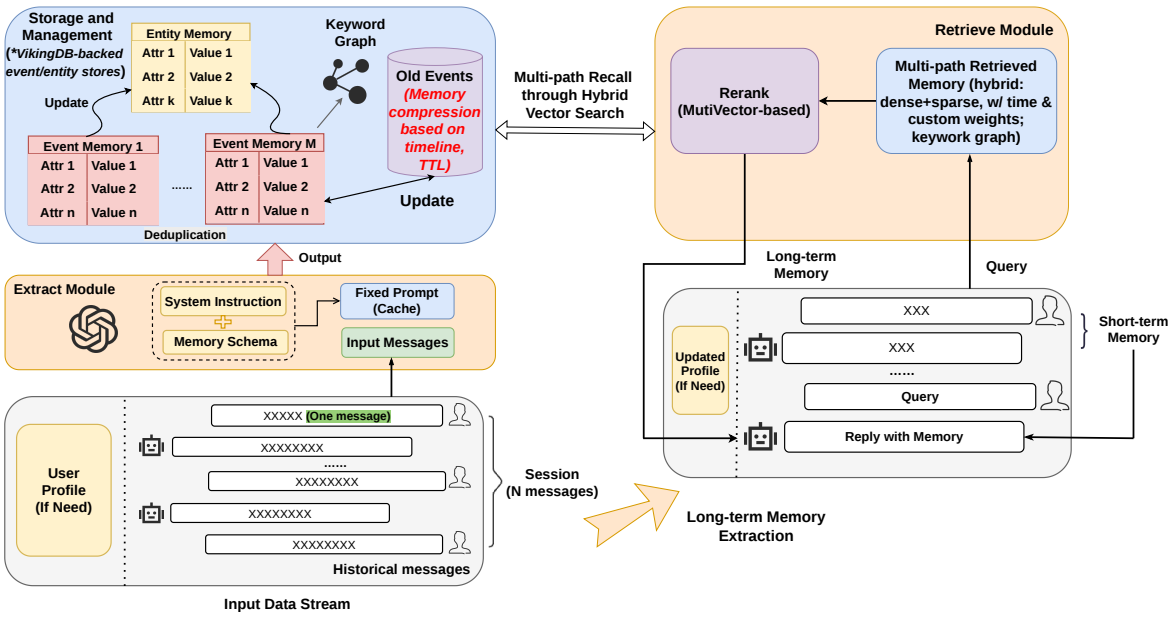

Figure 2: The pipeline of VikingMem

3.1 Memory Extract

As illustrated in Figure 2, the memory extraction module receives a session (e.g., a batch of N messages) from the input data stream and processes it as input messages to perform memory extraction. This process is activated once a logical batch of data has been accumulated (i.e., a completed session). While this batching can be triggered dynamically, we find that a well-defined threshold (≥ 20 messages) for data accumulation often yields the most consistent and high-quality memories . This design is also crucial for balancing short-term and long-term memory. We define short-term memory as the most recent data within the user's active session, which is retained for immediate contextual relevance. Meanwhile, this session is then converted into persistent long-term memories. This conversion prevents the LLM's active context from becoming overloaded—a condition that can lead to hallucinations and heightened computational complexity [22, 32]. By systematically converting accumulated data into persistent long-term memories, our approach preserves contextual richness while enabling real-time updates.

Next, we will first detail our one-pass memory extraction paradigm. We elucidate how we leverage the structured event and entity schemas to instruct a large language model to complete the entire memory workflow within a single inference pass. This entire mechanism is built upon the LLM's in-context learning (ICL) ability. Afterwards, we describe the prefix-cache optimizations we designed for the memory extraction prompt, i.e., reusing the cached representations of the fixed prompt prefix (such as the system instruction and memory schema) across requests [42]. It substantially reduces token consumption during the process. Finally, we explain how we leverage event schemas and sophisticated memory segmentation technique to derive more comprehensive and meaningful memory information from low-information-density input data stream.

One-pass Memory Extraction. Traditional memory extraction paradigms, as illustrated in Figure 3, suffer from significant inefficiencies. Prior works, such as [38, 54], typically define each memory type using a distinct, hard-coded prompt. This not only demands extensive prompt engineering and yields inconsistent results, but more critically, it necessitates a multi-pass approach: to extract N memory types, the same raw input data must be fed to the LLM N times. This repeated processing leads to prohibitive token consumption and computational costs, far exceeding practical industrial limits. In contrast, VikingMem implements a *one-pass memory extraction paradigm*. Instead of multiple unstructured prompts, we leverage our configurable event and entity schemas (§ 2.2.1). These schemas are programmatically compiled into a single, comprehensive prompt. This allows the LLM to process the raw input stream just once and, in that single pass, simultaneously extract all defined memory types—both new events and the precise content for entity updates. This schema-driven, one-pass approach achieves high-quality, structured extraction while drastically reducing computational costs, making complex memory operations efficient and economically viable (demonstrated in § 5.3).

Faster Entity Update. Although VikingMem can extract all memory types in a single LLM pass, updating **entity** memories may still introduce an extra LLM call when an entity field is of *string* type: we typically feed the extracted *entity-related events* together with the old entity back to the LLM to synthesize the updated entity. This is acceptable for offline processing, but in latency-sensitive scenarios (e.g., real-time, in-dialogue knowledge/attribute summarization where users only care about entity memory), a second LLM call significantly increases both response time and cost. To enable online, low-cost updates, we propose an Entity Update Algorithm (EUA, Algorithm 1) inspired by Roocode [47], which updates entities *without any additional LLM*. Specifically, for each entity field defined

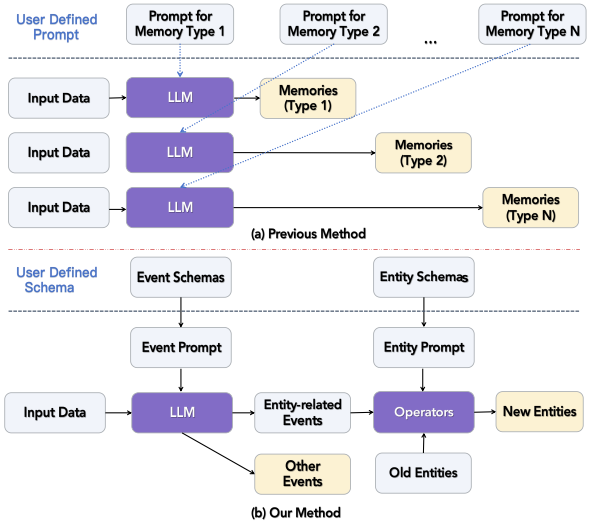

Figure 3: The extract paradigm in VikingMem

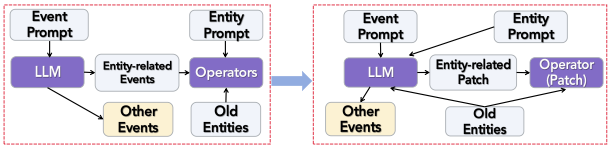

Figure 4: Faster Entity Update w/o LLM

in the schema, the extractor outputs a compact *patch* in the form "«« SEARCH ... ==== ... »» REPLACE"; EUA then applies this patch to the corresponding old field by locating the best-matching span using **edit-distance-based approximate search** [9] (robust to minor LLM inaccuracies that break exact string matching) and replacing it with the REPLACE content. This turns entity maintenance into a lightweight deterministic post-processing step: the one-pass extraction produces field-wise patches, and EUA applies them to obtain the new entity instantly, achieving near-perfect update reliability with drastically reduced latency and token cost. Importantly, this patching process does not require enumerating all stored entities. For each update, VikingMem first retrieves only a small candidate set of relevant existing entities (top-5 in our deployment) using ANN search over entity representations, and includes only these candidates in the prompt for patch generation. This keeps the prompt length bounded while preserving sufficient context for accurate updates.

Intelligent Memory Segmentation for Event-intertwined Sessions. In memory extraction, input data streams (particularly conversational ones) often exhibit low information density and interleaved topic discussions (e.g., shifting from topic A to B, then back to A) [20]. Traditional methods are suboptimal: message-level storage creates fragmented memories, session-level storage introduces excessive noise, and sequential topic-level approaches fail to merge non-contiguous segments [43, 61]. To illustrate, consider the example where a conversation on recursion (topic A) is interrupted

Algorithm 1: EUA: Patch-based Entity Update w/o LLM

Input: Entity schema $\mathcal{S}$; old entity E^{old}; field-wise patches $\{p_f\}$
Output: Updated entity E^{new}

$E^{new} \leftarrow E^{old}$;
foreach *field* f *in* $\mathcal{S}$ **do**
  $(s, r) \leftarrow \textsc{ParsePatch}(p_f)$; // s=SEARCH, r=REPLACE
  if $s = \emptyset$ **then**
    continue
  $(i, j) \leftarrow \textsc{BestApproxSpan}(E^{old}[f], s)$; // min edit distance
  $E^{new}[f] \leftarrow E^{old}[f]_{[0:i]} \,\|\, r \,\|\, E^{old}[f]_{[j:]}$;
return E^{new};

by a game algorithm discussion (topic B). For a query about recursion, fragmented retrieval (A1, B, A2) would yield incomplete answers, while session-level consolidation would introduce noise from B, leading to factually inaccurate outputs. In VikingMem, we introduce a novel Intelligent Memory Segmentation Strategy to address these challenges. This strategy leverages an LLM to execute a two-phase process. The first phase, semantic saliency filtering, identifies and isolates only the most meaningful segments from the raw dialogue. This effectively prunes conversational filler, such as initial greetings, that are not worth storing—a significant advantage over traditional RAG chunking strategies. This ensures only high-value information is processed, reducing both noise and storage overhead. In the second phase, event-centric partitioning, the LLM determines the precise start and end positions for each coherent topic, outputting them as a list of tuples, a format that is not only precise but also highly token-efficient. This coordinate-based mapping allows for the consolidation of semantically related but non-contiguous dialogue segments. This mechanism enables the formation of more complete and cohesive memories by consolidating scattered content from the same topic. For the illustrated query, our method accurately merges A1 and A2 into a unified memory while excluding B's noise, thereby producing a fully correct and comprehensive response that *encompasses the recursion definition, base case warnings, merge sort application, and efficiency analysis without distortions.*

3.2 Memory Management

The memory management module, as illustrated in Figure 2, receives extracted memories and processes them for efficient storage and updating. This module encompasses three core functionalities: (1) event memory management, (2) entity memory management, and (3) keyword graph updates. We next present event handling, entity updates, time-based compression, and keyword graph construction in detail.

Event Memory Management. Upon extracting new memories, the event update process begins with deduplication to ensure efficiency and accuracy [13]. Specifically, within the same session, highly similar memories are identified and merged to eliminate redundancy by using LLM-based resolution [45].

Entity Memory Update by Novel Operators. As outlined in Section § 2.2.1, entity memories are aggregated from related event memories via a suite of operators implemented in VikingMem,

which we abstracted from practical applications. These operators are broadly categorized into statistical and LLM-based.

The statistical operators (`SUM`, `COUNT`, `AVG`, `MAX`) operate without invoking large language models, focusing on straightforward computations—such as calculating a user's accuracy rate in educational exercises or tracking behavior patterns. This approach addresses the known limitations of LLMs in arithmetic tasks [63], thereby reducing token usage while ensuring greater precision. In contrast, the LLM-based operators handle complex synthesis. The `LLM_MERGE` operator leverages an LLM to perform incremental updates, handling tasks like content deduplication and resolving conflicts between new and existing information. The `TIME_COMPRESS` operator provides a mechanism for long-term memory lifecycle management, implementing our "progressive consolidation" design. Instead of treating memories as a flat log, it groups related events into topic-centric timelines and orders them chronologically, which enables a "lazy merge" strategy. Recent memories are retained with high fidelity, whereas older events in an inactive timeline are synthesized only after they age into a predefined temporal window (e.g., forming a weekly or monthly summary), emulating human memory mechanisms [39]. After such synthesis, the system stores the resulting higher-level summary and assigns a Time-To-Live (TTL) to the underlying fine-grained events. Once the summary has preserved their salient information, expired low-level events can be safely pruned unless subsequent recalls reinforce that timeline. In this way, `TIME_COMPRESS` balances contextual richness with storage and retrieval efficiency.

Keyword Graph. VikingMem's default memory retrieval employs a dense-sparse hybrid approach [62], yet it inadequately addresses queries such as "Do you remember my nickname?" This limitation arises as the semantic similarity between the direct query and the indirect memory is very low. Drawing from [23], we construct a keyword graph where each keyword's embedding is averaged from the embeddings of memory segments containing it, better capturing semantic features, with keywords connected to associated memory.

3.3 Memory Retrieve

Within the workflow outlined in Figure 2, the memory retrieval module takes user queries as input and selects appropriate memories (with time-weighting) from the organized memory repository mentioned above.

Multi-path Recall with Time-Decay and Business Weights. In VikingMem, memory recall is divided into two paths to enhance retrieval effectiveness. The primary path performs hybrid retrieval directly over the memory store by combining dense semantic matching and sparse lexical matching for robust results [62]. Beyond the original retrieval score, this path incorporates two explicit priors: *recency* via a time-decay score and *business importance* via a business-specific score. Specifically, for each candidate memory we compute a final score S_{final} as a weighted combination of the normalized original retrieval score S_{origin}, the temporal score S_{time}, and the business score S_{busi} (all normalized to $[0, 1]$): $S_{\text{final}} = (1 - w_{\text{time}} - w_{\text{busi}}) \cdot S_{\text{origin}} + w_{\text{time}} \cdot S_{\text{time}} + w_{\text{busi}} \cdot S_{\text{busi}}$, where $w_{\text{time}}, w_{\text{busi}} \in [0, 1]$ are configurable and satisfy $w_{\text{time}} + w_{\text{busi}} \leq 1$. The temporal score S_{time} is computed from the memory age with a user-configurable freshness tolerance window (e.g., 1 day) where recent memories receive $S_{\text{time}} = 1$; for memories older than this window, S_{time} decays exponentially following a fast-then-slow curve. For business importance, we support two complementary definitions of S_{busi}: (i) *type-level weighting*, where users assign different importance weights to different event-memory types to prioritize critical categories during retrieval; and (ii) *instance-level weighting*, where a dedicated weight field is included in the memory schema and extracted together with the event to represent the importance of a specific memory instance within the same type, which is then used to promote high-value instances at recall time. Meanwhile, the auxiliary path leverages a keyword graph to retrieve complementary memories, addressing scenarios where standard hybrid search may fall short [23]. Through empirical evaluations, we observed that simply merging results from both paths yields inferior performance compared to independently ranking each path's outputs, assigning distinct quotas (with a larger allocation to the primary path), and then merging the reranked selections, which leads to a more balanced and higher-quality final set of retrieved memories.

Multi-vector Rerank. In memory retrieval scenarios, end-to-end latency is subject to stringent requirements, with reranking operations typically needing to achieve p99 latencies in the order of hundreds of milliseconds. Certain cross-encoder-based reranking algorithms, such as mGTE [64], often result in p99 latencies in the order of seconds, especially when the candidate memory set is relatively large, rendering them unsuitable for real-time applications. To address this, we adopt a reranking strategy inspired by ColBERT [26], which enables efficient and effective memory search through contextualized late interaction. Specifically, we precompute and store ColBERT vectors for memories during the extraction step, applying a series of compression techniques including quantization [15, 27], and token-merge operations [35]. These optimizations ensure that storage overhead remains comparable to that of dense vectors, thereby facilitating efficient reranking with minimal additional storage costs.

4 APPLICATIONS AND EXAMPLE USE CASES

4.1 Downstream Application Scenarios

As introduced in § 1, VikingMem is designed to support a broad range of stateful LLM applications. We summarize our primary deployment domains into five representative categories. These capabilities can be experienced via our commercial services [52].

Social & Companionship. Personalized chatbots [24, 57] use VikingMem to capture salient episodic memories and user preferences, enabling more contextual and empathetic responses.

Search & Recommendation. Search and recommendation systems [1, 33] use VikingMem to consolidate cross-domain activity traces into evolving user profiles for long-term personalization.

Efficiency & Collaboration. Meeting assistants and note-taking tools [5] use VikingMem to extract key points and actionable todos from unstructured streams and consolidate them into periodic reports for better efficiency.

Education. Personalized learning systems [8, 50] use VikingMem to track fine-grained mastery of knowledge points and support adaptive exercises.

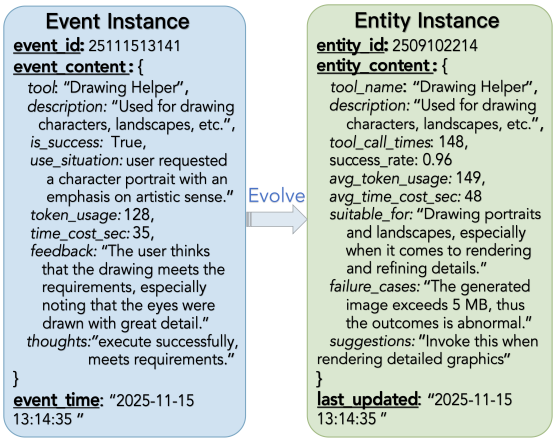

Figure 5: Use case for Agent Memory

Agent Memory. Autonomous agents use VikingMem to capture execution trajectories and tool usage logs [56, 59], progressively refining reusable tool experience and standard operating procedures (SOPs) for future tasks.

4.2 Example Event-Entity Instance

To provide a concrete understanding of how our abstract event-entity model is applied in practice, we focus on one representative application scenario in greater detail. Specifically, we use Agent Memory to illustrate how raw interaction streams are transformed into structured event instances and further consolidated into persistent entities. More comprehensive and complete use case examples can be found in [12].

Agent Tool Memory. One of our primary customers is an internal Agent platform at ByteDance, which aims to leverage memory capabilities to enhance the reliability and efficiency of autonomous tasks. As illustrated in Figure 5, the platform utilizes VikingMem to capture historical tool usage logs and transform them into reusable operational knowledge. Since full execution traces are often verbose and noisy, the system selectively extracts each salient tool invocation as a discrete event instance, recording key information such as the tool used, the usage situation, the task goal, and its success status. These compact event instances preserve actionable signals from past executions while avoiding the redundancy of raw logs. They then evolve the persistent entity instance for that tool, which serves as a living, long-term profile rather than a static record. As new events are ingested, the entity is continuously refined to accumulate qualitative "lessons-gleaned" insights, including what the tool is suitable for, documented failure cases, and actionable suggestions for future invocations. This allows the agent platform to make more informed decisions, avoid repeated failures, and select the optimal tool for new tasks, thereby illustrating how VikingMem turns raw histories into structured and evolving memory for downstream use.

5 EXPERIMENTAL EVALUATION

In this section, we empirically evaluate our proposed system as well as the competing approaches on effectiveness and efficiency. All methods are implemented in Python. The code and datasets can be found at [11]. The vector database service was deployed on a single node with 32 vCPUs (Intel Xeon Platinum 8582C) and 8GB memory, while the embedding service ran on a single NVIDIA A30 GPU (24GB VRAM) with 16 CPU cores and 64GB memory.

5.1 Experiment Setup

5.1.1 Dataset. We used two of the most widely utilized datasets for evaluating the long-term memory capabilities of large language models to assess the ability of our proposed memory system:
LOCOMO [36]: a high-quality benchmark for evaluating long-term conversational memory in large language model agents. It consists of 10 conversations, each characterized by multiple dialogues with an average of 1000+ messages per conversation and a substantial number of tokens. Each conversation is annotated for the question-answering, which are categorized into different reasoning types, including single-hop, multi-hop, temporal reasoning, and open-domain knowledge.
LongMemEval [57]: a longer benchmark for evaluating interactive memory in chat assistants. Following [45], we use one of the subsets: *LongMemEval_s* for evaluation, with 500 long conversations averaging approximately 115,000 tokens in length. The token length of *LongMemEval_s* is 346× larger than *LOCOMO*. The dataset features timestamped conversational histories and includes hundreds of high-quality questions across different abilities like multi-session reasoning and memory updates.

5.1.2 LLM Settings. Following [45, 54]. We evaluated the *LOCOMO* dataset using `GPT-4.1-mini` and `GPT-4o-mini`, and the *LongMemEval* dataset using `GPT-4o-mini` and `GPT-4o`. In all configurations, the specified model served as both the answer generator and the LLM-as-a-Judge evaluator.

5.1.3 Baselines. We compare VikingMem with eight representative long-term memory baselines covering vector retrieval, graph memory, modular memory, Markdown-first memory, native compacted memory, and brute-force full-context prompting.
Mem0 [7]: Extracts candidate facts from dialogue, updates memory via ADD/UPDATE/DELETE/NOOP operations, and retrieves memories with semantic search.
Mem0-graph [7]: Extends Mem0 with a knowledge graph over extracted entities and relations, enabling graph-based retrieval.
Zep [45]: Builds episodic, semantic, and community memory layers with temporal signals and performs hybrid retrieval with reranking.
RAG [16]: This serves as a standard RAG baseline. For our comparison, dialogue history is segmented by grouping 8 messages from the same session into a single text chunk. This chunking strategy ensures that the token count of each memory unit is comparable to the granular memories managed by the other methods.
Mirix [54]: Uses six specialized memory modules and performs hybrid retrieval across memory types.
Full-Context: A naive *inject-everything* baseline that directly feeds the entire available dialogue history into the model for each query, without any memory construction or retrieval.

Table 1: LLM-as-a-Judge evaluation scores (%, with higher values denoting superior performance) and Search Latency (second) for each question category in the *LOCOMO* and *LongMemEval* dataset. Scores for baselines annotated with a superscript † are sourced from published evaluation results, while the search latencies for other methods in *LOCOMO* and *LongMemEval* were measured from our own experiments. Best results are in bold; second-best results are with underline. In *LongMemEval*, "SSU" stands for "single-session-user", "MS" denotes "multi-session", "SSP" represents "single-session-preference", "TR" indicates "temporal-reasoning", "KU" refers to "knowledge-update", and "SSA" means "single-session-assistant".

LOCOMO

LLM Model	Methods	LLM Judge Score					Search Latency	
		Single Hop	Multi-Hop	Open Domain	Temporal	Overall	p50	p95
GPT-4o-mini	Mem0†	72.93	67.13	51.15	55.51	66.88	**0.15**	**0.20**
	Mem0-graph†	75.71	65.71	47.19	58.13	68.44	0.48	0.66
	Zep†	74.11	66.04	67.71	79.79	75.14	0.42	0.63
	RAG	65.16	50.35	48.96	59.50	60.26	0.22	0.43
	Full-Context	78.00	74.82	59.38	**83.80**	77.47	/	/
	Claude	63.50	55.67	50.00	47.35	57.86	18.90	25.42
	Openclaw	17.60	13.48	30.21	13.40	16.75	17.19	26.26
	Mirix	79.08	76.01	66.67	80.86	78.66	10.90	25.76
	VikingMem	**94.89**	**81.91**	**78.12**	82.24	**88.83**	0.20	0.39
GPT-4.1-mini	Mem0†	62.41	57.32	44.79	66.47	62.47	**0.15**	**0.20**
	Mem0-graph	74.44	68.44	51.04	54.52	67.73	0.48	0.66
	Zep†	79.43	69.16	73.96	83.33	79.09	0.42	0.63
	RAG	69.56	56.74	57.29	64.17	65.32	0.27	0.91
	Full-Context	88.70	77.30	72.92	**92.21**	86.36	/	/
	Claude	72.29	68.44	56.25	55.45	67.08	10.81	18.62
	Openclaw	26.52	18.09	26.04	21.81	23.96	13.73	19.26
	Mirix†	85.11	83.70	65.62	88.39	85.38	10.90	25.76
	VikingMem	**93.46**	**85.89**	**79.79**	88.16	**90.12**	0.20	0.39

LongMemEval

LLM Model	Methods	LLM Judge Score							Search Latency	
		SSU	MS	SSP	TR	KU	SSA	Overall	p50	p95
GPT-4o-mini	Mem0	68.57	41.35	20.00	37.59	66.67	5.36	42.80	0.40	1.00
	Mem0-graph	72.86	38.35	16.67	34.59	60.26	35.71	44.00	1.10	2.09
	Zep†	**92.90**	47.40	**53.30**	54.10	74.40	75.00	63.21	3.80	5.45
	Full-Context	81.43	40.60	30.00	36.84	**76.92**	82.14	55.00	/	/
	VikingMem	90.00	**53.31**	46.67	**55.89**	71.23	**96.43**	**66.36**	0.25	0.89
GPT-4o	Mem0	80.00	50.38	26.67	42.11	74.36	10.71	50.20	0.40	1.00
	Mem0-graph	84.29	48.87	30.00	41.35	75.64	48.21	54.80	1.10	2.09
	Zep†	**92.90**	57.90	**56.70**	62.40	**83.30**	80.40	70.40	3.80	5.45
	Full-Context	81.43	44.36	20.00	45.11	78.21	94.64	59.20	/	/
	VikingMem	**92.90**	**69.92**	**56.70**	**66.92**	76.92	**98.21**	**75.80**	0.25	0.89

OpenClaw [40]: A Markdown-first memory baseline that incrementally maintains persistent memories as text files and retrieves relevant entries at inference time.

Claude Native Memory: A baseline built on Claude Code's native memory mechanism, where interaction history is periodically compacted into concise memory notes for later reuse, without explicit schema-driven retrieval.

5.1.4 Evaluation Metrics. For the evaluation of our VikingMem, following [7, 45, 54], we adopt three widely used metrics: LLM-as-a-Judge (LLM Judge Score), F1-score and search latency (seconds), which are employed to evaluate the efficiency and effectiveness of our method. To mitigate randomness, we run each experiment three times and report the average result for the final results.

5.1.5 Implementation Details. To reflect the latest performance of each method, we used API keys to call the memory interface of each baseline to complete our evaluation (except for the RAG method). Since the memory ingestion cost is high, we only imported the data once per system. For each query, we repeated the answer generation and evaluation three times and reported the average results. Furthermore, to assess the practical throughput of each system, we enforced a 24-hour time limit for the entire end-to-end process of memory extraction and answering. Consequently, baselines that failed to complete the task within this timeframe were

considered to have timed out, and their results are not reported. Additionally, since both datasets consist primarily of fact-based queries, where time-weighting offers little benefit, we disabled this feature by default during our experiments. For LLM-based answer generation and evaluation, all compared methods were tested under the same prompt setup. Our prompts are consistent with the evaluation protocol used in prior work [45], ensuring a fair comparison. Meanwhile, for VikingMem, all results were obtained on the production implementation backed by VikingDB: event memories, entity states, and auxiliary keyword-linked indices were stored in VikingDB, while the reported retrieval latency includes candidate generation from VikingDB together with VikingMem-side score fusion and reranking.

5.2 End-to-end Evaluation

We evaluate VikingMem against state-of-the-art baselines along two dimensions of a memory system: effectiveness, i.e., providing accurate and relevant information, and efficiency, i.e., supporting real-time retrieval. On the public *LOCOMO* and *LongMemEval* benchmarks, we report LLM-Judge scores for effectiveness and p50/p95 search latency for efficiency, aligning with the design objectives in § 2.3. For each benchmark, we follow the judge settings used in prior work [7, 30, 45] to preserve comparability. Since LLM-Judge scores depend on the evaluation setup, we report them separately for each benchmark and judge model, and use two judge models to test whether relative rankings remain stable. In contrast, the efficiency metrics are measured directly from the memory system and are independent of the LLM judge.

5.2.1 Effectiveness. As shown in Table 1, VikingMem consistently achieves the best overall results on both benchmarks under their respective evaluation settings, validating its effectiveness in long-term memory tasks. Our key observations are as follows. (1) VikingMem obtains the highest overall LLM-Judge scores across all baselines. On *LOCOMO*, it scores 88.83% and 90.12% with GPT-4o-mini and GPT-4.1-mini, respectively, suggesting that it not only stores facts but also organizes them for cross-session reasoning. Its lead also holds on *LongMemEval*, where it achieves the best overall scores, 66.36% with GPT-4o-mini and 75.80% with GPT-4o. Although the absolute score scale varies with the judge model, the relative advantage of VikingMem remains stable in our results. (2) Mirix, with six specialized memory modules, is a strong second-best performer, achieving 85.38% overall on *LOCOMO*. However, its complex multi-module architecture appears less streamlined than VikingMem's integrated design, leading to a consistent performance gap. (3) Among stronger baselines, Zep and Full-Context are competitive in some categories but do not consistently match VikingMem. Zep's layered memory design is effective on several *LongMemEval* categories, while Full-Context remains a strong brute-force baseline on *LOCOMO*, showing that injecting all history can help when the context fits within the usable window. However, both fall short in overall accuracy, indicating that explicit memory organization and retrieval are more robust than generic graph memory or raw context stuffing. This advantage is especially clear for queries requiring precise factual details and cross-turn evidence, where filtering irrelevant long-context information helps surface useful evidence more reliably. Meanwhile, this trade-off remains practically important even as context windows increase: current LLMs are still susceptible to interference from irrelevant context in long inputs, which can hurt end-to-end accuracy. Even if future models become strong enough to fully overcome this issue, naively injecting the full history would still incur substantially higher token cost in production settings. (4) Other baselines, including Mem0, Mem0-graph, RAG, Claude Code Native Memory, and OpenClaw, perform substantially worse in most settings. Mem0/Mem0-graph rely on brief memory units and struggle with complex multi-hop or temporal reasoning. RAG retrieves raw conversational chunks, which often introduce contextual noise. OpenClaw performs worst overall, likely because its Markdown-based memory uses coarse-grained dialogue records rather than compact structured memories. Claude Code Native Memory is stronger than OpenClaw but still lags behind ours, likely because its memory access is largely inference-time and agent-driven, which may not always surface all relevant memory items.

5.2.2 Efficiency. Beyond accuracy, low latency is critical for user experience. Our evaluation, focusing on the order of magnitude of retrieval delays, confirms VikingMem's practical advantages. With a p50 latency around 0.20-0.25s and a p95 latency also remaining low at (e.g., 0.89s for *LongMemEval*), VikingMem is not only highly responsive on average but also maintains stability under load. It operates on the same low-latency scale as lightweight methods like Mem0 and Zep, demonstrating that its sophisticated capabilities do not compromise operational speed. In stark contrast, other high-accuracy systems make significant efficiency trade-offs. For instance, Mirix exhibits p50 latencies (10.90s) that are orders of magnitude higher, with their p95 latencies climbing even further to (e.g., 25.76s), rendering it impractical for interactive applications. Furthermore, this method failed to complete the *LongMemEval* evaluation within the 24-hour time limit.

5.3 Effect of One-pass Extraction and EUA

To quantify the benefits of our one-pass memory extraction paradigm and EUA algorithm described in § 3.1, we conduct a comparative analysis on the *LOCOMO* dataset, for which we configured three distinct memory types: one event memory and two entity memories (a user profile and a topic-based compressed memory). We benchmark our schema-driven method against the conventional multi-pass baseline, a paradigm representative of prior systems [38, 54] where each memory type is extracted in a separate LLM inference call.

We evaluate both approaches on two critical dimensions: efficiency and effectiveness. We use Extraction Cost ($) to measure the efficiency by calculating the total monetary cost required to process and extract all defined memories from all dialogue sessions as well as the Extract Time (s). Using LLM Judge Score to assess effectiveness of the extract memories. Note that the multi-pass baseline is implemented with parallel LLM calls across memory types for a fair latency comparison.

As shown in Table 2, the results validate both our one-pass extraction paradigm and the proposed EUA algorithm. First, compared with the conventional multi-pass baseline (*Multiple Prompts*), one-pass extraction substantially improves cost efficiency while preserving memory quality: the extraction cost drops from $0.35

Table 2: Effect of One-pass Extraction and EUA

Techniques	Cost($)	Time(s)	Score
Multiple Prompts	0.35	11.02	88.86
One-pass (w/ EUA)	0.07	7.42	88.77
One-pass (w/o EUA)	0.11	12.32	88.83

Table 3: Storage Efficiency on LongMemEval

Method	Storage (Tokens)	Score
Naive RAG [16]	100% (Baseline)	63.81
VikingMem	**16.82% (83.18% ↓)**	**75.80**

Table 4: Ablation study on LOCOMO using GPT-4o-mini. We report the LLM Judge Score and the p95 search latency impact of each component. "IMSM" stands for "intelligent memory segmentation method for event-intertwined sessions".

Removed Component	LLM Judge Score	Search Latency
/	88.83	-
Multi-Vector Rerank	85.19	-6.8ms
Entity Memory	86.93	≈ 0
IMSM	83.51	≈ 0
Keyword Graph	86.92	-25.8ms

to $0.11 (one-pass w/o EUA), i.e., a 3.2× reduction, with an almost unchanged LLM-judge score. Second, EUA further reduces the end-to-end latency and cost of one-pass extraction, decreasing the wall-clock time from 12.32 s to 7.42 s and the cost from $0.11 to $0.07, while maintaining comparable quality.

5.4 Storage Efficiency and Retention Analysis

We evaluate retention efficiency on the *LongMemEval* dataset by comparing VikingMem with the Naive RAG baseline [16]. We report storage consumption, measured by the number of tokens required to persist the memory state, and retrieval accuracy, measured by the LLM-Judge score.

Table 3 shows that VikingMem substantially reduces storage while preserving retrieval quality. Unlike Naive RAG, which stores all raw tokens, VikingMem retains only extracted events and entity snapshots, reducing the storage footprint to 16.82% of the original size, i.e., an 83.18% reduction or approximately a 6:1 compression ratio. Despite this reduction, VikingMem achieves a higher LLM-Judge Score than Naive RAG (75.80 vs. 63.81), indicating that selective retention effectively filters low-value content while preserving task-relevant memory.

5.5 Ablation Studies

We conduct an ablation study on *LOCOMO* using GPT-4o-mini to evaluate the contribution of each major component in VikingMem. Table 4 reports the impact of removing each module on effectiveness (LLM Judge Score) and efficiency (p95 search latency). All components contribute positively, with IMSM and reranking providing the largest gains.

5.5.1 Impact of Rerank. The Rerank module plays a crucial role in refining search results to enhance response accuracy. Removing it caused a 3.64-point drop in the LLM Judge Score, confirming its importance. Meanwhile, the late-interaction mechanism rerank provides a substantial quality boost at a modest cost of only 6.8ms in p95 search latency, proving it to be an effective and efficient component for high-precision memory retrieval.

5.5.2 Intelligent memory segmentation method. As shown in Table 4. The special design of IMSM is the most impactful component for memory quality. Its removal triggered the most severe performance drop, causing the LLM Judge Score to plummet by 5.32 points. This significant contribution stems from its ability to solve the core challenge of noisy and interleaved dialogues by extracting only semantically rich segments and merging related but non-contiguous content into a single, coherent memory. As this is an offline process performed during memory ingestion, it provides this substantial boost to effectiveness without incurring any search latency (≈ 0), making it a highly efficient and foundational element of our system.

5.5.3 Entity Memory. The entity memory enhances response accuracy by leveraging long-term, evolving entity-level context. Its removal caused a 1.9-point drop in the LLM Judge Score, confirming its value in tailoring outputs to individual users.

5.5.4 Keyword Graph. The Keyword Graph provides an auxiliary recall path, designed to retrieve complementary memories that the primary hybrid search might miss. Its value is confirmed by the 1.91-point drop in the LLM Judge Score upon its removal , an impact comparable to that of the entity memory. However, it also introduces a 25.8ms search latency, making it the most computationally expensive module. .

5.6 Operator Usage Frequency Across Real-World Application Scenarios

In this part, we analyze the usage frequency of different operators across three representative downstream application scenarios: Education, Agent Memory, and Social Companionship. The statistics are derived from entity schema definitions collected from over 100 customer applications in these three scenarios, with each customer using 2.6 schemas on average. For each scenario, we report the percentage of schemas that invoke a given operator. This analysis provides a fine-grained view of how different operators are used in practical applications and helps characterize the operator requirements of different scenarios.

As shown in Table 5, different scenarios exhibit distinct operator usage patterns. Education and Agent Memory frequently use aggregation-oriented operators, especially `MAX`, `AVG`, and `LLM_M`, indicating that these scenarios often require summarizing and consolidating accumulated user states, historical records, or execution statistics. In contrast, Social Companionship relies less on numerical aggregation, but shows strong demand for semantic merging and temporal compression, with `LLM_M` and `TIME_C` used in 100% and 72% of the schemas, respectively. Overall, these results show

Table 5: Operator usage frequency across different application scenarios. Each percentage denotes the proportion of entity schemas in the corresponding scenario that use the operator. `LLM_M` and `TIME_C` stand for `LLM_MERGE` and `TIME_COMPRESS`, respectively. Social Comp. denotes Social Companionship.

Scenario	SUM	MAX	AVG	COUNT	LLM_M	TIME_C
Education	85%	90%	96%	88%	91%	26%
Agent Memory	73%	96%	98%	64%	98%	42%
Social Comp.	24%	16%	32%	11%	100%	72%

Table 6: F1-score (%) on LOCOMO. SH: Single-Hop, MH: Multi-Hop, OD: Open-Domain, Temp: Temporal.

Methods	SH	MH	OD	Temp	Overall
Mem0	47.65	38.72	28.64	48.93	45.09
Mem0-graph	49.27	38.09	24.32	51.55	46.14
Zep	49.56	35.74	41.37	52.04	47.03
Full-Context	55.64	43.52	40.43	**58.32**	53.03
Claude	41.23	34.23	28.06	46.32	40.19
Openclaw	20.12	11.36	10.04	21.32	18.14
VikingMem	**59.59**	**44.52**	**43.13**	55.62	**54.98**

that the proposed operators are widely used in real downstream applications, while different scenarios require different operator combinations, which justifies our operator-based design.

5.7 Evaluation with F1-Score

As LLM-as-a-judge scores can be sensitive to the choice of judge model and evaluation prompt, we additionally report *F1-score* [19] to provide a more robust and transparent evaluation. Specifically, for all methods, we use the same answer generation model (`gpt-4o-mini`) to produce the final answer based on the retrieved memory, and then compute the token-level F1-score against the ground-truth answer on *LOCOMO*. The F1-score is defined as the harmonic mean of token-level precision and recall: As shown in Table 6, VikingMem achieves the best overall F1-score of 54.98%, outperforming all baselines, including Full-Context (53.03%). It also performs best on Single-Hop, Multi-Hop, and Open-Domain questions, while remaining competitive on Temporal questions. These results are consistent with the conclusions drawn from the LLM judge evaluation, further supporting the effectiveness of VikingMem.

6 RELATED WORK

6.1 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) integrates external knowledge retrieval to mitigate LLM hallucinations and enhance factual accuracy without retraining [16, 28, 65]. Modern variants extend this paradigm via modular query rewriting [34], graph structures for relational precision [21, 23], agentic iterative reasoning [49], and multimodal inputs [4]. These methods have shown great gains on benchmarks like HotpotQA [60]. However, traditional RAG remains constrained in long-term memory management. Its reliance on static, chunk-based retrieval struggles with low-information-density memory streams, where critical user details are scattered across non-contiguous segments, often leading to noisy or incomplete extractions [53].

6.2 Memory-centric Architecture and Systems

To bypass finite context windows, recent memory-centric architectures implement specialized storage layers to enable coherent, long-term LLM interactions [24, 41]. Representative systems like Mem0 [7] and MemOS [30] dynamically extract salient facts or manage memory lifecycles, while Zep [45] and MIRIX [54] utilize hierarchical structures and specialized modules to preserve granular contextual relevance.

Despite these advances, existing designs fall short of a general-purpose, task-agnostic memory substrate. Most are strictly optimized for companion-style dialogue flows, failing to natively support the procedural, auditable traces required by autonomous agents or the fine-grained preference trajectories in education and recommendation domains [7, 38]. Furthermore, heavy operations such as dynamic knowledge-graph maintenance under continuous memory influx introduce substantial build-time latency and resource overhead, conflicting with the strict cost and performance constraints of real-world production deployments [45, 54].

7 CONCLUSION

In this paper, we addressed the fundamental data management challenge facing the next generation of LLM applications, which are shifting from stateless queries to long-term, stateful interactions. We proposed the Memory Base, a new general framework designed specifically to manage persistent state. This paradigm moves beyond static repositories, providing inherent capabilities for selective memory extraction, continuous stateful evolution, and a generalizable design. We presented VikingMem, an end-to-end Memory Base Management System that operationalizes this framework. The core of VikingMem's design is its interconnected event and entity abstractions, which use intelligent segmentation to capture high-value, episodic events and a flexible, operator-based mechanism to progressively evolve persistent entity states. Extensive evaluations on public benchmarks confirmed that VikingMem achieves SOTA retrieval effectiveness, low latency, and low extraction cost, validating VikingMem as a robust, efficient, and scalable solution for enabling long-term contextual intelligence in LLM applications.

ACKNOWLEDGMENTS

The authors are deeply grateful to colleagues from the `Viking` team for their foundational contributions and insightful discussions. This work was supported in part by the NSFC (Grants No. 62502434, U23A20296, 62502426), Zhejiang Province's "Jianbing" R&D Project (Grant No. 2025C01195), and Yongjiang Talent Introduction Programme (Grant No. 2022A-237-G). Xiangyu Ke is the corresponding author of this work.

REFERENCES

[1] Arkadeep Acharya, Brijraj Singh, and Naoyuki Onoe. 2023. LLM Based Generation of Item-Description for Recommendation System. In *Proceedings of the 17th ACM Conference on Recommender Systems*. Association for Computing Machinery, New York, NY, USA, 1204–1207. doi:10.1145/3604915.3610647

[2] Federico Barbero, Andrea Banino, Steven Kapturowski, Dharshan Kumaran, João Madeira Araújo, Oleksandr Vitvitskyi, Razvan Pascanu, and Petar Veličković. 2024. Transformers need glasses! information over-squashing in language tasks. *Advances in Neural Information Processing Systems* 37 (2024), 98111–98142.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Davide Caffagni, Federico Cocchi, Nicholas Moratelli, Sara Sarto, Marcella Cornia, Lorenzo Baraldi, and Rita Cucchiara. 2024. Wiki-LLaVA: Hierarchical Retrieval-Augmented Generation for Multimodal LLMs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, Seattle, WA, USA, 1818–1826. doi:10.1109/CVPRW63382.2024.00188

[5] Ningyuan Chen. 2023. Enterprises Build Sustainable Innovation Loop: Take ByteDance Launching Lark as an Example. In *Digitalization and Management Innovation*, A. J. Tall'on-Ballesteros and P. Santana-Morales (Eds.). IOS Press, Amsterdam, The Netherlands, 126–132. doi:10.3233/FAIA230013

[6] Sibei Chen, Ju Fan, Bin Wu, Nan Tang, Chao Deng, Pengyi Wang, Ye Li, Jian Tan, Feifei Li, Jingren Zhou, et al. 2025. Automatic database configuration debugging using retrieval-augmented language models. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–27.

[7] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. In *ECAI 2025 (Frontiers in Artificial Intelligence and Applications, Vol. 413)*. IOS Press, Amsterdam, The Netherlands, 2993–3000. doi:10.3233/FAIA251160

[8] Zhendong Chu, Shen Wang, Jian Xie, Tinghui Zhu, Yibo Yan, Jingheng Ye, Aoxiao Zhong, Xuming Hu, Jing Liang, Philip S. Yu, and Qingsong Wen. 2025. LLM Agents for Education: Advances and Applications. In *Findings of the Association for Computational Linguistics: EMNLP 2025*. Association for Computational Linguistics, Suzhou, China, 13782–13810. doi:10.18653/v1/2025.findings-emnlp.743

[9] Dong Deng, Guoliang Li, Jianhua Feng, and Wen-Syan Li. 2013. Top-k String Similarity Search with Edit-Distance Constraints. In *2013 IEEE 29th International Conference on Data Engineering*. IEEE Computer Society, Brisbane, QLD, Australia, 925–936. doi:10.1109/ICDE.2013.6544886

[10] Weizhi Fei, Xueyan Niu, Pingyi Zhou, Lu Hou, Bo Bai, Lei Deng, and Wei Han. 2024. Extending Context Window of Large Language Models via Semantic Compression. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, Bangkok, Thailand, 5169–5181. doi:10.18653/v1/2024.findings-acl.306

[11] Jiajie Fu, Junwen Chen, Mengzhao Wang, Aoxiang He, Maojia Sheng, Xiangyu Ke, Yifan Zhu, and Yunjun Gao. 2025. Evaluation Code for VikingMem. https://github.com/BytedanceFu/VikingMem

[12] Jiajie Fu, Junwen Chen, Mengzhao Wang, Aoxiang He, Maojia Sheng, Xiangyu Ke, Yifan Zhu, and Yunjun Gao. 2025. Use cases for VikingMem. https://github.com/FuJiaJie123/VikingMem

[13] Jiajie Fu, Haitong Tang, Arijit Khan, Sharad Mehrotra, Xiangyu Ke, and Yunjun Gao. 2025. In-context clustering-based entity resolution with large language models: A design space exploration. *Proceedings of the ACM on Management of Data* 3, 4 (2025), 1–28.

[14] Aoran Gan, Hao Yu, Kai Zhang, Qi Liu, Wenyu Yan, Zhenya Huang, Shiwei Tong, and Guoping Hu. 2025. Retrieval Augmented Generation Evaluation in the Era of Large Language Models: A Comprehensive Survey. arXiv:2504.14891 [cs.CL] doi:10.48550/arXiv.2504.14891

[15] Jianyang Gao and Cheng Long. 2024. Rabitq: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[16] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL] https://arxiv.org/abs/2312.10997

[17] Gemini Team, Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-Baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. *Gemini 1.5: Unlocking Multimodal Understanding Across Millions of Tokens of Context*. arXiv:2403.05530 [cs.CL] doi:10.48550/arXiv.2403.05530

[18] Julie Gonneaud, Géraldine Rauchs, Mathilde Groussard, Brigitte Landeau, Florence Mezenge, Vincent de La Sayette, Francis Eustache, and Béatrice Desgranges. 2014. How do we process event-based and time-based intentions in the brain? An fMRI study of prospective memory in healthy individuals. *Human brain mapping* 35, 7 (2014), 3066–3082.

[19] Gaurav Goswami. 2025. Dissecting the Metrics: How Different Evaluation Approaches Yield Diverse Results for Conversational AI. TechRxiv preprint. doi:10.36227/techrxiv.174016591.19306108/v1

[20] Qinyu Han, Zhihao Yang, Hongfei Lin, and Tian Qin. 2024. Let topic flow: A unified topic-guided segment-wise dialogue summarization framework. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 32 (2024), 2021–2032.

[21] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Advances in Neural Information Processing Systems* 37 (2024), 132876–132907.

[22] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems* 43, 2 (2025), 1–55.

[23] Yiqian Huang, Shiqi Zhang, and Xiaokui Xiao. 2025. KET-RAG: A Cost-Efficient Multi-Granular Indexing Framework for Graph-RAG. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Toronto, ON, Canada) *(KDD '25)*. Association for Computing Machinery, New York, NY, USA, 1003–1012. doi:10.1145/3711896.3737012

[24] Brennan Jones, Yan Xu, Qisheng Li, and Stefan Scherer. 2024. Designing a Proactive Context-Aware AI Chatbot for People's Long-Term Goals. In *Extended Abstracts of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '24)*. Association for Computing Machinery, New York, NY, USA, Article 104, 7 pages. doi:10.1145/3613905.3650912

[25] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 6769–6781. doi:10.18653/v1/2020.emnlp-main.550

[26] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 39–48. doi:10.1145/3397271.3401075

[27] Andrey Kuzmin, Mart Van Baalen, Yuwei Ren, Markus Nagel, Jorn Peters, and Tijmen Blankevoort. 2022. Fp8 quantization: The power of the exponent. *Advances in Neural Information Processing Systems* 35 (2022), 14651–14662.

[28] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[29] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems* 37 (2024), 22947–22970.

[30] Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, Junpeng Ren, Zehao Lin, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhiqiang Yin, Qingchen Yu, Bo Tang, Hongkang Yang, Zhi-Qin John Xu, and Feiyu Xiong. 2025. *MemOS: An Operating System for Memory-Augmented Generation (MAG) in Large Language Models*. arXiv:2505.22101 [cs.CL] doi:10.48550/arXiv.2505.22101

[31] Patricia W Linville. 1987. Self-complexity as a cognitive buffer against stress-related illness and depression. *Journal of personality and social psychology* 52, 4 (1987), 663.

[32] Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, Yuanxing Zhang, Zhuo Chen, Hangyu Guo, Shilong Li, Ziqiang Liu, Yong Shan, Yifan Song, Jiayi Tian, Wenhao Wu, Zhejian Zhou, Ruijie Zhu, Junlan Feng, Yang Gao, Shizhu He, Zhoujun Li, Tianyu Liu, Fanyu Meng, Wenbo Su, Yingshui Tan, Zili Wang, Jian Yang, Wei Ye, Bo Zheng, Wangchunshu Zhou, Wenhao Huang, Sujian Li, and Zhaoxiang Zhang. 2025. *A Comprehensive Survey on Long Context Language Modeling*. arXiv:2503.17407 [cs.CL] doi:10.48550/arXiv.2503.17407

[33] Hanjia Lyu, Song Jiang, Hanqing Zeng, Yinglong Xia, Qifan Wang, Si Zhang, Ren Chen, Chris Leung, Jiajie Tang, and Jiebo Luo. 2024. LLM-Rec: Personalized Recommendation via Prompting Large Language Models. In *Findings of the Association for Computational Linguistics: NAACL 2024*. Association for Computational Linguistics, Mexico City, Mexico, 583–612. doi:10.18653/v1/2024.findings-naacl.39

[34] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query Rewriting in Retrieval-Augmented Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Singapore, 5303–5315. doi:10.18653/v1/2023.emnlp-main.322

[35] Yubo Ma, Jinsong Li, Yuhang Zang, Xiaobao Wu, Xiaoyi Dong, Pan Zhang, Yuhang Cao, Haodong Duan, Jiaqi Wang, Yixin Cao, and Aixin Sun. 2025. Towards Storage-Efficient Visual Document Retrieval: An Empirical Study on Reducing Patch-Level Embeddings. In *Findings of the Association for Computational Linguistics: ACL 2025*. Association for Computational Linguistics, Vienna, Austria, 19568–19580. doi:10.18653/v1/2025.findings-acl.1003

[36] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating Very Long-Term Conversational Memory of LLM Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 13851–13870. doi:10.18653/v1/2024.acl-long.747

[37] Hazel Markus. 1977. Self-schemata and processing information about the self. *Journal of personality and social psychology* 35, 2 (1977), 63.

[38] Memobase. 2025. Source code of Memobase. https://github.com/memodb-io/memobase.

[39] Mite Mijalkov, Ludvig Storm, Blanca Zufiria-Gerbolés, Dániel Veréb, Zhilei Xu, Anna Canal-Garcia, Jiawei Sun, Yu-Wei Chang, Hang Zhao, Emiliano Gómez-Ruiz, et al. 2025. Computational memory capacity predicts aging and cognitive decline. *Nature communications* 16, 1 (2025), 2748.

[40] Openclaw. 2026. Openclaw AI. https://openclaw.ai/

[41] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. *MemGPT: Towards LLMs as Operating Systems.* arXiv:2310.08560 [cs.AI] doi:10.48550/arXiv.2310.08560

[42] Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. 2025. *KVFlow: Efficient Prefix Caching for Accelerating LLM-Based Multi-Agent Workflows.* arXiv:2507.07400 [cs.DC] doi:10.48550/arXiv.2507.07400

[43] Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Xufang Luo, Hao Cheng, Dongsheng Li, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Jianfeng Gao. 2025. SeCom: On Memory Construction and Retrieval for Personalized Conversational Agents. In *Proceedings of the Thirteenth International Conference on Learning Representations*. OpenReview.net, Singapore, 35 pages. https://openreview.net/forum?id=xKDZAW0He3

[44] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. YaRN: Efficient Context Window Extension of Large Language Models. In *Proceedings of the Twelfth International Conference on Learning Representations*. OpenReview.net, Vienna, Austria, 17 pages. https://openreview.net/forum?id=wHBfxhZu1u

[45] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. *Zep: A Temporal Knowledge Graph Architecture for Agent Memory.* arXiv:2501.13956 [cs.CL] doi:10.48550/arXiv.2501.13956

[46] Henry L Roediger III and Jeffrey D Karpicke. 2006. Test-enhanced learning: Taking memory tests improves long-term retention. *Psychological science* 17, 3 (2006), 249–255.

[47] RooCodeInc. 2026. Roocode Source Code. Retrieved January 11, 2026 from https://github.com/RooCodeInc/Roo-Code

[48] Vit'oria S. Santos and Carina F. Dorneles. 2024. Unveiling the Segmentation Power of LLMs: Zero-Shot Invoice Item Description Analysis. In *Anais do XXXIX Simp'osio Brasileiro de Banco de Dados* (Florian'opolis, SC, Brazil). Sociedade Brasileira de Computaç ao, Porto Alegre, RS, Brazil, 549–561. doi:10.5753/sbbd.2024.240820

[49] Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025. *Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG.* arXiv:2501.09136 [cs.AI] doi:10.48550/arXiv.2501.09136

[50] Jakub Swacha and Michał Gracel. 2025. Retrieval-Augmented Generation (RAG) Chatbots for Education: A Survey of Applications. *Applied Sciences* 15, 8 (2025), 4234.

[51] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. *LLaMA: Open and Efficient Foundation Language Models.* arXiv:2302.13971 [cs.CL] doi:10.48550/arXiv.2302.13971

[52] VikingMem. 2025. VikingMem User Guide. https://docs.byteplus.com/en/docs/VikingDB/Integrating_LLM_with_Memory

[53] Xi Wang, Procheta Sen, Ruizhe Li, and Emine Yilmaz. 2025. Adaptive Retrieval-Augmented Generation for Conversational Systems. In *Findings of the Association for Computational Linguistics: NAACL 2025*. Association for Computational Linguistics, Albuquerque, New Mexico, 491–503. doi:10.18653/v1/2025.findings-naacl.30

[54] Yu Wang and Xi Chen. 2025. *MIRIX: Multi-Agent Memory System for LLM-Based Agents.* arXiv:2507.07957 [cs.AI] doi:10.48550/arXiv.2507.07957

[55] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025. Agent Workflow Memory. In *Proceedings of the 42nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 267)*. PMLR, Vancouver, BC, Canada, 63897–63911. https://proceedings.mlr.press/v267/wang25bx.html

[56] Chunlong Wu and Zhibo Qu. 2025. *Meta-Policy Reflexion: Reusable Reflective Memory and Rule Admissibility for Resource-Efficient LLM Agent.* arXiv:2509.03990 [cs.AI] doi:10.48550/arXiv.2509.03990

[57] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. In *Proceedings of the Thirteenth International Conference on Learning Representations*. OpenReview.net, Singapore, 28 pages. https://openreview.net/forum?id=UBvm2bIyxz

[58] Weiqi Wu, Shen Huang, Yong Jiang, Pengjun Xie, Fei Huang, and Hai Zhao. 2025. Unfolding the Headline: Iterative Self-Questioning for News Retrieval and Timeline Summarization. In *Findings of the Association for Computational Linguistics: NAACL 2025*. Association for Computational Linguistics, Albuquerque, New Mexico, 4385–4398. doi:10.18653/v1/2025.findings-naacl.248

[59] Yunzhong Xiao, Yangmin Li, Hewei Wang, Yunlong Tang, and Zora Zhiruo Wang. 2025. *ToolMem: Enhancing Multimodal Agents with Learnable Tool Capability Memory.* arXiv:2510.06664 [cs.AI] doi:10.48550/arXiv.2510.06664

[60] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 2369–2380. doi:10.18653/v1/D18-1259

[61] Linhao Ye, Zhikai Lei, Jianghao Yin, Qin Chen, Jie Zhou, and Liang He. 2024. Boosting Conversational Question Answering with Fine-Grained Retrieval-Augmentation and Self-Check. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington, DC, USA) *(SIGIR '24)*. Association for Computing Machinery, New York, NY, USA, 2301–2305. doi:10.1145/3626772.3657980

[62] Haoyu Zhang, Jun Liu, Zhenhua Zhu, Shulin Zeng, Maojia Sheng, Tao Yang, Guohao Dai, and Yu Wang. 2024. *Efficient and Effective Retrieval of Dense-Sparse Hybrid Vectors Using Graph-Based Approximate Nearest Neighbor Search.* arXiv:2410.20381 [cs.IR] doi:10.48550/arXiv.2410.20381

[63] Xiang Zhang, Juntai Cao, and Chenyu You. 2024. *Counting Ability of Large Language Models and Impact of Tokenization.* arXiv:2410.19730 [cs.CL] doi:10.48550/arXiv.2410.19730

[64] Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, Meishan Zhang, Wenjie Li, and Min Zhang. 2024. mGTE: Generalized Long-Context Text Representation and Reranking Models for Multilingual Text Retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Association for Computational Linguistics, Miami, FL, USA, 1393–1412. doi:10.18653/v1/2024.emnlp-industry.103

[65] Siyun Zhao, Yuqing Yang, Zilong Wang, Zhiyuan He, Luna K. Qiu, and Lili Qiu. 2024. *Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey on How to Make Your LLMs Use External Data More Wisely.* arXiv:2409.14924 [cs.CL] doi:10.48550/arXiv.2409.14924

[66] Wei Zhou, Yuyang Gao, Xuanhe Zhou, and Guoliang Li. 2025. Cracking SQL Barriers: An LLM-based Dialect Translation System. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–26.