HIPPOCAMPUS: AN EFFICIENT AND SCALABLE MEMORY MODULE FOR AGENTIC AI

Yi Li [1 2] **Lianjie Cao** [1] **Faraz Ahmed** [1] **Puneet Sharma** [1] **Bingzhe Li** [2]

ABSTRACT

Agentic AI systems require persistent memory to store user-specific histories beyond the limited context window of LLMs. Existing memory systems rely on the dense vector databases, knowledge-graph traversal, or hybrids, which incur high retrieval latency and poor storage scalability. We introduce HIPPOCAMPUS, an agentic memory management system that uses compact binary signatures for semantic search and lossless token-ID streams for exact content reconstruction. Its core is a Dynamic Wavelet Matrix (DWM) that compresses and co-indexes both streams to support ultra-fast search in the compressed domain, thus avoiding costly dense-vector or graph computations. For a fixed tokenizer vocabulary, the storage footprint of this design grows linearly with memory size, making it suitable for long-horizon agentic deployments. Empirically, across LoCoMo and LongMemEval, HIPPOCAMPUS achieves end-to-end retrieval latency that is comparable to or lower than the evaluated agentic memory baselines, with 1.1×–31.5× speedups over the evaluated baselines, and reduces per-query token footprint by 1.1×–14.5×, while maintaining competitive task accuracy.

1 INTRODUCTION

Agentic AI represents a transformative shift in how intelligent systems interact with the real world. Unlike traditional software, which executes predefined logic, agentic systems autonomously perceive, plan, act, and adapt over time. Powered by large language models (LLMs) (Vaswani et al., 2017; Touvron et al., 2023; Team et al., 2023; Liu et al., 2024), these agents can decompose the complex tasks, invoke external tools, reflect on their own behavior, and revise strategies without explicit human supervision. Early prototypes such as AutoGPT (Chen et al., 2023) and BabyAGI (Nakajima, 2023) demonstrated that coupling the LLMs with goal-driven loops unlocks emergent capabilities far beyond static prompting. As agentic AI transitions from research novelty to production infrastructure, it is poised to reshape productivity tools, DevOps workflows (Ali & Puri, 2024), and knowledge systems (Zhu et al., 2024).

Memory is a core component of agentic AI systems. While perception and planning enable agents to respond to immediate stimuli, memory allows them to accumulate experience, maintain coherence across interactions, and reason over long temporal horizons. In practice, agentic systems operate within an *observe–plan–act–learn* loop (Srivastava, 2019; Hayes-Roth & Hayes-Roth, 1979), where memory serves as the persistent substrate connecting past observations to future decisions. Without reliable recall, even sophisticated planners, such as those based on ReAct (Yao et al., 2023) or GoalAct (Chen et al., 2025), can lose track of prior actions, repeat failed strategies, or misinterpret context (Li et al., 2023a; Xu et al., 2025). This limitation is exacerbated by the bounded context window of LLMs (Su et al., 2024; Wu et al., 2024b), which restricts the amount of information that can be considered during inference. Empirical studies, including the "*Lost-in-Middle*" effect (Liu et al., 2023a), show that reasoning accuracy degrades sharply as prompts grow longer. As a result, context engineering (Mei et al., 2025; LangChain, 2025) has emerged as a workaround, treating the context window as a scarce computational resource. However, this approach is brittle and labor-intensive.

A dedicated memory management system offers a principled alternative: by externalizing long-term knowledge, it enables agents to retrieve only the most relevant fragments of prior information, e.g., dialogue, tool outputs, or retrieved facts, preserving context for immediate reasoning while maintaining continuity across tasks. Major agentic AI frameworks, e.g., LangChain and CrewAI (Chase, 2022; CrewAI, 2025), already provide memory management systems to enhance agent capabilities, and real-world deployments report improved coherence and personalization when memory is enabled (Li et al., 2023b).

*Equal contribution [1]Network and Distributed Systems Labs (NDSL), Hewlett Packard Enterprise (HPE) Labs, Milpitas, California, USA [2] Department of Computer Science, University of Texas at Dallas, Richardson, Texas, USA. Correspondence to: Yi Li <Yi.Li3@utdallas.edu>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026.

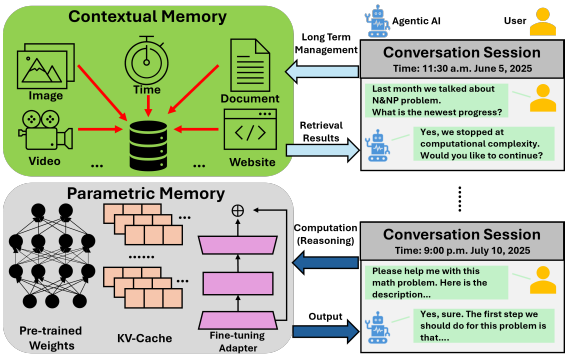

Figure 1. An illustration of memory taxonomy of Agentic AI.

Agentic memory can be broadly categorized into two types: *parametric* and *contextual* (Du et al., 2025) (as shown in Figure 1). Parametric memory is natively embedded within the LLM, encoded directly into its model weights, KV caches, or adapter layers (Hu et al., 2022). While powerful, this memory remains inherently opaque, computationally expensive to update, and tightly coupled to the model's underlying architecture (Wang et al., 2024). In contrast, contextual memory is decoupled and managed externally via explicit retrieval. This enables agents to dynamically store and query interaction histories, tool outputs, and empirical knowledge. This externalization offers three key advantages: (1) effectively unbounded capacity (De Cao et al., 2021; Jiang et al., 2024); (2) fast, selective updates without retraining; and (3) schema-level interpretability and control. In this work, we focus on the contextual memory, which has emerged as a critical enabler for scalable, coherent, and responsive agentic AI systems.

Despite architectural diversity, existing contextual memory systems share a critical limitation: low efficiency in both memory insertion and retrieval. A detailed performance analysis is presented in section 2.2. Whether based on RAG (Zhang et al., 2024; Kagaya et al., 2024), knowledge graph (Kim et al., 2024; Anokhin et al., 2024), or hybrid designs, these systems incur substantial overhead when storing new memory entries and retrieving relevant content. Insertion often requires costly embedding generation and preprocessing, while retrieval relies on high-dimensional similarity search or multi-hop graph queries, both of which are computationally expensive and latency-prone. This inefficiency is especially problematic in agentic AI, where agents operate in iterative loops and frequently update or consult memory across steps. Slow memory operations stall the observe–plan–act–learn cycle, reducing agent throughput and responsiveness. As agents scale to longer horizons and more complex tasks, the need for a memory substrate that supports fast, streaming writes and low-latency recall becomes paramount.

To address the above limitations, we argue for a fundamentally different memory substrate that abandons dense-vector and graph-centric retrieval in favor of lightweight, compression-native structures. The goal is to support efficient memory insertion and retrieval without sacrificing retrieval quality. HIPPOCAMPUS addresses these challenges by employing the Dynamic Wavelet Matrix (DWM), an innovative extension of the canonical wavelet matrix. While the wavelet matrix is a succinct data structure renowned for its space efficiency and rapid access primitives (Gog & Petri, 2014; Dietzfelbinger & Pagh, 2008), our DWM extension is specifically augmented to support the dynamic updates required by streaming agentic memory workloads. At a high level, HIPPOCAMPUS employs a dual-representation strategy: it stores memory content as lossless token-ID sequences for exact reconstruction (Content DWM), and parallel binary signatures for semantic search (Signature DWM). These two streams are co-indexed, enabling fast, bitwise retrieval directly in the compressed domain. The Signature DWM is constructed using random indexing, which produces compact binary representations of semantic content. Queries are executed via Hamming-ball search over these signatures, allowing fast, approximate matching with minimal computational cost. This design preserves both the fidelity of raw content and the semantic richness required for accurate, context-aware recall, while ensuring scalability and responsiveness in long-horizon agentic operations. Our main contributions are:

1. **A Compression-Native Memory Substrate.** We introduce an embedding-free memory substrate that replaces high-dimensional dense vectors with compact binary signatures and lossless token-ID streams. This enables a fundamental transition from embedding-heavy paradigms toward lightweight, compressed memory structures
2. **The HIPPOCAMPUS Architecture.** We introduce a contextual memory system powered by the *Dynamic Wavelet Matrix* (DWM), supporting streaming writes and the co-indexing of semantic and exact representations. This design facilitates Hamming-ball search directly within the compressed domain, achieving superior retrieval latency while maintaining storage costs that scale linearly with token count for a fixed vocabulary.
3. **Empirical Validation.** Extensive evaluations on the LoCoMo and LongMemEval-S benchmarks demonstrate that HIPPOCAMPUS achieves end-to-end retrieval latencies comparable to or better than state-of-the-art agentic memory baselines—yielding speedups of $1.1\times$ to $31.5\times$. Furthermore, HIPPOCAMPUS reduces the per-query token footprint by $1.1\times$ to $14.5\times$ relative to high-resource baselines while preserving competitive task accuracy.

The remainder of this paper is organized as follows: Section 2 reviews the existing agentic memory management systems and quantifies critical performance bottlenecks. Sec-

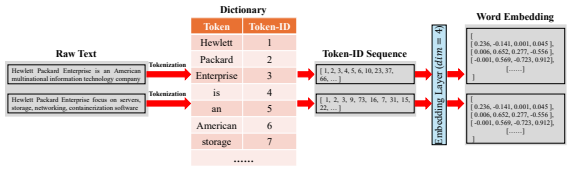

Figure 2. Illustration of raw text, token ID, and word embedding.

tion 3 details the architectural design of HIPPOCAMPUS, while Section 4 provides a comprehensive empirical evaluation of its latency, accuracy, and storage efficiency. Finally, Section 5 discusses related work, and Section 6 concludes the paper.

2 BACKGROUND AND MOTIVATION

2.1 Memory Representation and Management

Agentic AI systems rely on a memory system to persist information across iterative reasoning cycles. Existing agentic memory systems typically represent memory content using either high-dimensional embeddings or structured graphs. In Retrieval-Augmented Generation (RAG) (Zhang et al., 2024; Kagaya et al., 2024; Singh et al., 2025), raw text is embedded into dense vectors and stored in vector databases, enabling semantic similarity search. Knowledge Graph (KG)-based systems (Rasmussen et al., 2025; Xu et al., 2025) encode memory as entity–relation–entity triples, supporting multi-hop traversal and schema-aware reasoning in graph databases such as Neo4j (Guia et al., 2017). Hybrid systems (Xu et al., 2025) combine both approaches to balance semantic richness and structural precision.

While these representations offer expressive retrieval capabilities, they introduce significant performance bottlenecks. Embedding-based systems suffer from high computational costs during both ingestion and retrieval: memory insertion requires expensive embedding generation and preprocessing (e.g., summarization), while retrieval relies on costly vector similarity computations (Mei et al., 2024; Arora et al., 2020). Graph-based systems, though more interpretable, incur latency from multi-hop traversal and schema resolution. These inefficiencies are particularly problematic in agentic workflows, where memory is frequently updated and queried across observe–plan–act–learn cycles. Because these cycles are often recursive and time-sensitive, slow memory operations do more than just delay a single task; they introduce a computational bottleneck that stalls agent execution, throttles overall system throughput, and degrades the responsiveness required for real-time interaction.

To support scalable, high-performance agents, memory systems must enable fast, streaming writes and low-latency recall without compromising retrieval quality. This motivates our exploration of compression-native representations and efficient indexing mechanisms that can meet the demands of long-horizon, multi-iteration agentic deployments.

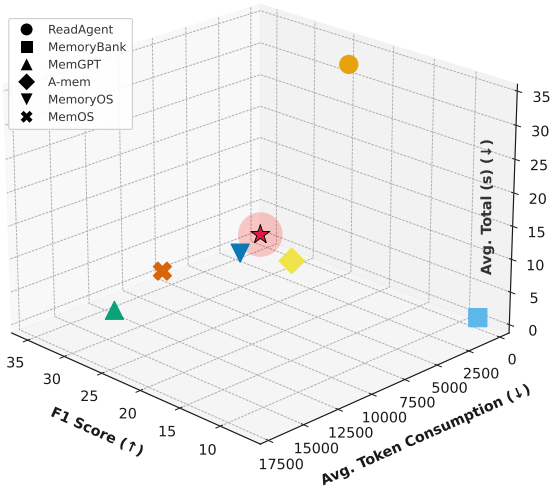

Figure 3. An analysis of SOTA agent memory systems across three critical metrics. Lower values are better for Avg. Token Consumption and Avg. Total Time, while higher is better for F1 Score. The plot illustrates that existing systems force a compromise, as none are able to simultaneously achieve high accuracy and high efficiency in the ideal design space indicated by the red star marker.

2.2 Performance Analysis

To understand the design trade-offs in existing agentic memory systems, we evaluated six representative state-of-the-art (SOTA) modules, ReadAgent (Lee et al., 2024), MemoryBank (Zhong et al., 2024a), MemGPT (Packer et al., 2023), A-Mem (Xu et al., 2025), MemoryOS (Kang et al., 2025), and MemOS (Li et al., 2025), across three critical metrics: retrieval accuracy (F1 score), operational cost (average token consumption per query), and user-perceived latency (average total query time). The evaluation was conducted on the LoCoMo benchmark (Maharana et al., 2024), which simulates long-horizon agentic tasks with frequent memory interactions.

As shown in Figure 3, current memory systems force developers into a difficult compromise. High-accuracy designs like MemGPT and A-Mem achieve strong F1 scores but incur significant latency and token overhead due to embedding generation, summarization, and multi-stage retrieval. Conversely, lightweight systems such as MemoryBank reduce latency and cost but suffer from degraded recall quality. None of the evaluated systems simultaneously optimize all three axes, leaving the ideal region of the design space, i.e., high accuracy with low latency and cost, unoccupied.

While above analysis focuses on the retrieval efficiency, the

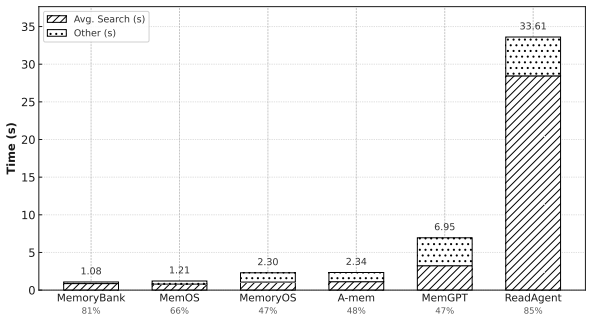

Figure 4. Breakdown of the end-to-end retrieval latency for SOTA agentic AI memory modules.

situation is further aggravated by insertion-side overhead during memory growth. In RAG, these arise from chunking, embedding, and index updates (Zhong et al., 2024b); in KG, from fact insertions and graph index maintenance (Anadiotis et al., 2024; Wandji & Calvanese, 2024); and in hybrid memories (e.g., A-Mem), from the additional note creation and cross-linking steps that improve read quality but inflate token and amortization cost. These write-path penalties add latency even before retrieval begins, exacerbating the trade-off illustrated in Figure 3 (Xu et al., 2025).

To pinpoint root cause of this inefficiency, Figure 4 shows a breakdown of end-to-end retrieval latency. The results show that the memory search phase dominates total execution time across all architectures. In ReadAgent, vector similarity search accounts for 85% of recall latency. Even in more streamlined systems like MemoryBank, search operations consume 81% of the time. Hybrid systems such as A-Mem and MemoryOS, which incorporate structured memory layouts and multi-hop reasoning, spend nearly half of their runtime in retrieval (48% and 47%, respectively).

These findings highlight a fundamental limitation: the retrieval substrate itself, whether based on dense vectors or graph traversal, is the primary bottleneck. In agentic workflows, where memory is accessed and updated repeatedly across observe–plan–act–learn cycles, such inefficiencies compound rapidly. Slow retrieval stalls the planning, while costly ingestion limits memory growth. To support scalable, responsive agents, we need a memory system that rethinks the underlying data structures and representations, enabling fast, streaming writes and low-latency recall at scale.

2.3 Tokenized Memory and Succinct Data Structures

Given that LLMs natively operate on integer sequences, i.e., token IDs (Qu et al., 2024; Yu et al., 2024), we adopt token IDs as the fundamental representation of memory. This compact, model-native format avoids repeated and costly tokenization cycles, enabling efficient storage and manipulation. More importantly, representing memory as integer sequences allows us to leverage powerful succinct data structures, e.g., the Wavelet Matrix (Gog & Petri, 2014; Claude & Navarro, 2012; Dietzfelbinger & Pagh, 2008), to build a high-performance retrieval system that operates directly in the compressed domain.

Wavelet Matrix. Succinct data structures are compact representations that approach the information-theoretic minimum space while supporting fast queries directly on compressed data (Dietzfelbinger & Pagh, 2008; Shamir, 2006). Among these, the Wavelet Matrix (Gog & Petri, 2014; Claude & Navarro, 2012) is particularly well-suited for representing long sequences of discrete symbols, such as token IDs in LLMs. It arranges the bits of each symbol into a multi-level structure and supports three core operations with logarithmic time complexity:

- $access(i)$: Retrieve the symbol at position i.
- $rank(c, i)$: Number of symbol c appears in prefix $[0, i)$.
- $select(c, j)$: Position of the j-th occurrence of symbol c.

However, canonical wavelet matrices are static and operate over a single homogeneous sequence, making them incompatible with agentic workloads where memory is continuously appended and must be immediately available for retrieval. Rebuilding the entire matrix for each new memory entry would be computationally prohibitive (Claude & Navarro, 2012), especially in long-horizon deployments. Moreover, as detailed in Section 3.2, HIPPOCAMPUS introduces two distinct but interrelated data streams, i.e., memory content and memory signatures, that must be co-indexed to support efficient retrieval. Nevertheless, the standard wavelet matrix (Gog & Petri, 2014; Claude & Navarro, 2012) lacks native support for co-indexing heterogeneous sequences, limiting its applicability in our design.

Semantic Hashing via Random Indexing. To enable efficient semantic search, we leverage Semantic Hashing, a form of Locality-Sensitive Hashing (LSH) (Indyk & Motwani, 1998), to convert high-dimensional vectors into compact binary signatures. Using a computationally inexpensive method called Random Indexing (Kanerva et al., 2000), we project each vector against a set of random hyperplanes to generate its signature. This ensures that semantically similar vectors are mapped to signatures with a small Hamming distance (i.e., differing in only a few bits) (Norouzi et al., 2012; Labib et al., 2019). This crucial property allows us to replace the expensive k-Nearest Neighbor (k-NN) search (de Vries et al., 2002) over floating-point vectors with an ultra-fast search for neighbors within a small Hamming radius, an operation that can be massively accelerated using native bitwise CPU operations (Seshadri et al., 2016).

3 DESIGN OF HIPPOCAMPUS

We present the technical design of HIPPOCAMPUS, a system built for scalable, high-throughput agentic AI memory management. At the core of the design is a dual-representation

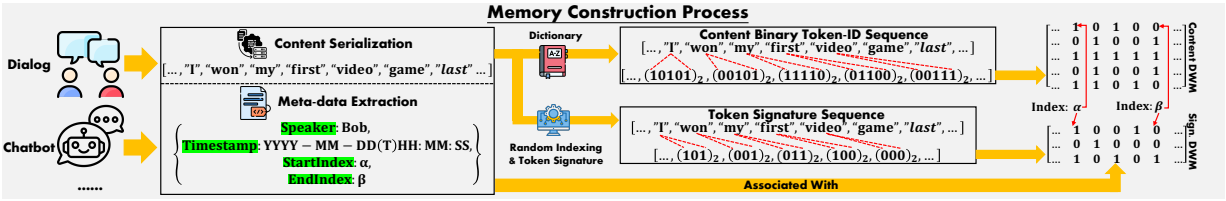

Figure 5. Illustration of the memory construction pipeline in Hippocampus. **DWM** denotes our proposed **D**ynamic **W**avelet **M**atrix, while the subscript $(\cdot)_2$ indicates the binary representation of an integer, for example, token ID. The first row of the DWM serves as the entry-level index, marking the start and end positions of each token in the **Content Serialization** (e.g., **StartIndex** α and **EndIndex** β).

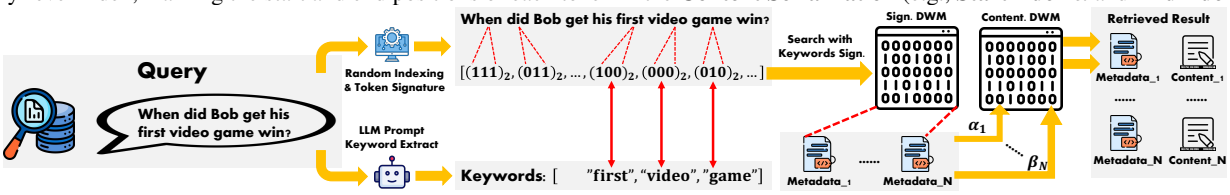

Figure 6. Illustration of the memory query pipeline in the Hippocampus. An LLM first extracts keywords from the natural language query. These keywords are converted into binary signatures and used to perform a fast, approximate search on the Signature (Sign.) Dynamic Wavelet Matrix (DWM), identifying candidate metadata blocks. The indices (e.g., α_1) from the retrieved metadata are then used to look up and reconstruct the exact, full-resolution content from the Content DWM.

strategy that simultaneously supports exact, high-fidelity content retrieval and fast, approximate semantic search. Central to this strategy is Dynamic Wavelet Matrix (DWM), a compressed, bit-level data structure that we develop and employ to index both representations. This approach enables HIPPOCAMPUS to achieve high data density while maintaining low-latency query performance.

We begin by outlining the high-level system architecture (Section 3.1), which illustrates the data flow during memory construction and retrieval. In Section 3.2, we provide a detailed description of the Dynamic Wavelet Matrix (DWM), including the construction and retrieval, i.e., memory recall processes. Further, Appendix E provides both empirical scaling results and theoretical analysis of the DWM's efficiency and storage overhead, while Appendix F quantifies how our Hamming-ball query scheme (Section 3.3) approximates dense vector retrieval and gives practical guidance for choosing the Hamming radius r.

3.1 Overall Architecture

The architecture of HIPPOCAMPUS memory module is composed of two primary components: a pipeline for memory construction (as shown in Figure 5) and a pipeline for memory querying (as shown in Figure 6).

Memory Construction Pipeline. As shown in Figure 5, memory construction pipeline begins by ingesting raw data, such as a dialogue turn in LoCoMo (Maharana et al., 2024) (see Section 4.1 for details), which is then processed through two parallel steps. First, content serialization converts unstructured text into a canonical sequence of tokens. Concurrently, metadata extraction captures essential contextual information. For LoCoMo dataset, this includes speaker/-role, a high-resolution timestamp, and the start (α) and end (β) indices of each utterance within serialized token list.

The core of HIPPOCAMPUS is a dual-representation strategy, realized through two distinct Dynamic Wavelet Matrices (DWMs) (see Section 3.2 for details): a Content DWM for exact data representation and a Signature DWM for efficient, approximate semantic search. To construct the Content DWM, each token in the serialized sequence (i.e., from content serialization) is mapped to its corresponding integer token ID, which is then converted into its binary representation. These binary codes are vertically arranged to form a bit matrix, constituting the Content DWM. In parallel, the Random Indexing & Token Signature module computes a low-dimensional binary hash, or signature, for each token (see Section 3.3 for details). This process produces a compact token signature sequence, which is used to construct the Signature DWM in the same manner. This dual-matrix structure enables HIPPOCAMPUS to support both precise content retrieval and fast, semantics-based similarity queries within a unified framework.

Memory Retrieval Process. We now describe the query process, illustrated in Figure 6, which leverages the two constructed DWMs (as shown in Figure 5) to enable highly efficient agentic AI memory retrieval. The query pipeline begins when a natural language query is received. First, the query is processed by a lightweight LLM Prompt module to extract a set of salient keywords. These keywords are then passed through the same Random Indexing & Token Signature module used during memory construction, converting them into their corresponding binary signatures. These

query signatures are used to perform a fast, approximate search, e.g., based on Hamming distance (see Section 3.3 for details), against the Signature DWM. This initial pass rapidly filters the entire memory space and identifies a small set of candidate data segments by retrieving their associated metadata blocks. The StartIndex (α) and EndIndex (β) from each candidate's metadata serve as direct pointers for exact, indexed retrieval from the Content DWM. This step reconstructs the original, full-resolution token sequences for the candidate segments. The retrieved content and its corresponding metadata are then returned as the final Retrieved Result. This two-stage design allows HIPPOCAMPUS to efficiently search over vast conversational histories by using the compact Signature DWM as a fast, low-cost index into the high-fidelity Content DWM.

With the end-to-end data flow established, we now dive into the core technical components that underpin the HIPPOCAMPUS architecture. The following subsections are organized as follows: we first provide a detailed formulation of the Dynamic Wavelet Matrix (DWM) (Section 3.2), which serves as the fundamental data structure in HIPPOCAMPUS. We then describe the Random Indexing and Hamming Ball search mechanism (Section 3.3), which is used to generate robust token signatures and perform approximate search.

3.2 Dynamic Wavelet Matrix

The core data structure underlying both the content store and the signature store in HIPPOCAMPUS is our Dynamic Wavelet Matrix (DWM). The DWM is a novel data structure we develop for agentic AI memory. It serves as an append-friendly adaptation of the conventional static Wavelet Matrix (WM) (Gog & Petri, 2014; Claude & Navarro, 2012), a well established structure in information retrieval for compressing and indexing large sequences. By extending the WM to support dynamic updates while preserving its compression and query efficiency, the DWM enables high-throughput memory construction and retrieval in continuously evolving agentic systems.

Primer on the standard WM. A standard WM is built level by level over an integer sequence $S[0, \ldots, n-1]$ drawn from an alphabet of size σ. Let $l = \lceil \log_2 \sigma \rceil$, and define $S^{(0)} = S$. At level $k \in \{0, \ldots, l-1\}$, the WM materializes a bit-vector $\boldsymbol{B}^k$ such that $\boldsymbol{B}^k[i] = bit_k(S^{(k)}[i])$, where $bit_k(\cdot)$ denotes the k-th bit when symbols are read from the most significant bit to the least significant bit. After emitting $\boldsymbol{B}^k$, the level-dependent sequence $S^{(k)}$ is stably partitioned into a zero side followed by a one side to form $S^{(k+1)}$. In other words, all symbols whose k-th bit is 0 are routed left, all symbols whose k-th bit is 1 are routed right, and the relative order within each group is preserved. We denote by Z_k the number of 0-bits in $\boldsymbol{B}^k$, which is also the starting offset of the one side at the next level. This stable-partition semantics is the reason the rows of a wavelet matrix should *not* be read as a simple vertical stacking of the original binary codes: the symbol order changes from level to level.

From WM to DWM. A standard WM is static: it is typically constructed once over a fixed sequence. In contrast, HIPPOCAMPUS requires an append-friendly index whose contents remain queryable immediately after each memory update. The DWM preserves the same level-wise stable-partition semantics as a standard WM, but supports dynamic appends by inserting one bit per level and computing the next-level insertion position from rank information and the zero-count offset Z_k. Thus, if the current sequence were rebuilt from scratch as a static WM after each append, the resulting bit-vectors would match the DWM.

Notation and Structure. We still view the DWM as an $l \times n$ bit-matrix, but its rows correspond to the permuted sequences $S^{(k)}$, not to the original sequence S in fixed column order. Figure 7 gives a worked construction example. The DWM retains the standard WM query primitives: $access(i)$, $rank(c, i)$, and $select(c, j)$, while extending the structure to streaming agentic-memory writes.

3.2.1 Dynamic Construction

We construct DWM through a sequence of $append(s)$ operations, each adding a new symbol $s \in [0, \sigma)$ to the end of the logical sequence S. To append a symbol, we traverse the l levels top-down while preserving the same stable-partition semantics as the static WM (as shown in Figure 7):

1. At level $k = 0$, insert the most significant bit $bit_0(s)$ into $\boldsymbol{B}^0$ at the current global position $p_0 = n$.
2. If the inserted bit is 0, the symbol belongs to the zero side of the next level and its next position is $p_{k+1} = rank_0(\boldsymbol{B}^k, p_k)$. If the inserted bit is 1, the symbol belongs to the one side and its next position is $p_{k+1} = Z_k + rank_1(\boldsymbol{B}^k, p_k)$, where Z_k is the number of 0-bits in level k.
3. Repeat this insertion-and-routing step for $k = 1, \ldots, l-1$, always writing $bit_k(s)$ at position p_k of $\boldsymbol{B}^k$ and then updating the insertion position for the next level according to the zero/one routing rule above.
4. After the last level, the symbol has been fully inserted and the logical sequence length increases by one. Because each level respects the same stable partition used by the static WM, the DWM remains consistent with the level-dependent sequences $S^{(k)}$.

With dynamic rank/select-supported bit-vectors, constructing a DWM over n stored tokens costs $\mathcal{O}(n \log n)$ total time for a fixed tokenizer vocabulary, while the storage footprint remains $\mathcal{O}(n \log \sigma)$ bits, i.e., linear in the number of stored tokens for fixed σ.

Our DWM design provides exactly query primitives needed

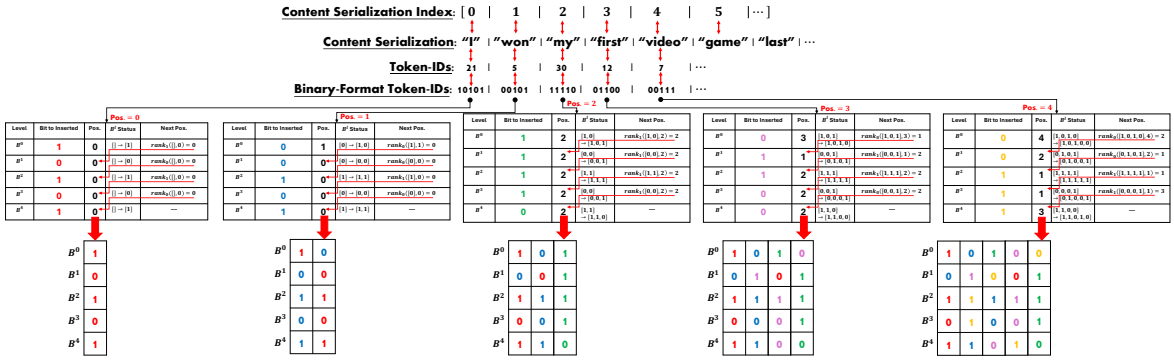

Figure 7. Construction process of Dynamic Wavelet Matrix.

Algorithm 1 DWM RANK(c, i) operation

Require: Token signature $\boldsymbol{c}$ with bits $(b_0, \ldots, b_{l-1})$ where b_0 is MSB; global prefix length i; Sign. DWM $\boldsymbol{D}$ with level bit-vectors $\boldsymbol{D}.B[0 \ldots l-1]$ and zero-counts $\boldsymbol{D}.Z[0 \ldots l-1]$
Ensure: $rank(c, i)$ = #occurrences of c in $S[0, i)$
$(b_0, b_1, \ldots, b_{l-1}) \leftarrow$ BitsMSBFirst$(\boldsymbol{c})$
$p_L \leftarrow 0$; $p_R \leftarrow i$
for $k = 0$ to $l-1$ **do**
  if $b_k = 0$ **then**
    $p_L \leftarrow rank_0(\boldsymbol{D}.B[k], p_L)$
    $p_R \leftarrow rank_0(\boldsymbol{D}.B[k], p_R)$
  else
    $Z_k \leftarrow \boldsymbol{D}.Z[k]$
    $p_L \leftarrow Z_k + rank_1(\boldsymbol{D}.B[k], p_L)$
    $p_R \leftarrow Z_k + rank_1(\boldsymbol{D}.B[k], p_R)$
  end if
end for
return $p_R - p_L$

Algorithm 2 DWM $select(c, j)$ operation

Require: Token signature $\boldsymbol{c}$ with bits $(b_0, \ldots, b_{l-1})$ where b_0 is MSB; 1-based occurrence j; sequence length n; Sign. DWM $\boldsymbol{D}$ with level bit-vectors $\boldsymbol{D}.B[0 \ldots l-1]$ and zero-counts $\boldsymbol{D}.Z[0 \ldots l-1]$
Ensure: $select(c, j)$ = global position of the j-th c in S
$(b_0, b_1, \ldots, b_{l-1}) \leftarrow$ BitsMSBFirst$(\boldsymbol{c})$
$p_L \leftarrow 0$; $p_R \leftarrow n$
for $k = 0$ to $l-1$ **do**
  if $b_k = 0$ **then**
    $p_L \leftarrow rank_0(\boldsymbol{D}.B[k], p_L)$
    $p_R \leftarrow rank_0(\boldsymbol{D}.B[k], p_R)$
  else
    $Z_k \leftarrow \boldsymbol{D}.Z[k]$
    $p_L \leftarrow Z_k + rank_1(\boldsymbol{D}.B[k], p_L)$
    $p_R \leftarrow Z_k + rank_1(\boldsymbol{D}.B[k], p_R)$
  end if
end for
$occ \leftarrow p_R - p_L$
if $j > occ$ **then**
  return NULL
end if
$p \leftarrow p_L + (j-1)$
for $k = l-1$ to 0 **step** -1 **do**
  if $b_k = 0$ **then**
    $p \leftarrow select_0(\boldsymbol{D}.B[k], p+1)$
  else
    $Z_k \leftarrow \boldsymbol{D}.Z[k]$
    $p \leftarrow select_1(\boldsymbol{D}.B[k], (p - Z_k) + 1)$
  end if
end for
return p

for HIPPOCAMPUS. Content DWM supports direct access to any token via the $access(i)$ operation. Signature DWM enables efficient approximate membership queries using $rank$ and $select$ operations, which we leverage to perform Hamming-distance-based searches for relevant signatures. We next describe how these queries operate within the DWM and how they enable memory recall in HIPPOCAMPUS.

3.2.2 *Memory Recall with Dynamic Wavelet Matrix*

When a query is issued, HIPPOCAMPUS uses the Signature DWM to identify likely relevant memory indices, and then uses the Content DWM to reconstruct the content at those indices. This process relies on the DWM's ability to efficiently count and locate symbols. In the Signature DWM, each "symbol" is a compact binary signature representing a token. A natural language query is transformed into a set of such signature symbols $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_m\}$ via the random indexing step. Our goal is to find memory entries where all (or many) of these query signatures co-occur. We accomplish this by using the DWM $rank$ and $select$ primitives to traverse the Signature DWM efficiently.

Searching in Signature DWM. Suppose we have a particular signature $\boldsymbol{c}$ (a binary code of length l bits) and we want to quickly find all positions in the Signature DWM where $\boldsymbol{c}$ appears. We can use $rank(\boldsymbol{c}, i)$ to count occurrences of $\boldsymbol{c}$ up to any position i, and $select(\boldsymbol{c}, j)$ to retrieve the position of the j-th occurrence. Algorithm 1 outlines the $rank$ query. Starting from the most significant bit of $\boldsymbol{c}$, we use the bit values to narrow an interval $[p_L, p_R)$ as we descend the levels. Initially, $p_L = 0$ and $p_R = i$ (meaning we consider the prefix $S[0..i-1]$). At each level k, if the k-th bit of $\boldsymbol{c}$ is 0, we map the current interval to the zero-prefixed subar-

ray of the next level by setting $p_L \leftarrow rank_0(\boldsymbol{B}^k, p_L)$ and $p_R \leftarrow rank_0(\boldsymbol{B}^k, p_R)$. If the bit is 1, we map to the one-prefixed subarray by setting $p_L \leftarrow Z_k + rank_1(\boldsymbol{B}^k, p_L)$ and $p_R \leftarrow Z_k + rank_1(\boldsymbol{B}^k, p_R)$, where Z_k is the total number of 0s in $\boldsymbol{B}^k$. After processing all l bits, the length of the final interval $(p_R - p_L)$ equals the number of occurrences of $\boldsymbol{c}$ in $S[0..i-1]$.

To retrieve the actual positions of occurrences, we use the $select(\boldsymbol{c}, j)$ operation, outlined in Algorithm 2. We first find the total number of occurrences $occ = rank(\boldsymbol{c}, n)$ in the entire sequence of length n. If $j > occ$, the j-th occurrence does not exist. Otherwise, we know the j-th occurrence lies in the interval $[p_L, p_R)$ obtained by running the $rank$ procedure (Algorithm 1) to the end of the sequence ($i = n$). We set $p = p_L + (j-1)$, which is the index of this occurrence in the bottom level. We then lift this index back up through the levels. For each level k (going from $l-1$ up to 0): if $b_k = 0$ (the k-th bit of $\boldsymbol{c}$ is 0), we call $p \leftarrow select_0(\mathbf{D}.B[k], p+1)$, which finds the global position of the $(p+1)$-th 0-bit in level k. If $b_k = 1$, we set $p \leftarrow select_1(\mathbf{D}.B[k], (p - Z_k) + 1)$, which finds the global position of the $(p - Z_k + 1)$-th 1-bit in level k (accounting for the offset of the one-block). After lifting through all levels, p gives the global position in S of the j-th occurrence of $\boldsymbol{c}$.

In HIPPOCAMPUS, we use these primitives to execute memory queries as follows. Given a set of query signatures: $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m\}$ extracted from the user's query, we first identify the least frequent signature $\boldsymbol{c}_{\min}$ by comparing $rank(\boldsymbol{c}_i, n)$ for all i. We then iterate through each occurrence of $\boldsymbol{c}_{\min}$ in the Signature DWM. For the j-th occurrence (where j ranges from 1 to $occ = rank(\boldsymbol{c}_{\min}, n)$), we find its global position $i = select(\boldsymbol{c}_{\min}, j)$. This position i corresponds to a specific token in the memory sequence S. We retrieve the metadata entry whose range $[\alpha, \beta]$ covers i (recall that each memory entry's start and end indices are stored in its metadata). This metadata tells us the span of token indices for that memory entry. If the query contains multiple keywords, we can quickly verify whether the other query signatures $\boldsymbol{c}_{2\ldots m}$ appear in the same span by checking if $rank(\boldsymbol{c}_k, \beta) - rank(\boldsymbol{c}_k, \alpha) > 0$ for each k. Entries that pass this check are collected as candidate results.

Retrieving from Content DWM. Finally, for each candidate memory entry identified via the above process, we perform a lossless reconstruction of its content using the Content DWM. This is achieved through the $access(i)$ primitive applied over the range $[\alpha, \beta]$ of token positions. Algorithm 3 shows how a single symbol is retrieved by $access(i)$. We start at the top level with the global position i. At level 0, we read the bit $b_0 = \boldsymbol{B}^0[i]$, which is the most significant bit of the symbol at $S[i]$. We append b_0 to a bit buffer and then determine the position at the next level: if $b_0 = 0$, we set $i_1 = rank_0(\boldsymbol{B}^0, i)$ (the number of 0s up to position i in level 0); if $b_0 = 1$, we set $i_1 = Z_0 + rank_1(\boldsymbol{B}^0, i)$ (the number of 0s in level 0 plus the number of 1s up to i). We then move to level 1, read $b_1 = \boldsymbol{B}^1[i_1]$, append it, and update the position for level 2 in a similar fashion.

Algorithm 3 DWM ACCESS(i) operation

Require: Global position i; Content DWM $\boldsymbol{D}$ with level bit-vectors $\boldsymbol{D}.B[0 \ldots l-1]$ and zero-counts $\boldsymbol{D}.Z[0 \ldots l-1]$
Ensure: ACCESS(i) = the symbol $S[i]$

```
p ← i
bits ← [ ]
for k = 0 to l − 1 do
    b ← D.B[k][p]
    bits.append(b)
    if b = 0 then
        p ← rank_0(D.B[k], p)
    else
        Z_k ← D.Z[k]
        p ← Z_k + rank_1(D.B[k], p)
    end if
end for
return SymbolFromBits(bits)
```

After we descend through all l levels, we have collected bits $(b_0, b_1, \ldots, b_{l-1})$, which constitute the binary representation of $S[i]$. We then convert these bits back to the original token-id (an integer) using SymbolFromBits. In practice, we execute $access(i)$ for each position i in the range $[\alpha, \beta]$ to retrieve the entire sequence of token-ids for that memory entry, and then detokenize to reconstruct the text.

3.3 Random Indexing and Hamming Ball

While the DWM supports efficient keyword-exact matching, many queries require semantic-level retrieval for improved accuracy and robustness. To enable this capability, HIPPOCAMPUS converts each token into a compact, context-aware binary signature. Instead of using static embeddings or precomputed vectors, we adopt a lightweight streaming random indexing mechanism (Indyk & Motwani, 1998) that continuously integrates local contextual information during memory construction.

Specifically, let D (e.g., 1024) denote the embedding dimensionality. At initialization, each token v is assigned a sparse random base vector $\boldsymbol{r}_v = \{-1, 0, +1\}^D$ with exactly t non-zero entries placed randomly (half $+1$ and half -1). These vectors remain fixed throughout content serialization. As the conversation stream arrives, we maintain a sliding window $W(i)$ around each token $S[i]$ and aggregate its contextual embedding via $\boldsymbol{e}_i = \sum_{j \in W(i)} \boldsymbol{r}_{S[j]}$, ensuring that tokens appear in slightly different semantic states depending on their conversational context (Kanerva et al., 2000). After one streaming pass, each token has a fully contextualized embedding $\boldsymbol{e}_i$. Directly hashing all D dimensions would incur unnecessary cost, so HIPPOCAMPUS selects only the

d ($d \ll D$) most activated components: $\mathcal{I}_i = \text{Top-}d(|\boldsymbol{e}_i|)$. A binary signature is then formed:

$$\boldsymbol{s}_i[k] = \begin{cases} 1, & \boldsymbol{e}_i[\mathcal{I}_i[k]] > 0 \\ 0, & \boldsymbol{e}_i[\mathcal{I}_i[k]] \leq 0 \end{cases} \qquad k = 1, \cdots, d$$

During querying (Figure 6), an LLM extracts a small set of keywords from the natural language query that best describe the user's intent. Each keyword is then converted into a d-bit signature using the same streaming random setting used during memory construction. We then perform an efficient Hamming-ball search on the Signature DWM. For each keyword signature $\boldsymbol{s}_q$ and a stored signature $\boldsymbol{s}_i$, we first compute a bitwise XOR, which returns a d-bit mask where 1s indicate mismatched bit positions. We then apply `POPCOUNT` (Sun, 2016), a native CPU instruction that counts the number of 1s in the mask in constant time, thus directly yielding the Hamming distance $\text{HammingDist}(\boldsymbol{s}_q, \boldsymbol{s}_i)$. A candidate is preserved only if this distance does not exceed a small threshold r, meaning we search within a Hamming-ball defined as: $\{\boldsymbol{s}_i \mid \text{HammingDist}(\boldsymbol{s}_q, \boldsymbol{s}_i) \leq r\}$, so that only entries differing in at most r bits (out of the d bits) are considered semantically relevant and passed forward for subsequent metadata validation. In Appendix F, we provide a practical rule-of-thumb for choosing the Hamming radius r. Under random-hyperplane-style hashing, the Hamming distance concentrates around $(\theta/\pi)d$, where θ is the angle between the underlying vectors. Therefore, if the target semantic threshold corresponds to cosine similarity $\cos\theta_0$, a natural initial choice is $r \approx (\theta_0/\pi)d$; a slightly recall-favoring setting is $r = (\theta_0/\pi + \epsilon)d$ for a small margin ϵ (e.g., 0.01). Section 4.3 evaluates this trade-off empirically.

3.4 Handling Forgetting and Deletion

Although the current DWM core is append-friendly, it can integrate cleanly with external forgetting policies. We envision a two-tier mechanism. **Logical forgetting:** once an external policy marks a memory segment as inactive or expired, HIPPOCAMPUS can place a tombstone in the metadata layer and exclude that segment during candidate selection and content reconstruction. **Physical deletion:** to reclaim space, the system can periodically compact the store by rebuilding both DWMs while skipping tombstoned segments. In this way, forgetting/editing policies remain orthogonal to our retrieval substrate: online updates stay append-friendly, while deletion is handled through metadata masking and offline compaction.

4 EVALUATION

4.1 Experimental Setup

Dataset. We adopt two of the most recent and widely used benchmarks designed to assess the long-term contextual memory capabilities of agentic AI: LoCoMo (Maharana et al., 2024) and LongMemEval (Wu et al., 2024a). For a detailed description of the datasets, please refer to Appendix B.

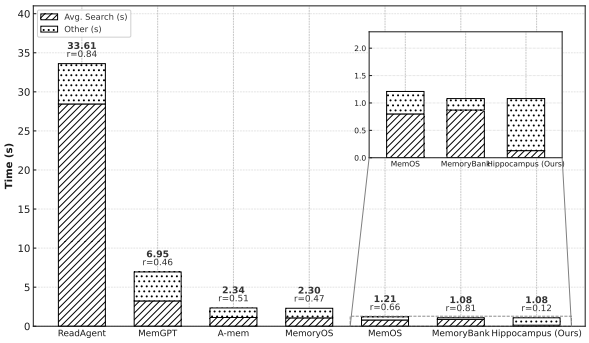

Figure 8. Average query retrieval latency (seconds) for various memory systems on LoCoMo dataset.

Metric. For LoCoMo, we adopt its default automatic evaluation metrics: F1 (Opitz & Burst, 2019) and BLEU-1 (Yang et al., 2008), which measure lexical overlap and token-level correctness in question answering tasks. For LongMemEval, the benchmark uses accuracy as its principal metric, defined as the fraction of evaluation questions answered correctly (Wu et al., 2024a). Beyond these standard metrics, we also introduce a *LLM-as-a-Judge* score (Gu et al., 2024) to better capture semantic correctness and deeper reasoning quality (Range: $[1, 2, 3, 4, 5]$). Refer to Appendix A for details. In addition to accuracy-oriented metrics, we evaluate efficiency along two axes:

- **Avg. Token Consumption:** average number of tokens read and processed per query, reflecting memory retrieval cost;
- **Avg. Total:** mean time from the moment memory recall is triggered to the moment the retrieved context is delivered (used for constructing the final prompt). Within this, we further decompose and report **Avg. Search**, which captures the pure retrieval cost.

Software and Hardware. All experiments were conducted on a HPE DL380a Gen11 server with 2× Intel Xeon Platinum 8470 CPU, 4 × NVIDIA H100 GPUs, and 1 TB of DDR4 DRAM. The software environment includes Ubuntu 22.04.5 LTS, Python 3.10.12, PyTorch 2.7.0, and CUDA 12.9 for GPU acceleration.

4.2 Overall Comparison

LoCoMo Analysis. On the LoCoMo tasks, HIPPOCAMPUS delivers strong retrieval accuracy that rivals or exceeds prior systems (Table 1). For example, on *Temporal Reasoning*, HIPPOCAMPUS achieves F1 ≈ 38.3, substantially higher than the 26.5 F1 reported for MemGPT. Similarly, on

Table 1. Overall comparison of different memory modules across four tasks in LoCoMo benchmark: **Single-Hop**, **Multi-Hop**, **Temporal**, and **Open-Domain**. We report the default metrics (F1 and BLEU-1) together with an LLM-as-a-Judge score reflecting human-aligned evaluation of answer quality. Reported F1 and BLEU-1 are multiplied by 100 for easier comparison and visualization.

Memory Module	Single-Hop			Multi-Hop			Temporal			Open-Domain		
	F1	BLEU-1	LLM-as-a-Judge	F1	BLEU-1	LLM-as-a-Judge	F1	BLEU-1	LLM-as-a-Judge	F1	BLEU-1	LLM-as-a-Judge
ReadAgent	8.78	5.93	1.03	5.44	5.03	1.01	11.24	11.12	1.08	9.32	8.1	1.45
MemoryBank	5.05	3.97	2.00	6.02	5.89	1.12	9.85	9.92	1.03	7.9	7.97	2.09
MemGPT	25.43	17.68	1.91	9.11	8.82	1.06	26.48	26.19	1.02	39.74	40.03	1.92
A-mem	19.82	19.86	2.66	12.97	12.81	1.85	34.63	34.87	2.18	41	41.41	2.72
MemoryOS	32.5	30.13	2.76	28.61	26.81	1.79	25.08	25.08	2.61	41.51	41.43	2.59
MemOS	**39.24**	**40.76**	2.75	30.11	30.91	2.56	31.06	31.34	2.81	40.31	40.51	2.60
HIPPOCAMPUS	34.36	30.04	**3.08**	**31.97**	**31.85**	**3.22**	**38.3**	**37.35**	**2.94**	**48.38**	**46.8**	**2.97**

Table 2. Overall comparison across six tasks in LongMemEval-S benchmark: **Single-session-preference**, **Single-session-assistant**, **Temporal-reasoning**, **Multi-session**, **Knowledge-update**, and **Single-session-user**. Reported F1 and Accuracy are multiplied by 100. Best F1 and Accuracy values are boldfaced for consistency with Table 1.

Memory Module	Single-session-preference			Single-session-assistant			Temporal-reasoning			Multi-session			Knowledge-update			Single-session-user		
	F1	Accuracy	LLM-as-a-Judge	F1	Accuracy	LLM-as-a-Judge	F1	Accuracy	LLM-as-a-Judge	F1	Accuracy	LLM-as-a-Judge	F1	Accuracy	LLM-as-a-Judge	F1	Accuracy	LLM-as-a-Judge
ReadAgent	3.54	4.17	1.25	4.48	15.18	1.46	3.76	4.32	0.86	1.65	4.89	1.03	2.96	8.01	1.05	4.87	17.14	1.55
MemoryBank	4.25	5.00	1.88	5.36	18.21	2.20	4.50	5.19	1.29	1.98	5.86	1.55	3.55	9.62	1.58	5.62	20.57	2.33
MemGPT	4.95	5.83	1.72	6.25	21.25	2.01	5.25	6.05	1.18	2.31	6.84	1.41	4.15	11.22	1.43	6.38	24.00	2.14
A-mem	7.78	9.17	2.50	9.86	33.39	2.93	8.28	9.51	1.72	3.64	10.75	2.06	6.54	17.63	2.10	10.03	37.71	3.10
MemoryOS	9.21	10.83	2.66	11.67	39.46	3.11	9.78	11.24	1.83	4.30	12.70	2.19	7.73	20.83	2.23	11.86	44.57	3.23
MemOS	10.61	12.50	2.81	13.39	45.53	3.29	11.23	12.97	1.94	4.94	14.66	2.31	8.88	24.04	2.36	13.63	51.43	3.29
HIPPOCAMPUS	**14.14**	**16.67**	**3.13**	**17.92**	**60.71**	**3.66**	**15.03**	**17.29**	**2.15**	**6.61**	**19.54**	**2.57**	**11.83**	**32.05**	**2.63**	**19.48**	**68.57**	**3.88**

Table 3. Sensitivity of HIPPOCAMPUS to the Hamming radius r on LoCoMo, averaged over the four tasks (single-hop, multi-hop, temporal, and open-domain).

Hamming radius r	Avg. F1 ($\times 100$)	Avg. BLEU-1 ($\times 100$)	Avg. LLM-as-a-Judge (1–5)	Avg. Search (s)	Avg. Token	Avg. Total (s)
1	38.12	36.83	3.06	0.13	1333.40	1.11
2	39.00	37.15	3.10	0.16	1645.38	1.26
3	39.10	37.25	3.08	0.21	2067.12	1.51
4	38.55	36.90	3.04	0.27	2716.74	2.00
5	37.85	36.30	2.97	0.34	3548.13	2.72

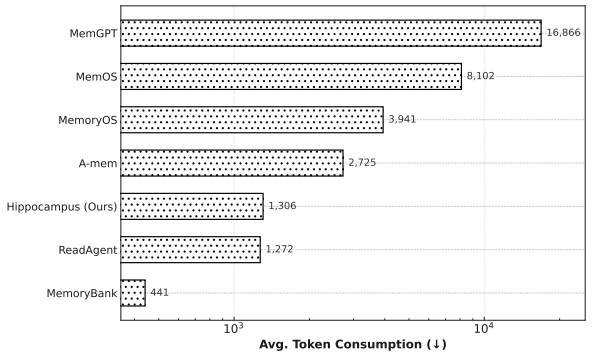

Figure 9. Average number of tokens consumption per query by each memory system on LoCoMo dataset.

Open-Domain, HIPPOCAMPUS attains 48.4 F1 compared to 41.5 for MemoryOS. HIPPOCAMPUS 's LLM-as-a-Judge scores are also the highest in all categories ($\approx$ 3.0–3.3 out of 5), reflecting answer quality that is equal or better than the baselines. These results demonstrate that the semantic-approximation mechanism in HIPPOCAMPUS (binary token signatures) incurs only minor accuracy loss, while still retrieving relevant context effectively. In contrast, lightweight baselines like MemoryBank, which sacrifice search overhead for speed, achieve very low accuracy (F1 < 10 across tasks). In summary, HIPPOCAMPUS matches or outperforms SOTA memory systems on LoCoMo while using a compact, compressed-index representation.

The efficiency advantages of HIPPOCAMPUS are dramatic. Figure 8 plots the average end-to-end query latency for each system. HIPPOCAMPUS responds in roughly 1.08 seconds on average—an order of magnitude faster than dense-vector approaches (MemGPT $\approx$ 33.6s) and substantially quicker than knowledge graph- or RAG-based memories. The breakdown in Figure 8 shows that HIPPOCAMPUS spends only a small fraction of that time in the search phase, whereas baselines incur a dominant search cost (often $> 80\%$ of total latency). Figure 9 displays average token consumption: HIPPOCAMPUS reads only $\approx$ 1.3K tokens on average, far fewer than MemGPT ($\approx$ 16.9K) or MemoryOS ($\approx$ 8.1K). This low token overhead arises from HIPPOCAMPUS 's compressed memory structure: rather than loading large text embeddings, it scans concise bitwise signatures and reconstructs exact token IDs on demand. These efficiency gains show that HIPPOCAMPUS achieves high retrieval accuracy with minimal latency and cost. The observed performance aligns with our design motivation: prior high-accuracy mem-

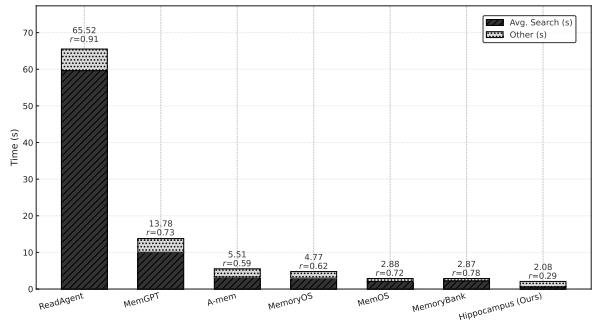

Figure 10. Average query retrieval latency (seconds) for various memory systems on LongMemEval-s.

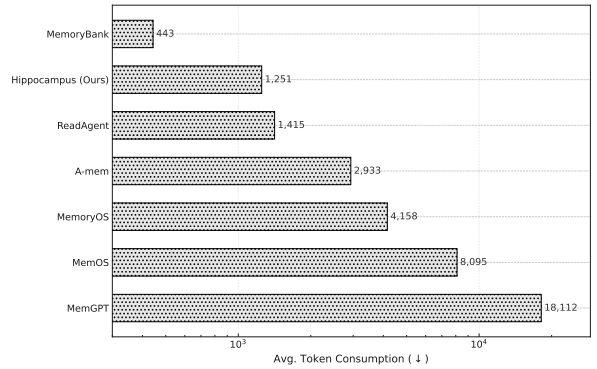

Figure 11. Average number of tokens consumption per query by each memory system on LongMemEval-s.

ory systems required heavy token usage and slow searches, whereas HIPPOCAMPUS breaks that trade-off through its space-efficient, bitwise index.

LongMemEval Analysis. On the LongMemEval-S, HIPPOCAMPUS consistently achieves the best accuracy–efficiency operating point among all baselines. As summarized in Table 2, HIPPOCAMPUS improves accuracy across all six tasks while dramatically reducing end-to-end retrieval time and minimizing token footprint. In Figure 10, our end-to-end latency lies near the floor of the plot, reflecting how bit-sliced Hamming-ball filtering on the Signature DWM eliminates the dominant search cost that burdens dense-vector and graph-traversal designs. Figure 11 shows a much smaller per-query token budget, as HIPPOCAMPUS scans compact binary signatures and reconstructs token IDs on demand, rather than streaming long textual passages or large embedding blocks. Together, these effects validate our design thesis from Section 3: approximate semantic access in the compressed domain (signatures) combined with exact reconstruction (Content DWM) breaks the classic trade-off—maintaining task accuracy while achieving order-of-magnitude gains in responsiveness and prompt-token economy. Please refer to Appendix C for complementary results on LongMemEval-M.

We additionally validate scalability beyond a single 16K-token conversation. Appendix E reports construction time and storage overhead from 16K to 160K tokens on LoCoMo. Construction follows the expected near-linear $\mathcal{O}(n \log n)$ trend, while the footprint grows linearly from 0.15MB to 1.46MB. Together with the LongMemEval-M results, these data show that HIPPOCAMPUS remains practical in the 100K-token regime and beyond.

4.3 Sensitivity to Hamming Radius

Table 3 sweeps $r \in \{1, 2, 3, 4, 5\}$ on LoCoMo and averages the results over the four tasks. The observed pattern is an inverted-U: accuracy peaks around $r = 2$ or 3, where the Hamming ball is wide enough to recover semantically related tokens but still tight enough to suppress irrelevant noise. When r increases to 4 or 5, the candidate set becomes too loose, token consumption rises sharply, and answer quality declines. Compared with the default $r = 1$, the best-accuracy setting $r = 2$ increases average token consumption by about 23% (1333.40 $\rightarrow$ 1645.38) and raises total latency from 1.11s to 1.26s. We therefore keep $r = 1$ as the default operating point because it deliberately favors prompt-token efficiency and latency while remaining close to the best accuracy regime. Appendix F explains why the natural radius scales with $(\theta_0/\pi)d$.

5 RELATED WORK

The landscape of memory systems for agentic AI is rapidly evolving, with recent work focusing on high-level architectural abstractions to manage long-term experiences. These approaches can be broadly categorized into two dominant philosophies. The first draws inspiration from operating systems, treating memory as a manageable system resource. This includes MemGPT (Packer et al., 2023), which introduces virtual context management analogous to OS-level memory paging; MemoryOS (Kang et al., 2025), which implements a hierarchical storage architecture with short, mid, and long-term tiers; and MemOS (Li et al., 2025), which proposes a standardized MemCube abstraction to unify parametric, activation, and plaintext memory. The second category is inspired by human cognitive science, such as ReadAgent (Lee et al., 2024), which compresses memories into gist memories (Abadie et al., 2013) akin to human summarization; MemoryBank (Zhong et al., 2024a), which employs an Ebbinghaus-inspired (Tulving, 1985) forgetting curve for dynamic memory updates; and A-mem (Xu et al., 2025), which organizes knowledge into an evolving, interconnected network based on the Zettelkasten method (Malashenko et al., 2023). Despite their architectural diversity, these systems converge on a common technological substrate where retrieval is predominantly powered by dense vector similarity search within a Retrieval-Augmented Generation (RAG) framework or by traversing explicit knowledge graph struc-

tures. For instance, A-mem leverages a vector store like ChromaDB , and MemoryBank uses FAISS (Douze et al., 2025) for efficient retrieval. A more detailed related work is presented in Appendix G.

6 Conclusion

This work presents HIPPOCAMPUS, a contextual memory module built around binary signatures and a Dynamic Wavelet Matrix co-index for compressed-domain search and lossless content reconstruction. For a fixed tokenizer vocabulary, its storage grows linearly with history length, while dynamic construction remains near-linear at $\mathcal{O}(n \log n)$. Across LoCoMo and LongMemEval, HIPPOCAMPUS preserves or improves task accuracy while substantially reducing query latency and prompt-token cost, validating approximate-then-exact retrieval on top of succinct data structures as a practical foundation for long-horizon agentic memory.

References

Abadie, M., Waroquier, L., and Terrier, P. Gist memory in the unconscious-thought effect. *Psychological Science*, 24(7):1253–1259, 2013.

Ali, M. S. and Puri, D. Optimizing devops methodologies with the integration of artificial intelligence. In *2024 3rd International Conference for Innovation in Technology (INOCON)*, pp. 1–5. IEEE, 2024.

Anadiotis, A. C., Khan, M. G., and Manolescu, I. Dynamic graph databases with out-of-order updates. *Proceedings of the VLDB Endowment*, 17(13):4799–4812, 2024.

Anokhin, P., Semenov, N., Sorokin, A., Evseev, D., Kravchenko, A., Burtsev, M., and Burnaev, E. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. *arXiv preprint arXiv:2407.04363*, 2024.

Arora, S., May, A., Zhang, J., and Ré, C. Contextual embeddings: When are they worth it? *arXiv preprint arXiv:2005.09117*, 2020.

Charikar, M. S. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 380–388, 2002.

Chase, H. Langchain: Build context-aware reasoning applications. https://github.com/langchain-ai/langchain, 2022. Accessed: 2025-08-04.

Chen, G., Dong, S., Shu, Y., Zhang, G., Sesay, J., Karlsson, B. F., Fu, J., and Shi, Y. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*, 2023.

Chen, J., Li, H., Yang, J., Liu, Y., and Ai, Q. Enhancing llm-based agents via global planning and hierarchical execution. *arXiv preprint arXiv:2504.16563*, 2025.

Claude, F. and Navarro, G. The wavelet matrix. In *International Symposium on String Processing and Information Retrieval*, pp. 167–179. Springer, 2012.

CrewAI. Core concept: Memory. https://docs.crewai.com/en/concepts/memory, 2025. Accessed: 2025-08-04.

De Cao, N., Aziz, W., and Titov, I. Editing factual knowledge in language models. *arXiv preprint arXiv:2104.08164*, 2021.

de Vries, A. P., Mamoulis, N., Nes, N., and Kersten, M. Efficient k-nn search on vertically decomposed data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pp. 322–333, 2002.

Dietzfelbinger, M. and Pagh, R. Succinct data structures for retrieval and approximate membership. In *International Colloquium on Automata, Languages, and Programming*, pp. 385–396. Springer, 2008.

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., and Jégou, H. The faiss library. *IEEE Transactions on Big Data*, 2025.

Du, Y., Huang, W., Zheng, D., Wang, Z., Montella, S., Lapata, M., Wong, K.-F., and Pan, J. Z. Rethinking memory in ai: Taxonomy, operations, topics, and future directions. *arXiv preprint arXiv:2505.00675*, 2025.

Gog, S. and Petri, M. Optimized succinct data structures for massive data. *Software: Practice and Experience*, 44 (11):1287–1314, 2014.

Gu, J., Jiang, X., Shi, Z., Tan, H., Zhai, X., Xu, C., Li, W., Shen, Y., Ma, S., Liu, H., et al. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024.

Guia, J., Soares, V. G., and Bernardino, J. Graph databases: Neo4j analysis. In *ICEIS (1)*, pp. 351–356, 2017.

Hayes-Roth, B. and Hayes-Roth, F. A cognitive model of planning. *Cognitive science*, 3(4):275–310, 1979.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Huang, M., Long, Y., Deng, X., Chu, R., Xiong, J., Liang, X., Cheng, H., Lu, Q., and Liu, W. Dialoggen: Multi-modal interactive dialogue system for multi-turn text-to-image generation. *arXiv preprint arXiv:2403.08857*, 2024.

Indyk, P. and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, 1998.

Jiang, Y., Wang, Y., Wu, C., Zhong, W., Zeng, X., Gao, J., Li, L., Jiang, X., Shang, L., Tang, R., et al. Learning to edit: Aligning llms with knowledge editing. *arXiv preprint arXiv:2402.11905*, 2024.

Kagaya, T., Yuan, T. J., Lou, Y., Karlekar, J., Pranata, S., Kinose, A., Oguri, K., Wick, F., and You, Y. Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents. *arXiv preprint arXiv:2402.03610*, 2024.

Kanerva, P., Kristoferson, J., and Holst, A. Random indexing of text samples for latent semantic analysis. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 22, 2000.

Kang, J., Ji, M., Zhao, Z., and Bai, T. Memory os of ai agent. *arXiv preprint arXiv:2506.06326*, 2025.

Kim, T., François-Lavet, V., and Cochez, M. Leveraging knowledge graph-based human-like memory systems to solve partially observable markov decision processes. *arXiv preprint arXiv:2408.05861*, 2024.

Kurisinkel, L. J. and Chen, N. F. Llm based multi-document summarization exploiting main-event biased monotone submodular content extraction. *arXiv preprint arXiv:2310.03414*, 2023.

Labib, K., Uznanski, P., and Wolleb-Graf, D. Hamming distance completeness. In *30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019)*, volume 128, pp. 14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019.

LangChain. The rise of "context engineering". http://blog.langchain.com/the-rise-of-context-engineering/, 2025. Accessed: 2025-08-04.

Lee, K.-H., Chen, X., Furuta, H., Canny, J., and Fischer, I. A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint arXiv:2402.09727*, 2024.

Li, B., Wu, P., Abbeel, P., and Malik, J. Interactive task planning with language models. *arXiv preprint arXiv:2310.10645*, 2023a.

Li, G., Hammoud, H., Itani, H., Khizbullin, D., and Ghanem, B. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008, 2023b.

Li, Z., Song, S., Xi, C., Wang, H., Tang, C., Niu, S., Chen, D., Yang, J., Li, C., Yu, Q., et al. Memos: A memory os for ai system. *arXiv preprint arXiv:2507.03724*, 2025.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023a.

Liu, Y., Li, L., Zhang, B., Huang, S., Zha, Z.-J., and Huang, Q. Matcr: Modality-aligned thought chain reasoning for multimodal task-oriented dialogue generation. In *Proceedings of the 31st ACM International Conference on Multimedia*, pp. 5776–5785, 2023b.

Maharana, A., Lee, D.-H., Tulyakov, S., Bansal, M., Barbieri, F., and Fang, Y. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*, 2024.

Malashenko, G. T., Kosov, M. E., Frumina, S. V., Grishina, O. A., Alandarov, R. A., Ponkratov, V. V., Bloshenko, T. A., Sanginova, L. D., Dzusova, S. S., and Hasan, M. F. A digital model of full-cycle training based on the zettelkasten and interval repetition system. *Emerging Science Journal*, 7:1–15, 2023.

Mei, K., Zhu, X., Xu, W., Hua, W., Jin, M., Li, Z., Xu, S., Ye, R., Ge, Y., and Zhang, Y. Aios: Llm agent operating system. *arXiv preprint arXiv:2403.16971*, 2024.

Mei, L., Yao, J., Ge, Y., Wang, Y., Bi, B., Cai, Y., Liu, J., Li, M., Li, Z.-Z., Zhang, D., et al. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*, 2025.

Nakajima, Y. Babyagi: An experimental framework for a self-building autonomous agent. https://github.com/yoheinakajima/babyagi, 2023. Accessed: 2025-08-04.

Norouzi, M., Fleet, D. J., and Salakhutdinov, R. R. Hamming distance metric learning. *Advances in neural information processing systems*, 25, 2012.

Opitz, J. and Burst, S. Macro f1 and macro f1. *arXiv preprint arXiv:1911.03347*, 2019.

Packer, C., Fang, V., Patil, S., Lin, K., Wooders, S., and Gonzalez, J. Memgpt: Towards llms as operating systems. 2023.

Peng, L., Wang, Z., Yao, F., Wang, Z., and Shang, J. Metaie: Distilling a meta model from llm for all kinds of information extraction tasks. *arXiv preprint arXiv:2404.00457*, 2024.

Qu, H., Fan, W., Zhao, Z., and Li, Q. Tokenrec: learning to tokenize id for llm-based generative recommendation. *arXiv preprint arXiv:2406.10450*, 2024.

Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., and Chalef, D. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Seshadri, V., Lee, D., Mullins, T., Hassan, H., Boroumand, A., Kim, J., Kozuch, M. A., Mutlu, O., Gibbons, P. B., and Mowry, T. C. Buddy-ram: Improving the performance and efficiency of bulk bitwise operations using dram. *arXiv preprint arXiv:1611.09988*, 2016.

Shamir, G. I. Universal lossless compression with unknown alphabets—the average case. *IEEE Transactions on Information Theory*, 52(11):4915–4944, 2006.

Singh, A., Ehtesham, A., Kumar, S., and Khoei, T. T. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136*, 2025.

Singh, S. Natural language processing for information extraction. *arXiv preprint arXiv:1807.02383*, 2018.

Srivastava, A. Sense-plan-act in robotic applications. In *Intelligent Robotics Seminar*, pp. 1–8, 2019.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Sun, C. Revisiting popcount operations in cpus / gpus. 2016. URL https://api.semanticscholar.org/CorpusID:5415415.

Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Tulving, E. Ebbinghaus's memory: What did he learn and remember? *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 11(3):485, 1985.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wandji, R. E. and Calvanese, D. Improving the cost of updates in virtual knowledge graphs. In *2024 Joint Ontology Workshops-Episode X: The Tukker Zomer of Ontology, and Satellite Events, JOWO 2024, Enschede, The Netherlands, July 15-19, 2024*. CEUR-WS, 2024.

Wang, S., Zhu, Y., Liu, H., Zheng, Z., Chen, C., and Li, J. Knowledge editing for large language models: A survey. *ACM Computing Surveys*, 57(3):1–37, 2024.

Wu, D., Wang, H., Yu, W., Zhang, Y., Chang, K.-W., and Yu, D. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024a.

Wu, Y., Gu, Y., Feng, X., Zhong, W., Xu, D., Yang, Q., Liu, H., and Qin, B. Extending context window of large language models from a distributional perspective. *arXiv preprint arXiv:2410.01490*, 2024b.

Xin, J., Tang, R., Yu, Y., and Lin, J. The art of abstention: Selective prediction and error regularization for natural language processing. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1040–1051, 2021.

Xu, W., Mei, K., Gao, H., Tan, J., Liang, Z., and Zhang, Y. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

Yang, M., Zhu, J., Li, J., Wang, L., Qi, H., Li, S., and Daxin, L. Extending bleu evaluation method with linguistic weight. In *2008 The 9th International Conference for Young Computer Scientists*, pp. 1683–1688. IEEE, 2008.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Yu, Y.-C., Kuo, C.-C., Ye, Z., Chang, Y.-C., and Li, Y.-S. Breaking the ceiling of the llm community by treating token generation as a classification for ensembling. *arXiv preprint arXiv:2406.12585*, 2024.

Zaib, M., Zhang, W. E., Sheng, Q. Z., Mahmood, A., and Zhang, Y. Conversational question answering: A survey. *Knowledge and Information Systems*, 64(12):3151–3195, 2022.

Zhang, R., Du, H., Liu, Y., Niyato, D., Kang, J., Sun, S., Shen, X., and Poor, H. V. Interactive ai with retrieval-augmented generation for next generation networking. *IEEE Network*, 38(6):414–424, 2024.

Zhong, W., Guo, L., Gao, Q., Ye, H., and Wang, Y. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024a.

Zhong, Z., Liu, H., Cui, X., Zhang, X., and Qin, Z. Mix-of-granularity: Optimize the chunking granularity for retrieval-augmented generation. *arXiv preprint arXiv:2406.00456*, 2024b.

Zhu, Y., Qiao, S., Ou, Y., Deng, S., Lyu, S., Shen, Y., Liang, L., Gu, J., Chen, H., and Zhang, N. Knowagent: Knowledge-augmented planning for llm-based agents. *arXiv preprint arXiv:2403.03101*, 2024.

A LLM-AS-A-JUDGE METRIC

When evaluating a generated answer, we feed both the reference and candidate into the judge prompt (see Listing 1), and have GPT-5 act as the impartial judge to assign a score in the range of $[1, 2, 3, 4, 5]$. The LLM judge supplements F1, BLEU-1, and accuracy, which may overestimate correctness in the edge cases.

As shown in Listing 1, we provide our prompt for LLM-as-a-Judge, which serves as the evaluation prompt for assessing the quality of generated answers. The prompt instructs an impartial evaluator to rate a candidate answer against a reference answer on a $[1, 2, 3, 4, 5]$ scale, focusing on **Correctness**, **Completeness**, and **Clarity/Coherence**. Specifically, a score of 5 indicates a perfectly correct, complete, and clear response; 4 reflects minor inaccuracies or slight omissions; 3 denotes partial correctness with missing major points; 2 corresponds to largely incorrect or irrelevant content; and 1 represents a completely wrong answer. This standardized prompt ensures consistent, interpretable, and reproducible evaluation across different experimental settings.

Listing 1. Judge Prompt Template

```
JUDGE_PROMPT = "
You are an impartial evaluator.
Your task is to rate the quality of a
candidate answer compared to a reference
    answer.
[Question/Query]: {question};
[Reference Answer]: {reference};
[Candidate Answer]: {candidate}.
Please assign a score from 1 to 5 based on
    how well the {candidate} matches
the {reference} in terms of correctness,
    completeness (coverage of key points),
and clarity and coherence.

Scoring Guidelines:
5: perfectly correct, complete, and clear;
4: mostly correct, with minor issues or
    slight omissions;
3: partially correct, with noticeable
    errors or missing major points;
2: largely incorrect, irrelevant, or
    nonsensical;
1: totally wrong.

Only output the final score as an integer
    between 1 and 5.
"
```

As shown in Listing 2, we present the prompt for answering the question, which is used to instruct the model to generate answers strictly based on the provided context. The prompt explicitly constrains the model to avoid relying on external knowledge or prior training data, ensuring that the generated responses are fully grounded in the given information. By including placeholders for the context and question, this design enforces factual consistency and prevents hallucination, making it suitable for controlled evaluations of context-dependent reasoning and information retrieval tasks.

Listing 2. Answer-from-Context Prompt

```
ANSWER_PROMPT = "
Based ONLY on the following context,
    answer the user's question directly.
Context:{context}
Question: {question}
"
```

B DETAILED DATASET DESCRIPTION

We use LoCoMo (Maharana et al., 2024) and LongMemEval (-S and -M) (Wu et al., 2024a) for the experiments. Below is the detailed description of these two benchmarks.

- LoCoMo is introduced to evaluate extremely long-term conversational memory in LLM agents. It is constructed via a machine-human hybrid pipeline: two LLM-powered agents carry multi-session dialogues grounded on persona profiles and temporal event graphs, generating coherent and causally linked conversations which humans then refine for consistency. Each conversation spans up to approximate 32 sessions and contains on the order of 600 turns and $\sim$16K tokens on average. The benchmark supports multiple tasks, including question answering (Zaib et al., 2022), event summarization (Kurisinkel & Chen, 2023), and multimodal dialogue generation (Liu et al., 2023b; Huang et al., 2024), allowing evaluation along dimensions such as single-hop, multi-hop, temporal, and open-domain memory reasoning.
- LongMemEval is a more recent benchmark tailored for chat assistants, designed to probe long-term memory in interactive, multi-session settings. It comprises 500 curated questions, each embedded within a dynamically constructed chat history spanning multiple sessions. The benchmark assesses five core memory abilities: information extraction (Singh, 2018; Peng et al., 2024), multi-session reasoning, temporal reasoning, knowledge updates, and abstention (Xin et al., 2021). During evaluation, models must parse incremental interactions, maintain memory over sessions, and deliver answers after the final session, thereby simulating realistic real-world continual-memory demands.

C OVERALL COMPARISON ON LONGMEMEVAL-M

Accuracy-only results on LongMemEval-M, as shown in Table 4, mirror the trends observed on the LongMemEval-S (Table 2): HIPPOCAMPUS attains the highest or near-highest accuracy across all categories, particularly on multi-session

Table 4. Overall comparison of different memory modules across six tasks in LongMemEval-M benchmark, under the same setting as Table 2.

Memory Module	Single-session-preference			Single-session-assistant			Temporal-reasoning			Multi-session			Knowledge-update			Single-session-user		
	F1	Accuracy	LLM-as-a-Judge	F1	Accuracy	LLM-as-a-Judge	F1	Accuracy	LLM-as-a-Judge	F1	Accuracy	LLM-as-a-Judge	F1	Accuracy	LLM-as-a-Judge	F1	Accuracy	LLM-as-a-Judge
ReadAgent	3.45	0.83	1.15	2.72	5.36	1.45	3.17	1.69	1.04	1.20	1.32	1.08	1.56	2.57	1.19	2.47	6.79	1.06
MemoryBank	4.14	1.00	1.72	3.27	6.43	1.43	3.81	2.03	1.11	1.45	1.58	1.17	1.88	3.08	1.18	2.98	8.14	1.54
MemGPT	4.83	1.17	1.58	3.81	7.50	1.31	4.45	2.37	1.02	1.70	1.84	1.07	2.19	3.59	1.08	3.49	9.50	1.39
A-mem	7.59	1.83	2.30	5.99	11.79	1.91	7.00	3.72	1.49	2.68	2.89	1.57	3.46	5.64	1.58	5.49	14.93	1.94
MemoryOS	8.98	2.16	2.44	7.10	13.93	2.02	8.29	4.40	1.57	3.18	3.42	1.64	4.12	6.67	1.65	6.53	17.64	2.01
MemOS	10.36	2.50	2.58	8.21	16.07	2.14	9.59	5.08	1.65	3.68	3.95	1.72	4.77	7.70	1.73	7.56	20.36	2.16
HIPPOCAMPUS	13.79	3.33	**2.87**	10.88	21.43	**2.38**	12.69	6.77	**1.86**	4.81	5.26	**1.94**	6.23	10.26	**1.98**	8.67	27.14	**2.40**

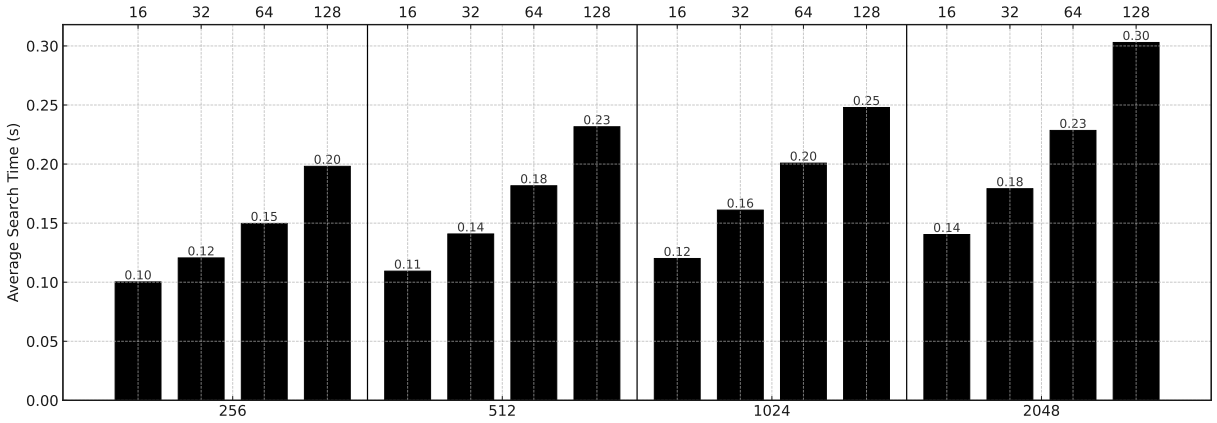

Figure 12. Ablation on the LoCoMo: Average search time versus Random Indexing dimension D and signature size d. Bars within each group (bottom axis) correspond to different d values (top axis).

and knowledge-update where signature-level association helps surface temporally and semantically related memories that are dispersed across sessions. We omit efficiency plots for brevity, the latency and token-consumption advantages follow the same pattern as on LongMemEval-S, since the retrieval substrate (signature filtering and content reconstruction) is identical; hence relative gaps against dense-vector and KG baselines persist at similar magnitudes.

D ABLATION STUDY OF HIPPOCAMPUS

We evaluated the trends on LoCoMo long-memory benchmark by varying the random-index dimension $D \in \{256, 512, 1024, 2048\}$ and binary signature length $d \in \{16, 32, 64, 128\}$. Figure 12 reports the average search time (in seconds), and Table 5 shows quality metrics (F1, BLEU-1, and LLM-as-a-Judge) under these settings.

Avg. Search Time. Retrieval involves computing Hamming distances between the query's binary signature and all stored signatures. Thus search time grows roughly linearly with the signature length d (and weakly with D). In our table, doubling d roughly doubles the time. Each extra bit adds a fixed cost, so larger d or D slows lookup.

Accuracy. Increasing D or d raises the representational capacity of the memory, reducing collisions and improving recall. A higher random indexing dimension D yields more nearly-orthogonal random codes, while a longer signature d captures more bits of information. Consequently, all quality metrics (F1, BLEU-1, and the LLM-as-a-Judge score) improve as D and d grow. This matches the known trends: expanding memory capacity or embedding dimensions consistently boosts retrieval performance (Li et al., 2025).

Trade-off Consideration. There is a clear trade-off. Larger D and d yield diminishing marginal gains in accuracy (the improvements taper off as the system saturates its capacity), but each added dimension/bit linearly increases search effort. In practice, one chooses D, d to balance these effects: enough capacity to achieve good recall accuracy (and thus higher LLM-judge scores), but not so large that retrieval becomes too slow.

D.1 Removing the Signature DWM

A physical ablation that removes the Signature DWM would require a substantial refactor of the retrieval pipeline, because the semantic index is structurally coupled to candidate generation. Nevertheless, its role can be quantified theoretically. Without the Signature DWM, HIPPOCAMPUS would lose its compressed-domain semantic filter and would need to reconstruct token-id spans from the Content DWM before comparing them at the token level. This would replace the current bitwise Hamming scan cost of $\mathcal{O}(n \cdot d/w)$ with token-level reconstruction whose dominant factor is $\mathcal{O}(n \log \sigma)$ from repeated ACCESS operations, where typically $d/w \ll \log \sigma$. More importantly, retrieval would regress from approximate semantic matching to largely exact token matching, the regime in which

Table 5. LoCoMo ablation: F1/BLEU-1/LLM-as-a-Judge vs. Random Indexing D and signature size d.

D	d	F1 (%)	BLEU-1 (%)	LLM-Judge
256	16	37.28	33.69	3.18
256	32	38.27	34.53	3.17
256	64	37.29	33.62	3.20
256	128	37.54	34.20	3.20
512	16	38.14	34.42	3.17
512	32	38.30	33.78	3.18
512	64	37.75	33.45	3.17
512	128	38.16	34.42	3.21
1024	16	38.72	33.83	3.20
1024	32	38.18	34.49	3.20
1024	64	37.31	33.63	3.19
1024	128	37.59	33.46	3.19
2048	16	37.79	33.82	3.18
2048	32	38.66	34.20	3.21
2048	64	38.35	33.54	3.19
2048	128	38.21	34.08	3.21

keyword-centric memories suffer substantial answer-quality degradation. Thus, Signature DWM is not merely an accelerator, it is the component that simultaneously provides semantic recall and latency advantage of HIPPOCAMPUS.

E TIME AND SPACE COMPLEXITY

We distinguish between two notions of scaling. For a fixed tokenizer vocabulary of size σ, the DWM stores an $l \times n$ bit-matrix with $l = \lceil \log_2 \sigma \rceil$, so the storage footprint is $\mathcal{O}(n \log \sigma)$ bits, linear in the number of stored tokens n. Construction, however, is not linear-time: because each append updates dynamic bit-vectors, building the matrix over n tokens costs near-linear $\mathcal{O}(n \log n)$ time.

Table 6 makes the distinction concrete. A $10\times$ increase in stored tokens from 16K to 160K raises end-to-end construction time by about $12.6\times$ (32s $\rightarrow$ 402s), which matches the expected near-linear $\mathcal{O}(n \log n)$ trend. In contrast, the storage footprint grows linearly, from 0.15MB to 1.46MB.

We also ground query-time scalability empirically. On LoCoMo (roughly 160K total tokens across the 10 conversations), HIPPOCAMPUS achieves an average total retrieval time of about 1.08s, with the search phase accounting for about 12% of that time. On LongMemEval-S, which is substantially larger, HIPPOCAMPUS still averages about 2.08s end-to-end, with the search phase accounting for about 29%. Although these are different benchmarks rather than a controlled scaling curve, they show that query latency remains on the order of seconds in the 100K-token regime.

Let n be the number of stored tokens, σ the tokenizer vocabulary size, and d the binary-signature length. **DWM Construction (memory insertion).** Each insertion touches all $l = \lceil \log_2 \sigma \rceil$ levels. With dynamic rank/select-supported bit-vectors, this is $\mathcal{O}(l \log n)$ per token and therefore $\mathcal{O}(n \log n)$ total for fixed σ. Space remains $nl + o(nl)$ bits, i.e., $\mathcal{O}(n \log \sigma)$ bits. **DWM Query (retrieval).** Exact navigation via $rank/select/access$ follows the standard WM logic, while the Hamming-ball scan over stored signatures costs $\mathcal{O}(n \cdot d/w)$ using machine-word size w. Since d is small and fixed in practice, query time is linear in n with a low bitwise constant. **Dense-vector ANN (e.g., FAISS).** A brute-force k-NN search in D dimensions costs $\mathcal{O}(nD)$ per query, and practical ANN indices trade preprocessing and index overhead for lower average-time retrieval. **Knowledge-graph traversal.** Multi-hop exploration can expand rapidly with the branching factor and remains at least linear in the size of the explored subgraph. **Comparison.** Asymptotically, HIPPOCAMPUS combines linear-in-n storage (for fixed σ) with near-linear $\mathcal{O}(n \log n)$ construction and linear-in-n query scanning with a small bit-level factor d/w, thereby avoiding both floating-point vector comparisons and multi-hop graph expansion.

F ACCURACY GAP AND PRACTICAL GUIDANCE FOR CHOOSING r

We formalize how HIPPOCAMPUS 's random indexing step followed by an r-bit Hamming ball search approximates dense-vector similarity. Let each token's context embedding be a vector $\boldsymbol{v} \in \mathbb{R}^D$, and consider two such vectors $\boldsymbol{v}, \boldsymbol{w}$. HIPPOCAMPUS generates a d-bit signature by random projection and thresholding: each bit is $\text{sign}(\langle b_k, v \rangle)$ for some random hyperplane b_k (or an analogous sparse random base-vector scheme). By known results for random hyperplane hashing, the probability that a single bit differs satisfies:

$$P(\text{bit}_k(\boldsymbol{v}) \neq \text{bit}_k(\boldsymbol{w})) = \frac{\theta}{\pi}$$

where $\theta = \arccos\left(\frac{\boldsymbol{v} \cdot \boldsymbol{w}}{|\boldsymbol{v}||\boldsymbol{w}|}\right)$. Thus the expected Hamming distance between the d-bit signatures is

$$\mathbb{E}[\text{Ham}(\boldsymbol{v}, \boldsymbol{w})] = d \cdot \frac{\theta}{\pi}$$

Equivalently, similarity $\frac{\boldsymbol{v} \cdot \boldsymbol{w}}{|\boldsymbol{v}||\boldsymbol{w}|} = \cos\theta$ can be recovered up to small error from the normalized Hamming similarity.

With d independent bits, the law of large numbers gives concentration: for any $\varepsilon > 0$, by Hoeffding's bound:

$$P(|\frac{1}{d}\text{Ham}(\boldsymbol{v}, \boldsymbol{w}) - \frac{\theta}{\pi}| \geq \epsilon) \leq 2e^{-2d\epsilon^2}$$

Hence with high probability, $\frac{1}{d}\text{Ham}(\boldsymbol{v}, \boldsymbol{w})$ is within $\mathcal{O}(1/\sqrt{d})$ of θ/π. In practice, choosing $d = \mathcal{O}(\epsilon^{-2} \log N)$

Table 6. Empirical scaling of DWM construction on LoCoMo as the total number of stored tokens grows from 16K to 160K.

#Tokens	16K	32K	48K	64K	80K	96K	112K	128K	144K	160K
Time (s)	32	70	108	149	189	231	273	316	359	402
Footprint (MB)	0.15	0.31	0.44	0.59	0.77	0.87	1.02	1.16	1.27	1.46

ensures that for any fixed query among N candidates, the Hamming distance will approximate the original cosine similarity within additive error ϵ.

Concretely, if we set a Hamming threshold r corresponding to a desired angle θ_0 (thus target similarity $\cos\theta_0$), then any $\boldsymbol{w}$ with $\frac{\boldsymbol{v}\cdot\boldsymbol{w}}{|\boldsymbol{v}||\boldsymbol{w}|} \geq \cos\theta_0$ will satisfy $\text{Ham}(\boldsymbol{v}, w) \leq r$ except with probability at most $e^{-\mathcal{O}(d)}$. Conversely, vectors with similarity below $\cos\theta_0$ will exceed the threshold with high probability. This establishes that HIPPOCAMPUS 's random indexing plus Hamming ball filter retrieves all sufficiently similar vectors (within angle θ_0) with bounded false-negative probability, and rejects dissimilar vectors, mirroring an approximate nearest-neighbor search in cosine similarity space. The sampling complexity matches known bounds for binary embeddings.

Theorem. Under the process in Section 3.3, for any two vectors $\boldsymbol{v}, \boldsymbol{w}$, the Hamming distance of their d-bit signatures concentrates around its mean $d\theta/\pi$. By choosing $d = \mathcal{O}(\delta^{-2}\log(1/\eta))$, one ensures $\text{Ham}(\boldsymbol{v}, \boldsymbol{w})/d$ approximates θ/π within $\pm\delta$ with probability $1-\eta$. In particular, setting the Hamming radius $r = \frac{d\theta_0}{\pi}$, the Hamming-ball $s : \text{Ham}(\boldsymbol{s}_v, \boldsymbol{s}) \leq r$ contains all items with cosine similarity at least $\cos(\theta_0)$ up to vanishing error.

Proof. HIPPOCAMPUS compresses high-dimensional embeddings into fixed-length binary signatures by sparse random indexing and binarization. In effect, each bit of a token signature can be viewed as the sign of a random hyperplane dot-product with the original vector. The Hamming distance between two signatures then equals the number of bits on which they differ. We will show that this Hamming distance concentrates around $(\theta/\pi)d$, where $\theta = \arccos\frac{\boldsymbol{v}\cdot\boldsymbol{w}}{|\boldsymbol{v}||\boldsymbol{w}|}$ is the angle between vectors $\boldsymbol{v}, \boldsymbol{w}$. In particular, by choosing a suitable radius $r \approx (\theta_0/\pi)d$ we retrieve all vectors with angle $\leq \theta_0$ (cosine similarity $\geq \cos\theta_0$) with high probability.

Binary hash and collision probability. For each bit index $i = 1, \ldots, d$, pick an independent random Gaussian vector $\boldsymbol{r}_i \sim N(0, I)$ in $\mathbb{R}^n$ and define the bit $b_i(v) = \text{sign}(\boldsymbol{r}_i \cdot \boldsymbol{v}) \in 0, 1$. (Equivalently, HIPPOCAMPUS selects a sparse random base vector and later binarizes the top-d components, which yields the same analysis.) Let X_i be the indicator that $b_i(v) \neq b_i(w)$ (a bit mismatch). It is a known fact (Charikar, 2002) that for any two vectors $\boldsymbol{v}, \boldsymbol{w}$ at angle θ, the probability their signs agree on a random hyperplane is

$$P(b_i(v) = b_i(w)) = 1 - \frac{\theta}{\pi} \Rightarrow P(X_i = 1) = \frac{\theta}{\pi}$$

considering a random line in the 2D plane of $\boldsymbol{v}, \boldsymbol{w}$ Thus $X_i \sim \text{Bernoulli}(p)$ with $p = \theta/\pi$, and the Hamming distance $H(\boldsymbol{v}, \boldsymbol{w}) = \sum_{i=1}^{d} X_i$ is a binomial random variable with mean

$$\mathbb{E}[H(\boldsymbol{v}, \boldsymbol{w})] = dp = \frac{\theta}{\pi}d$$

Concentration (Hoeffding/Chernoff bound). The X_i are independent and bounded in $[0, 1]$, so by Hoeffding's inequality, we have for any $\epsilon > 0$:

$$P(|H(\boldsymbol{v}, \boldsymbol{w}) - dp| \geq \epsilon d) \leq 2e^{-2\epsilon^2 d}$$

Equivalently:

$$P(|\frac{H(\boldsymbol{v}, \boldsymbol{w})}{d} - p| \geq \epsilon) \leq 2e^{-2\epsilon^2 d}$$

Hence with high probability $H(v, w)/d$ lies in the interval $[p - \epsilon, ; p + \epsilon]$, i.e.

$$H(\boldsymbol{v}, \boldsymbol{w}) = (\frac{\theta}{\pi})d \pm \epsilon d \quad \text{with probability} 1 - 2e^{-2\epsilon^2 d}$$

Containment in the Hamming ball (false negatives). Fix a target angle θ_0 (so we want $\cos(\boldsymbol{v}, \boldsymbol{w}) \geq \cos\theta_0$). Consider any vector w with $\theta(\boldsymbol{v}, \boldsymbol{w}) \leq \theta_0$. Then $p = \theta/\pi \leq \theta_0/\pi$. Define a search radius: $r = (\frac{\theta_0}{\pi} + \epsilon) + d$, by the above tail bound:

$$P(H(\boldsymbol{v}, \boldsymbol{w}) > r) = P(\frac{H}{d} > \frac{\theta_0}{\pi} + \epsilon) \leq e^{-2\epsilon^2 d}$$

since $\mathbb{E}[H/d] \leq \theta_0/\pi$. Thus with probability at least $1 - e^{-2\epsilon^2 d}$ we have $H(v, w) \leq r$. By choosing d large enough (see below), this failure probability can be made $\leq \delta/N$ (union-bounding over N candidates). In summary, any vector within angle θ_0 will lie inside the Hamming ball of radius $r \approx (\theta_0/\pi)d$ with high probability.

False positives (outside angle). Conversely, if $\theta(\boldsymbol{v}, \boldsymbol{w}) > \theta_0$, then $p = \theta/\pi > \theta_0/\pi$. In particular, if $\theta \geq \theta_0 + 2\epsilon\pi$ then $p \geq \theta_0/\pi + 2\epsilon$. In that case:

$$P(H(\boldsymbol{v}, \boldsymbol{w}) \leq (\frac{\theta_0}{\pi} + \epsilon)d) = P(\frac{H}{d} \leq p - \epsilon) \leq e^{-2\epsilon^2 d}$$

by the lower-tail Hoeffding bound. Hence vectors with angle substantially above θ_0 will (with probability $1 - \exp(-2\epsilon^2 d)$) have Hamming distance exceeding $(\theta_0/\pi + \epsilon)d$ and will not be included in the ball of radius $r =$

$(\theta_0/\pi + \epsilon)d$. This bounds the false-positive rate.

Parameter choice (d vs. ϵ, δ, N). To guarantee both error probabilities $e^{-2\epsilon^2 d}$ are at most $\delta/(2N)$ (so that a union bound over N vectors still yields failure probability $\leq \delta$), it suffices to choose $d \geq \frac{1}{2\epsilon^2}\ln(\frac{2N}{\delta})$. In big-O terms, $d = O\big((1/\epsilon^2)(\log N + \log(1/\delta))\big)$ is enough. For such d, we have with probability $1-\delta$ (over the randomness of the projections) that all vectors within angle θ_0 lie in the Hamming ball of radius $r = (\theta_0/\pi + \epsilon)d$, and vectors with angle significantly larger than θ_0 lie outside this ball.

The above calculation shows that the normalized Hamming distance $H(\boldsymbol{v}, \boldsymbol{w})/d$ concentrates near (θ/π). Hence a Hamming-ball query of radius $r \approx (\theta_0/\pi)d$ retrieves exactly those vectors with angle $\leq \theta_0$ (cosine $\geq \cos\theta_0$), up to an error margin controlled by ϵ, δ. In other words, HIPPOCAMPUS 's random indexing plus Hamming ball method yields an (ϵ, δ)-approximation of cosine-similarity search: with $d = O((1/\epsilon^2)\log(N/\delta))$ bits, one finds all high-cosine neighbors with bounded false-positive/negative rates.

Practical guidance. Given a desired cosine threshold $\cos\theta_0$, first compute $\theta_0 = \arccos(\cos\theta_0)$ and then set $r \approx (\theta_0/\pi)d$. If recall is more important than precision, add a small positive margin ϵ and use $r = (\theta_0/\pi + \epsilon)d$; if prompt-token efficiency is more important, reduce r. Table 3 shows this trade-off empirically on LoCoMo.

G Detailed Related Work

G.1 Memory System for Agentic AI

The design of memory modules for agentic AI has become a central research area, with a primary focus on developing high-level architectural frameworks that enable agents to store, organize, and recall past experiences effectively. A close examination of the state-of-the-art reveals that innovation has largely concentrated on the conceptual layer of memory management, how an agent should reason about its history, while relying on a common set of underlying retrieval technologies.

A prominent school of thought approaches agent memory by drawing analogies to the memory management principles of traditional operating systems (OS), emphasizing concepts like hierarchy, resource allocation, and control flow.

MemGPT (Packer et al., 2023) pioneers the concept of virtual context management for LLMs. This technique provides the illusion of an infinite context window by creating a two-tiered memory hierarchy. The main context is analogous to physical RAM and consists of the tokens directly within the LLM's prompt, while the external context serves as disk storage for out-of-context information. The core mechanism of MemGPT is that the LLM itself orchestrates the movement of data between these tiers through self-directed function calls, effectively managing its own limited context as a constrained resource.

MemoryOS (Kang et al., 2025) extends this OS metaphor with a more rigidly defined three-tier hierarchical storage architecture: Short-Term Memory (STM) for real-time conversations, Mid-Term Memory (MTM) for topic-based summaries, and Long-term Personal Memory (LPM) for persistent user and agent personas. It formalizes the data lifecycle with explicit update policies borrowed from OS design, such as a dialogue-chain-based FIFO (First-In, First-Out) principle for promoting information from STM to MTM and a heat-based replacement strategy for archiving less relevant information from MTM, mirroring OS page management techniques.

MemOS (Li et al., 2025) presents the most abstract and comprehensive OS-level vision, proposing that memory should be treated as a first-class operational resource within the AI system. Its central innovation is the MemCube, a standardized data structure and abstraction layer designed to unify three fundamentally different memory types: parametric memory (knowledge encoded in model weights), activation memory (transient states like the KV-cache), and plaintext memory (external knowledge sources). By providing a unified framework for the full lifecycle of these memory units, including their creation, scheduling, and evolution, MemOS aims to imbue LLMs with system-level controllability, plasticity, and evolvability.

A second major approach draws inspiration from psychology and cognitive science, seeking to build memory systems that emulate the more nuanced and adaptive characteristics of human memory.

ReadAgent (Li et al., 2023a) is modeled on how humans read and comprehend very long documents. Instead of attempting to process entire texts verbatim, it implements a system that creates short, compressed gist memories. This design is grounded in the fuzzy-trace theory of human memory, which posits that humans quickly forget precise details but retain the core substance or gist of information for much longer. ReadAgent uses the LLM's own reasoning capabilities to decide what content to group into a memory episode, how to compress it into a gist, and when to perform an interactive look-up of the original text for specific details, transforming retrieval into an active reasoning task.

MemoryBank (Zhang et al., 2024) explicitly incorporates a model of human forgetting to achieve more natural long-term interactions. Its memory update mechanism is directly inspired by the Ebbinghaus Forgetting Curve theory, a psychological principle describing the decay of memory over time. This allows the agent to selectively forget less significant or infrequently accessed memories while reinforcing more important ones, aiming for a more anthropomorphic and engaging user experience, particularly in long-term AI companion scenarios. Its storage is also hierarchical, distill-

ing verbose dialogues into concise daily summaries, which are then aggregated into a global summary.
A-mem (Xu et al., 2025) is architected around the principles of the Zettelkasten method, a sophisticated technique for knowledge management that emphasizes the creation of a network of interconnected atomic notes. When a new memory is formed, A-mem uses an LLM to generate a structured note containing attributes like keywords, tags, and a rich contextual description. The system then agentically analyzes historical memories to establish meaningful links, creating an evolving web of knowledge. This process also enables memory evolution, where the integration of new information can trigger updates to the attributes of existing memories, allowing the network to continuously refine its understanding over time.

A distinct architectural approach structures memory explicitly as a knowledge graph (KG), which excels at representing the relational and temporal dependencies between entities. While many systems rely on vector search for amorphous semantic similarity, KGs provide a structured representation that is particularly well-suited for tasks requiring multi-hop reasoning or a precise understanding of how the information evolves.
Zep and Graphiti (Rasmussen et al., 2025) exemplify this approach. Zep is a memory layer service for agents that is powered by Graphiti, a temporally-aware knowledge graph engine. Unlike static RAG systems that retrieve from unchanging document collections, Graphiti dynamically ingests and synthesizes both unstructured conversational data and structured business data into a KG that explicitly maintains historical relationships and their periods of validity. This bi-temporal model, which tracks both event time and transaction time, enables agents to perform complex temporal reasoning queries (e.g., "What was the status of Project X last week?"), a capability that is fundamentally challenging for standard vector-based RAG systems.

G.2 Succinct Data Structures

The core data structure of HIPPOCAMPUS: Dynamic Wavelet Matrix, is rooted in the field of succinct data structures, a specialized area of computer science focused on high-performance information retrieval in space-constrained environments.

Succinct data structures are data representations that occupy an amount of space that is very close to the information-theoretic minimum required to store the data, while still supporting efficient queries. For example, a binary tree with n nodes requires at least $2n$ bits to be represented uniquely, and succinct representations achieve this bound while still allowing for navigation operations (e.g., finding a parent or child) in constant time. A crucial feature that distinguishes them from simple compression algorithms is that they are designed to be queried directly in their compressed form, without needing to be decompressed first. This combination of extreme space efficiency and fast query performance makes them ideal for managing massive datasets that must be held in memory. This field has matured from purely theoretical results to practical, highly-engineered libraries such as SDSL.

The Wavelet Matrix is a powerful and flexible succinct data structure designed to represent long sequences of symbols, such as a stream of integers drawn from a fixed alphabet. It is an optimized and more practical implementation of the conceptual Wavelet Tree. Structurally, it reorganizes the bits of the symbols in the input sequence into a collection of bit-vectors, where each bit-vector corresponds to a specific bit-plane of the alphabet (e.g., the most significant bits of all symbols form the first bit-vector, the second-most significant bits form the second, and so on).
By augmenting these bit-vectors with small auxiliary structures that allow for constant-time binary *rank* and *select* operations, the Wavelet Matrix can efficiently support three fundamental queries on the original sequence in time logarithmic in the alphabet size ($\mathcal{O}(log\ \sigma)$):
access(i): Returns the original symbol at position i;
rank(c, i): Counts the number of occurrences of symbol c in the prefix of the sequence up to position i.
select(c, j): Finds the position of the j-th occurrence of symbol c in the sequence.
These primitives are the computational building blocks used by HIPPOCAMPUS. However, canonical wavelet matrices are static, they are built once over a fixed dataset and do not support efficient updates.

A key technical contribution of our work is the development of a Dynamic Wavelet Matrix (DWM), an append-friendly adaptation specifically designed to handle the high-throughput, continuously growing memory stream of an agentic system. Furthermore, the application of this structure to co-index two heterogeneous data streams: compact semantic signatures for search and lossless token-IDs for reconstruction, is a novel use case that extends the traditional application of wavelet matrices in information retrieval.