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], Zhengren 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

Presented by:

John Tan Chong Min

Conventional Memory in LLMs: Retrieval Augmented Generation

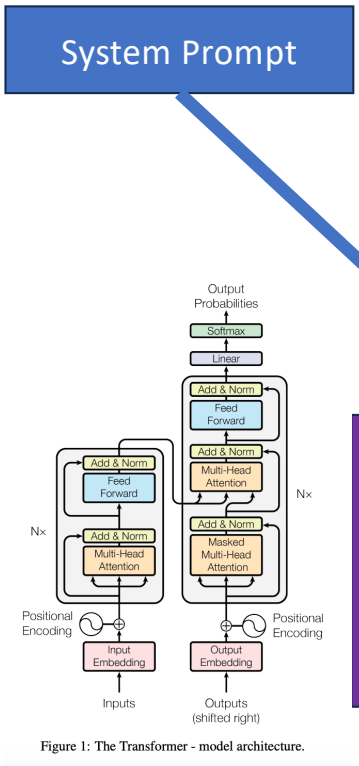


Figure 1: The Transformer - model architecture.

System Prompt

User Query

top k chunks

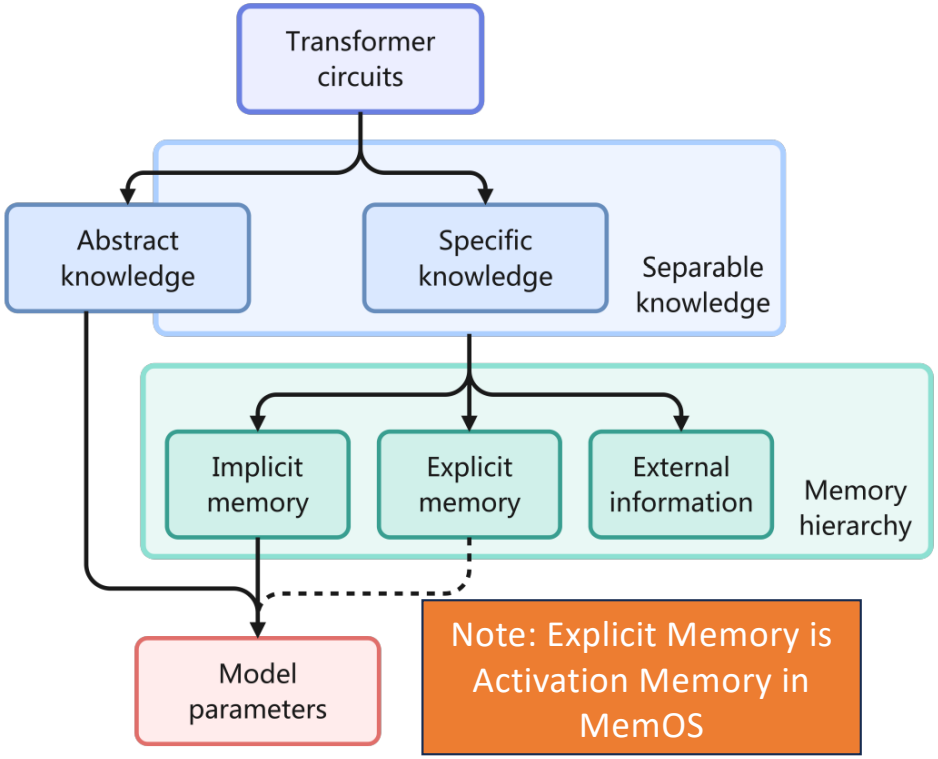
External Database (for augmented context)

LLM (Internal memory as parameters)

Response to User

- Pre-training memory is in LLM model parameters
- Inference time memory is in database

LLM Knowledge Structure



- Most memory are already ”hard-coded” into parameters of LLM
- Little or no learning of external memory information

Memory^3 : Language modeling with explicit memory. Yang et al. 2024.

Explicit Memory and how humans learn

- Plain LLMs to patients with impaired explicit memory, e.g. due to injury to the medial temporal lobe. These patients are largely unable to learn semantic knowledge (usually stored as explicit memory), but can acquire sensorimotor skills through repetitive priming (stored as implicit memories)

- Thus, one may hypothesize that due to **the lack of explicit memory**, the **training of plain LLMs is as inefficient** as repetitive priming, and thus has ample room for improvement.

Memory^3 : Language modeling with explicit memory. Yang et al. 2024.

Explicit Memory and how humans learn

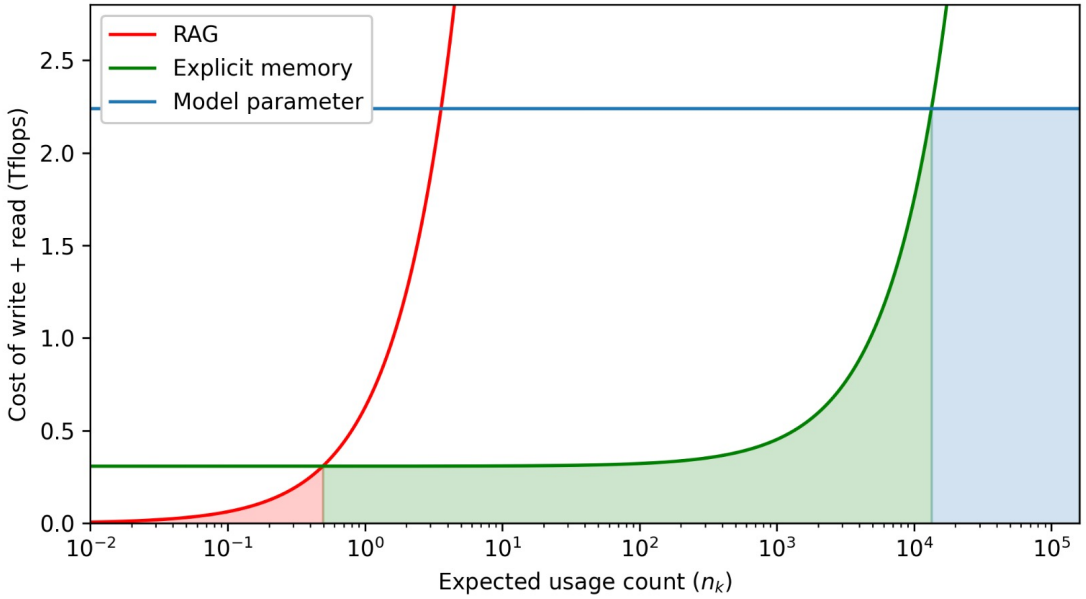


Figure 4: The total cost (TFlops) of writing and reading a piece of knowledge by our 2.4B model with respect to its expected usage count. The curves represent the cost of different memory formats, and the shaded area represents the minimum cost given the optimal format. The plot indicates that $(0.494, 13400)$ is the advantage interval for explicit memory. The calculations are provided in Appendix A. (The blue curve is only a lower bound on the cost of model parameters.)

- Here, explicit memory is stored as a **KV cache generated from each reference prior to inference**, but made sparse
- This means there is no need to recompute the KV representation from the base tokens of the reference again
- Memory encoded across **all layers** of transformer

Memory^3 : Language modeling with explicit memory. Yang et al. 2024.

Explicit Memory Math

Each explicit memory is a subset of the attention key-values from a subset of attention heads when encoding a reference. Thus, during inference, the LLM can directly read the retrieved explicit memories through its self-attention layers by concatenating them with the usual context key-values (Figure 9). Specially, for each attention head h at layer l, if it is chosen as a memory head, then its output $Y^{l,h}$ changes from the usual

$$Y_i^{l,h} = \text{softmax}\Big(\frac{X_i^{l,h} W_Q^{l,h} \big(X_{[:i]}^{l,h} W_K^{l,h}\big)^T}{\sqrt{d_h}}\Big) X_{[:i]}^{l,h} W_V^{l,h} W_O^{l,h}$$

where $X_{[:i]}$ denotes all tokens before or at position i and d_h denotes the head dimension, to

$$Y_i^{l,h} = \text{softmax}\Big(\frac{X_i^{l,h} W_Q^{l,h} \cdot \text{concat}\big(K_0^{l,h}, \dots K_4^{l,h}, X_{[:i]}^{l,h} W_K^{l,h}\big)^T}{\sqrt{d_h}}\Big) \text{concat}\big(V_0^{l,h}, \dots V_4^{l,h}, X_{[:i]}^{l,h} W_V^{l,h}\big) W_O^{l,h} \quad (4)$$

where each (K_j, V_j) denotes the keys and values of an explicit memory.

Memory^3 : Language modeling with explicit memory. Yang et al. 2024.

Why memory as an OS?

- Existing memory is static
  - Static parameters in the neural network
  - Static external memory chunks

- Memory needs to be updated dynamically for fast adaptation
  - Retain, Compress, Discard, Prioritise what is needed

- Memory to be stored and created within the model processing pipeline rather than an external one

Overview of MemOS

- **MemOS**: a memory operating system that treats **memory as a manageable system resource**

- **Unifies** the representation, scheduling, and evolution of **plaintext, activation-based, and parameter-level memories**, enabling cost-efficient storage and retrieval

- Basic unit, **MemCube**, encapsulates both memory content and metadata such as provenance and versioning
  - Can be composed, migrated, and fused over time, enabling flexible transitions between memory types and bridging retrieval with parameter-based learning

Good benchmark results on LOCOMO

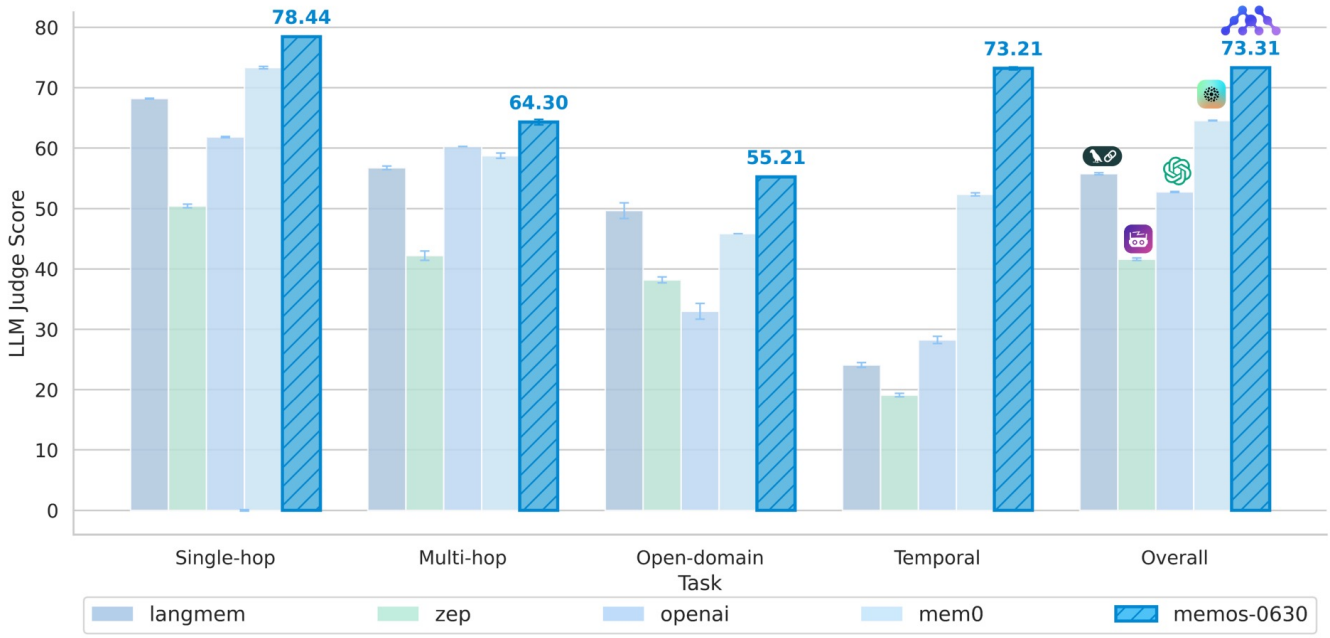


Figure 1 MemOS achieves state-of-the-art performance across all reasoning tasks. This figure summarizes LLM-Judge scores on the LOCOMO benchmark, covering four task categories (Single-hop, Multi-hop, Open-domain, Temporal Reasoning) and the overall average. MemOS (MemOS-0630) consistently ranks first 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.

Memory Development: From static to dynamic

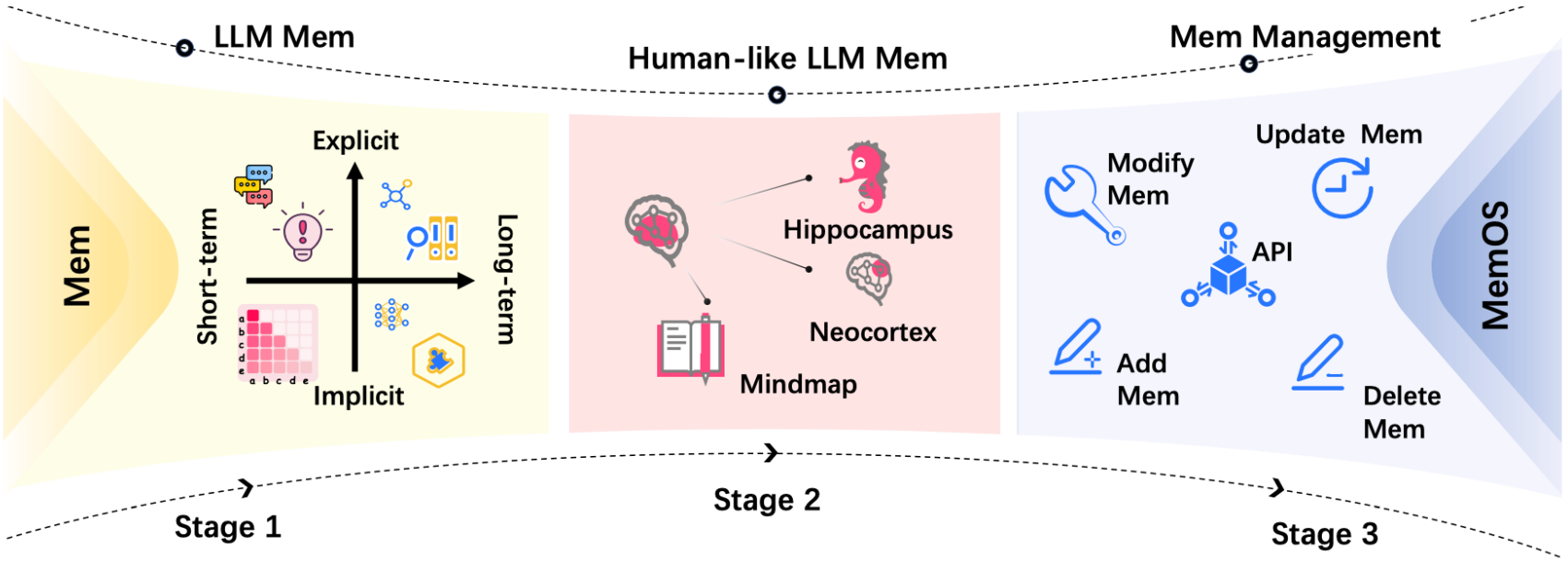


Figure 1 Memory (Mem) in LLMs.

Memory consolidation along abstraction spaces

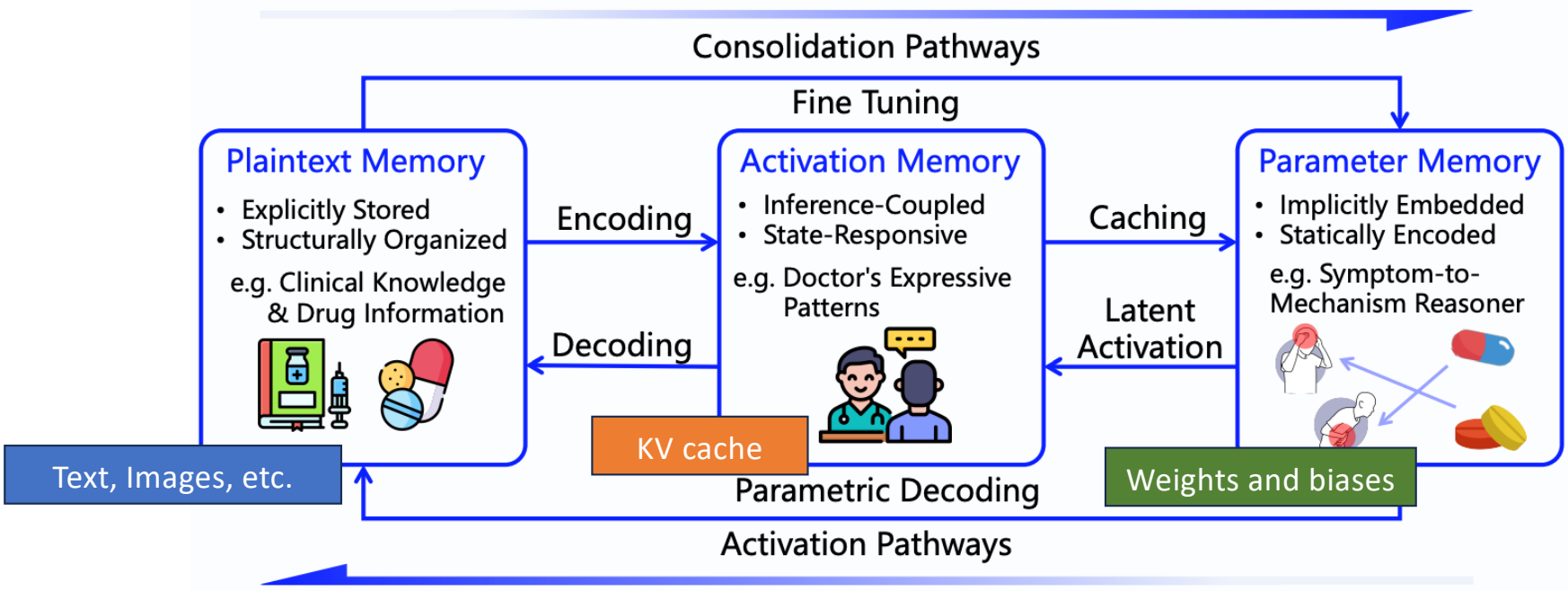
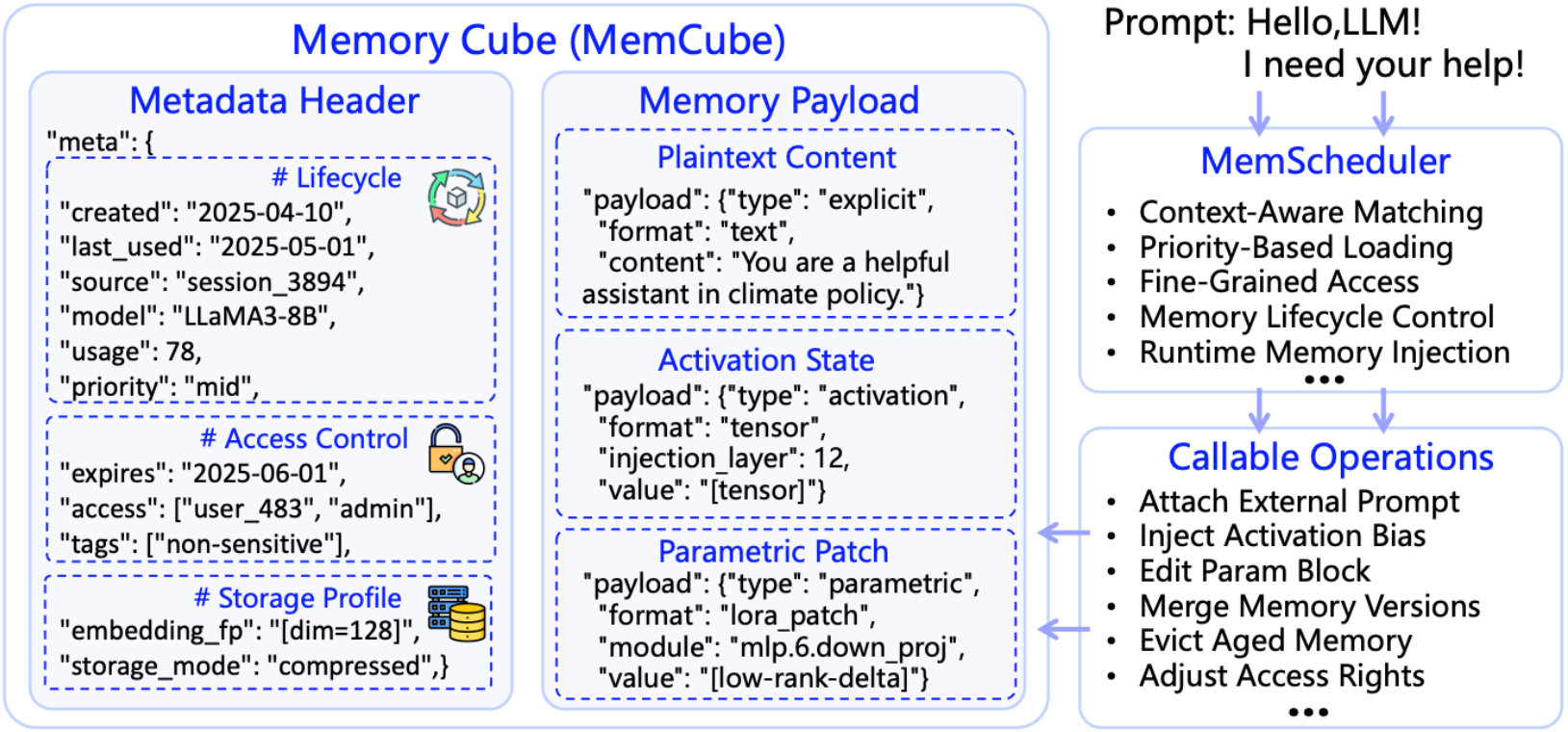


Figure 3 Transformation paths among three types of memory, forming a unified, controllable, and evolvable memory space.

Conversion between abstraction spaces

- **Plaintext ⇒ Activation**: Frequently accessed plaintext memory is converted into activation templates to reduce re-decoding costs;
- **Plaintext/Activation ⇒ Parametric**: Stable, reusable knowledge is distilled into parametric structures to boost inference efficiency;
- **Parametric ⇒ Plaintext**: Rarely used or outdated parameters are externalized into editable plaintext for greater flexibility.

MemCube – Unified Abstraction for Heterogeneous Memory



MemCube Contents

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.

MemCube Contents

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.

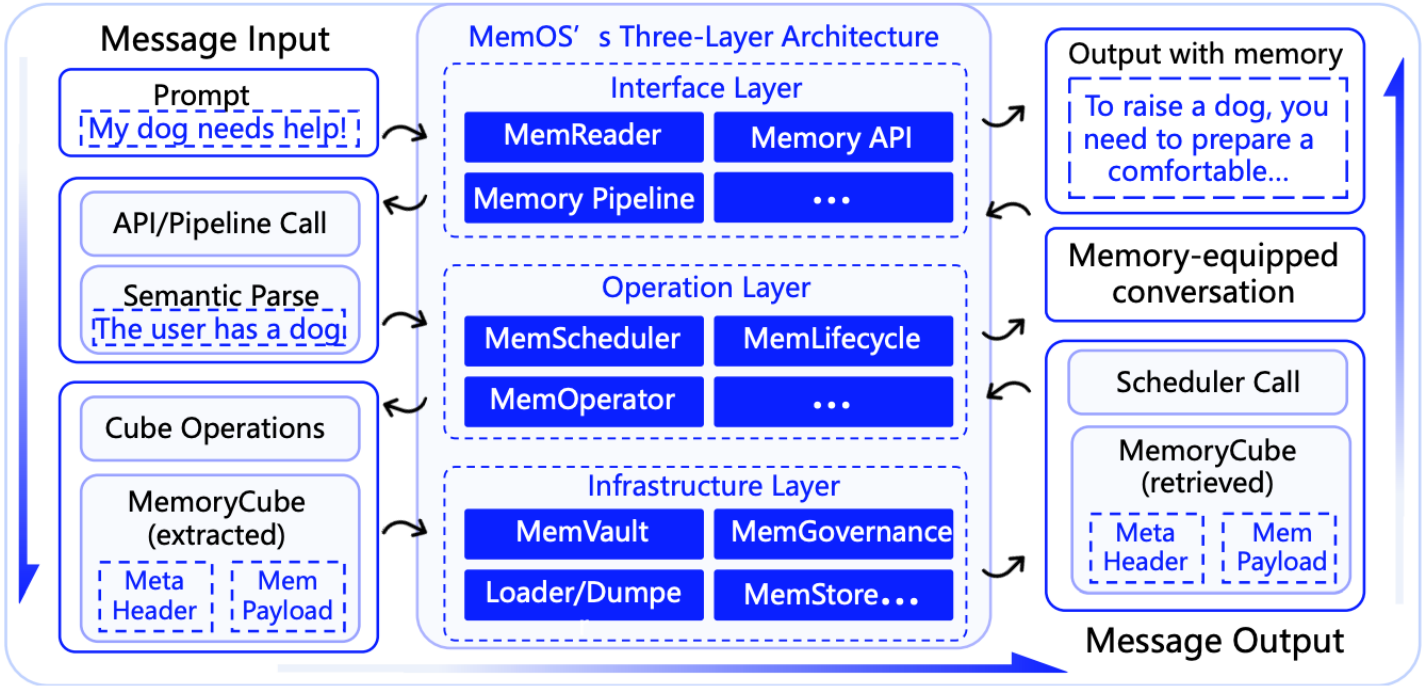
Various Processing Components across Memory Layers

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

3-layer architecture and memory I/O path for MemOS

- Different Memory Layers have different operations available for memory manipulation



Memory Lifecycle

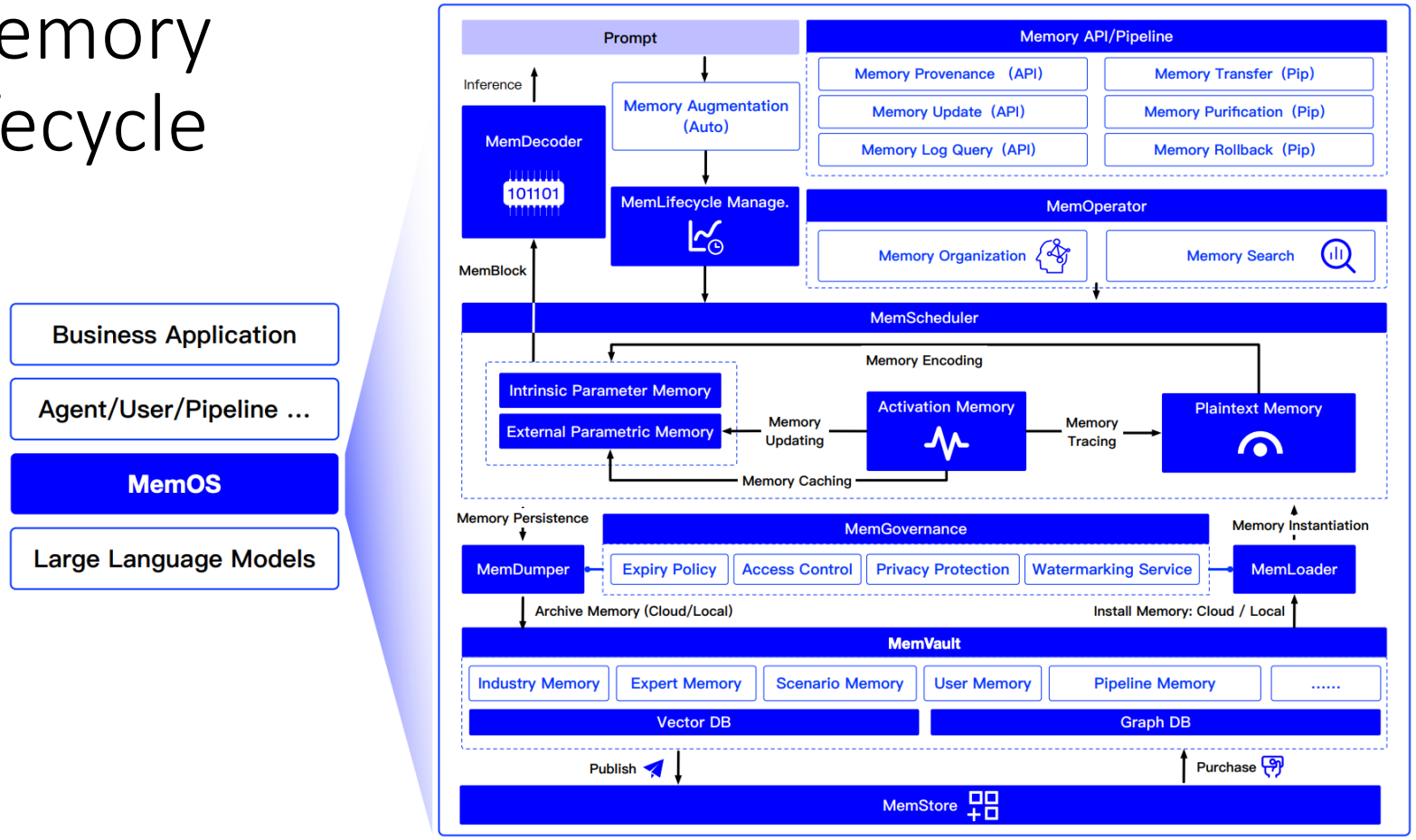


Figure 5 Overview of the MemOS architecture: showing the end-to-end memory lifecycle from user input to API parsing, scheduling, activation, governance, and evolution—unified via MemCube.

Future Plans

- **Cross-LLM Memory Sharing**: Enable interoperability and module reuse across different foundation models by sharing parametric and activation memories. Standard formats and protocols will be needed.

- **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.

My thoughts: Curse of Memory

- Using prior memory can cause agents to keep selecting previous actions, which may not be ideal if the environment has changed
- There should be a way to break out of a negative cycle via future simulation or expert knowledge from others
- Blindly relying on memory alone to make decisions may not be ideal

Question to Ponder

- Can we do the same as what MemOS is proposing using memory tools given to an agent to selectively forget, compress, and store important memory?
- Could a unifying OS framework for memory be too bloated for most basic use cases? Would a task-specific memory be better and more efficient?
- How can we implement MemOS with any LLM, including closed-sourced LLMs?
- How can memory be shared across agents/models with different pre-training parameters?