MemTensor

MemOS: A Memory OS for AI System

Zhiyu Li[1,3], **Shichao Song**[1,8], **Chenyang Xi**[1], **Hanyu Wang**[1,8], **Chen Tang**[1], **Simin Niu**[1,8], **Ding Chen**[10], **Jiawei Yang**[1,8], **Chunyu Li**[1], **Qingchen Yu**[1,9], **Jihao Zhao**[1,8], **Yezhaohui Wang**[1], **Peng Liu**[8], **Zehao Lin**[1,3], **Pengyuan Wang**[1], **Jiahao Huo**[1], **Tianyi Chen**[1,2], **Kai Chen**[1,3], **Kehang Li**[1,2], **Zhen Tao**[8], **Junpeng Ren**[1], **Huayi Lai**[1], **Hao Wu**[1], **Bo Tang**[1], **Zhenren Wang**[7,3], **Zhaoxin Fan**[9], **Ningyu Zhang**[5], **Linfeng Zhang**[2], **Junchi Yan**[2], **Mingchuan Yang**[10], **Tong Xu**[6], **Wei Xu**[8], **Huajun Chen**[5], **Haofeng Wang**[4], **Hongkang Yang**[1,3], **Wentao Zhang**[7,†], **Zhi-Qin John Xu**[2,†], **Siheng Chen**[2,†], **Feiyu Xiong**[1,3,†]

[1]**MemTensor (Shanghai) Technology Co., Ltd.**, [2]**Shanghai Jiao Tong University**, [3]**Institute for Advanced Algorithms Research, Shanghai**, [4]**Tongji University**, [5]**Zhejiang University**, [6]**University of Science and Technology of China**, [7]**Peking University**, [8]**Renmin University of China**, [9]**Beihang University**, [10]**Research Institute of China Telecom**

Abstract

Large Language Models (LLMs) have become an essential infrastructure for Artificial General Intelligence (AGI), yet their lack of well-defined memory management systems hinders the development of long-context reasoning, continual personalization, and knowledge consistency.Existing models mainly rely on static parameters and short-lived contextual states, limiting their ability to track user preferences or update knowledge over extended periods.While Retrieval-Augmented Generation (RAG) introduces external knowledge in plain text, it remains a stateless workaround without lifecycle control or integration with persistent representations.Recent work has modeled the training and inference cost of LLMs from a memory hierarchy perspective, showing that introducing an explicit memory layer between parameter memory and external retrieval can substantially reduce these costs by externalizing specific knowledge [1]. Beyond computational efficiency, LLMs face broader challenges arising from how information is distributed over time and context, requiring systems capable of managing heterogeneous knowledge spanning different temporal scales and sources. To address this challenge, we propose MemOS, a memory operating system that treats memory as a manageable system resource. It unifies the representation, scheduling, and evolution of plaintext, activation-based, and parameter-level memories, enabling cost-efficient storage and retrieval. As the basic unit, a MemCube encapsulates both memory content and metadata such as provenance and versioning. MemCubes can be composed, migrated, and fused over time, enabling flexible transitions between memory types and bridging retrieval with parameter-based learning. MemOS establishes a memory-centric system framework that brings controllability, plasticity, and evolvability to LLMs, laying the foundation for continual learning and personalized modeling.

Author Legend: †Correspondence

Project Website: https://memos.openmem.net/

Code: https://github.com/MemTensor/MemOS

1 Introduction

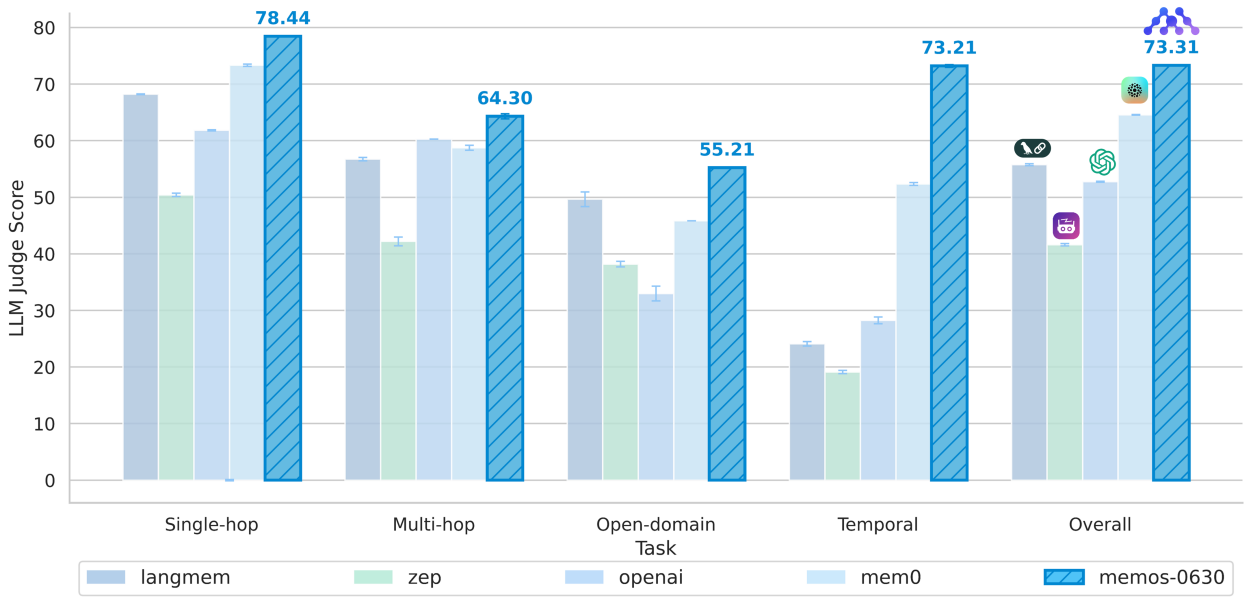

Figure 1 MemOS achieves state-of-the-art performance across all reasoning tasks. This figure summarizes LLM-Judge scores on the LOCOMO benchmark, covering four task categories (Single-hop, Multi-hop, Open-domain, Temporal Reasoning) and the overall average. MemOS (MemOS-0630) consistently ranks first in all categories, outperforming strong baselines such as mem0, LangMem, Zep, and OpenAI-Memory, with especially large margins in challenging settings like multi-hop and temporal reasoning. Error bars indicate standard deviation. Full metric breakdown is provided in Table 3.

With the advent of the Transformer architecture and the maturation of self-supervised pretraining, Large Language Models (LLMs) have become the cornerstone of modern NLP. Trained on large-scale corpora, LLMs exhibit near-human performance in open-domain QA, text generation, and summarization tasks [2]. With increasing model size and compute, their capabilities have expanded to structured code generation [3], cross-modal reasoning [4], multi-turn dialogue, and complex planning—positioning LLMs as a leading paradigm toward Artificial General Intelligence (AGI).

Looking ahead, the presence of LLMs, or more generally, AGI systems, will expand vastly in both time and space. Temporally, models will shift from stateless, session-based tools to persistent agents embedded in long-running workflows. Much like humans, they will need to accumulate interaction histories, adapt internal states, and reason over extended contexts. Spatially, LLMs are evolving into foundational intelligence layers across users, platforms, and ecosystems. Whether deployed in cloud services or embedded in enterprise systems, they must support consistency, adaptability, and personalization across users, roles, and tasks. As such omnipresence becomes the norm, a critical challenge emerges: how should knowledge be organized, stored, and retrieved?

With expanding interaction histories, models face a potentially unbounded context space. We anticipate that future LLMs will seek to leverage as much of their accessible temporal and spatial context as possible, to support deeper reasoning, decision-making, and adaptation. No longer reprocessing all past information per inference, they will decide what to retain, compress, discard, or prioritize. In this always-on paradigm, memory becomes a necessity, not an add-on, for maintaining coherent behavior and identity over time. This requires efficient management of large-scale, multi-source information and dynamic scheduling of memory conditioned on context. This motivates a layered memory hierarchy, similar to how OSs manage memory, consisting of working memory, long-term storage, and cold archives, governed by recency, access frequency, and importance. Sharing memory across users and agents requires scoping, permission control, and migratable, reusable representations. These capabilities are vital not only for system efficiency, but for the long-term

sustainability of model-based knowledge evolution.

The management of memory will become model-defined instead of human-defined. Just as deep learning replaced feature engineering, the transition of memory management from hard-coded pipelines (e.g., RAG) to learnable strategies is natural and necessary. Future agents will autonomously decide whether to retrieve memory, summarize interaction into reusable rules, abstract preferences, or transfer knowledge across contexts. In essence, models must take on the responsibility of shaping their own memory architectures and strategies. Yet, existing infrastructures fall short of enabling this shift.

Mainstream LLMs rely on implicit **parameter memory**, encoding knowledge in billions or trillions of model weights. While this approach affords generalization, it suffers from high update cost, poor interpretability, and limited flexibility. Retraining or fine-tuning requires significant computational resources and risks issues such as catastrophic forgetting.

To address this bottleneck, Retrieval-Augmented Generation (RAG) has emerged as a popular augmentation strategy. By incorporating external retrieval modules, RAG allows models to dynamically access fresh information at inference time, enabling augmentation without parameter updates [5–11]. It is now widely deployed in systems such as Copilots [12] and enterprise search [13–16]. Nonetheless, RAG remains fundamentally an "on-the-fly retrieval and transient composition" pipeline, rather than an integrated memory management system. It lacks core memory manageability features such as lifecycle tracking, versioning, and permission-aware scheduling, limiting its ability to support long-term, adaptive knowledge systems. As a result, models continue to exhibit short-memory behavior in multi-turn dialogue, planning, and personalization tasks, struggling to maintain behavioral consistency or long-horizon adaptation.

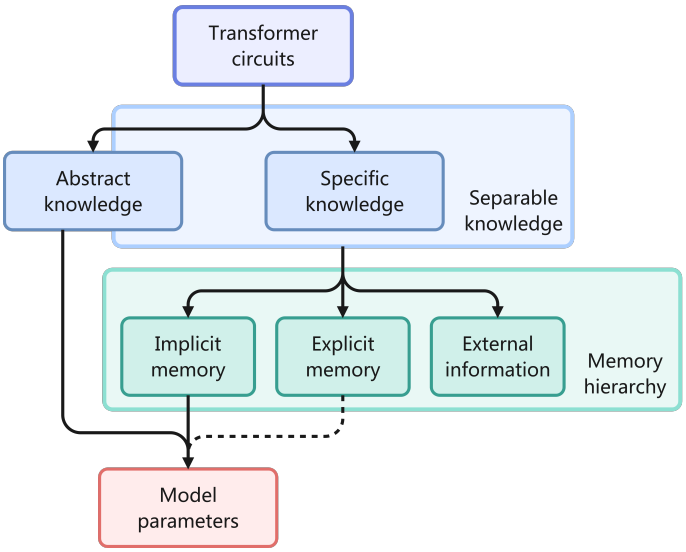

Figure 2 Categorization of LLM knowledge, including the memory hierarchy. The explicit memories, extracted from model activations, lie half-way between raw data and model parameters, so a dotted line is used to indicate that they may or may not be regarded as parameters. Reproduced from [1].

Recent work has shown that the limitations of current memory mechanisms are not incidental, but stem from the architectural absence of explicit and hierarchical memory representations within LLMs. For example, [1] argues that without an intermediate explicit memory layer bridging external retrieval and parametric storage, models become suboptimal in terms of read-write cost, and cannot balance storage cost against retrieval efficiency. This distinction is illustrated in Figure 2, which categorizes knowledge and memory formats and highlights the intermediate role of explicit memory.

From a systems perspective, neither parametric memory nor RAG treats memory as a schedulable and evolvable system resource. This structural gap remains a core bottleneck preventing LLMs from becoming persistent and collaborative intelligent agents. As application scenarios grow more complex, these limitations

become particularly evident in the following four typical contexts.

- **Long-range Dependency Modeling**: As tasks and dialogues grow in length, models must preserve instruction and state consistency across multiple turns or stages. However, current Transformer architectures face three major obstacles: limited context windows constrain input capacity, quadratic attention cost leads to high compute overhead, and user instructions often detach from model behavior over long horizons. For example, in complex tasks, user-defined code structures or writing styles are frequently forgotten, and model outputs revert to default modes. As LLMs are deployed in multi-turn dialogue, long-form generation, and persistent workflows, long-context—and even infinite-context—will become a general requirement rather than a rare exception. This limitation indicates the lack of mechanisms for persistent state maintenance and structured context retention.
- **Adapting to Knowledge Evolution**: Real-world knowledge evolves continuously (e.g., legal updates, scientific discoveries, current events), but static parameters prevent timely reflection. RAG allows dynamic retrieval, yet remains a stateless patching mechanism lacking unified versioning, provenance, or temporal awareness. For instance, it may cite outdated and new regulations simultaneously without reconciliation. It cannot retire obsolete facts, prioritize reliable ones, or track knowledge evolution—limiting long-term consistency.
- **Personalization and Multi-role Support**: LLMs lack durable "memory traces" across users, roles, or tasks. Each session resets to a blank state, ignoring accumulated preferences or styles. Although tools like ChatGPT and Claude now offer memory, issues persist: capacity limits, unstable access, opaque updates, and missing editability. Current systems emphasize passive recording over structured control, making them ill-suited for long-term personalization across diverse use cases.
- **Cross-platform Memory Migration and Ecosystem Diversity**: As LLMs expand from single interfaces to multi-end deployments (web, mobile, enterprise), user memories (e.g., profiles, task history, preferences) should persist across contexts. Yet most systems trap memory within specific instances, forming "memory islands." For example, ideas explored in ChatGPT [17] can't carry over to Cursor [18], forcing context rebuilding. This impairs continuity and blocks memory reuse. Deeper yet, centralization vs. decentralization poses a systemic challenge: while monopolized platforms benefit from feedback loops, distributed models risk stagnation. Making memory portable and reusable is key to balancing evolution efficiency with ecosystem diversity.

A review of the four challenges reveals a shared pattern: models lack the ability to coherently manage and coordinate information distributed across time and space. This is not due to any single failing module, but to the absence of a system-level mechanism for organizing and operating over memory.

Modern LLMs lack an intermediate layer between parametric storage and external retrieval, making it difficult to manage memory lifecycle, integrate evolving knowledge, or maintain behavioral continuity. While RAG provides access to external information, its lack of unified structure and operational semantics prevents long-term, controllable use of knowledge.

Therefore, we argue that building future-capable language intelligence systems requires treating memory as a system-level resource that can be explicitly modeled and scheduled. In modern operating systems, computational resources (CPU), storage (RAM/disks), and communication (I/O) are uniformly scheduled and managed across their lifecycle. In contrast, memory in large model architectures exists as implicit parameters or temporary retrievals—neither schedulable nor traceable, and incapable of integration or transfer. Therefore, the key to enhancing memory in LLMs is not simply "adding a cache" or "attaching an external retrieval module," but redefining the operational logic and resource management of memory from a systems-level perspective.

To address these challenges, we propose MemOS (Memory Operating System), a dedicated memory operating system designed for large language models. The core philosophy of MemOS is that, in order to fully utilize temporally and spatially distributed information, models require a unified framework for organizing memory, maintaining internal state, and supporting long-term adaptation.

Inspired by recent work on memory hierarchy for improving model efficiency and adaptability [1], MemOS

extends this idea into a system-level design by modeling memory as schedulable and evolvable resource units. It builds a modular architecture around the memory lifecycle—including generation, activation, fusion, archiving, and expiration—supported by components such as `MemReader`, `MemScheduler`, `MemLifecycle`, and `MemOperator`, which together orchestrate memory flow, state transitions, and access control.

Much like traditional operating systems coordinate CPU, memory, and I/O, MemOS provides an abstraction layer and unified `Memory API`, enabling consistent and auditable access to memory units across users, tasks, and sessions. The system supports structured storage, provenance tagging, lifecycle tracking, and fine-grained permission enforcement, forming a scalable foundation for memory-driven reasoning. More importantly, MemOS lays a cognitive foundation for the next generation of AGI systems with long-term memory and continual evolution, and provides efficient infrastructure for memory-centric architectural innovation.

The system provides three core capabilities:

- **Controllability**: MemOS offers full lifecycle management of memory units, enabling unified scheduling of memory creation, activation, fusion, and disposal. It implements multi-level permission control and context-aware activation strategies, ensuring safety and traceability in multi-task and multi-user environments through access control and operation auditing. For instance, user preference memories can be scoped to specific agent instances and automatically expire or archive after task completion.
- **Plasticity**: MemOS supports memory restructuring and migration across tasks and roles. It provides memory slicing, tagging, hierarchical mapping, and context binding capabilities, allowing developers or systems to construct highly adaptable memory structures based on inference objectives. This enables models to activate different memory views for different tasks or update memory associations dynamically during role transitions, facilitating rapid cognitive adaptation and behavior shaping.
- **Evolvability**: MemOS enables dynamic transitions and unified scheduling among different memory types—including parameter memory (knowledge embedded in model weights), activation memory (contextual inference state), and plaintext memory (structured knowledge fragments). The system supports seamless transitions, such as converting user-defined rules from multiple dialogues into active memory, or compressing long-term structured knowledge into parametric form. This cross-memory adaptation provides a robust foundation for knowledge integration, autonomous learning, and model evolution.

Therefore, as a novel infrastructure for the continual evolution of LLMs, MemOS aims to reconstruct the representation, management, and scheduling of memory from a systems perspective. It addresses core limitations in structured memory, lifecycle management, and multi-source integration, while providing OS-level support for cross-task adaptation, cross-modal evolution, and cross-platform migration. The introduction of MemOS marks a critical transition in the development of large models: from mere perception and generation to memory and evolution.

2 Memory in Large Language Models

Research in memory capabilities in large language models has generally progressed through four key stages: **(1) The stage of definition and exploration**, which focuses on categorizing and analyzing LLM memory systems from multiple perspectives, while identifying effective optimization mechanisms applicable in real-world scenarios. **(2) The stage of human-like memory development**, which addresses performance gaps in complex tasks arising from discrepancies between LLM and human memory by introducing various forms of cognitively inspired memory mechanisms. **(3) The stage of tool-based memory management**, where modular interfaces for memory operations begin to emerge, yet are largely limited to basic insert, delete, and update functionalities over existing memory structures. Our proposed MemOS introduces operating system–inspired resource management mechanisms to LLM memory, offering standardized and unified interfaces for full-lifecycle memory management and scheduling. This paves the way toward **(4) The stage of systematic memory governance**, enabling structured evolution, abstraction, and secure control over memory resources. In this subsection, we review existing research on memory in large models along this developmental trajectory.

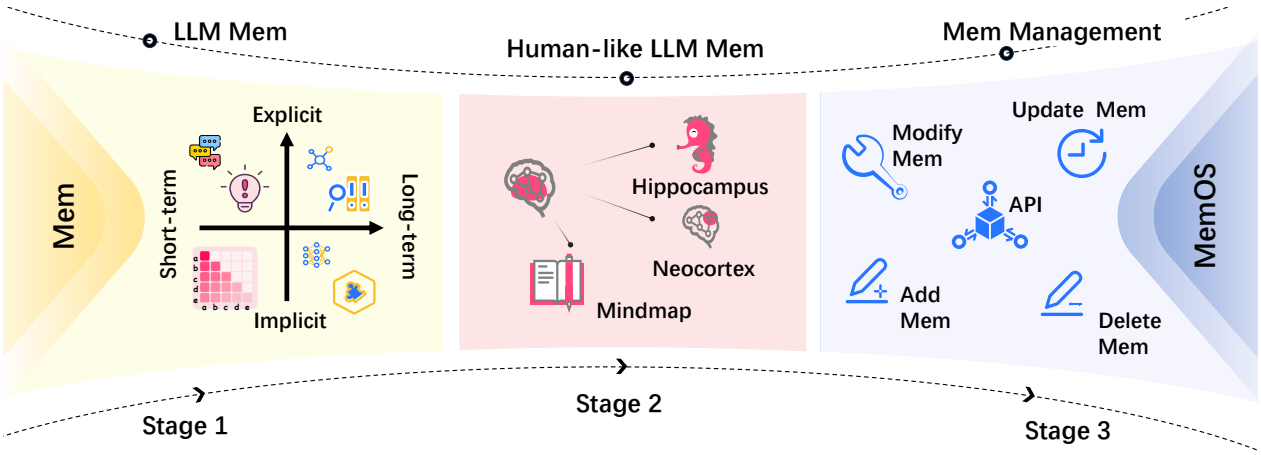

Figure 3 Illustration of the evolution of memory systems in large language models, highlighting the progression from definition and exploration, to human-like memory development, and to tool-based memory management.

2.1 Stage 1: Memory Definition and Exploration

Several recent studies have proposed systematic classifications and analyses of memory in LLMs from various dimensions. For example, [19] categorizes memory into three types: parameter memory, unstructured contextual memory, and structured contextual memory. [20] classifies memory based on object (personal vs. system), form (parametric vs. non-parametric), and temporal aspects (short-term vs. long-term). [21] further divides memory into four types: parameter-based, key-value cache-based, hidden state-based, and text-based, and introduces retention duration as a standard to distinguish sensory memory, short-term memory, and long-term memory.

Building on these works, we propose that LLM memory can be characterized along two primary dimensions: **implicit** and **explicit**. Implicit memory includes parameter memory, key-value cache, and hidden states, while explicit memory involves text- and context-based information storage. Memory can be classified temporally as sensory, short-term, or long-term. Sensory memory captures fleeting impressions of perceptual input, with extremely short duration and no conscious processing. While traditionally treated as a separate stage, we include it under short-term memory for unified scheduling and handling of initial information. This work adopts this two-dimensional framework to analyze memory mechanisms in the first and second stages (see Figure 3 left, Table 1).

2.1.1 Implicit Memory in LLMs

Implicit Long-term Memory in LLMs Through large-scale pretraining, LLMs encode syntactic structures, conceptual relationships, and language usage patterns from corpora into their weight matrices. These parameters serve as implicit long-term memory, internalized into the model's inherent capabilities. Although they lack explicit expression, they continuously influence language generation behavior, knowledge expression, and even semantic generalization.

Training: In LLMs, training is the most fundamental and direct method for forming implicit long-term memory. For example, pretraining [23, 56] and post-training [28, 57] enable large-scale parameter updates, fundamentally reconstructing the internal knowledge distribution and behavioral structure of the model. Some studies introduce memory explicitly during training. For instance, CTRL [58] includes control codes in training data to help models automatically associate contextual information during text generation. Memory&Reasoning [74] fine-tunes the model to decouple output into separate memory and reasoning components, fully leveraging memory for inference. SLayer [59] identifies memory-relevant layers in the model and locally fine-tunes them to enhance specific knowledge representation. It is worth noting that relying solely on memorization of training data can be limited in real-world deployment due to distributional shifts between real-world and training data. Titans [75] proposes a dynamic memory mechanism by encoding historical information into

Table 1 Classification of Memory Types, Mechanisms, and Example References

Timescale	Consciousness	Mechanism	Example References
Short-term	Explicit	Prompt-Based Context	GPT-2 [22], GPT-3 [23], Prefix-Tuning [24], Prompt-Tuning [25], P-Tuning [26, 27], InstructGPT [28]
	Implicit	Key-Value Cache Mechanism	vLLM [29], StreamingLLM[30], H2O[31], LESS [32], KVQuant [33], RetrievalAttention [34], Memory^3 [1]
		Hidden State Steering	Steer [35], ICV [36], ActAdd [37], StyleVec [38], CAA [39], FreeCtrl [40], EasyEdit2 [41]
		Activation Circuit Modulation	SAC [42], DESTEIN [43], LM-Steer [44]
Long-term	Explicit	Non-parametric Retrieval-Augmented Generation	kNN-LMs [45, 46], MEMWALKER [9], Graph RAG [10], LightRAG [11], NodeRAG [47, 48], HyperGraphRAG [49], HippoRAG [50, 51], PGRAG[52], Zep [53], A-MEM [54], Mem0[55]
	Implicit	Parametric Knowledge	BERT [56], RLHF [57], CTRL [58], SLayer [59]
		Modular Parameter Adaptation	LoRA [60], PRAG [61], DyPRAG [62], SERAC [63], CaliNet [64], DPM [65], GRACE [66]
		Parametric Memory Editing	ROME [67], MEMIT [68], AlphaEdit [69], AnyEdit [70], EasyEdit [71], AdaPLE [72], MEMAT [73]

neural network parameters and training a pluggable online meta-model. This meta-model can adaptively decide retention or forgetting strategies for specific data during real usage, thereby improving generalization across distribution shifts.

Adaptor: Full-scale training or fine-tuning is costly and often impractical for rapid memory updates in real-world scenarios. To address this, adapter-based methods freeze the core model parameters and introduce small, trainable modules that adapt quickly to new memory with minimal disruption to original capabilities. LoRA [60] inserts low-rank adapters into the model, enabling lightweight parameter tuning without modifying the original parameter structure, supporting efficient loading and storage of implicit memory. PRAG [61] treats LoRA adapter modules trained for specific documents or tasks as "memory units" and merges them into the main model as needed, enabling rapid access to specialized knowledge. Furthermore, DyPRAG [62] introduces a neural generator that directly maps input documents to LoRA parameters, significantly reducing explicit memory storage cost.

Editing: Memory editing refers to targeted interventions on model parameters to induce new knowledge or behaviors for specific inputs while preserving existing capabilities as much as possible. Most existing research focuses on editing objective factual knowledge, such as correcting answers to questions like "Who is the president of the United States?" However, memory in LLMs also includes abstract competencies such as language style, semantic preferences, and reasoning modes, for which systematic editing methods are still lacking. If not carefully controlled, local parameter edits can lead to undesirable global behavior shifts. Thus, edit precision and retention of existing capabilities are key evaluation metrics. This paper follows [76] in categorizing knowledge editing techniques into three types: **(1) Locate-then-edit intuitive methods**[67, 68, 77]: These methods use causal tracing to locate where the target knowledge is stored, followed by targeted parameter updates. **(2) Meta-learning-based methods**[78–80]: These use hypernetworks to directly predict parameter changes. Another important direction is preserving prior knowledge and abilities during editing[69, 70, 72]. **(3) Adapter-based editing strategies**[63–66]: These preserve the LLM backbone, offering a degree of edit controllability.

Implicit Short-term Memory in LLMs Beyond the internalized parametric long-term memory, LLMs also depend on dynamically generated and transient intermediate representations during inference—such as KV-caches and hidden states. Although these representations lack explicit forms, they continually influence attention distributions and behavioral strategies in autoregressive generation, forming the implicit short-term memory of LLMs. They play a vital role in maintaining contextual coherence, enabling instant control, and facilitating behavior transition, and have become a crucial entry point for understanding and enhancing

dynamic capabilities of language models.

KV-cache: KV-cache stores key-value representations of previously processed tokens, enabling persistent access to historical memory during autoregressive generation. Although users cannot directly manipulate these caches, they implicitly modulate attention and output behavior during inference [1]. Subsequent optimization work has focused primarily on improving compute and memory efficiency. Techniques such as low-rank compression and quantization are adopted by LESS [32] and KVQuant [33], while StreamingLLM [30] and H_2O [31] dynamically prune less relevant KV pairs based on attention patterns. More recent studies introduce retrieval-based memory activation [34, 81], enabling selective access to cached content. Meanwhile, vLLM [29] draws from operating system design by implementing PagedAttention—using virtual memory-style page caching to reduce redundant storage and improve KV access.

While most existing work focuses on optimizing KV-cache for inference efficiency, its capacity to represent structured and controllable knowledge remains underexplored. Memory^3 [1] takes a first step in this direction by encoding external knowledge bases as sparse key-value pairs, which are injected into the model's self-attention layers. This enables dynamic, non-parametric retrieval of relevant information during inference, effectively externalizing knowledge and improving memory controllability—offering new directions for the structured use of short-term memory. Building on the foundation laid by Memory^3, MemOS advances the notion of structured memory by proposing the first hierarchical memory architecture for LLMs that models and unifies three distinct substrates: plaintext memory, activation memory, and parameter memory. It introduces an integrated retrieval and scheduling framework that enables explicit control, efficient fusion, and dynamic activation. The MemCube module further organizes semantic fragments into a multi-dimensional structure, enabling query-based aggregation and multi-granularity activation—paving the way for more systematic and scalable memory utilization in LLMs.

Hidden States: Hidden states represent the layer-wise intermediate activations within LLMs during processing, encoding the model's semantic understanding and generation trajectory. Compared to modifying model parameters, directly manipulating hidden states offers a more flexible, instantaneous, and efficient means of memory control. Among the various mechanisms, steering vectors [35] stand out as a representative method. These vectors are derived by computing activation differences between inputs with contrasting semantic attributes, forming directionally meaningful control signals. Injecting such vectors into the intermediate activations of other inputs can steer generation toward specific semantic directions without altering the model architecture. To avoid reliance on supervised corpora, methods like Self-Detoxifying [82], ActAdd [37], ICV [36], StyleVec [38], and CAA [39] propose unsupervised contrastive approaches. These construct semantically similar yet attribute-opposing input pairs (e.g., emotion, stance, politeness) to extract hidden state differences and generate steering vectors, enabling automated, lightweight signal derivation. This not only enhances the portability of steerable control but also lowers its entry barrier. As an implicit short-term memory mechanism, hidden states have been validated in various practical tasks. For example, steering vectors have been employed in hallucination mitigation and factual consistency enhancement in ACT [83], ITI [84], and InferAligner [85]. IFS [86] extends their application to controlling low-level generation features such as text formatting and sentence length, indicating that hidden state interventions are effective not only for abstract semantics but also for structural behavior modulation.

2.1.2 Explicit Memory in LLMs

Explicit Short-term Memory in LLMs LLMs' explicit short-term memory primarily resides in their input context window—namely, the prompt and directly concatenated historical dialogues, including user task descriptions, interaction history, and reference documents. These explicitly injected elements are directly perceived and utilized during inference, forming the basis for understanding the current context and generating responses. With the increasing scale and capabilities of LLMs, their ability to manage explicit short-term memory has significantly improved. From early general-purpose language models relying on static text input [22, 23], to parameterized prompt techniques using learnable continuous vectors [24–27], to advanced instruction-following models [87–89], and the InstructGPT-style instruction tuning paradigm [28], mechanisms for expressing and managing explicit short-term memory have evolved from static configuration to dynamic interaction, becoming increasingly structured and flexible. However, explicit short-term memory in LLMs is

physically constrained by context window length. When handling lengthy texts or multi-turn dialogues, models often encounter truncation of early content and memory fading, leading to diminished semantic coherence or loss of key information [90, 91]. Recent research has attempted to alleviate these bottlenecks through longer windows, external retrieval, or more efficient caching, yet the capacity of explicit short-term memory remains a key limiting factor in real-time comprehension and interaction.

Explicit Long-term Memory in LLMs Unlike short-term memory dependent on context windows, LLMs' explicit long-term memory emphasizes sustained access to external non-parametric knowledge, with a focus on optimizing memory organization structures and retrieval strategies. Early research focused on identifying effective retrieval mechanisms for recalling relevant content from standalone external memory stores. Common approaches include off-the-shelf retrievers such as BM25 [92], Dense Passage Retrieval (DPR) [93], and hybrid retrieval methods [94]. However, such retrieve-then-generate approaches impose an inherent bottleneck in integrating retrieved content into model reasoning. Thus, some studies have explored tighter coupling of retrieval with inference. Non-parametric language models (NPLMs) such as kNN-LMs [45, 46] propose a linear fusion of neural language models (e.g., Transformers) with k-nearest-neighbor retrieval. At each prediction step, they retrieve top-matching context chunks from memory and blend their influence into the model's output distribution to improve reference fidelity.

Due to the limited representational capacity of flat memory structures, optimizing retrieval alone often fails to surpass performance ceilings. As a result, research has increasingly shifted toward enhancing memory organization itself. Traditional key-value formats have gradually evolved into more hierarchical and relational structures, such as tree-based [9] and graph-based formats [10, 11]. To further represent diverse memory relationships, researchers have introduced heterogeneous graphs [47, 48] and hypergraph structures [49], enabling unified modeling and dynamic control of varied knowledge types and complex semantic links. These advances greatly enhance the expressive power and generalization of memory networks. To endow LLMs with structured, dynamic, and persistent memory, Zep [53] builds on GraphRAG [10] by adding timeline modeling to track memory evolution over time. A-MEM [54] draws from dynamic memory networks to support automatic memory linking and semantic updating, allowing LLM memory to evolve across multi-turn interactions.

2.2 Stage 2: Development of Human-like Memory

To enhance the memory capabilities of LLMs in complex tasks, some studies have drawn inspiration from human memory mechanisms and knowledge management methods, proposing various forms of human-like memory.

In the early stages of human-like memory research, the focus was on simulating the structural and functional mechanisms of human memory. One representative early work is the HippoRAG series of models[50, 51], inspired by the "hippocampal indexing theory" in human long-term memory. The model integrates LLMs, knowledge graphs, and the Personalized PageRank algorithm to emulate the roles of the neocortex and hippocampus in memory, achieving more efficient knowledge integration and retrieval. Memory^3[1], inspired by the hierarchical structure of human memory, makes the KV-cache in the attention mechanism explicit as a memory carrier for the model. This approach offers a lower-cost alternative to parameter storage or traditional RAG, significantly reducing the resource consumption for training and inference.

As research advanced, system designs began emphasizing human-like behavior and function, simulating how humans actually use memory. For instance, PGRAG[52] mimics the act of note-taking during reading, automatically generating mind maps as explicit long-term memory to enhance organization and durability. Second-Me[95] proposes a multi-level architecture centered on human-like memory behaviors, emphasizing experience-driven personalized retrieval. The system consists of three layers: L0 retains raw data for completeness; L1 enhances organization and retrievability through structured natural language; L2 internalizes user preferences via parameter tuning, enabling associative reasoning similar to humans. AutoGen[96] introduces a multi-agent framework to simulate human group collaboration, forming a dialog ecosystem of interacting agents. Each agent has distinct roles, and they collaborate through dialog to share information and accomplish complex tasks like mathematical reasoning, information retrieval, and code generation.

2.3 Stage 3: Tool-based Memory Management

With the growing understanding of memory in LLMs, researchers have begun exploring explicit manipulation of knowledge, pushing memory management from implicit representations toward tool-based interfaces.

This stage witnessed the emergence of standardized frameworks for memory editing, enabling users to dynamically update the model's semantic behavior through insert, modify, and delete operations. For example, early approaches like EasyEdit[41, 71] offer unified interfaces to manipulate model parameters and hidden states for fine-grained control. Another representative line of work is Mem0[97], which targets the context window bottleneck by introducing external memory modules maintained through extract-update workflows. Follow-ups to Mem0 even structure conversational memory into graphs to enable richer semantic modeling and long-term evolution. Among these, Letta[98] stands out as a system-oriented attempt. It draws inspiration from traditional operating systems by modularizing context and introducing function-style paging for dynamic memory access.

However, most work in this stage remains limited to interface-level utilities. While tool-based management introduces basic CRUD operations, it lacks systematic modeling and governance of memory as a core resource—making it insufficient for tasks requiring memory evolution, coordination, or security.

2.4 Stage 4: Systematic Memory Governance

Although tool-based management introduces explicit memory operation interfaces, it essentially patches implicit mechanisms. CRUD capabilities alleviate short-term issues but fall short of addressing systemic challenges like memory evolution, access control, and version management. Just as system calls alone cannot build a complete OS, "tooling" memory lacks a sustainable and scalable governance architecture.

To overcome the limitations of tool-based management, we propose **MemOS**, a memory operating system purpose-built for LLMs, marking the entry into the stage of systematic memory governance. MemOS treats memory units as first-class resources and builds upon operating system design principles to introduce comprehensive governance mechanisms including scheduling, layering, API abstraction, permission control, and exception handling. Unlike the tool-based phase, MemOS not only enables operations but also emphasizes the evolution and integration of memory across tasks, sessions, and agent roles. With core modules such as MemScheduler, Memory Layering, and Memory Governance, MemOS enables unified scheduling and behavior-driven evolution of heterogeneous memory types—building a long-term cognitive structure essential for AGI. We envision the "memory-as-OS" paradigm pioneered by MemOS as the infrastructural backbone for future general-purpose agents, enabling sustainable knowledge accumulation and self-evolution.

3 MemOS Design Philosophy

3.1 Vision of MemOS

As AGI advances toward increasingly complex systems involving multiple tasks, roles, and modalities, LLMs must go beyond merely "understanding the world"—they must also "accumulate experience," "retain memory," and "continuously evolve." However, current mainstream LLM architectures lack systematic support for memory as a core intelligence capability: knowledge is rigidly encoded in parameters, context cannot be preserved across sessions, personalization cannot be retained, and knowledge updates are prohibitively expensive. We argue that the next-generation LLM architecture must adopt a memory-centric design paradigm.

As shown in Figure 4, model performance is approaching the upper limits predicted by traditional scaling laws. The prevailing research paradigm is transitioning from data- and parameter-centric pretraining to post-training, which emphasizes reinforcement alignment and instruction tuning [99]. Yet this shift faces two major challenges: diminishing returns and growing system complexity. To unlock the next leap in capability, we must transcend the current paradigm by incorporating continuous memory modeling and dynamic memory scheduling—thereby enabling long-term knowledge accumulation, task adaptation, and behavioural evolution.

Beyond the temporal benefits of continual learning, memory training also introduces a spatial scaling effect. Thousands of heterogeneously deployed model instances can gather experience in situ and exchange compact

memory units—rather than expensive parameters or gradients—to build a collective knowledge base. This memory-parallel regime blurs the line between training and deployment, effectively extending data parallelism to a society-scale, distributed intelligence ecosystem. Two technical challenges arise: (1) efficient knowledge exchange across highly heterogeneous environments, and (2) strict governance that protects private or sensitive data while maximising shared utility.

We therefore advocate a memory-centric training strategy—the Mem-training Paradigm. Instead of relying solely on sporadic parameter updates, Mem-training drives continuous evolution through explicit, controllable memory units. Unlike traditional workflows that modify the model only during pretraining or fine-tuning, Mem-training allows knowledge to be collected, re-structured, and propagated at runtime, enabling self-adaptation across tasks, time horizons, and deployment environments.

In this paradigm, "training" is no longer limited to large-scale corpora but extends to dynamic knowledge accumulation via continuous interaction with users and the environment. The focus shifts from how much knowledge the model learns once to whether it can transform experience into structured memory and repeatedly retrieve and reconstruct it. MemOS serves as the system-level foundation for this paradigm, enabling end-to-end capabilities in memory generation, scheduling, fusion, and updating.

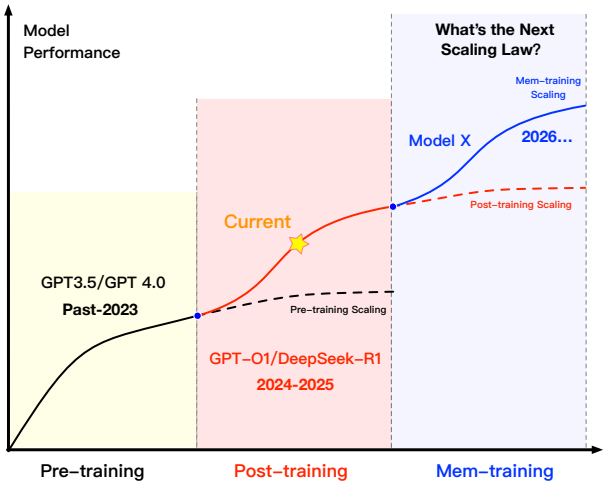

Figure 4 Phased transitions in model performance: from pretraining and post-training to the Mem-training stage. MemOS serves as the foundational infrastructure enabling the next era of scaling laws.

Our vision is for MemOS to become the foundational memory infrastructure for next-generation intelligent agents, with its core mission expressed through the following three pillars:

- **Memory as a System Resource**: Abstract memory from a latent, internal dependency into a first-class, schedulable, and manageable resource. Build memory pathways that span agents, users, applications, and sessions, breaking down "memory silos" across platforms, significantly reducing memory management complexity, and improving the effectiveness and efficiency of memory access.
- **Evolution as a Core Capability**: Enable continuous learning, structural reorganization, and task transfer throughout long-term memory usage. Build a co-evolutionary infrastructure for models and memory, allowing LLMs to self-adapt and upgrade in response to changing tasks, environments, and feedback—achieving truly sustainable, evolving intelligence.
- **Governance as the Foundation for Safety**: Provide lifecycle-wide memory governance mechanisms including access control, versioning, provenance auditing, and more. Ensure controllability, traceability, and explainability of memory, laying the groundwork for secure, trustworthy, and compliant intelligent agent systems.

We believe that just as traditional operating systems laid the foundation for modern computing by unifying computation and storage management, MemOS will elevate memory to a core system resource, forming an indispensable foundation for both general-purpose and embodied intelligent agents. This will drive a paradigm shift from reactive, perception-based systems to memory-driven, evolving agents.

3.2 From Computer OS to Memory OS

In traditional computing systems, the operating system (OS) centrally manages key hardware resources—such as the central processing unit (CPU), memory, storage devices, and peripherals—to support efficient execution and stable operation of applications. The OS's abstraction of resources, unified scheduling, and lifecycle governance serve as the foundation for the scalability and reliability of modern computing infrastructures.

As large language models (LLMs) scale in inference and application complexity, both internal and external memory resources—ranging from static parameter memory to runtime activation memory and dynamically retrieved explicit memory modules—exhibit increasingly dynamic and heterogeneous behavior. These memory forms are not only foundational to inference but also continuously evolve with task shifts and knowledge updates. Therefore, LLMs similarly require a systematic resource management framework akin to traditional operating systems, enabling standardized abstraction, dynamic scheduling, and autonomous lifecycle governance of memory.

MemOS proposes a design philosophy for the unified and systematic management of memory resources in LLMs, drawing extensively on mature mechanisms from traditional OS domains such as resource scheduling, interface abstraction, access control, and fault handling. Table 2 illustrates the mapping between classical OS components and MemOS modules: MemOS coordinates inference and memory block scheduling via the LLM Core and MemScheduler, manages hierarchical memory through Memory Layering and MemStore, offers standardized API abstraction through MemAPI and Backend Adapter, enforces security and access governance through Memory Governance, and supports monitoring and anomaly detection through the Memory Observability framework. These modules work in concert to adapt traditional resource management principles to the evolving demands of memory in LLMs.

Table 2 Mapping of Traditional OS Components to MemOS Modules

Layer	OS Component	MemOS Module	Role
	Core Operation Layer		
Parameter Memory	Registers / Microcode	Parameter Memory	Long-term ability
Activation Memory	Cache	Activation Memory	Fast working state
Plaintext Memory	I/O Buffer	Plaintext Memory	External episodes
	Management Layer		
Scheduling	Scheduler	MemScheduler	Prioritise ops
Persistent Store	File System	MemVault	Versioned store
System Interface	System Call	Memory API	Unified access
Backend Driver	Device Driver	MemLoader / Dumper	Move memories
Package Deploy	Package Manager	MemStore	Share bundles
	Governance & Observability		
Auth / ACLs	Auth Module, ACLs	MemGovernance	Access control
Logging	Syslog	Audit Log	Audit trail
Fault Handling	Excp. Handler	Error Recovery	Error recover

4 Memory Modeling in MemOS

4.1 Types of Memory in MemOS

The concept of hierarchical memory was originally introduced in our prior work Memory^3[1], which proposed a distinction between explicit and implicit memory paths in LLMs and investigated their interaction mechanisms.

Building on this foundation, MemOS systematizes the idea by delineating three core memory types—**Plaintext Memory**, **Activation Memory**, and **Parameter Memory**—that together reflect a full semantic evolution trajectory from perception to consolidation.

To coordinate scheduling and evolution across heterogeneous memory types, MemOS introduces the **MemCube**—a unified abstraction that standardizes memory representation, lifecycle management, cross-modal fusion, and dynamic memory state transitions. Its design is inspired by the controllable externalization proposed in Memory^3, while advancing it into a composable and schedulable memory substrate suitable

for intelligent agent construction. This design forms the semantic memory backbone of MemOS, enabling seamless integration and transformation of multiple memory types during inference.

Plaintext Memory Plaintext memory refers to explicit, dynamically retrieved knowledge modules accessed via external interfaces—editable, traceable, and storable independently. Examples include retrieved passages, structured graphs, and prompt templates. Injected into model input, it bypasses the limitations of parameter capacity and context window size. It enables rapid knowledge updates, task customization, and user personalization.

MemOS encapsulates plaintext memory into tunable `MemCube`s, with lifecycle control, access policies, and version tracking. It supports graph-structured and multimodal memory, contextual fingerprinting, and timestamp-based loading. Plaintext memory is not merely an external plugin. MemOS deeply integrates it into the inference loop, enabling interaction with activation memory. High-frequency plaintext can be transformed into activation paths, achieving dynamic externalization and internalization of knowledge. To enhance scheduling efficiency and long-term evolvability, MemOS manages plaintext memory in a hierarchical graph structure organized by task–concept–fact paths. Task parsing combined with semantic similarity and topic-aware strategies enables structured query routing and prioritized retrieval. It supports conflict detection, deduplication, versioning, and forgetting policies to maintain memory quality and evolution.

Plaintext memory is particularly suited for fact-heavy, personalized, and multi-agent tasks—serving as a core enabler of transparent and collaborative intelligence.

Activation Memory Activation memory consists of intermediate states generated during inference, with the KV-cache as the central structure. It retains key-value representations of context, enabling efficient long-range dependency modeling and recursive reasoning. It supports instant contextual response and reusable inference pathways through cache-stable behaviors. Other elements include hidden states (h_i^l) and attention weights (α_{ij}^l), comprising the model's runtime semantic perception. These are characterized as short-term, dynamic, and implicitly activated.

MemOS offers unified scheduling and lifecycle management for activation memory. It enables lazy loading, selective freezing, and priority-driven adjustments. Frequent KV patterns are cached to form low-latency "instant memory paths". Beyond KV patterns, strategic behaviors that are repeatedly triggered can also be abstracted into persistent memory structures, such as steering vectors or semantic templates. KV memory proves valuable in multi-turn dialogue, code assistance, and runtime safety management. For instance, in medical agent systems, stable and frequently accessed knowledge—such as patient histories, routine diagnostic procedures, or clinical commonsense—can be abstracted into cached KV segments, enabling rapid recall and minimizing redundant decoding. It is essential for maintaining contextual continuity, stylistic coherence, and precise response control.

Parameter Memory Parameter memory refers to knowledge and capabilities encoded in the model's fixed weights. It serves as the primary repository of long-term semantic knowledge within the model. It encodes deep representations of linguistic structure, commonsense knowledge, and general semantics—typically instantiated as feedforward weight matrices (e.g., W_{MLP}^l) and attention key/value matrices (e.g., W_K^l, W_V^l). Unlike other memory types, parameter memory is activated implicitly without retrieval or explicit context, forming the foundation for zero-shot inference, general QA, and language generation.

In MemOS, parameter memory includes both pre-trained linguistic and world knowledge and can be modularly enhanced via lightweight fine-tuning methods such as LoRA or adapters. MemOS enables distilling domain-specific knowledge into parameter blocks, loadable as "capability modules" (e.g., summarization expert, legal assistant, style generator). While offering strong expressivity and high efficiency, parameter memory suffers from high update costs, limited customizability, and poor interpretability. To address this, MemOS links parameter memory with plaintext and activation memories. For instance, frequently used and structurally stable plaintext may be distilled into parametric form for embedded efficiency. Conversely, outdated or inconsistent parameter memory can be backpatched by reverting to plaintext. Parameter memory is ideal for capability-centric agents, such as legal advisors, financial auditors, technical writers, or summarizers, or as

composable "capability plugins". Compared with frequently updated plaintext or transient activation memory, it better supports long-term, structurally stable capabilities.

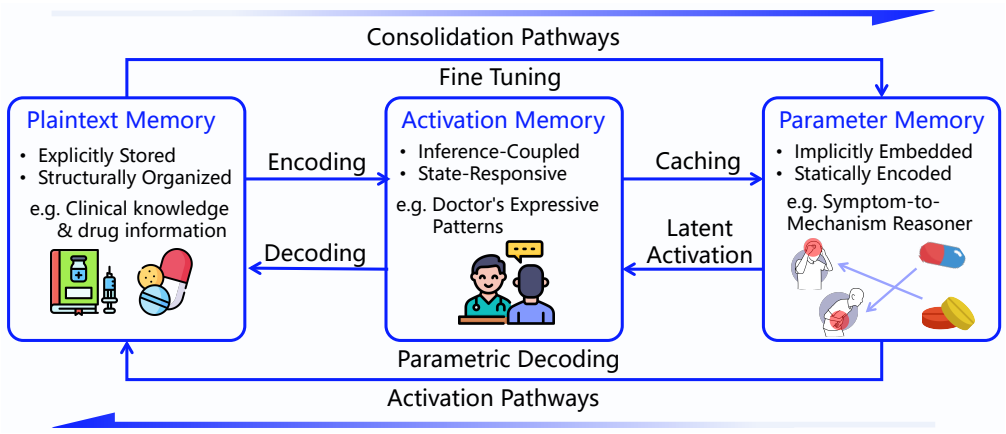

Figure 5 Transformation paths among three types of memory, forming a unified, controllable, and evolvable memory space.

4.2 Memory Cube (`MemCube`) as a Core Resource

In MemOS, the foundation of a unified and structured memory management system lies in the standardized abstraction and system-level governance of heterogeneous memory resources. To this end, we propose the `Memory Cube (MemCube)` as a universal encapsulation unit for memory resources (see Figure 6).

Memory in LLMs is highly diverse, spanning long-term knowledge embedded in model parameters, intermediate activation states generated during inference, and externally injected structured knowledge fragments (e.g., retrieved passages, knowledge graph nodes). These resources differ significantly in origin, lifecycle, representation, and scheduling method, making unified control, evolution, and governance a systemic challenge.

The design of `MemCube` aims to encapsulate all memory types as unified scheduling units, each with standard interfaces, behavioral properties, and governance strategies. Each `MemCube` instance consists of two components: the `Memory Payload`, which contains the semantic content, and the `Metadata`, which encodes identity, control, and behavioral metrics. These metadata elements serve as foundational interfaces for MemOS scheduling and governance and as central anchors for long-term system evolution, task adaptation, and security control.

The metadata of each `MemCube` is categorized into three groups: **descriptive identifiers**, **governance attributes**, and **behavioral usage indicators**. Together, these enable full-spectrum memory management across structural identification, access control, and behavioral evolution. We elaborate below on their motivations, components, and system-level implications.

Descriptive Identifiers define each memory block's identity, classification, and organization. Unified memory scheduling at scale relies on precise identification of these "semantic fingerprints." `MemCube` embeds key fields such as: **Timestamp**, indicating creation or last update for lifecycle modeling; **Origin Signature**, identifying whether the memory comes from inference extraction, user input, external retrieval, or parameter finetuning; and **Semantic Type**, specifying its use (e.g., task prompt, fact, user preference) to support semantic composition. These jointly enable layered memory structuring and contextual navigation.

Governance Attributes provide systemic controls for memory access, security, and scheduling. In dynamic, multi-user, long-running systems, default model reasoning is insufficient for robust memory governance. MemOS defines a comprehensive rule set per memory unit, including: **Access Control** (read/write/share scope), **Lifespan Policy** (TTL or decay rules), **Priority Level** (for scheduling), and **Compliance & Traceability** (e.g., sensitivity tags, watermarks, logs). Together, they form the memory governance kernel—critical for system stability, transparency, and accountability.

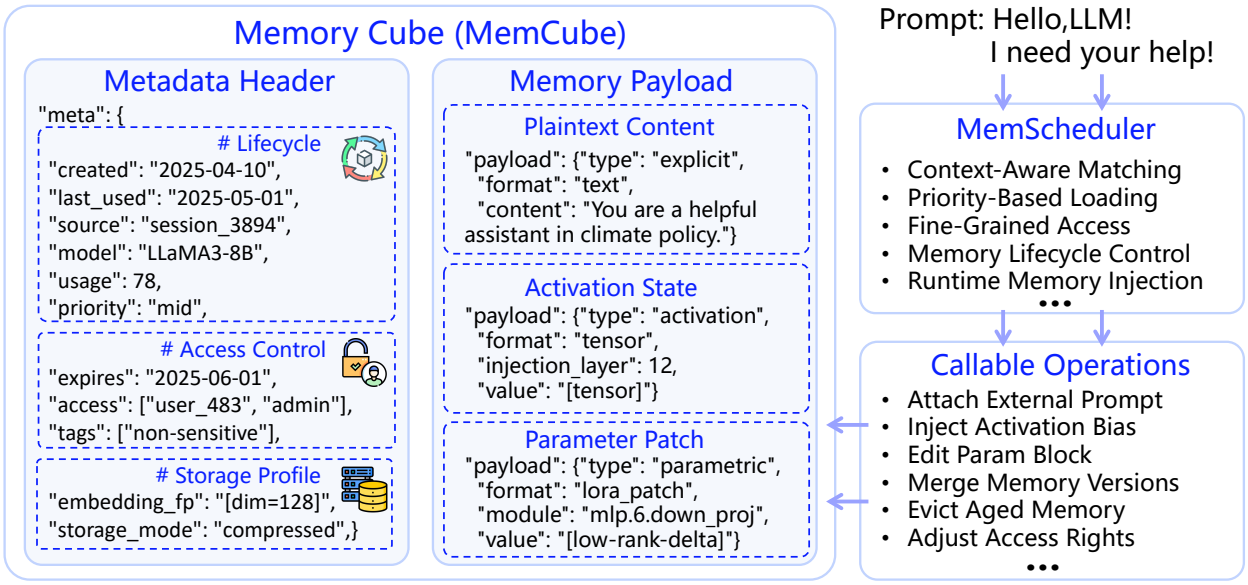

Figure 6 `MemCube`: A unified encapsulation structure for heterogeneous memory scheduling. Each `MemCube` consists of a structured Metadata Header (supporting lifecycle, permission, and storage policy) and a Memory Payload (encapsulating plaintext, activation states, or parameter deltas). It is the minimal memory unit within MemOS that can be scheduled and composed for downstream reasoning.

Behavioral Usage Indicators reflect real-time memory usage during inference, enabling "value-driven" scheduling and cross-type transformation. Unlike static labels, these runtime metrics empower adaptive orchestration of memory.

Access Patterns, such as frequency and recency, inform whether a memory is "hot" or "cold" during inference. MemOS uses this to adjust caching priority—for example, promoting high-frequency plaintext memory into fast-access layers to reduce latency.

These indicators also support **Cross-Modality Memory Transformation**, allowing dynamic transitions across memory types:

- **Plaintext ⇒ Activation:** Frequently used plaintext memory can be pre-transformed into activation vectors or attention templates for faster decoding.
- **Plaintext/Activation ⇒ Parameter:** Stable knowledge across tasks can be distilled into parameter modules, internalized as efficient capability plugins.
- **Parameter ⇒ Plaintext:** Cold or outdated parameters can be offloaded into external plaintext storage to increase flexibility and reduce structural overhead.

To support such transformations, MemOS introduces **Policy-Aware Scheduling**: the system dynamically adjusts a memory block's tier and format based on usage frequency, contextual dependency, and task fit—enabling layered memory evolution. Additionally, each memory is associated with a **Contextual Fingerprint**, a lightweight semantic signature for fast retrieval and task alignment. A **Version Chain** logs each memory's modification history and derivation lineage, enabling version control, conflict resolution, and rollback. These behavioral metrics allow MemOS to perceive the "value" of memory, forming the basis for adaptive scheduling, memory transformation, and knowledge evolution. As a result, memory becomes a self-regulating and self-evolving intelligent resource unit. Through the coordinated design of these three metadata types, `MemCube` enables structured abstraction, permissioned control, and behavior-driven evolution of heterogeneous memory resources.

5 Architecture of MemOS

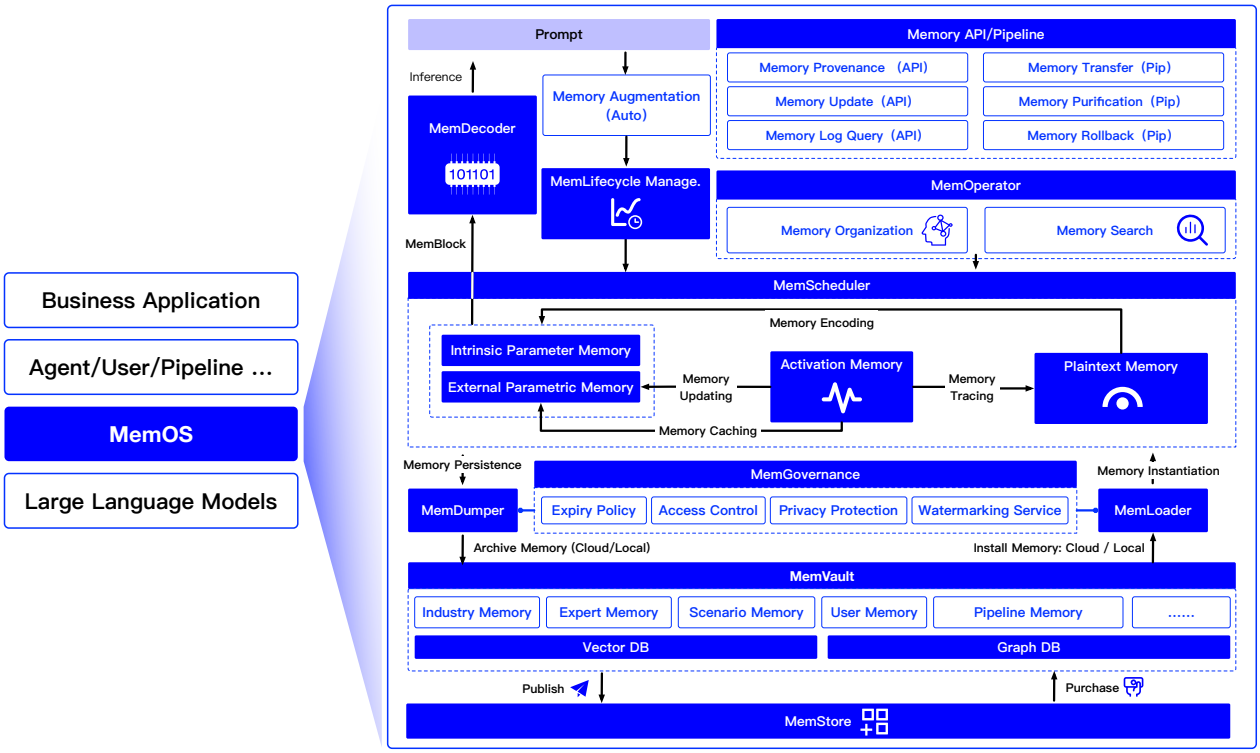

Figure 7 Overview of the MemOS framework. The architecture illustrates the full pipeline from user input through semantic parsing and API abstraction in the interface layer, to memory scheduling and lifecycle control in the operation layer, and finally interaction with the infrastructure layer for memory injection, retrieval, and governance. The unified data structure, `MemCube`, serves as the foundation for dynamic memory flow throughout model execution.

5.1 Overview: Three-layer Architecture of MemOS

MemOS adopts a modular three-layer architecture to support efficient invocation, dynamic scheduling, and compliant governance of complex memory tasks (see Figure 7). It consists of the Interface Layer, Operation Layer, and Infrastructure Layer, each with distinct responsibilities and collaborative interfaces—together building a unified execution and governance framework for heterogeneous memory types that enables robust intelligent agent performance across complex tasks.

Memory Interface Layer The interface layer interacts with users or upstream systems and serves as the entry point for all memory operations. It provides a standardized `Memory API` suite that supports querying, writing, updating, transferring, and composing memory units. All user requests are parsed by the interface layer into specific memory manipulation commands. The built-in `MemReader` module plays a central role in this process. It converts natural language inputs into structured memory operation chains, extracting time expressions, task intents, contextual anchors, and memory scopes. For instance, given a request like "Summarize my meeting notes from last month," `MemReader` extracts the time range (last month), memory type (meeting notes), and output target (summary), and formulates a labeled `MemoryQuery` with proper window parameters. In multi-turn conversations, `MemReader` uses context to infer omitted details, ensuring consistency in memory invocation. This layer also performs permission checks, parameter encapsulation, and call sequence management. It coordinates with `MemGovernance` to validate the compliance and traceability of every operation.

Memory Operation Layer The operation layer serves as the control center of MemOS, organizing, planning, and scheduling memory resources during inference. Its core components include `MemOperator`, which builds tag systems, semantic indexes, and graph-based topologies across heterogeneous memory types and contexts, facilitating efficient retrieval and contextual adaptation. `MemScheduler` selects appropriate memory types (e.g., Plaintext, activation, parameter) based on task intent and context, and dynamically plans invocation order and integration strategy to optimize for low latency and task relevance. `MemLifecycle` tracks the lifecycle transitions of each memory unit—creation, activation, expiration, and reclamation—to ensure memory resource controllability and freshness. In a multi-turn QA or complex dialogue, the operation layer first retrieves relevant memory (e.g., user preferences, past conversations, external structured documents) via `MemOperator`, determines the optimal invocation path via `MemScheduler`, and updates memory states using `MemLifecycle`. Thanks to this design, memory becomes a dynamic, context-aware resource rather than a static data fragment.

Memory Infrastructure Layer The infrastructure layer handles storage, security, migration, and flow of memory data, serving as the foundation for reliable system execution. `MemGovernance` enforces access control, retention policies, audit logging, and sensitive content handling. `MemVault` manages multiple memory repositories (e.g., user-specific, domain knowledge, shared pipelines) and provides standardized access interfaces. `MemLoader` and `MemDumper` enable memory import/export and cross-platform synchronization. `MemStore` provides a publish-subscribe mechanism for open memory sharing among multiple agents. In organizational QA systems, for instance, a locally updated memory entry can be validated and synchronized to a central memory hub, becoming available to authorized users.

Together, these three layers form the complete memory operation loop in MemOS—from task input to execution scheduling to governance and archival. The standard interface decoupling allows rapid iteration and extensibility, laying the foundation for multi-model, multi-task, and cross-platform memory sharing in future intelligent systems.

5.2 Execution Path and Interaction Flow of MemOS

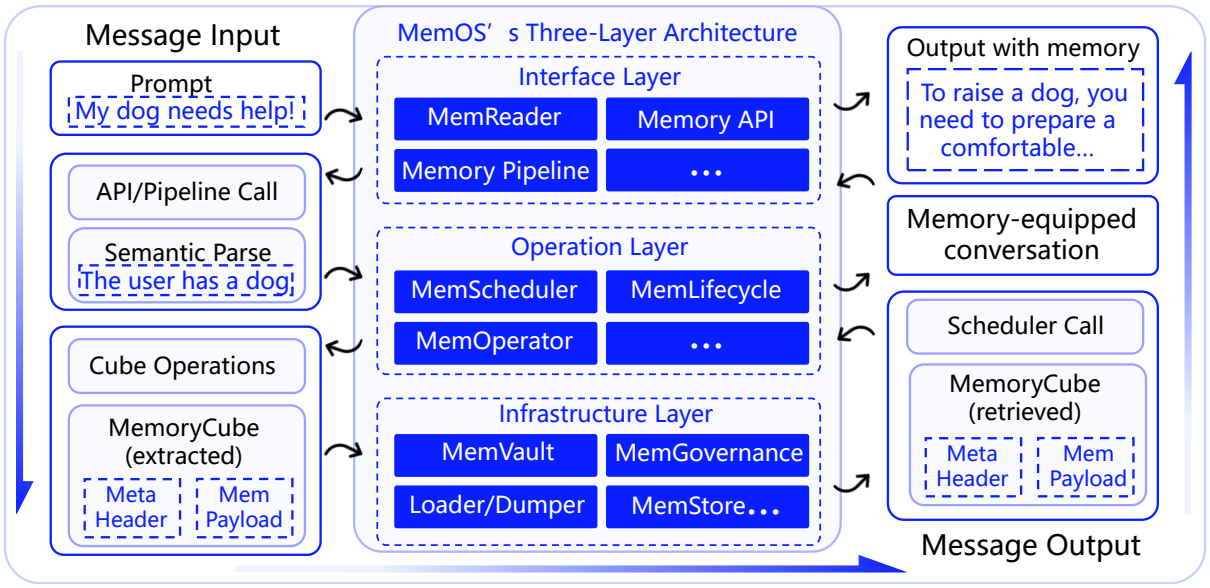

Figure 8 Overview of MemOS architecture and memory interaction flow. The system is composed of the interface layer, operation layer, and infrastructure layer. From left to right, it shows the complete memory processing pipeline from user input to parsing, scheduling, injection, and response generation. Each stage corresponds to coordinated module invocation, with MemoryCube serving as the carrier across layers for structured, governable, and traceable memory lifecycle management.

The execution of MemOS is triggered by either user interaction or automated tasks. It follows a closed-loop process through input parsing, memory scheduling, state management, and storage archiving(Figure 8).

Prompt Input and Memory API Packaging System execution begins with a user-issued natural language prompt or an automatically triggered task. The interface layer processes the input through the built-in `MemReader` module, which identifies task intent, time scope, topic entities, and contextual anchors to determine if memory access is involved. If so, `MemReader` converts the prompt into a structured `MemoryCall`, including the caller ID, context scope, memory type, access intent, and time window. This is encapsulated into a standardized `Memory API` request and passed to the operation layer for execution. For example, in a healthcare scenario, when a patient inputs, "Please retrieve my inpatient records from last year," `MemReader` identifies the time range (last year), topic tag (diagnostic records), contextual anchor (hospitalization period), and intent (historical query), and generates a structured `MemoryCall`, which proceeds to the memory retrieval and scheduling pipeline.

Memory Retrieval and Organization The `MemOperator` in the operation layer uses intent and context info from the `Memory API` to perform semantic matching and organize memory units. It constructs task-specific indexes (user preferences, anchors, keyword vectors) and memory graphs (temporal chains, entity relations, dependencies) to filter relevant candidates. For instance, if a patient asks the system to reference past cases for diagnosis, the operator retrieves memory blocks with symptom keywords, treatment periods, and associated physician notes to construct a structured retrieval path.

Memory Scheduling and Activation After the candidate set is identified, `MemScheduler` optimizes memory selection using metrics like contextual similarity, access frequency, temporal decay, and priority tags. It dynamically computes the optimal injection strategy. In a follow-up appointment, the system injects recent consultation summaries (activation memory), diagnosis templates (parameter memory), and lifestyle advice (plaintext memory), ensuring integrated, semantically coherent support.

Lifecycle Modeling and State Transitions Scheduled memory units are passed to `MemLifecycle` for state management. Each memory item transitions through five states—Generated, Activated, Merged, Archived, and Expired—based on access patterns, time decay, and task labels. For example, in medical use, generated medication advice starts in "Generated" state. If frequently accessed, it becomes "Activated"; after repeated user confirmations, it is "Merged" into frequent-use suggestions; and eventually archived or expired if unused.

Storage Archiving and Access Governance Evolved memories are archived in `MemVault` and organized by user, task, or context. Archiving may be triggered by policy, user command, or scheduling, keeping frequently accessed data active and less-used data cold or long-term stored. The archiving phase also invokes `MemGovernance` for permission encapsulation and compliance checks. Each memory unit is assigned a set of access control strategies—such as Access Control List (ACL), Time-To-Live (TTL), and conditional activation policies—that determine its availability based on user roles and task context. For example, a treatment summary may be fully visible to the care team but partially visible to the patient. After redaction and watermarking, it can be registered in `MemStore` for sharing across institutions.

The full governance and archiving pipeline ensures that all memory units—across diverse modalities and agents—are handled in a structured, transparent, and traceable manner, maintaining compliance and efficiency across collaborative healthcare environments.

5.3 Interface Layer

5.3.1 MemReader

In MemOS, the first step of any memory operation is interpreting natural language inputs from users or system tasks. This responsibility is handled by the `MemReader`, which serves as the semantic abstraction module for memory-level reasoning. It parses incoming prompts to extract key memory-related features—such as task intent, temporal scope, entity focus, memory type, and contextual anchors—and outputs a structured

intermediate representation. For example, a prompt like "Remind me what the doctor said about my medication during last year's hospitalization" would be parsed by `MemReader` into a structured memory access plan: task intent (retrieval), time scope (last year), topic (medication guidance), and context anchor (hospitalization period). This plan is passed downstream as a `MemoryCall` to be processed by the memory operation layer. `MemReader` also supports prompt rewriting, coreference resolution, and dialogue memory slot filling across multi-turn interactions. It functions as both an intent recognizer and memory orchestrator, ensuring the system issues precise and traceable calls to the underlying memory infrastructure.

5.3.2 Memory API

The interface layer of MemOS is built around a unified and composable `Memory API`, which bridges upper-level tasks with backend memory operations. All memory-related actions—including creation, updates, retrieval, and auditing—are performed via standardized APIs that ensure extensibility, composability, and governance. `Provenance API` enables provenance tracking by embedding metadata into memory objects at creation or modification time. This includes event triggers, contextual state, model identifiers, and external links. Each memory is tagged with a unique provenance ID that persists throughout its lifecycle. Provenance metadata supports explainability, debugging, access control, and memory lineage tracing. `Update API` supports mutation operations such as append, merge, or overwrite. It is version-aware, allowing snapshots and label-based differential writes. Typical use cases include task result logging, user correction, and fine-grained memory consolidation. When paired with `MemLifecycle`, update operations can trigger state transitions and index refreshes. `LogQuery API` allows structured access to memory access logs and execution traces. It supports filtering by timestamp, caller identity, memory type, and operation kind. It is essential for debugging, hotspot analysis, auditing, and governance enforcement. For instance, developers can investigate memory usage that led to faulty responses, or validate whether specific memories were invoked. All `Memory API` calls use `MemoryCube` as their parameter carrier and response format. They support transactional safety, structured status reporting, and are governed by `MemGovernance`, which enforces access control based on users, roles, models, and tasks.

5.3.3 Memory Pipeline

To support complex workflows in enterprise and multi-agent settings, MemOS offers a pipeline-style composition mechanism for chaining memory operations. Developers or agent systems can define a sequence of memory actions—e.g., retrieve → augment → update → archive—and execute them as a cohesive pipeline. Each pipeline step operates on a shared `MemoryCube` object, which carries input-output state, metadata, and intermediate artifacts. For example, a medical assistant might define a pipeline that (1) retrieves past medication notes via `LogQuery`, (2) adds doctor's latest instructions via `Update`, (3) tags the memory with a new provenance entry, and (4) archives it post-consultation. Pipelines support transactional consistency, rollback, and fault isolation. They can be defined declaratively through a domain-specific language (DSL), or constructed programmatically. For agent orchestration, `MemScheduler` interprets dependencies across steps and coordinates scheduling. Pipeline templates can be reused across agents—e.g., for follow-up generation in customer support, or for diagnosis tracking in clinical triage.

By enabling compositional memory flows, MemOS empowers developers to model higher-level cognition patterns, task-specific knowledge shaping, and auditable memory workflows.

5.4 Operation Layer

5.4.1 MemOperator

In MemOS, efficient memory organization and accurate retrieval are fundamental to enabling intelligent behavior generation, contextual reasoning, and knowledge reuse. The `MemOperator` module fulfills this role by structuring memory content both logically and semantically. It incorporates tag-based annotation, graph-based linking, and hierarchical abstraction to support multi-perspective memory modeling. Simultaneously, it provides unified interfaces for hybrid retrieval, serving diverse agents across tasks, models, and user contexts.

Multi-perspective Memory Structuring MemOS employs three complementary mechanisms for organizing memory. First, a flexible tagging system allows each memory unit to be annotated with metadata such as topic, source, credibility, and sentiment, supporting both user-defined and model-predicted labels. Second, a knowledge-graph structure treats memory as nodes connected via semantic edges, enabling traversable relations across memory items. Third, a semantic layering scheme segments memory into private, shared, and global layers, facilitating memory isolation and coordinated access across tasks and roles.

Hybrid Retrieval and Dynamic Dispatch The `MemOperator` module supports hybrid retrieval mechanisms that combine symbolic and semantic strategies. Structured retrieval applies rule-based filtering over tags, time spans, Boolean conditions, and access control policies. Semantic retrieval uses embedding-based vector representations to identify contextually relevant memory units via similarity search. These two mechanisms can be composed into complex query expressions—such as tag filters combined with semantic ranking—to serve applications like multi-turn dialogue, question answering, or knowledge integration.

Pipeline Coupling and Caching Strategy Retrieved memory units are passed downstream as inputs to execution pipelines, tightly coupled with the `Memory API` and `MemoryCube` modules. To minimize latency, MemOS implements a local index caching strategy whereby frequently accessed memory is automatically migrated to high-speed intermediate storage. Cache invalidation is managed by heuristics based on usage frequency and contextual drift, with the `MemScheduler` module overseeing refresh operations in a dynamic, workload-aware manner.

Task-Aligned Memory Routing To address the complexity of real-world tasks, MemOS employs a task-aligned routing mechanism that resolves memory navigation paths based on hierarchical semantic goals. User inputs are decomposed into a topic–concept–fact structure, forming a three-layered task schema. The `MemoryPathResolver` component then formulates a retrieval strategy that answers three key questions: what to search, where to search, and in what order. This structured approach enhances interpretability, scheduling relevance, and alignment between memory selection and task intent.

5.4.2 MemScheduler

`MemScheduler` is the central memory dispatcher of the MemOS operation layer. Its purpose goes beyond simply "retrieving" stored memories; it dynamically transforms and loads them into the runtime context based on task semantics, call frequency, and content stability. Relying on the three memory types defined in `MemCube`—Activation Memory (KV-Cache), Plaintext Memory, and Parameter Memory—`MemScheduler` supports classification, transformation, and hierarchical dispatch to deliver adaptive, high-performance memory operations.

Type-Aware Transformation and Loading Mechanism During memory scheduling, `MemScheduler` analyzes task semantics, window size, and resource constraints to determine the best-fit memory type. Stable, frequently accessed content is transformed into **Activation Memory** for KV caching, minimizing prefill latency. Abstract rules and reusable patterns are encoded as **Parameter Memory**—e.g., via distillation or adapters embedded into model weights. Time-sensitive or session-specific knowledge is preserved as **Plaintext Memory**, inserted into the prompt as raw text. Adaptive triggers guide the loading process. For coherence-heavy tasks like multi-turn dialogue, the scheduler favors KV-cache recall. For procedural or expert-driven flows, parametric modules take precedence. For on-demand factual queries, plain memory is retrieved and contextualized. All decisions are logged to `MemCube` and coordinated with `MemOperator`'s memory structure to maintain traceability and interpretability.

Cross-Type Conversion and Migration To maintain long-term performance and adaptive memory utilization, `MemScheduler` supports cross-type memory migration. For example, plain memories frequently recalled across sessions may be promoted to Activation Memory (KV cache). Stable templates used repeatedly can be distilled into parameter Memory. Conversely, underutilized KV entries may be downgraded to Plain Memory

and archived to cold storage. This type-shifting mechanism ensures memory units evolve toward their optimal invocation form while conserving system resources.

Execution Path Integration and Governance `MemScheduler` integrates upstream with `MemReader` and the `Memory API` to parse structured calls and semantic goals. Downstream, it collaborates with model execution paths to determine how and where to inject memory. Scheduling logic is optimized in real time, guided by task type, model load, cache hit rates, and access history. All dispatch actions are governed by `MemGovernance`, which enforces user-role boundaries, rate limits, and lifecycle policies. This ensures proper memory isolation and secure usage across users, models, and tasks, while maintaining an auditable record of every memory interaction.

5.4.3 MemLifecycle

In MemOS, each memory object is treated as a dynamic entity with evolving states, managed centrally by the `MemLifecycle` module. The system models memory as a finite state machine, cycling through four key states: Generated, Activated, Merged, and Archived. This framework supports semantic evolution, dynamic memory management, and stable, controlled resource scheduling at the storage layer.

State Modeling and Evolution Logic State transitions are triggered by a combination of system policies and user actions. For instance, in a smart meeting assistant, an auto-generated summary is initially labeled as "Generated". If that summary is later referenced in a follow-up task—like agenda tracking or meeting comparison—it transitions into the "Activated" state. When the user adds supplementary data, or the system detects semantic overlap with historical memory, these entries are consolidated into a new version and marked as "Merged". If a memory is no longer accessed for a prolonged period, it is demoted to the "Archived" state and moved to cold storage. Transitions can be explicitly initiated by user actions, or implicitly driven by system heuristics such as recency, contextual salience, or successful merge events.

Time Machine and Freezing Mechanism To ensure long-term consistency and recovery, MemOS offers a "Time Machine" capability that snapshots memory states and supports historical rollbacks. Users or developers can invoke this feature to restore an archived or merged memory back to a specific version, re-enabling its use in inference and context injection. This is critical for scenarios such as detecting model forgetting, handling user retractions, or conducting counterfactual simulations. In a policy collaboration platform, a user might unarchive an old clause to perform "what-if" simulations, without impacting the canonical frozen version and its audit trail. MemOS also supports a "Frozen" state for critical memories—like legal agreements or standard guidelines—where updates are disabled and full modification histories are retained for auditing, compliance, or education.

Scheduling and Storage Integration Strategy Lifecycle states directly influence scheduling priority and storage allocation strategies. "Activated" memories are preferentially cached in local memory or fast-access `MemoryCube` instances for low-latency retrieval. "Archived" or "Frozen" memories are offloaded to `MemVault`, a cold storage layer optimized for durability over speed. Based on lifecycle rules, the system can batch-trigger operations like cleanup, compression, or migration to balance call availability with efficient resource usage.

5.5 Infrastructure Layer

5.5.1 MemGovernance

`MemGovernance` is the core module in MemOS responsible for memory access control, compliance enforcement, and auditability. As memory systems evolve toward multi-user collaboration and long-horizon task reasoning, `MemGovernance` ensures that memory remains secure, interpretable, and controllable throughout its sharing, transfer, and inference processes.

It establishes a ternary permission model involving the user identity, the memory object, and the calling context, supporting private, shared, and read-only access policies. Each memory request undergoes identity authentication and contextual validation to prevent unauthorized access. For example, in clinical settings,

only physicians may access a patient's diagnostic records; in enterprise systems, only authorized managers can retrieve archived policy documents.

It manages memory lifecycle policies such as time-to-live (TTL) enforcement and access-frequency-based garbage collection or archiving of inactive items. It also tracks memory usage heat to monitor high-traffic memory segments. Its privacy control subsystem includes sensitive content detection, automatic redaction, and access logging to ensure personal and behavioral data remain secure.

All memory objects carry full provenance metadata, including creation source, invocation lineage, and mutation logs. Generated content can be watermarked semantically and tagged with behavioral fingerprints, allowing attribution and copyright tracking in multi-platform scenarios.

The module also exposes audit interfaces for integration with enterprise compliance systems, supporting export of access logs and permission revision reports. These features support regulatory compliance in high-stakes environments such as healthcare and finance.

5.5.2 MemVault

`MemVault` is the central memory storage and routing infrastructure in MemOS, responsible for managing and serving diverse categories of memory. Memory is organized into namespaces such as user-private stores, expert knowledge bases, industry-shared repositories, contextual memory pools, and pipeline-aligned caches. Each is assigned a dedicated namespace and path structure to support efficient lookup and access control.

To support heterogeneous backends, `MemVault` interfaces with vector stores, relational databases, and blob storage through a unified `MemoryAdapter` abstraction. This allows API-level consistency for querying, writing, and syncing memory regardless of backend heterogeneity. Stores may be configured as read-only caches or write-enabled repositories, depending on latency or learning objectives.

At runtime, `MemVault` works in concert with `MemScheduler` and `MemLifecycle` to dynamically load memory based on access history, contextual relevance, and memory state. It supports tag-based, semantic, and full-text loading patterns, and triggers migration for hot memory to fast storage or cold data to archival zones. This architecture is vital for multi-model collaboration, domain-level knowledge fusion, and consistency in multi-turn dialogue—forming the knowledge backbone for scalable intelligent systems.

5.5.3 MemLoader & MemDumper

`MemLoader` and `MemDumper` form a bi-directional channel for memory migration across platforms in MemOS. They support injection, export, and synchronization of structured units like `MemoryCube`. This capability is essential for system handover, edge-cloud integration, and knowledge continuity across distributed agents.

During ingestion, `MemLoader` accepts memory from local caches, third-party systems, or archives and maps it to target stores. It auto-fills provenance metadata, tagging, and lifecycle status to ensure governance readiness.

`MemDumper` exports selected memory in portable formats with permission metadata, redacted fields, and access logs. Both components support periodic and event-driven updates, such as automatic export upon tag activation. The migration process is governed by `MemGovernance` to validate policies, trace operations, and isolate sensitive data. For instance, a mobile device may export patient interaction logs to the cloud, which remote agents later load to preserve task context.

5.5.4 MemStore

`MemStore` is the open-access interface in MemOS that enables controlled publishing, subscription, and distribution of memory units. It supports memory exchange between models, institutions, and even industry-wide networks.

Users may declare memory as publishable and define visibility, usage conditions, and access control rules. Each shared unit carries unique IDs and provenance metadata; `MemGovernance` ensures masking, watermarking, and policy validation during dissemination.

`MemStore` enables both push and pull models of memory exchange. Consumers can define subscriptions using tags or semantic filters, and the system delivers matched updates proactively. Licensed memory assets can enforce contract-bound access frequencies and expiry policies. All access is logged with invocation traces to support audit and accountability.

For example, a hospital may publish de-identified diagnostic records for remote triage agents, with every call validated for context and provenance.

6 Evaluation

To systematically evaluate the capabilities of MemOS, we conduct both holistic and component-level experiments. We begin by benchmarking the full system on the LOCOMO benchmark suite to assess its performance in memory-intensive reasoning tasks, comparing against several state-of-the-art baselines. In addition, we present targeted evaluations of key architectural subsystems, including multi-perspective memory organization, hybrid semantic retrieval, task-aligned scheduling, and KV-based activation memory injection. These experiments assess the individual effectiveness of each component and its contribution to overall system performance.

6.1 End-to-End Evaluation on LOCOMO

Table 3 Comparison of LLM Judge Scores across five major tasks in the LOCOMO benchmark. Each bar shows the mean evaluation score judged by LLMs for a given method-task pair, with standard deviation as error bars. MemOS-0630 consistently outperforms baseline methods (LangMem, Zep, OpenAI, Mem0) across all task types, especially in multi-hop and temporal reasoning scenarios.

Category	Method	Chunk / Mem Tok	Top-K	LLMJudge Score	F1	RL	B1	B2	METEOR	BERT-F1	Sim
single hop	langmem	165	-	68.21±0.06	41.72	44.80	35.61	24.73	36.42	43.77	76.25
	zep	2320	20	50.42±0.29	32.49	35.07	27.38	18.81	28.49	35.26	66.95
	openai	4141	-	61.83±0.10	36.96	41.45	30.72	22.02	35.39	40.56	69.84
	mem0	1176	20	73.33±0.20	**47.26**	**51.44**	**40.34**	**30.02**	43.96	**48.53**	**76.56**
	memos-0630	1600	20	**78.44±0.11**	45.55	51.00	38.32	28.32	**44.46**	47.53	74.70
multi hop	langmem	185	-	56.74±0.29	**36.03**	**36.32**	**27.22**	**17.03**	29.14	33.03	**73.05**
	zep	2351	20	42.20±0.77	23.14	24.63	14.96	8.49	17.38	25.15	64.26
	openai	3924	-	60.28±0.00	33.10	35.36	23.84	15.36	27.25	32.36	68.82
	mem0	1163	20	58.75±0.44	35.24	34.87	25.91	16.55	27.90	32.65	70.71
	memos-0630	1528	20	**64.30±0.44**	35.57	36.25	26.71	16.59	**29.42**	**33.85**	69.60
open domain	langmem	209	-	49.65±1.30	**29.79**	30.54	**23.17**	11.72	21.03	**32.27**	**67.34**
	zep	2276	20	38.19±0.49	19.76	20.62	13.17	6.18	14.20	21.07	58.59
	openai	4071	-	32.99±1.30	17.19	15.88	11.04	5.23	12.05	19.37	57.53
	mem0	1141	20	45.83±0.00	27.80	28.67	20.01	10.59	20.33	28.38	63.74
	memos-0630	1511	20	**55.21±0.00**	29.64	**31.54**	22.40	**11.78**	**23.74**	30.36	63.06
temporal reasoning	langmem	134	-	24.09±0.39	38.10	38.33	32.23	18.81	27.55	48.30	74.57
	zep	2295	20	19.11±0.29	17.59	19.03	14.57	8.11	13.81	17.59	59.38
	openai	4048	-	28.25±0.59	23.90	24.47	18.25	11.87	19.35	23.11	59.53
	mem0	1173	20	52.34±0.25	45.40	46.90	38.15	22.27	34.60	44.59	76.15
	memos-0630	1655	20	**73.21±0.25**	**53.67**	**53.69**	**46.37**	**29.69**	**43.45**	**48.48**	**76.97**
overall	langmem	165	-	55.76±0.16	39.18	41.01	32.59	21.27	32.28	42.03	**74.76**
	zep	2318	20	41.62±0.21	26.88	28.91	21.55	13.90	22.51	28.84	64.36
	openai	4077	-	52.75±0.08	32.30	35.20	25.64	17.64	29.10	34.10	66.74
	mem0	1171	20	64.57±0.06	43.46	46.04	35.97	24.72	37.60	43.54	74.61
	memos-0630	1593	20	**73.31±0.05**	**44.42**	**47.65**	**36.88**	**25.43**	**40.20**	**44.15**	73.51

We evaluate MemOS against a diverse set of strong baselines, each representing a distinct memory system design paradigm. Specifically, *LangMem* applies hierarchical semantic retrieval over flat textual history; *Zep* integrates time-aware knowledge graphs with structured query resolution; *OpenAI-Memory* represents a commercial, closed-source memory module with opaque internal logic; and *Mem0* implements slot-based long-term memory with top-k semantic search. To ensure architectural parity, all methods are implemented over the same LLM backbone (GPT-4o-mini).

All experiments are conducted on an 80GB H800 GPU under identical hardware and software configurations.

For memory-augmented systems, we vary the number of retrieved items (Top-K) and chunk granularity (Chunk / Mem Tok), which controls the length of each retrieved memory segment. The configuration for each method is selected based on its best validation performance, ensuring a fair and optimized comparison across all metrics.

We report LLM-judge scores as the primary evaluation metric (Table 3), supported by standard generation quality indicators including F1, ROUGE-L (RL), BLEU-1/2 (B1/B2), METEOR, and BERTScore-F1 (BERT-F1), as well as cosine similarity (Sim) computed over semantic embeddings.

Overall, MemOS achieves the best average performance across all task categories, consistently outperforming strong baselines such as mem0, openai-memory, and zep. Across all sub-tasks in the LOCOMO benchmark, MemOS ranks among the top performers, maintaining first or second place in nearly every category. It demonstrates clear advantages in multi-hop and temporal reasoning, where long-range memory and contextual integration are especially critical. Beyond LLM-judge scores, MemOS also delivers strong generation quality across F1, ROUGE-L, and BLEU, particularly in long-form completeness and stylistic consistency. At the representation level, it maintains tight semantic alignment with reference answers, as indicated by consistently high cosine similarity in semantic embeddings across tasks.

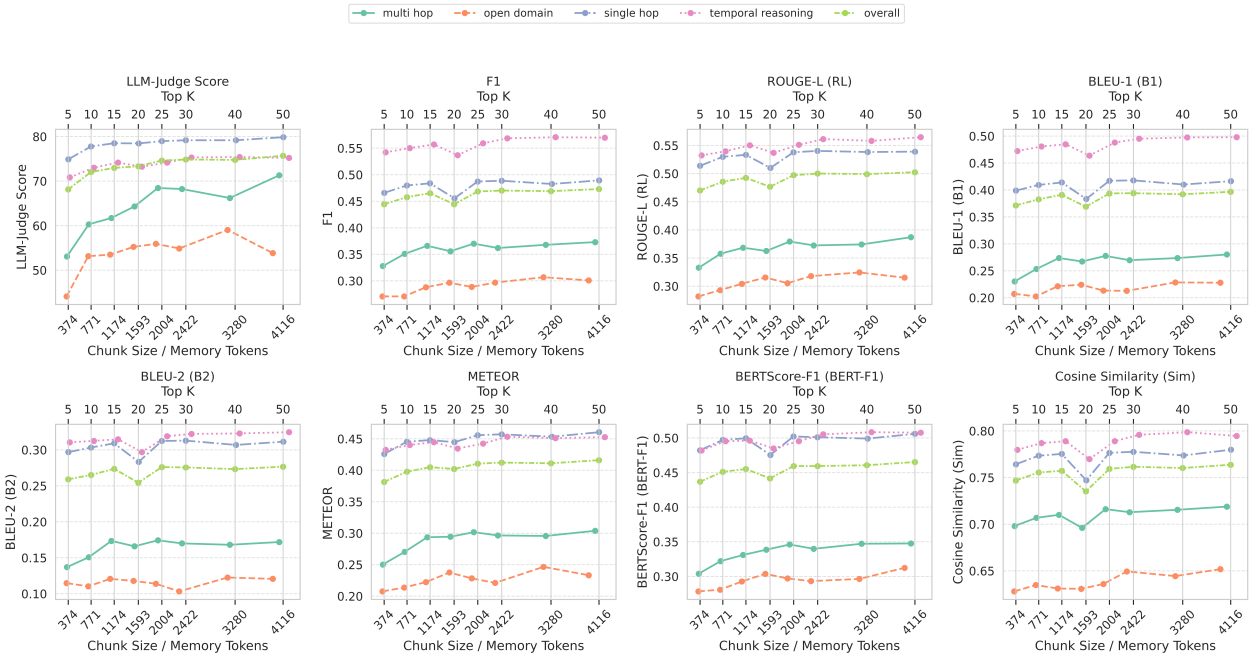

Figure 9 Performance trends of MemOS across memory configurations. We vary the number of retrieved memory chunks (Top-K, upper x-axis) and chunk size (lower x-axis, total memory tokens), and report performance on the LOCOMO benchmark across multiple metrics and task types. MemOS consistently maintains top-tier performance as memory capacity increases, with clear gains on multi-hop and temporal reasoning tasks. Cosine similarity results indicate stable semantic alignment throughout.

To better understand the impact of memory configuration, we conduct an ablation study by varying chunk sizes and Top-K retrieval depth. As shown in Figure 9, MemOS demonstrates stable and strong performance across all LOCOMO sub-tasks, with performance steadily improving as memory capacity increases—particularly for multi-hop and temporal reasoning tasks that demand long-range retrieval and contextual integration. In addition to higher LLM-Judge scores, generation metrics such as F1, ROUGE-L, and BLEU also benefit from memory expansion. Cosine similarity remains consistently high, indicating stable semantic alignment even with deeper retrieval.

These results collectively validate the effectiveness of MemOS's architectural innovations—particularly its hybrid semantic retrieval and memory-centric design—which enable accurate, fluent, and contextually aligned

responses under long-horizon constraints.

Table 4 Latency and LLM evaluation scores across various methods on the LOCOMO benchmark. RAG is evaluated under different chunk sizes and retrieval depths (Top-K = 1 or 2), while other baselines include standard retrieval systems and memory-augmented models. Metrics include LLMJudge scores (evaluating answer quality), search latency (P50/P95), and total end-to-end latency (P50/P95). MemOS-0630 achieves the highest LLM score with competitive latency performance.

Method	Chunk / Mem Tok	Top-K	LLMJudge Scores	search duration (ms)		total duration (ms)	
				P50	P95	P50	P95
RAG	128	1	44.61±0.05	516	800	1306	1963
		2	55.71±0.05	523	850	1325	2040
	256	1	45.13±0.19	553	1288	1438	2606
		2	56.54±0.25	575	1371	1496	2843
	512	1	41.36±0.24	481	1979	1331	4129
		2	54.29±0.19	482	2070	1351	4252
	1024	1	36.04±0.09	1008	2436	2061	4443
		2	46.97±0.03	468	808	1466	2193
	2048	1	33.70±0.05	460	986	1387	2311
		2	44.81±0.05	456	903	1476	2479
	4096	1	33.9±0.05	449	715	1432	2324
		2	48.53±0.13	459	1055	1606	3324
	8192	1	41.45±0.17	692	1733	2016	5037
		2	58.20±0.08	688	1773	2335	6008
Full-Context	22636	-	71.58±0.08	-	-	2339	7016
langmem	165	-	55.76±0.16	17226	29344	18025	30139
zep	2318	20	41.62±0.21	1364	1901	9777	20197
openai	4077	-	52.75±0.08	-	-	1184	2240
mem0	1171	20	64.57±0.06	1297	1416	4906	5962
memos-0630	1593	20	**73.31±0.05**	1758	1969	4942	7937

6.2 Evaluation of Memory Retrieval

We conduct a focused evaluation to analyze the efficiency and effectiveness of memory retrieval across representative system designs. As shown in Table 4, we compare latency and generation quality under different memory configurations, including standard RAG pipelines, memory-augmented models, and our proposed MemOS.

To test RAG-style retrieval systems, we systematically vary chunk sizes (from 128 to 8192 tokens) and Top-K values (1 or 2) to observe the trade-offs between context size, search latency, and LLM output quality. Larger chunk sizes reduce retrieval depth but increase encoding and integration cost, while smaller chunks allow finer granularity at the expense of retrieval breadth.

In addition to standard retrieval, we include full-context and commercial memory systems to establish upper and lower bounds. Notably, the full-context baseline—where the entire dialogue history is loaded into the model—achieves strong LLMJudge scores but suffers from prohibitively high latency due to extreme context length. LangMem and Zep incur substantial retrieval delays from graph traversal or multi-level indexing. OpenAI-Memory offers low latency but only moderate output quality, likely limited by opaque memory heuristics.

Remarkably, MemOS not only matches but surpasses the full-context baseline in LLMJudge scores—while operating at significantly lower latency. Despite managing over 1500 memory tokens, its retrieval time remains

close to smaller baselines such as mem0. This demonstrates that MemOS's hybrid semantic organization and activation-based memory loading can achieve superior performance without the cost of full-context inference.

6.3 Evaluation of KV-Based Memory Acceleration

To evaluate the effectiveness of KV-form memory acceleration within **MemOS**, we design a controlled experiment simulating realistic memory reuse scenarios.

During typical usage, the *MemScheduler* module in MemOS continuously monitors model interactions and automatically identifies the most frequently accessed and semantically stable plaintext memory entries. These entries are then converted into **activation memory**—a KV-format structure injected into the model's attention cache and proactively transferred to GPU memory for low-latency reuse.

Our evaluation assumes this realistic deployment: memory has already been preprocessed and cached on GPU in KV format, avoiding the need for repeated prompt encoding.

We compare two memory usage strategies: prompt-based memory injection, where memory entries are prepended to the input sequence, and KV-cache injection, where memory is injected directly as key-value pairs into the model's attention mechanism.

To simulate realistic inference conditions, we evaluate across three context lengths—short (583 tokens), medium (2773 tokens), and long (6064 tokens)—as well as three query types of increasing length and complexity: short (167 tokens), medium (302.7 tokens), and long (952.7 tokens). All experiments are conducted using the HuggingFace `transformers` library, running on a single NVIDIA H800 GPU with 80GB of memory under consistent system settings.

We report four metrics as shown in Table 5. "Build" time refers to the preprocessing duration needed to convert memory into KV format. "KV TTFT" denotes the first-token latency under KV-based memory injection, while "Dir TTFT" indicates the latency under prompt-based injection. "Speedup" reflects the relative latency reduction achieved by KV injection compared to direct prompt injection.

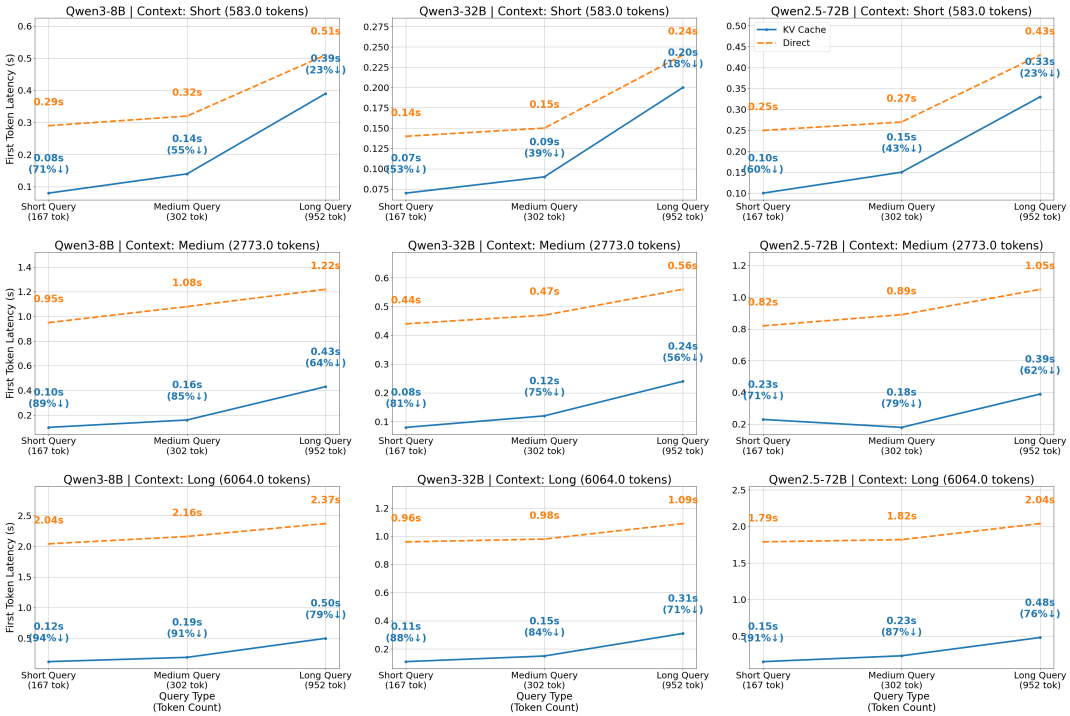

Figure 10 Time to First Token (TTFT) comparison across models, context lengths, and query lengths. KV-based memory injection consistently achieves lower latency with identical output.

Table 5 Evaluation of Time to First Token (TTFT) and acceleration effect across different models, context lengths, and query lengths using the HuggingFace `transformers` library. We compare two memory injection strategies: direct prompt-based injection and KV-based attention cache injection. Gray-highlighted rows correspond to MemOS's strategy, which uses KV-form memory injection and consistently achieves faster response without altering output semantics.

Model	Ctx	CtxTok	Qry	QryTok	Build (s)	KV TTFT (s)	Dir TTFT (s)	Speedup (%)
Qwen3-8B	long	6064	long	952.7	0.92	0.50	2.37	79.1
			medium	302.7	0.93	0.19	2.16	91.1
			short	167	0.93	0.12	2.04	94.2
	medium	2773	long	952.7	0.41	0.43	1.22	64.6
			medium	302.7	0.41	0.16	1.08	85.1
			short	167	0.43	0.10	0.95	89.7
	short	583	long	952.7	0.12	0.39	0.51	23.0
			medium	302.7	0.12	0.14	0.32	55.6
			short	167	0.12	0.08	0.29	71.3
Qwen3-32B	long	6064	long	952.7	0.71	0.31	1.09	71.4
			medium	302.7	0.71	0.15	0.98	84.3
			short	167	0.71	0.11	0.96	88.8
	medium	2773	long	952.7	0.31	0.24	0.56	56.9
			medium	302.7	0.31	0.12	0.47	75.1
			short	167	0.31	0.08	0.44	81.2
	short	583	long	952.7	0.09	0.20	0.24	18.6
			medium	302.7	0.09	0.09	0.15	39.6
			short	167	0.09	0.07	0.14	53.5
Qwen2.5-72B	long	6064	long	952.7	1.26	0.48	2.04	76.4
			medium	302.7	1.26	0.23	1.82	87.2
			short	167	1.27	0.15	1.79	91.4
	medium	2773	long	952.7	0.58	0.39	1.05	62.7
			medium	302.7	0.58	0.18	0.89	79.2
			short	167	0.71	0.23	0.82	71.6
	short	583	long	952.7	0.16	0.33	0.43	23.8
			medium	302.7	0.16	0.15	0.27	43.2
			short	167	0.16	0.10	0.25	60.5

The results (Table 5 and Figure 10) confirm that KV-based memory injection yields substantial TTFT reduction across all models and configurations. The output sequences remain identical under both methods, validating their semantic equivalence. Acceleration is especially significant for larger models and longer contexts—for instance, Qwen2.5-72B achieves a 91.4% reduction in TTFT under long-context, short-query conditions. These findings highlight KV memory as a practical and effective technique for low-latency execution in memory-augmented language models.

7 MemOS for Architecture Innovation and Applications

7.1 Architectural Innovations Enabled by MemOS

MemOS treats memory as a first-class system resource, enabling unified lifecycle management and orchestration of memory in multiple forms. This abstraction supports architectural innovations that focus on memory-driven modules and services, facilitating the modularization and reusability of knowledge assets.

7.1.1 Paid Memory as Modular Installables (User-Facing Paradigm)

MemOS is designed around a memory-centric architecture, offering modularized and assetized memory interfaces that allow knowledge to be uploaded, mounted, and invoked like a digital resource. Under this paradigm, memory is no longer bound to training pipelines or development workflows but becomes a composable and user-controllable intelligence unit.

Concretely, domain experts can publish structured experiential memories via `MemStore`, akin to publishing a knowledge plugin or an expert tip. Consumers—students, enterprise agents, or assistant models—can install these memories using a standardized loading interface, subject to permission control. This entire flow

abstracts away the need for understanding the underlying model architecture or performing manual alignment. It drastically reduces the barrier to memory usage and makes memory-driven intelligence available beyond developers and platform operators.

For instance, a medical student in clinical rotation may wish to study how to manage a rare autoimmune condition. An experienced physician can encapsulate diagnostic heuristics, questioning paths, and typical case patterns into a structured memory and upload it to `MemStore`. The student can then search, install, and invoke this memory locally via their assistant model. This bypasses the need for building formal ontologies or coordinating structured knowledge base design, as is common in traditional clinical AI.

MemOS encapsulates this process as a standardized "Memory-as-a-Service" capability, greatly expanding the accessibility and reusability of expert knowledge. Furthermore, `MemGovernance`, the dedicated control module in MemOS, offers full-spectrum privacy and access control for memory assets. It enables memory providers to define custom access conditions over their published content. For example, a medical expert may restrict installation rights to users who have completed a micropayment, enabling a form of licensed intelligence delivery.

7.1.2 Painless Memory Management (Task-Oriented Paradigm)

MemOS abstracts memory as a universal, long-lived, and shareable infrastructure resource, architecturally analogous to storage subsystems in traditional operating systems. This design elevates memory from a model-embedded utility to a first-class system-level asset with its own lifecycle and invocation semantics.

Unlike conventional transient memory techniques limited to context windows or parameter embeddings, MemOS offers standardized memory interfaces, a unified access protocol, and structured persistence formats. This enables runtime tasks to flexibly read, write, mount, fuse, or replace memory blocks on demand, without requiring manual state tracking or architectural alignment.

Neither users nor developers need to handle low-level vector indexing, KV-caching, or context orchestration logic. Instead, they can access and update memory seamlessly through task-level `Memory API` calls. This infrastructure-level abstraction proves especially valuable in multi-stage, long-horizon, and evolving tasks.

For example, in an intelligent legal assistant system, a user may complete a corporate contract review task in distinct phases: the first phase may focus on structural layout and terminological consistency; the second phase may highlight risky clauses and compare precedent cases; and the final phase may involve checking compliance against current regulations. MemOS dynamically loads the appropriate memory sets at each stage (e.g., “Contract Template Memory”, “Risk Clause Case Logs”, “Recent Regulation Digest”), and performs hot-swapping and cache eviction as task contexts evolve. Throughout the task lifecycle, the user need not explicitly manage memory policies; the system automatically schedules the relevant memory assets based on context semantics, delivering a “memory-as-resource, use-on-demand” intelligent task execution experience.

7.2 MemOS Application Scenarios

7.2.1 Supporting Multi-Turn Dialogue and Cross-Task Continuity

Real-world interactions rarely reveal user intent in a single turn; instead, goals are refined progressively over multiple exchanges. However, traditional LLMs rely on static context windows, making it difficult to retain key semantic states across turns, resulting in “memory loss” between dialogue rounds.

For instance, in a procurement negotiation task, a user might set a budget cap of ¥300,000 in round 5, later revise product preferences in round 12 to prioritize domestic alternatives, yet by round 15, the model reverts to recommending high-priced imports based on earlier defaults.

MemOS addresses this at the system level by extracting salient elements (e.g., budget, preferences, delivery constraints) after each user input and encoding them into structured “conversation memory units.” These are linked to the ongoing task’s long-term memory path via `MemLink`.

During inference, `MemScheduler` retrieves relevant historical fragments based on current context and integrates them into the active reasoning path. This ensures continuity of semantic state and prevents logic drift due to

"context sliding."

Furthermore, MemOS supports cross-task memory reuse to enable dialogue continuity and state persistence. For example, after completing an auto-form-filling task, the system retains memory of ID details or user habits. When the user later initiates a "visa application" task, MemOS recalls the previously stored data (e.g., from "passport issuance"), enabling seamless state transition across tasks.

7.2.2 Supporting Knowledge Evolution and Continuous Update

Modern knowledge is dynamic, yet LLMs are generally trained once with static datasets. Updating their internal knowledge either requires expensive fine-tuning or introduces risks like catastrophic forgetting. Even RAG approaches lack lifecycle, version, or governance mechanisms—leading to fragmented, unverifiable external knowledge.

MemOS redefines knowledge as dynamic, lifecycle-governed memory. Each memory unit evolves independently, with defined stages for generation, replacement, fusion, and deprecation. The system schedules updates based on usage frequency, contextual alignment, and semantic overlap.

For example, when updated clinical guidelines are published, medical authorities can release them as explicit memory blocks via `MemStore`. MemOS tags them as "trusted sources," compares them with older versions, and suggests updates to users.

At inference time, `MemScheduler` prioritizes trusted and active versions, while obsolete entries are archived. This allows the model to remain up-to-date without retraining or harming prior knowledge structures.

MemOS also supports personalized knowledge development. For instance, a cancer specialist may iteratively add interpretations and heuristics to drug usage. Over time, these refinements are integrated into their personal memory path, coexisting with official guidelines and selected based on task context.

7.2.3 Enabling Personalization and Multi-Role Modeling

LLMs today often operate statelessly across users and roles, unable to remember stylistic preferences or distinguish between user roles in complex settings. As a result, users must re-specify information every time, and models struggle to maintain consistent identity or behavior.

MemOS provides system-level support for identity-aware memory and role-based behavior. Each user identity is associated with dedicated memory spaces, and multiple roles can coexist under one account.

For example, a user may interact as both a "parent" managing home tasks and a "manager" handling contracts. MemOS keeps memory streams separate and dynamically loads the appropriate persona during inference.

In addition, long-term interaction patterns are encoded into "personal memory units" capturing language tone, response preferences, or value leanings. These units are incorporated into inference, yielding a personalized and coherent AI behavior.

In enterprise contexts, MemOS allows deployment of predefined role templates with task scopes, permission controls, and memory sync strategies. For example, an organization may define roles for analysts, assistants, and project leads, each with distinct memory access and agent behavior.

7.2.4 Enabling Cross-Platform Memory Migration

In a world of multi-device, multi-agent environments, valuable user-model memories often become locked within individual platforms, creating "memory silos" that break continuity and fragment knowledge accumulation.

MemOS resolves this through standardized memory representations, encryption, and platform-agnostic mount protocols. All memory blocks are portable across environments—from mobile to cloud to enterprise infrastructure.

For example, a user's "family travel preference" memory built via mobile assistant—including flight timing, hotel type, and budget—can be selectively migrated to a corporate travel planning agent on desktop, enabling consistent and efficient decision-making.

By breaking the memory silo, MemOS transforms memory from a private asset embedded in a single model to a distributed, governable, and reusable intelligence layer across platforms.

8 Conclusion

In this work, we introduce a memory operating system designed for Large Language Models, aimed at collaboratively building foundational memory infrastructure for next-generation LLM applications.

MemOS provides a unified abstraction and integrated management framework for heterogeneous memory types, including parameter memory, activation memory, and explicit plaintext memory. We propose a standardized memory unit, `MemCube`, and implement key modules for scheduling, lifecycle management, structured storage, and transparent augmentation. These components collectively enhance reasoning coherence, adaptability, and system scalability in LLMs.

Building on this foundation, we envision a future intelligent ecosystem centered on modular memory resources and supported by a decentralized memory marketplace. This paradigm shift enables the creation of next-generation AI systems capable of continual learning and long-term evolution.

Looking ahead, we plan to explore the following directions:

- **Cross-LLM Memory Sharing**: Enable interoperability and module reuse across different foundation models by sharing parametric and activation memories. To support consistent semantics and secure exchange, we plan to extend the **Memory Interchange Protocol (MIP)** to define standard formats, compatibility rules, and trust mechanisms for cross-model/app memory transmission—facilitating collaborative knowledge transfer among agents.
- **Self-Evolving MemBlocks**: Develop memory units capable of self-optimization, reconstruction, and evolution based on usage feedback, reducing the need for manual maintenance and supervision.
- **Scalable Memory Marketplace**: Establish decentralized mechanisms for memory exchange, supporting asset-level transactions, collaborative updates, and distributed evolution to foster a sustainable AI ecosystem.

Overall, with the introduction of MemOS, we aim to transform LLMs from closed, static generation systems to continuously evolving intelligent agents equipped with long-term memory, integrated knowledge, and behavioral plasticity. MemOS not only addresses critical architectural limitations in current models but also lays the groundwork for cross-task, cross-platform, and multi-agent collaborative intelligence. Building on prior work demonstrating the potential of explicit memory and hierarchical memory representations in LLMs [1], we look forward to advancing the frontiers of MemOS in collaboration with the community, making memory a first-class computational resource in the age of general-purpose AI.

References

[1] Hongkang Yang, Zehao Lin, Wenjin Wang, Hao Wu, Zhiyu Li, Bo Tang, Wenqiang Wei, Jinbo Wang, Zeyun Tang, Shichao Song, Chenyang Xi, Yu Yu, Kai Chen, Feiyu Xiong, Linpeng Tang, and Weinan E. Memory^3: Language modeling with explicit memory. Journal of Machine Learning, 3(3):300–346, January 2024.

[2] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 1(2), 2023.

[3] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. Codet5+: Open code large language models for code understanding and generation. arXiv preprint arXiv:2305.07922, 2023.

[4] Shengsheng Qian, Zuyi Zhou, Dizhan Xue, Bing Wang, and Changsheng Xu. From linguistic giants to sensory maestros: A survey on cross-modal reasoning with large language models. arXiv preprint arXiv:2409.18996, 2024.

[5] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. Retrieval-augmented generation for ai-generated content: A survey. CoRR, abs/2402.19473, 2024.

[6] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. CoRR, abs/2312.10997, 2023.

[7] Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Junnan Dong, Hao Chen, Yi Chang, and Xiao Huang. A survey of graph retrieval-augmented generation for customized large language models. CoRR, abs/2501.13958, 2025.

[8] Bo Ni, Zheyuan Liu, Leyao Wang, Yongjia Lei, Yuying Zhao, Xueqi Cheng, Qingkai Zeng, Luna Dong, Yinglong Xia, Krishnaram Kenthapadi, Ryan A. Rossi, Franck Dernoncourt, Md. Mehrab Tanjim, Nesreen K. Ahmed, Xiaorui Liu, Wenqi Fan, Erik Blasch, Yu Wang, Meng Jiang, and Tyler Derr. Towards trustworthy retrieval augmented generation for large language models: A survey. CoRR, abs/2502.06872, 2025.

[9] Howard Chen, Ramakanth Pasunuru, Jason Weston, and Asli Celikyilmaz. Walking down the memory maze: Beyond context limit through interactive reading. CoRR, abs/2310.05029, 2023.

[10] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. CoRR, abs/2404.16130, 2024.

[11] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. Lightrag: Simple and fast retrieval-augmented generation. CoRR, abs/2410.05779, 2024.

[12] Microsoft. Retrieval augmented generation (rag) in azure ai search, 2025.

[13] Google. Vertex ai search, 2025.

[14] Elastic. Build innovative ai search experiences, 2025.

[15] Nuclia. Agentic rag-as-a-service company, 2025.

[16] Tu Vu, Mohit Iyyer, Xuezhi Wang, Noah Constant, Jerry Wei, Jason Wei, Chris Tar, Yun-Hsuan Sung, Denny Zhou, Quoc Le, and Thang Luong. Freshllms: Refreshing large language models with search engine augmentation, 2023.

[17] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

[18] Cursor - The AI Code Editor.

[19] Yiming Du, Wenyu Huang, Danna Zheng, Zhaowei Wang, Sebastien Montella, Mirella Lapata, Kam-Fai Wong, and Jeff Z. Pan. Rethinking Memory in AI: Taxonomy, Operations, Topics, and Future Directions, May 2025. arXiv:2505.00675 [cs].

[20] Yaxiong Wu, Sheng Liang, Chen Zhang, Yichao Wang, Yongyue Zhang, Huifeng Guo, Ruiming Tang, and Yong Liu. From Human Memory to AI Memory: A Survey on Memory Mechanisms in the Era of LLMs, April 2025. arXiv:2504.15965 [cs].

[21] Lianlei Shan, Shixian Luo, Zezhou Zhu, Yu Yuan, and Yong Wu. Cognitive Memory in Large Language Models, April 2025. arXiv:2504.02441 [cs].

[22] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners.

[23] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020. arXiv:2005.14165 [cs].

[24] Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation. 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), pages 4582–4597, Online, 2021. Association for Computational Linguistics.

[25] Brian Lester, Rami Al-Rfou, and Noah Constant. The Power of Scale for Parameter-Efficient Prompt Tuning. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 3045–3059, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics.

[26] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. GPT Understands, Too, October 2023. arXiv:2103.10385 [cs].

[27] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-Tuning: Prompt Tuning Can Be Comparable to Fine-tuning Across Scales and Tasks. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 61–68, Dublin, Ireland, 2022. Association for Computational Linguistics.

[28] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, March 2022. arXiv:2203.02155 [cs].

[29] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In Proceedings of the 29th Symposium on Operating Systems Principles, pages 611–626, Koblenz Germany, October 2023. ACM.

[30] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks. October 2023.

[31] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. Advances in Neural Information Processing Systems, 36:34661–34710, December 2023.

[32] Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. Get More with LESS: Synthesizing Recurrence with KV Cache Compression for Efficient LLM Inference. June 2024.

[33] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun S. Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. Advances in Neural Information Processing Systems, 37:1270–1303, December 2024.

[34] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. RetrievalAttention: Accelerating Long-Context LLM Inference via Vector Retrieval, December 2024. arXiv:2409.10516 [cs].

[35] Nishant Subramani, Nivedita Suresh, and Matthew Peters. Extracting Latent Steering Vectors from Pretrained Language Models. In Findings of the Association for Computational Linguistics: ACL 2022, pages 566–581, Dublin, Ireland, 2022. Association for Computational Linguistics.

[36] Sheng Liu, Haotian Ye, Lei Xing, and James Zou. In-context Vectors: Making In Context Learning More Effective and Controllable Through Latent Space Steering, February 2024. arXiv:2311.06668 [cs].

[37] Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering Language Models With Activation Engineering, October 2024. arXiv:2308.10248 [cs].

[38] Kai Konen, Sophie Jentzsch, Diaoulé Diallo, Peer Schütt, Oliver Bensch, Roxanne El Baff, Dominik Opitz, and Tobias Hecking. Style vectors for steering generative large language models. In Yvette Graham and Matthew Purver, editors, Findings of the Association for Computational Linguistics: EACL 2024, pages 782–802, St. Julian's, Malta, March 2024. Association for Computational Linguistics.

[39] Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering Llama 2 via Contrastive Activation Addition. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 15504–15522, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[40] Zijian Feng, Hanzhang Zhou, Kezhi Mao, and Zixiao Zhu. FreeCtrl: Constructing Control Centers with Feedforward Layers for Learning-Free Controllable Text Generation. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 7627–7640, Bangkok, Thailand, 2024. Association for Computational Linguistics.

[41] Ziwen Xu, Shuxun Wang, Kewei Xu, Haoming Xu, Mengru Wang, Xinle Deng, Yunzhi Yao, Guozhou Zheng, Huajun Chen, and Ningyu Zhang. EasyEdit2: An Easy-to-use Steering Framework for Editing Large Language Models, April 2025. arXiv:2504.15133 [cs].

[42] Yuxin Xiao, Chaoqun Wan, Yonggang Zhang, Wenxiao Wang, Binbin Lin, Xiaofei He, Xu Shen, and Jieping Ye. Enhancing Multiple Dimensions of Trustworthiness in LLMs via Sparse Activation Control. November 2024.

[43] Yu Li, Han Jiang, Chuanyang Gong, and Zhihua Wei. DESTEIN: Navigating Detoxification of Language Models via Universal Steering Pairs and Head-wise Activation Fusion, August 2024. arXiv:2404.10464 [cs].

[44] Chi Han, Jialiang Xu, Manling Li, Yi Fung, Chenkai Sun, Nan Jiang, Tarek Abdelzaher, and Heng Ji. Word Embeddings Are Steers for Language Models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 16410–16430, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[45] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through Memorization: Nearest Neighbor Language Models. September 2019.

[46] Luiza Pozzobon, Beyza Ermis, Patrick Lewis, and Sara Hooker. Goodtriever: Adaptive Toxicity Mitigation with Retrieval-augmented Models. In Findings of the Association for Computational Linguistics: EMNLP 2023, pages 5108–5125, Singapore, 2023. Association for Computational Linguistics.

[47] Tianyang Xu, Haojie Zheng, Chengze Li, Haoxiang Chen, Yixin Liu, Ruoxi Chen, and Lichao Sun. Noderag: Structuring graph-based rag with heterogeneous nodes, 2025.

[48] Peiru Yang, Xintian Li, Zhiyang Hu, Jiapeng Wang, Jinhua Yin, Huili Wang, Lizhi He, Shuai Yang, Shangguang Wang, Yongfeng Huang, and Tao Qi. Heterag: A heterogeneous retrieval-augmented generation framework with decoupled knowledge representations, 2025.

[49] Haoran Luo, Haihong E, Guanting Chen, Yandan Zheng, Xiaobao Wu, Yikai Guo, Qika Lin, Yu Feng, Ze-min Kuang, Meina Song, Yifan Zhu, and Luu Anh Tuan. Hypergraphrag: Retrieval-augmented generation with hypergraph-structured knowledge representation. CoRR, abs/2503.21322, 2025.

[50] Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, 2024.

[51] Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From RAG to memory: Non-parametric continual learning for large language models. CoRR, abs/2502.14802, 2025.

[52] Xiang Liang, Simin Niu, Zhiyu Li, Sensen Zhang, Shichao Song, Hanyu Wang, Jiawei Yang, Feiyu Xiong, Bo Tang, and Chenyang Xi. Empowering large language models to set up a knowledge retrieval indexer via self-learning. CoRR, abs/2405.16933, 2024.

[53] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. CoRR, abs/2501.13956, 2025.

[54] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: agentic memory for LLM agents. CoRR, abs/2502.12110, 2025.

[55] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025.

[56] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[57] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback, April 2022. arXiv:2204.05862 [cs].

[58] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. CTRL: A Conditional Transformer Language Model for Controllable Generation, September 2019. arXiv:1909.05858 [cs].

[59] Tianxiang Chen, Zhentao Tan, Tao Gong, Yue Wu, Qi Chu, Bin Liu, Jieping Ye, and Nenghai Yu. Llama SLayer 8B: Shallow Layers Hold the Key to Knowledge Injection. In Findings of the Association for Computational Linguistics: EMNLP 2024, pages 5991–6002, Miami, Florida, USA, 2024. Association for Computational Linguistics.

[60] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models, October 2021. arXiv:2106.09685 [cs].

[61] Weihang Su, Yichen Tang, Qingyao Ai, Junxi Yan, Changyue Wang, Hongning Wang, Ziyi Ye, Yujia Zhou, and Yiqun Liu. Parametric Retrieval Augmented Generation, January 2025. arXiv:2501.15915 [cs].

[62] Yuqiao Tan, Shizhu He, Huanxuan Liao, Jun Zhao, and Kang Liu. Better wit than wealth: Dynamic Parametric Retrieval Augmented Generation for Test-time Knowledge Enhancement, March 2025. arXiv:2503.23895 [cs].

[63] Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. Memory-Based Model Editing at Scale. In Proceedings of the 39th International Conference on Machine Learning, pages 15817–15831. PMLR, June 2022. ISSN: 2640-3498.

[64] Qingxiu Dong, Damai Dai, Yifan Song, Jingjing Xu, Zhifang Sui, and Lei Li. Calibrating Factual Knowledge in Pretrained Language Models. In Findings of the Association for Computational Linguistics: EMNLP 2022, pages 5937–5947, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics.

[65] Xin Cheng, Yankai Lin, Xiuying Chen, Dongyan Zhao, and Rui Yan. Decouple knowledge from paramters for plug-and-play language modeling. In Findings of the Association for Computational Linguistics: ACL 2023, pages 14288–14308, Toronto, Canada, 2023. Association for Computational Linguistics.

[66] Thomas Hartvigsen, Swami Sankaranarayanan, Hamid Palangi, Yoon Kim, and Marzyeh Ghassemi. Aging with GRACE: Lifelong Model Editing with Discrete Key-Value Adaptors, October 2023. arXiv:2211.11031 [cs].

[67] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and Editing Factual Associations in GPT, January 2023. arXiv:2202.05262 [cs].

[68] Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-Editing Memory in a Transformer, August 2023. arXiv:2210.07229 [cs].

[69] Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Shi Jie, Xiang Wang, Xiangnan He, and Tat-seng Chua. AlphaEdit: Null-Space Constrained Knowledge Editing for Language Models, March 2025. arXiv:2410.02355 [cs].

[70] Houcheng Jiang, Junfeng Fang, Ningyu Zhang, Guojun Ma, Mingyang Wan, Xiang Wang, Xiangnan He, and Tat-seng Chua. AnyEdit: Edit Any Knowledge Encoded in Language Models, February 2025. arXiv:2502.05628 [cs].

[71] Ningyu Zhang, Yunzhi Yao, Bozhong Tian, Peng Wang, Shumin Deng, Mengru Wang, Zekun Xi, Shengyu Mao, Jintian Zhang, Yuansheng Ni, Siyuan Cheng, Ziwen Xu, Xin Xu, Jia-Chen Gu, Yong Jiang, Pengjun Xie, Fei Huang, Lei Liang, Zhiqiang Zhang, Xiaowei Zhu, Jun Zhou, and Huajun Chen. A Comprehensive Study of Knowledge Editing for Large Language Models, November 2024. arXiv:2401.01286 [cs].

[72] Qi Li and Xiaowen Chu. Can We Continually Edit Language Models? On the Knowledge Attenuation in Sequential Model Editing. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, Findings of the Association for Computational Linguistics: ACL 2024, pages 5438–5455, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[73] Daniel Tamayo, Aitor Gonzalez-Agirre, Javier Hernando, and Marta Villegas. Mass-Editing Memory with Attention in Transformers: A cross-lingual exploration of knowledge. In Findings of the Association for Computational Linguistics ACL 2024, pages 5831–5847, 2024. arXiv:2502.02173 [cs].

[74] Mingyu Jin, Weidi Luo, Sitao Cheng, Xinyi Wang, Wenyue Hua, Ruixiang Tang, William Yang Wang, and Yongfeng Zhang. Disentangling Memory and Reasoning Ability in Large Language Models, November 2024. arXiv:2411.13504 [cs].

[75] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. CoRR, abs/2501.00663, 2025.

[76] Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. Editing Large Language Models: Problems, Methods, and Opportunities, November 2023. arXiv:2305.13172 [cs].

[77] Xin Xu, Wei Xu, Ningyu Zhang, and Julian McAuley. BiasEdit: Debiasing Stereotyped Language Models via Model Editing, March 2025. arXiv:2503.08588 [cs].

[78] Nicola De Cao, Wilker Aziz, and Ivan Titov. Editing Factual Knowledge in Language Models, September 2021. arXiv:2104.08164 [cs].

[79] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Fast Model Editing at Scale. October 2021.

[80] Chenmien Tan, Ge Zhang, and Jie Fu. Massive Editing for Large Language Models via Meta Learning. October 2023.

[81] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-Aware Sparsity for Efficient Long-Context LLM Inference. June 2024.

[82] Chak Leong, Yi Cheng, Jiashuo Wang, Jian Wang, and Wenjie Li. Self-Detoxifying Language Models via Toxification Reversal. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pages 4433–4449, Singapore, 2023. Association for Computational Linguistics.

[83] Tianlong Wang, Xianfeng Jiao, Yinghao Zhu, Zhongzhi Chen, Yifan He, Xu Chu, Junyi Gao, Yasha Wang, and Liantao Ma. Adaptive Activation Steering: A Tuning-Free LLM Truthfulness Improvement Method for Diverse Hallucinations Categories. In Proceedings of the ACM on Web Conference 2025, WWW '25, pages 2562–2578, New York, NY, USA, April 2025. Association for Computing Machinery.

[84] Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-Time Intervention: Eliciting Truthful Answers from a Language Model. Advances in Neural Information Processing Systems, 36:41451–41530, December 2023.

[85] Pengyu Wang, Dong Zhang, Linyang Li, Chenkun Tan, Xinghao Wang, Mozhi Zhang, Ke Ren, Botian Jiang, and Xipeng Qiu. InferAligner: Inference-Time Alignment for Harmlessness through Cross-Model Guidance. pages 10460–10479, November 2024.

[86] Alessandro Stolfo, Vidhisha Balachandran, Safoora Yousefi, Eric Horvitz, and Besmira Nushi. Improving Instruction-Following in Language Models through Activation Steering, April 2025. arXiv:2410.12877 [cs].

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

[88] Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open Foundation and Fine-Tuned Chat Models, July 2023. arXiv:2307.09288 [cs].

[89] Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, Jie Zhou, Siming Chen, Tao Gui, Qi Zhang, and Xuanjing Huang. A Comprehensive Capability Analysis of GPT-3 and GPT-3.5 Series Models, December 2023. arXiv:2303.10420 [cs].

[90] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the Middle: How Language Models Use Long Contexts. Transactions of the Association for Computational Linguistics, 12:157–173, 2024. Place: Cambridge, MA Publisher: MIT Press.

[91] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A Survey on In-context Learning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 1107–1128, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[92] Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. Found. Trends Inf. Retr., 3(4):333–389, 2009.

[93] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 3980–3990, 2019.

[94] Langchaln. Ensemble retriever. https://python.langchain.com/v0.1/docs/modules/data_connection/retrievers/ensemble.

[95] Jiale Wei, Xiang Ying, Tao Gao, Fangyi Bao, Felix Tao, and Jingbo Shang. Ai-native memory 2.0: Second me, 2025.

[96] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversation framework. CoRR, abs/2308.08155, 2023.

[97] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory, April 2025. arXiv:2504.19413 [cs].

[98] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems, February 2024. arXiv:2310.08560 [cs].

[99] Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. Lima: Less is more for alignment, 2023.