２０２５年度　修士論文

A Multi-Layered Agentic Memory Architecture: Integrating Dynamic Memory Evolution via Prompt Engineering

指導教員　岩井原　瑞穂　教授

早稲田大学大学院情報生産システム研究科

情報生産システム工学専攻　データ工学　研究

44241009　　YING JIAXUAN

Abstract

As Large Language Models (LLMs) are increasingly applied in the field of Autonomous Agents, empowering agents with long-term, coherent, and efficient memory capabilities has become a key challenge in realizing Artificial General Intelligence (AGI). While Retrieval-Augmented Generation (RAG) techniques mitigate context window limitations by introducing external vector databases, most existing memory systems adopt a static "append-only" storage strategy. These systems lack structured memory organization and dynamic maintenance, leading to severe issues such as memory homogenization, retrieval noise interference, and unbounded storage expansion as interaction cycles prolong.

To address these limitations, this thesis proposes a novel memory framework named **MLA-ARC (Multi-Layered Agentic Architecture)**. Inspired by human memory models in cognitive psychology, this architecture constructs a three-tier mechanism comprising **User Profile**, **Short-Term Memory**, and **Long-Term Memory**. Unlike traditional static RAG, MLA-ARC deeply integrates a dynamic evolution strategy based on **Prompt Engineering**: the system utilizes carefully designed **Prompt Chains** to automatically extract core user attributes from interactions and perform real-time correction and fusion of nodes within the Long-Term Memory.

Furthermore, this thesis innovatively proposes a **Frequency-based Eviction and Consolidation Mechanism**. When system storage reaches a preset threshold, MLA-ARC automatically identifies low-activity memory fragments and consolidates them into high-density archived memories through an LLM-driven summarization algorithm, thereby achieving active forgetting and information compression. We evaluated MLA-ARC on simulated long-term conversational datasets. Experimental results demonstrate that the system significantly improves the retrieval accuracy (Recall) of key information and Agent persona consistency while substantially reducing storage overhead. Through theoretical construction, system implementation, and empirical analysis, this thesis validates the effectiveness and robustness of MLA-ARC in building long-lifecycle agents.

Chapter 1: Introduciton

1.1 Research Background

1.1.1 From Large Language Models to Autonomous Agents

In the past few years, the field of artificial intelligence has witnessed a paradigm shift from Task-Specific Models to general-purpose Large Language Models (LLMs). Foundation models represented

by GPT-4 [11], Claude 3 [28], and LLaMA 3 [29] have demonstrated astonishing language understanding and generation capabilities by pre-training on massive corpora containing trillions of tokens. More importantly, these models have exhibited the emergence of so-called "Reasoning Capabilities" and "In-Context Learning" [10], enabling them to exist not merely as passive text generators but as active decision-making hubs.

This technological leap has catalyzed the concept of Autonomous Agents. Unlike traditional Chatbots that focus solely on current question-answering interactions, Autonomous Agents are defined as intelligent entities capable of perceiving the environment, formulating long-term plans, invoking external tools, and self-correcting based on feedback [9]. From Devin in the software development domain to the "Generative Agents" simulation at Stanford [2], Agents are redefining the boundaries of human-computer interaction.

1.1.2 Memory: The Core Bottleneck of Agents

Although LLMs possess powerful instantaneous reasoning capabilities, building an Agent capable of accompanying users over the long term and handling complex continuous tasks faces a fundamental obstacle: Memory Amnesia.
Standard LLM architectures are inherently Stateless. Each model call is an independent function execution; the model itself does not "remember" the state of the previous call. While current models maintain short-term dialogue history through the Context Window, this mechanism has significant physical and algorithmic bottlenecks:

1. **Limited Context Capacity and High Cost**: Although context windows have expanded from early 4k to 128k or even 1M tokens, indiscriminately stuffing all historical interactions into the Context is engineered to be unsustainable. The time and space complexity of the Transformer architecture's Self-Attention mechanism grows quadratically with sequence length ($O(N^2)$). This implies that as the interaction cycle prolongs, inference costs will rise exponentially, and response Latency will reach levels unacceptable to users.

2. **"Lost in the Middle" Phenomenon**: Recent research [30] indicates that when the context is excessively long, the LLM's ability to extract key information located in the middle of the sequence declines significantly. This dispersion of attention causes Agents to easily overlook early user instructions or key facts when dealing with long-term tasks.

3. **Lack of Persistence**: Information in the context window exists only within the current Session. Once the session ends or the system restarts, all non-parameterized memory is instantly lost.

Therefore, developing an External Memory System that is independent of model weights and capable of persistent storage and efficient retrieval across sessions is a necessary condition for achieving Artificial General Intelligence (AGI).

1.2 Problem Statement

To address the aforementioned bottlenecks, academia and industry have widely adopted the Retrieval-Augmented Generation (RAG) paradigm [6]. By encoding historical interactions into vectors (Embeddings) and storing them in vector databases (e.g., Pinecone, ChromaDB), systems can retrieve historical information based on relevance.

However, directly applying standard RAG architectures to the memory of long-lifecycle Agents presents three core unresolved issues, which are the pain points this thesis aims to address:

1.2.1 Lack of Cognitive Hierarchy

Human memory systems are not flat databases. According to the classic cognitive model by Atkinson and Shiffrin [7], human memory is strictly divided into Sensory Memory, Short-Term Memory (STM), and Long-Term Memory (LTM). Different levels of memory have distinct storage durations, capacity limits, and access mechanisms. Conversely, existing Agent memory systems typically employ a flattened indexing structure. Core user attributes (e.g., "I am allergic to seafood," belonging to a stable long-term profile) are mixed with trivial dialogue flows (e.g., "This step needs to be retried," belonging to short-term context) in the same vector space. This structural homogenization makes it difficult for retrieval algorithms to distinguish the importance of information, often resulting in the retrieval of a large amount of semantically relevant but insignificant "Noise," which crowds out the context space for truly critical information.

1.2.2 Static Nature vs. Dynamic Evolution

Information in the real world is dynamic. User preferences change over time (e.g., shifting from "likes fitness" to "recently injured and needs rest"), and new facts supersede old ones. However, the vast majority of existing RAG systems adopt an "Append-only" storage strategy. Once memory is written to the database, it becomes an immutable record. Over time, the database accumulates a vast amount of contradictory or obsolete information. When an Agent performs retrieval, due to the lack of a Memory Evolution mechanism, it may simultaneously retrieve conflicting old and new information, leading to severe logical confusion (Hallucination).

1.2.3 Storage Bloat & Absence of Forgetting

In computer science system design, any cache or storage system requires a Garbage Collection (GC) mechanism. However, in the field of Agent memory, this aspect has long been overlooked. As the

Agent's runtime increases, unrestricted writing leads to "Memory Bloat." This not only incurs huge storage and index maintenance costs but, more seriously, reduces the Signal-to-Noise Ratio (SNR) of retrieval. The human brain utilizes Active Forgetting and Memory Consolidation mechanisms to transform short-term details into long-term concepts and discard useless details. Currently, there is a lack of an effective algorithm that can automatically "prune" and "consolidate" the Agent's memory repository based on the Frequency and Recency of information access.

1.3 Proposed Solution

In response to the challenges above, this thesis proposes a novel memory framework named **MLA-ARC (Multi-Layered Agentic Architecture)**. Based on memory evolution theory, this framework constructs a memory system with a complete "Perceive-Store-Evolve-Forget" lifecycle by introducing a deep layered architecture and dynamic control logic based on Prompt Engineering.

1.3.1 Prompt-Engineered Dynamic Evolution

The core perspective of this thesis is that memory maintenance should not rely solely on database operations but should be driven by the cognitive capabilities of the LLM itself. We designed a complex set of **Prompt Chains** acting as the system's "Cognitive Controller":

- **Extraction and Correction**: Whenever a new interaction occurs, the system triggers specific Prompts to analyze whether the new information conflicts with old memories in the LTM. If a conflict exists, it directly updates the old node rather than adding a redundant one.

- **Context Compression**: During the transfer of information from STM to LTM, Prompts are used to compress lengthy multi-turn dialogues into concise **Fact-based Statements**.

1.3.2 Multi-Layered Architecture

MLA-ARC discards flat vector indexing and constructs three memory layers with distinct functions:

1. **User Profile (UP)**: Serving as the system's "Metadata," it stores explicit, global attributes of the user (e.g., name, occupation, core constraints). This information is automatically extracted from dialogues by the system using specific Prompts and is always present as part of the System Prompt, ensuring the Agent's persona consistency.
2. **Short-Term Memory (STM)**: Simulating human working memory. It adopts a memory-based **Cache Queue** structure, retaining only the complete dialogue interactions of the most recent K turns, used to maintain the coherence of the current dialogue flow.
3. **Long-Term Memory (LTM)**: Persistent storage based on ChromaDB. Unlike standard RAG, each memory node here is an independent knowledge point that has been "summarized and reconstructed" by the LLM, rather than a raw dialogue fragment.

1.3.3 Frequency-based Eviction & Consolidation

To address the storage bloat issue, MLA-ARC introduces a memory lifecycle management algorithm. The system maintains a "Temperature" value for each memory node, determined jointly by **Access Frequency** and **Time Decay**. When storage space reaches a preset threshold (e.g., 8GB), the system triggers the **Eviction and Consolidation** mechanism:

1. **Identification**: Select the Top-N memory nodes with the lowest temperature.
2. **Consolidation**: Utilize a specialized **Summary Prompt** to attempt to fuse these fragmented cold memories into a more generalized "**Archived Node**."
3. **Eviction**: Physically delete the original fragments, retaining only the archived node.

1.4 Thesis Contributions

The main contributions of this thesis can be summarized in the following four aspects:

1. **Theoretical Framework Contribution**: This thesis proposes a general, biomimetic Agent memory architecture, MLA-ARC, providing a practical perspective for solving the stateless problem of LLMs. This system equips LLM agents with long-term interaction capabilities without requiring predetermined memory operations.
2. **Methodological Innovation**: A memory evolution method based on **Prompt Engineering** is proposed. Through carefully designed Prompt logic, the transformation from unstructured dialogue to structured memory and the automatic resolution of conflicts between memories are realized. This is an agentic memory system for LLM agents that supports the autonomous generation of contextual descriptions, dynamic establishment of memory connections, and intelligent evolution of existing memories based on new experiences.
3. **Agentic Memory Update Mechanism**: New memories automatically trigger two key operations: link generation and memory evolution. Link generation automatically establishes connections between memories by identifying shared attributes and similar contextual descriptions. Memory evolution enables existing memories to dynamically adapt as new experiences are analyzed, leading to the emergence of higher-order patterns and attributes.
4. **Eviction Mechanism Design**: Designed and formally defined a **Frequency-based Eviction and Consolidation Algorithm**. This is one of the earlier works in the literature attempting to implement "controlled forgetting" and "lossy compression" in Agent memory systems, solving the storage explosion problem under long lifecycles to a certain extent.
5. **System Implementation and Evaluation**: A prototype of MLA-ARC was implemented based on Python, LangChain, and ChromaDB. The system was comprehensively evaluated using long-term conversational datasets, comparing the performance of multiple foundation models across

several distinct evaluation metrics, demonstrating significant improvements. Additionally, T-SNE visualization charts are provided to illustrate the structured organization of the agentic memory system.

1.5 Thesis Organization

The subsequent chapters of this thesis are organized as follows:

- **Chapter 2 (Related Work)**: Reviews LLM-based agents, cognitive science memory models, and existing neuro-symbolic memory systems.
- **Chapter 3 (Methodology)**: Elaborates on the mathematical definitions, layered architecture design, and the core eviction and consolidation algorithm of MLA-ARC.
- **Chapter 4 (Implementation)**: Introduces the engineering implementation of the system, including data structure design, prompt strategies, and key code logic.
- **Chapter 5 (Experiments)**: Presents experimental setups, dataset construction, and the analysis of quantitative and qualitative experimental results.
- **Chapter 6 (Discussion & Limitations & Conclusion)**: Discusses the limitations of the system and directions for future improvement. Summarizes the entire thesis.
- **References**: Shows the references and the open-source projects used as references.
- **Appendix**: Provides more detailed experimental data and a complete reference prompt.

Chapter 2: Related Work

This chapter aims to systematically review the academic work and technological evolution closely related to this research. We organize the technical lineage into four progressive levels: first, we explore the cognitive science memory theories that served as the inspiration for our architectural design; second, we review neuro-symbolic memory systems in the pre-LLM era; subsequently, we elaborate on Retrieval-Augmented Generation (RAG), which serves as the current mainstream technical foundation; finally, we provide an in-depth analysis of the latest advancements in specialized memory architectures for LLM Agents. Through this review, we will identify the limitations of existing technical solutions in terms of long-term interaction, memory evolution, and storage efficiency, thereby establishing the research motivation for the MLA-ARC architecture proposed in this thesis.

2.1 Theoretical Foundations in Cognitive Science

Understanding how the human brain processes, stores, and forgets information serves as the theoretical cornerstone for building AI memory systems with "human-like" characteristics. The layered

design and eviction mechanisms in the MLA-ARC architecture proposed in this thesis directly correspond to classic models in cognitive psychology.

2.1.1 The Multi-Store Memory Model

In 1968, Atkinson and Shiffrin proposed the foundational Multi-Store Model [7]. This model posits that human memory is not a single unitary entity but is composed of three structured components:

1. **Sensory Memory**: Responsible for retaining the raw state of sensory inputs for an extremely short duration (milliseconds).

2. **Short-Term Memory (STM)**: Also known as Working Memory. It has extremely limited capacity; Miller's research suggests its capacity is approximately 7 ± 2 information chunks [19]. STM is responsible for current cognitive processing, and information will rapidly fade if not rehearsed.

3. **Long-Term Memory (LTM)**: Possesses nearly infinite capacity and persistence, used for storing factual knowledge (semantic memory) and experiential events (episodic memory).

This theory directly supports the design decision in this thesis to divide memory into STM (cache-based) and LTM (vector database-based): STM ensures the immediate fluidity of dialogue, while LTM provides deep background support.

2.1.2 Memory Consolidation & Adaptive Forgetting

Memory is not a static record. Neuroscience research indicates that memory undergoes a **Consolidation** process, where unstable short-term memory traces are gradually transformed into stable long-term memories stored in the Neocortex via the Hippocampus [20]. During this process, the brain reorganizes, compresses, and abstracts information.

Simultaneously, **Forgetting** is not a system failure but a highly efficient adaptive mechanism. Hermann Ebbinghaus's Forgetting Curve [8] revealed the law that memory retention decays exponentially over time. Modern research has further proposed the concept of **Active Forgetting** [21], suggesting that the brain actively clears insignificant memories to optimize the allocation of cognitive resources. The dynamic eviction and consolidation mechanism in MLA-ARC is a computational simulation of this biological process.

2.2 Memory Systems in Pre-LLM Era

Before the advent of Large Language Models, the field of Natural Language Processing (NLP) had long explored memory mechanisms.

2.2.1 RNNs and Implicit Memory

Early sequence models such as RNNs and LSTMs primarily relied on **Hidden States** to pass historical information [22]. This approach can be viewed as a form of "Implicit Memory." However, due to the vanishing gradient problem and the limitations of fixed-dimensional state vectors, these models struggled to capture long-range dependencies and could not store external knowledge beyond their training corpora.

2.2.2 Memory Networks & Neural Turing Machines

To overcome the bottlenecks of implicit memory, researchers began attempting to introduce readable and writable external storage.

- **Memory Networks (MemNN)** [23]: Proposed a framework combining external memory components with neural networks, assisting reasoning by reading relevant memories through an addressing mechanism.

- **Neural Turing Machines (NTM)** [24]: Inspired by Turing machines, designed neural networks equipped with Read/Write Heads capable of performing precise read and write operations on an external memory matrix.

Although these models achieved success in simple question-answering tasks, they were typically based on symbol matching or simple attention mechanisms, lacking deep semantic understanding capabilities for complex natural language.

2.3 Retrieval-Augmented Generation

With the proliferation of the Transformer architecture, Retrieval-Augmented Generation (RAG) [6] has become the mainstream paradigm for addressing the issues of lagged knowledge updates and Hallucination in LLMs. RAG combines parametric memory (model weights) with non-parametric memory (external indices) and serves as the technological foundation for current agent memory systems.

2.3.1 RAG Architecture

A standard RAG system typically consists of two core components:

1. **Retriever**: Usually based on Dense Passage Retrieval (DPR) [25]. It encodes the Query and Document into high-dimensional vectors (Embeddings) and utilizes Maximum Inner Product Search (MIPS) to rapidly locate relevant documents.

2. **Generator**: The LLM itself, which receives the retrieved documents as context to generate the final response.

2.3.2 Vector Databases

The popularity of RAG has driven the development of vector databases (e.g., FAISS [26], Chroma, Pinecone). These databases support efficient Approximate Nearest Neighbor (ANN) search, making millisecond-level retrieval possible at the scale of billions of data points.

2.3.3 Limitations for Agents

While RAG performs excellently in knowledge-intensive tasks (such as open-domain QA), applying it directly to Agent memory has limitations:

- **Static Nature**: Traditional RAG assumes the knowledge base is static (e.g., Wikipedia dumps), whereas an Agent's memory is a dynamically accumulating history of interactions.

- **Retrieval Granularity**: RAG typically operates on fixed-length text blocks (Chunks), making it difficult to handle global information like user profiles that require cross-document synthesis.

2.4 Memory for LLM Agents

With the rise of Autonomous Agents, researchers have begun designing memory architectures specifically tailored to agent characteristics. This body of work is most relevant to this thesis.

2.4.1 Generative Agents & Memory Streams

The work on **Generative Agents** by Park et al. [2] is a milestone in this field. They designed a **Memory Stream** structure that records all of the Agent's Observations. Its retrieval mechanism is based on three dimensions:

- **Recency**: Events that occurred more recently have higher weight.

- **Importance**: Events are scored by the LLM.
- **Relevance**: Vector similarity.

Furthermore, this work proposed a "Reflection" mechanism, which periodically summarizes the memory stream to generate higher-level inferences. This is similar in concept to the "Memory Consolidation" in this thesis, but we further introduce a physical deletion (eviction) mechanism to address the problem of unbounded storage growth.

2.4.2 System-Level Memory Management (MemGPT)

MemGPT [3] introduces the concept of Virtual Memory from operating systems to Agents. It treats the LLM's context window as restricted Main Context (analogous to RAM) and external storage as External Context (analogous to Disk). MemGPT allows the LLM to autonomously manage the swapping (Paging) of information by defining specific system calls (e.g., archival_memory_insert, archival_memory_search). MemGPT focuses on solving the physical bottleneck of the context window, whereas MLA-ARC focuses more on the **Semantic Evolution** of memory content and the structured construction of user profiles.

2.4.3 Evolving Memory Mechanisms

A-Mem [1] focuses on the dynamic correction of memory. Unlike the traditional Append-only mode, A-Mem allows the Agent to modify old vector nodes in a Prompt-driven manner when new information conflicts with old memory. Similarly, **Voyager** [27] constructed a Skill Library in Minecraft agents, updating stored code snippets through continuous trial-and-error feedback.

2.4.4 Summary of Gaps

Despite the contributions of the aforementioned works, the following issues remain insufficiently addressed:

1. **Lack of a Unified Layered View**: Most systems do not explicitly distinguish between "User Profile" and "Dialogue History," causing long-term preferences to be easily drowned out by short-term noise.

2. **Absence of Eviction Mechanisms**: Current systems are mostly append-only (except for MemGPT's paging, which is not true deletion). As runtime increases, the database inevitably encounters performance bottlenecks and a decline in signal-to-noise ratio.

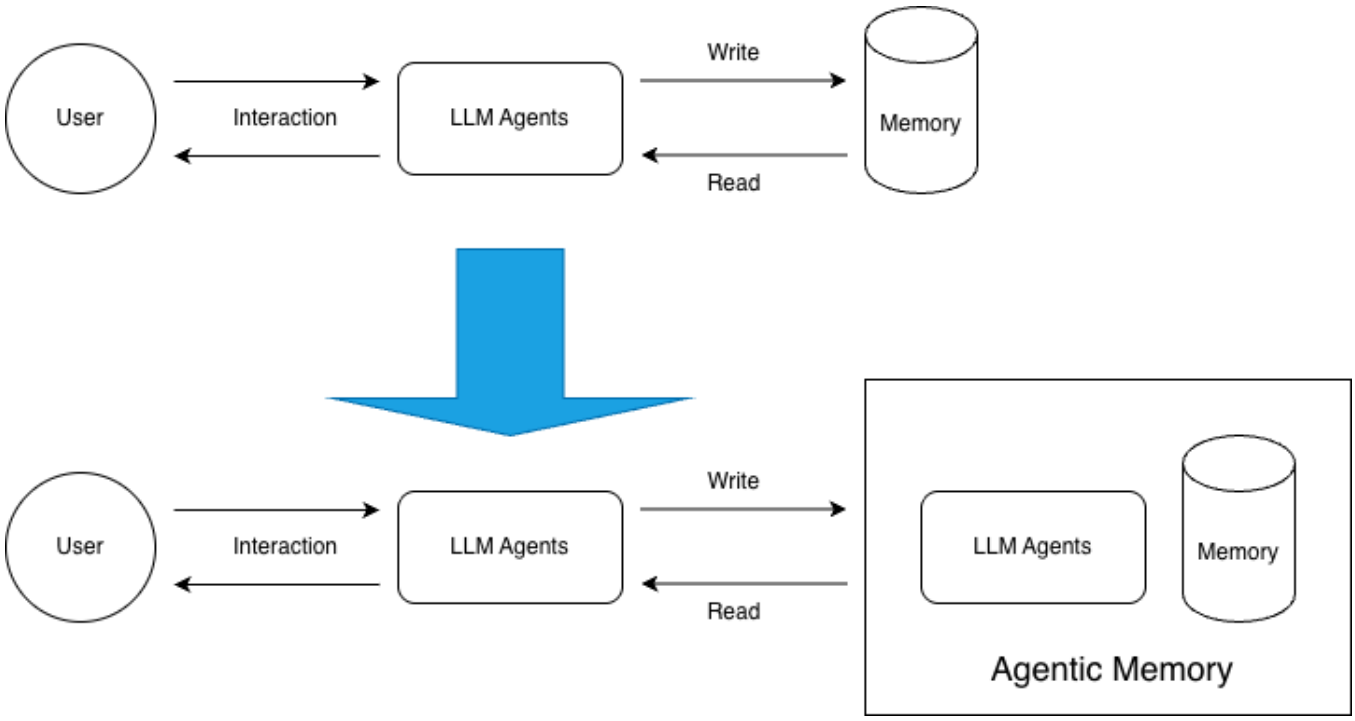


Figure 1: Traditional Memory System and My Purpose Memory System

The **MLA-ARC** proposed in this thesis aims to address some of these problems by combining the layered theory of cognitive science, RAG retrieval technology, and the evolutionary capabilities of Prompt Engineering to build a more efficient, biomimetic, and sustainable agent memory system.

Chapter 3: Methodology

This chapter elaborates on the design principles and implementation methodology of **MLA-ARC (Multi-Layered Agentic Architecture)**. We first mathematically formalize the problem of agent interaction and memory management; subsequently, we introduce the system's three-layer architecture from a top-down perspective; finally, we delve into the construction logic, data structures, and state transition mechanisms of each layer.

3.1 Problem Formulation

Before detailing the architecture, we first define the core symbols and mathematical concepts involved in this thesis.

3.1.1 Interaction and Context

We define a series of discrete interactions between an Autonomous Agent A and a user. At any given time step, an interaction I_t can be represented as a tuple:

$$I_t = (q_t, r_t, \tau_t)$$

Where:

- q_t represents the query or instruction input by the user.
- r_t represents the response generated by the agent.
- τ_t represents the timestamp of the interaction.

We denote the complete interaction history up to time t as $\mathcal{H}_t = \{I_1, I_2, \ldots, I_t\}$. In traditional LLM applications, constrained by the **Context Window** length L_{max}, the model can only receive a truncated historical fragment $C_t \subset \mathcal{H}_t$, where $|C_t| \leq L_{max}$. The objective of this thesis is to construct an external memory system $\mathcal{M}$ that enables the model to utilize global information within $\mathcal{M}$ to generate the optimal response r_t without violating the L_{max} constraint.

3.1.2 Memory Node Definition

In MLA-ARC, memory is not stored as raw text fragments but is structured as **Memory Nodes**. When a memory or dialogue first enters the system, we invoke Prompt P_{s1} to construct a structured data object.

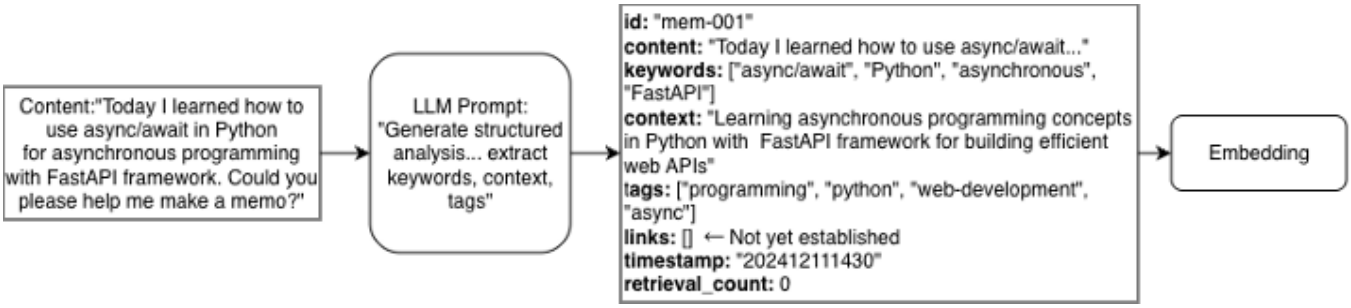


Figure 2: The processing of translating raw input into structured data.

We define the memory pool as a set of nodes $\mathcal{M} = \{m_1, m_2, \ldots, m_N\}$. Each memory node m_i is defined as a multidimensional tuple:

$$m_i = (c_i, v_i, attr_i, meta_i)$$

Where:

- c_i is the textual **Content** of the memory, typically a factual statement that has been summarized or rewritten by the LLM.
- $v_i \in \mathbb{R}^d$ is the **Embedding Vector** of the content, generated by a pre-trained encoder model $\phi(\cdot)$, such that $v_i = \phi(c_i)$.
- $attr_i$ is the set of **Attributes** for the memory, used to enhance semantic matching during retrieval (detailed in Section 3.5).
- $meta_i$ is the Metadata for system management, containing timestamps and access counters to support the eviction algorithm:

$$meta_i = \{t_{created}, t_{last\_access}, n_{access}\}$$

3.2 System Architecture Overview

To simulate human cognitive processing, MLA-ARC discards flattened storage structures in favor of a three-layered memory architecture. This architecture consists of the following three subsystems:

1. **User Profile (UP)**: Located at the top layer, this stores global, static, or semi-static attributes about the user (e.g., name, occupation, personality preferences). This information acts as "metadata" for the agent's cognition and is injected into the **System Prompt** during every generation to establish the agent's service tone.

2. **Short-Term Memory (STM)**: Located at the intermediate layer, this simulates **Working Memory**. It employs a memory-based **Cache Queue** to store the complete raw dialogue interactions of the most recent K turns. Its primary responsibility is to maintain the coherence of the current dialogue flow and serve as a "buffer" for long-term memory.

3. **Long-Term Memory (LTM)**: Located at the bottom layer and constructed upon a vector database (ChromaDB), this stores historical experiences that have been filtered, compressed, and evolved. This layer supports complex semantic retrieval and incorporates the **Memory Evolution** and **Eviction** mechanisms proposed in this thesis.

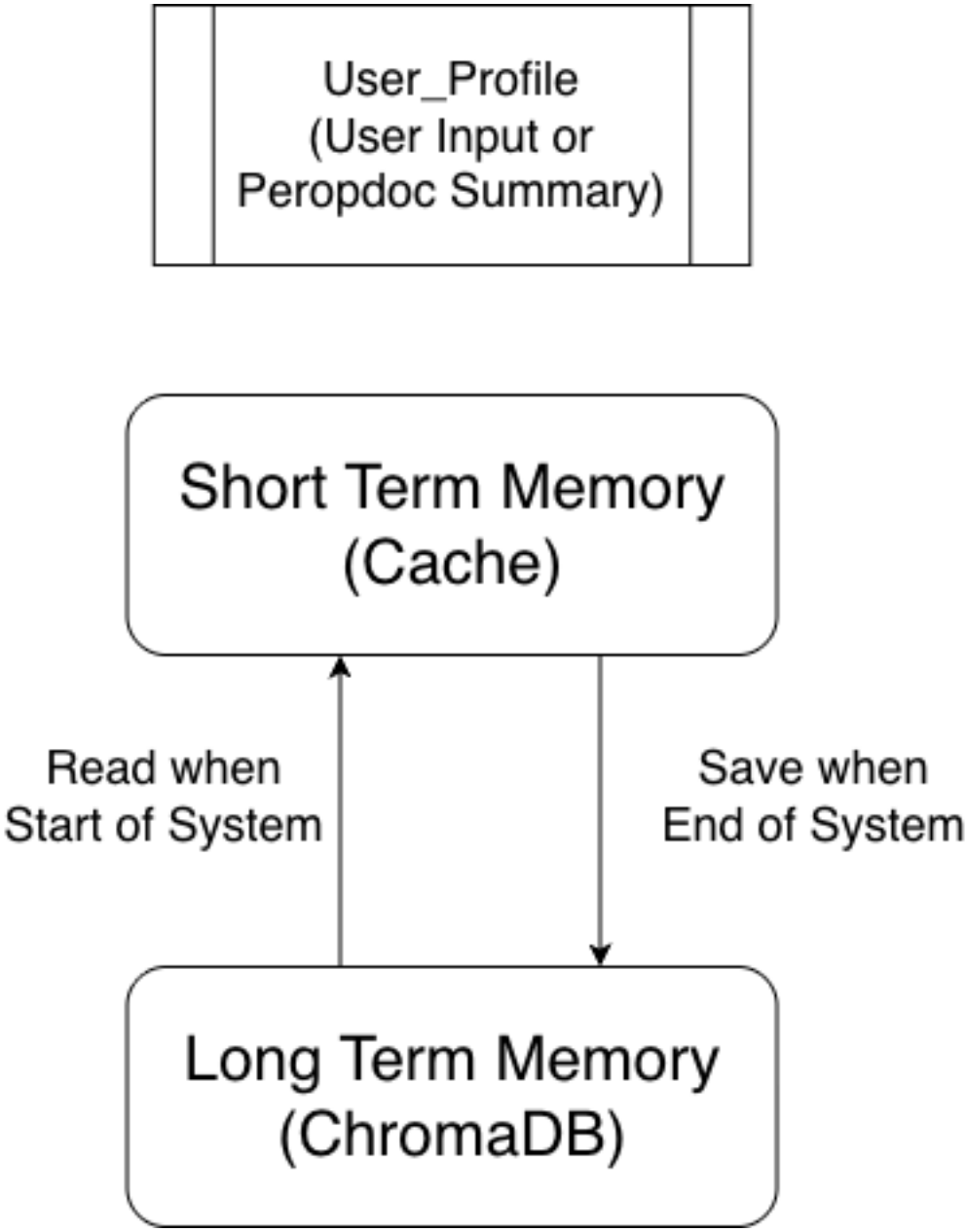


Figure 3: Overview of 3-layered Memory Architechture

The overall data flow of the system is as follows: A new interaction I first enters the Short-Term Memory S; when S reaches capacity or the session ends, older interactions are extracted, transformed into Long-Term Memory nodes m, and stored in L. Simultaneously, the system periodically runs a **Profile Extractor** to identify user characteristics from the context C and update the User Profile P.

3.3 Layer 1: User Profile Construction

The User Profile P serves as the "metadata" for the Agent's cognition. Unlike fragmented dialogue history, the profile layer aims to maintain a set of highly generalized, structured facts about the user.

3.3.1 Data Structure

We define P as a collection of Key-Value Pairs or natural language statements:

$\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$

Where each p_i represents an independent user attribute, such as {"Name": "Alice"} or {"Diet": "Vegetarian"}.

3.3.2 Prompt-based Profile Extraction

To achieve automated updates, we designed an LLM-based Profile Extraction Prompt. Formally, we define a periodic extraction function $f_{profile}$:

$\Delta\mathcal{P}_t = f_{profile}(I_t, \mathcal{P}_{t-1})$

This function accepts the current interaction and the existing profile $\mathcal{P}_{t-1}$ as input, outputting an incremental update $\Delta\mathcal{P}_t$.

The specific logic is designed as follows:

1. **Input Analysis**: The LLM analyzes the user's query to determine if it contains explicit **Self-Disclosure**.
2. **Redundancy Check**: The LLM compares new information against the logical structure of the existing profile $\mathcal{P}_{t-1}$; if the information is already known, it is ignored.
3. **Conflict Resolution**: If new information conflicts with old information (e.g., the user previously stated "I like cats" but now says "I am allergic to cats"), an update instruction is generated to overwrite the old entry.

Prompt Example:

```
Analyze the user's input and historical query. Does it contain explicit
personal information (name, job, preferences)?
Current Profile: {Current_Profile_JSON}
User Input: {User_Query}
Task: Return a JSON delta. If no new info, return empty. If conflict,
overwrite." Please be careful, the content "up_i" in the format need to be
changed as actual profile, like "up_i" -> "name".

### OUTPUT FORMAT ###
{
    "up_1":"",
    "up_2":"",
    "up_3":"",
```

```
    ...
}
```

This mechanism ensures the Agent maintains the most up-to-date cognition of the user, minimizing interference from historical data noise.

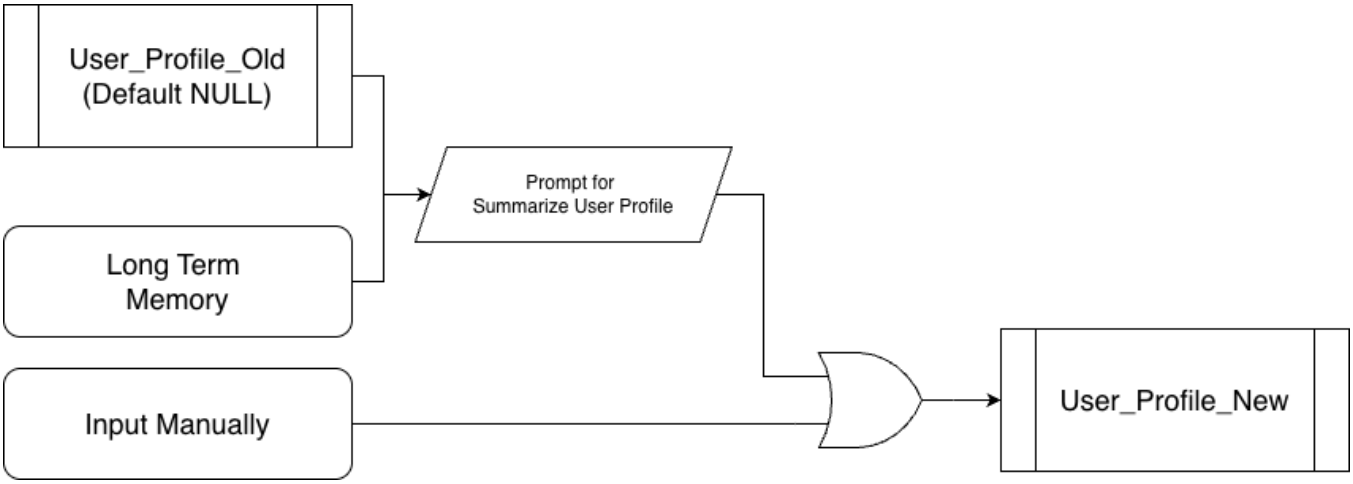


Figure 4: Overview of User Profile updating mechanism

3.4 Layer 2: Short-Term Memory & Caching

Short-Term Memory S addresses the "**Recency Effect**." In human conversation, pronouns (e.g., "it," "that person") typically refer to entities mentioned in the most recent sentences. Therefore, retaining the complete original text of recent interactions is crucial.

3.4.1 Cache Queue Model

We implement S using a Double-Ended Queue (Deque) with a fixed capacity K:

$Q_{STM} = [I_{t-K+1}, \ldots, I_{t-1}, I_t]$

Where K is a hyperparameter (e.g., $K = 10$ dialogue turns).

3.4.2 State Transition Logic

The operation of S follows a FIFO (First-In, First-Out) principle, with a specific trigger mechanism for transferring data to Long-Term Memory:

1. **Push**: The new interaction I_t is appended to the tail of Q_{STM}.
2. **Overflow**: When the queue length exceeds K, the earliest interaction I_{t-K+1} is removed.
3. **Memorization Trigger**: The removed interaction I_{t-K+1} is not discarded; instead, it is sent to the **LTM Processing Pipeline**.

This design reflects cognitive coherence: only when an experience leaves "working memory" and is no

longer under immediate attention does the brain attempt to "encode" it into long-term memory.

3.5 Layer 3: Long-Term Memory Construction

The Long-Term Memory layer (LTM, $\mathcal{L}$) serves as the core knowledge base of the MLA-ARC system. Unlike the **Raw Dialogue Stream** stored in the Short-Term Memory layer, LTM aims to store knowledge and experiences that have been **Processed**, **Indexed**, and **De-noised**.

To support efficient semantic retrieval and simulate the context-dependency of human memory, we employ an attribute-based memory structure rather than simply storing text embeddings.

3.5.1 Memory Attribute Extraction

When an interaction fragment I overflows from the Q_{STM} queue and prepares to enter LTM, it first passes through an Attribute Extractor. This module utilizes the LLM's language understanding capabilities to decompose the raw interaction into a set of structured attributes.

Formally, for an input interaction I, the extraction function f_{attr} generates the attribute set $attr$ for the memory node m:

$$m.attr = f_{attr}(I) = \{c_{xt}, c_{nt}, k_w, t_s\}$$

The definitions for each attribute are as follows:

1. **Context (c_{xt})**: Describes background information, including "Who," "Where," and "Cause and Effect." This is critical for disambiguation. For example, the isolated sentence "It is red" is meaningless, but with the context "User is discussing his Ferrari model," the information becomes complete.

2. **Content (c_{nt})**: A summary of the substantive information contained in the interaction. The system removes **Phatic Expressions** and redundant modifiers, retaining only **Fact-based Statements**.

3. **Keywords (k_w)**: Entity nouns or technical terms extracted from c_{nt} (e.g., "Python", "Gundam", "Visa Application"). These keywords assist with **Inverted Indexing** or **Hybrid Search**.

4. **Timestamp (t_s)**: The absolute time when the interaction occurred.

Implementation Logic: We designed a specific **Attribute Extraction Prompt**(See more detail in

Appendix) that instructs the LLM to output the above fields in JSON format. If the input I consists only of meaningless chitchat (e.g., "Haha, okay"), the extractor returns NULL, and the fragment is directly discarded without entering LTM, thereby reducing noise at the source.

3.5.2 Vector Embedding and Storage

The system vectorizes the input content and the core content c_{nt}. We use a pre-trained Transformer encoder (specifically all-MiniLM-L6-v2 in our experiments) [14] to map the text into a high-dimensional vector space:

$$v = \phi(m.c_{nt}) \in \mathbb{R}^d$$

Finally, the memory node m is stored in the vector database (ChromaDB), where v is used for similarity retrieval, and $attr$ is stored as metadata for post-retrieval **Filtering** or **Reranking**.

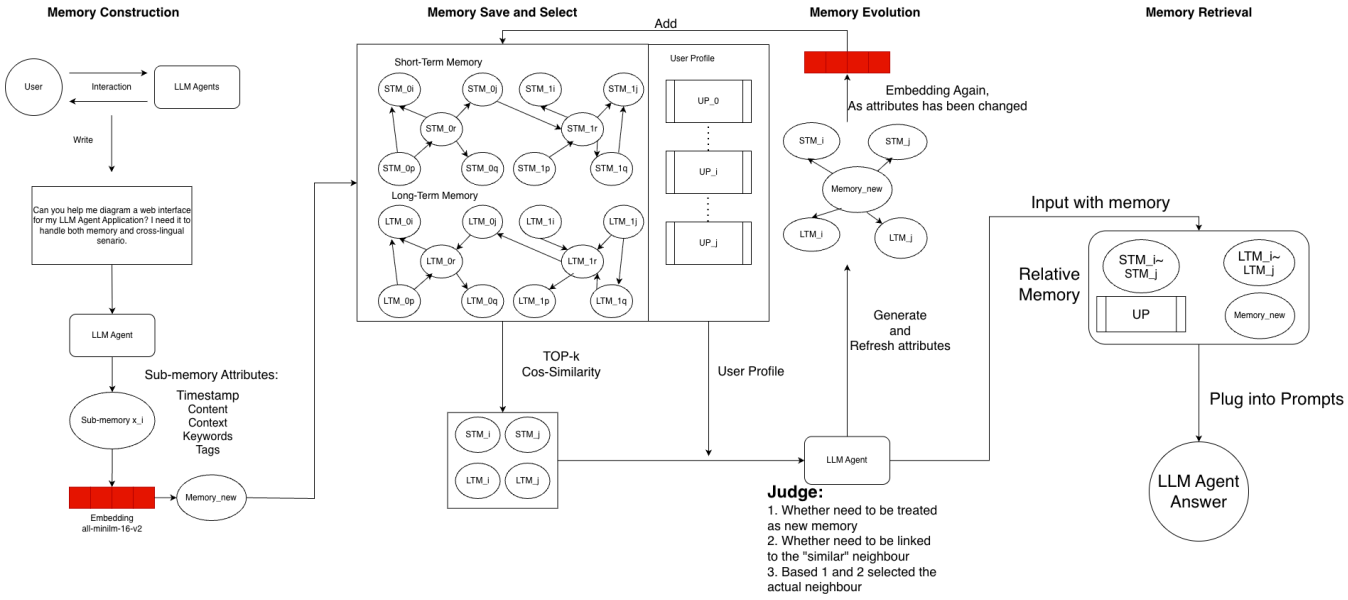


Figure 5: Overview of the memory system from input to output

3.6 Dynamic Memory Evolution

Most existing RAG systems adopt an "Append-only" write strategy, which leads to two significant problems:

1. **Information Redundancy**: Similar information is stored repeatedly (e.g., the user mentions "I like sci-fi movies" multiple times).
2. **Logical Conflict**: New information contradicts old information (e.g., the user status changes from "Single" to "Married"), causing the Agent to hallucinate during retrieval.

To address these issues, MLA-ARC introduces a **Dynamic Memory Evolution** mechanism. The core

philosophy is that writing a new memory is not merely an **Insertion** but a process of **Interaction** and **Reshaping** with old memories.

3.6.1 Formalization of the Evolution Process

When a new memory node m_{new} is prepared to enter the system, the system first executes a Retrieval. It identifies the k existing nodes most semantically similar to m_{new} from both the STM cache and the LTM repository, denoted as sets $\mathcal{M}_{rel_{stm}}$ and $\mathcal{M}_{rel_{ltm}}$. Subsequently, m_{new} along with $\mathcal{M}_{rel_{stm}}$ and $\mathcal{M}_{rel_{ltm}}$ are input into the LLM to execute the evolution function $\mathcal{E}$:

$$\{m'_{new}, \mathcal{M}'_{rel}\} = \mathcal{E}(m_{new}, \mathcal{M}_{rel})$$

The evolution function performs one of three operations based on the logical relationship between m_{new} and $\mathcal{M}_{rel}$:

A. Reinforcement

If the information in m_{new} already exists in $\mathcal{M}_{rel}$ and is consistent:

- **Operation**: Do not create a new node.
- **Update**: Increase the confidence or access count of the corresponding old node $m_i \in \mathcal{M}_{rel}$.
- **Purpose**: To avoid duplicate storage while reinforcing the weight of high-frequency facts.

B. Correction & Update

If the information in m_{new} presents a Fact Conflict with old information in $\mathcal{M}_{rel}$:

- **Operation**: Determine that m_{new} represents the latest state.
- **Update**: Modify the content of the old node m_i to reflect m_{new}, or mark the old node as "**Outdated**."
- **Example**:
  - Old memory m_{old}: "User lives in Tokyo."
  - New input m_{new}: "I moved to Osaka."
  - Evolution result m'_{old}: "User moved to Osaka (previously lived in Tokyo)."

C. Link, Evolution and Merge

If m_{new} and $\mathcal{M}_{rel}$ describe different aspects of the same topic and can be combined:

- **Operation**: Fuse multiple fragmented pieces of information into a more complete node.
- **Update**: Generate a new merged node m_{merged} and delete the original m_{new} and $\mathcal{M}_{rel}$.
- **Example**:
  - m_1: "User likes red."

- m_2: "User likes sports cars."
- Evolution result m_{merged}: "User likes red sports cars."

3.6.2 Prompt-based Evolution Implementation

This complex logical judgment is partly implemented via the Evolution Prompt. Specifically, we utilize the capabilities of LLM to determine whether the information requires enhancement, correction, or merging. The prompt design follows the Chain-of-Thought (CoT) paradigm, guiding the LLM to think in steps:

```
**Evolution Prompt Structure:**
1. **Analyze Relationship**: Compare the New Memory with Retrieved Memories.
Are they Supporting, Contradicting, or Unrelated?
2. **Determine Action**: Choose one of [INSERT, UPDATE, MERGE, IGNORE].
3. **Execute**:
   * If UPDATE: Rewrite the old memory content to reflect the change.
   * If MERGE: Synthesize a single comprehensive statement.
   * If INSERT: Return the new memory as is.
```

Through this mechanism, MLA-ARC's long-term memory repository becomes a self-purifying, self-growing knowledge network rather than a static storage dump. This fundamentally ensures the Agent maintains a real-time and consistent cognition of the user's state.

3.7 Hierarchical Retrieval and Context Construction

When the agent receives a new user query q_t, the MLA-ARC system must extract relevant information from the three memory layers to construct the final Prompt context. To balance **Accuracy** and **Utilization** of the context window, we designed a **Dual-Route Hybrid Retrieval** mechanism.

3.7.1 Retrieval Process

The retrieval process involves both parallel and serial stages: 40

1. Profile Injection: The User Profile $\mathcal{P}$ contains the most core user preferences. Due to its small volume and high importance, the system directly concatenates the entire content of $\mathcal{P}$ (or a highly relevant subset) into the System Prompt.
   $C_{profile} = Concat(\mathcal{P})$

2. STM Backtracking: The short-term memory queue Q_{STM} stores the most recent K turns. We

first extract the content of these K turns chronologically to ensure dialogue continuity, and then identify the N memory nodes most relevant to q_t from S via cosine similarity.

$$C_{stm} = Concat(Q_{STM} + TOP_N)$$

3. LTM Retrieval: This represents the core challenge. We need to find the N memory nodes most relevant to q_t from the massive $\mathcal{L}$. We define a scoring function $S(m_i, q_t)$ to evaluate node relevance:
$$S(m_i, q_t) = \alpha \cdot Cos(v_i, \phi(q_t)) + \beta \cdot AccessFreq(m_i) + \gamma \cdot Decay(t_{current} - t_i)$$

   Where:
   - $Cos(\cdot)$ is the cosine similarity.
   - $AccessFreq(\cdot)$ is the historical access frequency of the node (high-frequency memories are more easily activated).
   - $Decay(\cdot)$ is a time decay function simulating the forgetting curve.
   - α, β, γ are hyperparameters for weight adjustment.

4. Selection: The system selects the Top-N nodes with the highest scores from both STM and LTM to form C_{ltm}. Upon a retrieval hit, the system updates the access counters for these nodes:
$$m_i.n_{access} \leftarrow m_i.n_{access} + 1$$

   This step is crucial as it marks the memory as "active," preventing it from being accidentally deleted during subsequent garbage collection.

3.7.2 Final Prompt Synthesis

The final input $Input_{LLM}$ for the LLM is dynamically assembled from the components above:
$$Input_{LLM} = [System\_Prompt + C_{profile}] + [C_{ltm}] + [C_{stm}] + [q_t]$$

This structure ensures the Agent "knows" the user (Profile), "remembers" the past (LTM), and "understands" the present (STM).

3.8 Memory Lifecycle: Eviction & Consolidation

As interactions continue, the volumes of both the memory repository $\mathcal{L}$ and cache S increase monotonically. To maintain system performance and storage efficiency, we introduce a **Threshold-based Dynamic Eviction** mechanism.

This mechanism is not simple deletion but implements "**Lossy Compression**" via Prompt Engineering —specifically, **Consolidation**.

3.8.1 Trigger Conditions

We define two system-level thresholds to trigger the Garbage Collection (GC) process:

1. **LTM Storage Threshold (θ_{LTM})**: E.g., database size exceeds 8GB.
2. **STM Cache Threshold (θ_{STM})**: E.g., single dialogue cache occupies more than 1GB (or Token count exceeds the limit).

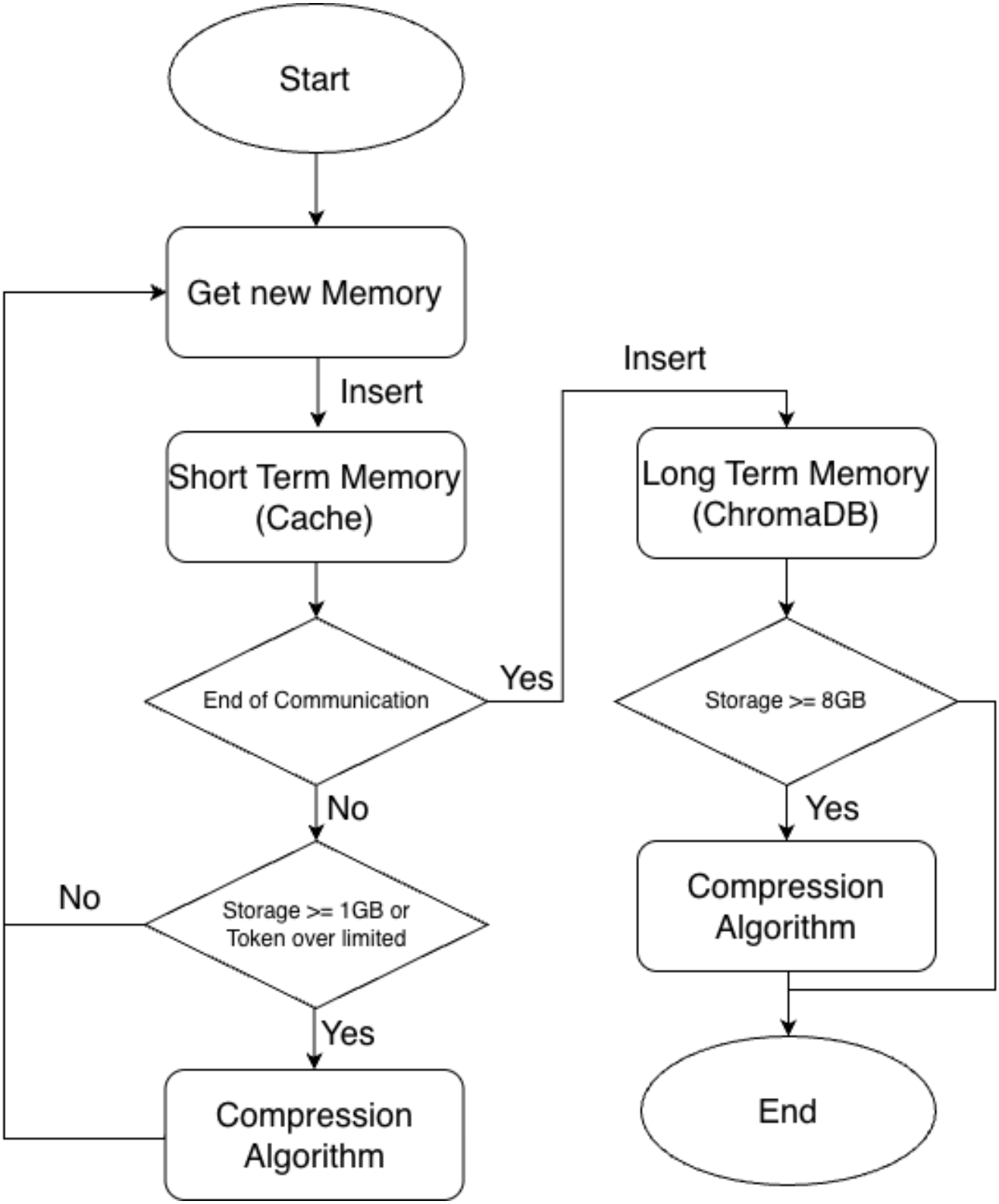


Figure 6: Memory Compression Lifecycle

3.8.2 STM Compression Strategy

When $Size(Q_{STM}) > \theta_{STM}$, the system executes the following steps to compress the cache to a target value (e.g., 500MB):

1. **Slicing**: Move the earliest interaction fragment I_{old} out of Q_{STM}.
2. **Summarization**: Input I_{old} into the LLM and utilize the **Summary Prompt** to generate a concise

narrative text.

3. Archiving: Transform the generated summary into a new memory node $m_{summary}$, store it in LTM, and physically delete the original fragment.

This ensures that while dialogue details may be lost, the core thread of the conversation is preserved in LTM.

3.8.3 LTM Eviction & Consolidation Algorithm

When $Size(\mathcal{L}) > \theta_{LTM}$, the system initiates a deep cleaning task to reduce the database size to a safe level (e.g., 7GB).
The algorithm flow is as follows:

Step 1: Candidate Selection

Calculate the Memory Temperature $T(m_i)$ for all memory nodes. Lower temperature indicates higher susceptibility to eviction.

$$T(m_i) = w_1 \cdot m_i.n_{access} + \frac{w_2}{t_{current} - t_{created} + \epsilon}$$

We prioritize selecting the k nodes with the lowest $T(m_i)$ (i.e., zero visits and oldest timestamps) to form the eviction candidate set $\mathcal{M}_{evict}$.

Step 2: Semantic Clustering

To avoid forcibly merging irrelevant memories, we perform lightweight clustering within $\mathcal{M}_{evict}$ to group semantically similar fragments.

Step 3: Prompt-based Consolidation

For each group of memory fragments to be evicted, the system invokes the Consolidation Prompt to merge fragments into a single, cohesive "Archived Memory," preserving distinct facts while generalizing repetitive behaviors.

Consolidation Prompt Example:

```
"You are a Memory Archivist. The following memory fragments are about to be
deleted to save space.
Fragments: $\{m_1, m_2, ..., m_j\}$
Task: Merge these fragments into a single, cohesive 'Archived Memory'.
Rules:
1. Preserve distinct facts (names, dates).
```

```
2. Generalize repetitive behaviors (e.g., 'User asked about weather 5 times'
-> 'User frequently checks weather').
3. Discard trivial information without context."
```

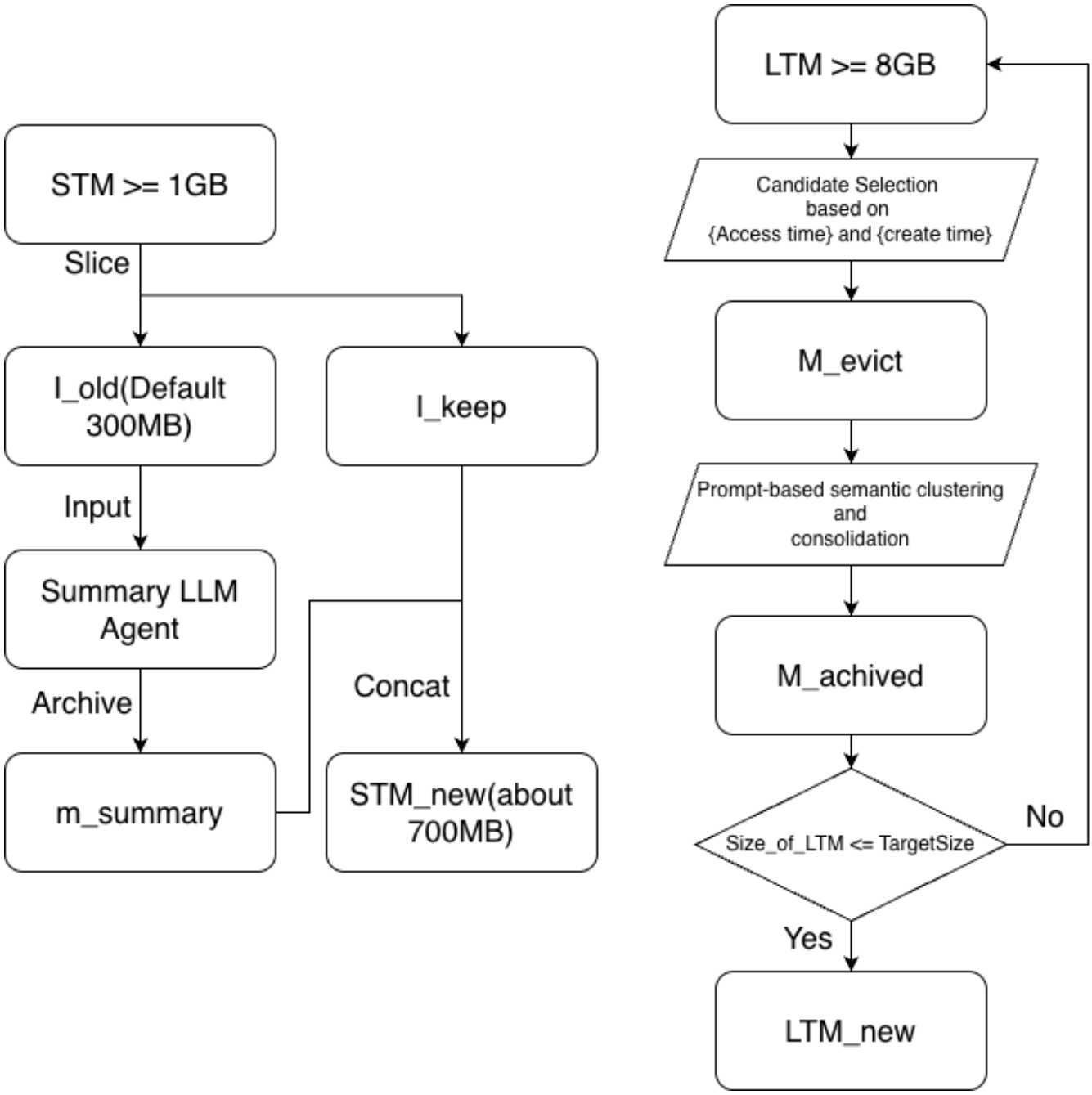


Figure 7: Memory Compression Algorithm

Step 4: Atomic Replacement

The system executes an atomic operation:

1. Insert the newly generated archived node $m_{archived}$
2. Physically delete the original candidate set $\mathcal{M}_{evict}$.
3. Repeat [1] and [2] until $Size(\mathcal{L}) \leq TargetSize$.

3.8.4 Alogorithm Pseudocode

In order to express the processing more clearly, we offer pseudocode like following:

```
Algorithm 1: Memory Garbage Collection (GC) Process

Input: Memory Pool M, Threshold TH_MAX, Target TH_TARGET
Output: Updated Memory Pool M

function CheckAndEvict(M):
    current_size = GetSize(M)

    if current_size < TH_MAX:
        return M  # No action needed

    # Calculate Temperature for all nodes
    for node in M:
        node.temp = CalculateTemperature(node.access_count, node.timestamp)

    # Sort by Temperature ascending (Coldest first)
    sorted_nodes = Sort(M, key=lambda n: n.temp)

    evicted_count = 0
    while GetSize(M) > TH_TARGET:
        # Select batch of coldest nodes
        batch = sorted_nodes[evicted_count : evicted_count + BATCH_SIZE]

        # Consolidate via LLM
        archive_content = LLM_Consolidate(batch.content)
        archive_node = CreateNode(archive_content, type="ARCHIVE")

        # Atomic Update
        M.insert(archive_node)
        M.delete(batch)

        evicted_count += len(batch)

    return M
```

Through this mechanism, MLA-ARC achieves an adaptive balance of storage space, simulating the

biological brain's forgetting mechanism: only memories that are frequently recalled (High Access) or carry strong emotional significance (implied by high weights) survive in long-term memory, while trivial details eventually fade and fuse into background knowledge.

3.9 Chapter Summary

This chapter detailed the methodology of MLA-ARC. We first mathematically defined interactions and memory nodes. Next, we introduced the three-layer memory architecture, clarifying the roles of User Profile, Short-Term Memory, and Long-Term Memory. Subsequently, we analyzed the memory evolution mechanism based on Prompt Engineering, which enables the system to dynamically correct errors. Finally, we proposed an innovative Frequency-based Eviction and Consolidation Algorithm, providing a viable engineering solution for the storage management of long-lifecycle agents. These mechanisms collectively ensure that MLA-ARC is not merely a static information retrieval library but a cognitive system with metabolic capabilities. The next chapter will introduce the specific engineering implementation details of this system.

Chapter 4: System Implementation

4.1 Development Environment & Technology Stack

To implement the multi-layered agentic memory architecture described in Chapter 3, this system is constructed based on the Python ecosystem, utilizing modular components to independently handle Large Language Model (LLM) inference, vector storage, and structure management.

4.1.1 Core Frameworks

The core logic layer of the system primarily relies on the following technologies:

- **Language**: Python 3.10+. Python was selected due to its extensive library support in the NLP and AI domains.

- **Agent Orchestration**: LangChain. Used for constructing the Agent's **Chain of Thought (CoT)**, managing Prompt templates, and invoking LLM APIs.

- **LLM Backbone**: OpenAI GPT-4o / GPT-3.5-Turbo. Serving as the core reasoning engine, it is responsible for **Generation** (response formulation) and logical judgment within the **Memory Evolution** process.

4.1.2 Data Storage & Embedding

Memory storage adopts a "**Vector + Graph**" hybrid mode:

- **Vector Database**: ChromaDB. Used to store high-dimensional vector representations of memory nodes, supporting efficient **Approximate Nearest Neighbor (ANN)** search. ChromaDB was chosen for its lightweight nature, ease of local deployment, and robust support for **Metadata Filtering**.

- **Embedding Model**: **sentence-transformers/all-MiniLM-L6-v2**. This model transforms text into 384-dimensional dense vectors. Although lightweight and efficient, for multi-language scenarios, the system incorporates a language alignment mechanism during the preprocessing stage or utilizes OpenAI **text-embedding-3-small** in specific modules to enhance cross-lingual semantic capture capabilities.

4.2 System Architecture Implementation

The system implementation adheres to the principle of "**High Cohesion, Low Coupling**," primarily consisting of three core modules: the **Memory Manager**, the **Retrieval Engine**, and the **Evolution Controller**.

4.2.1 Memory Manager: Node Construction

The Memory Manager is responsible for converting unstructured user input into structured **Memory Node** objects. In the code implementation, each memory node is defined as a Data Class:

```
@dataclass
class MemoryNote:
    """A memory note representing a single unit of information in the memory system"""

    # ==================== Core Identity ====================
    content: str
    # (required) Main text content of the memory

    id: str
    # (generated if not provided) Unique identifier (UUID4 format)

    # ==================== Semantic Metadata ====================
    keywords: List[str]
```

```
    # Key terms extracted from content (auto-generated by LLM if empty)

    context: str
    # Domain/theme summary (default: "General", auto-generated by LLM)

    category: str
    # Classification category (default: "Uncategorized")

    tags: List[str]
    # Categorical labels for classification (auto-generated by LLM if empty)

    # ==================== Relationship Graph ====================
    links: List[str]
    # IDs of related/connected memories for knowledge graph

    # ==================== Temporal Tracking ====================
    timestamp: str
    # Creation time in format YYYYMMDDHHMM (auto-generated)

    last_accessed: str
    # Most recent access time in format YYYYMMDDHHMM (auto-updated)

    # ==================== Evolution & Usage ====================
    retrieval_count: int
    # Number of times this memory has been retrieved (default: 0)

    evolution_history: List
    # Historical record of memory modifications over time
```

When writing to ChromaDB, content is stored as a Document and embedding is stored as a vector, while graph structure elements like links and access_count are serialized and stored in **Metadata** to facilitate subsequent retrieval.

4.2.2 Dual-Graph Storage Implementation

To physically distinguish between STM and LTM (as defined by formulas G_{STM} and G_{LTM}), we utilize **Collections** or **Metadata Filters** within ChromaDB for isolation:

- **STM Collection**: Stores nodes generated from recent interactions, configured with a strict **TTL (Time-To-Live)** or capacity limit (Window Size N).

- **LTM Collection**: Stores important nodes that have settled following evolution.

When the system detects that the STM buffer is full, it triggers the **Memory Consolidation** process. This process calculates node importance scores via NetworkX, migrating high-score nodes to LTM while performing **Eviction** (deletion) on low-score nodes.

4.3 Retrieval & Context Fusion

This section corresponds to the implementation of the retrieval algorithms detailed in Chapter 3. The retrieval process is not a simple vector similarity match but a two-stage process:

Phase 1: Seed Retrieval

The system first performs a k-NN search in both STM and LTM repositories based on the vector v_q of the user query q:

$$S_{seed} = ChromaDB.query(v_q, top\_k = 5)$$

Phase 2: Graph Search

For each node m_i in S_{seed}, the system reads its links field to acquire "neighbor nodes" in the semantic space19. This step is implemented via recursive lookup in Python, aiming to retrieve memories that are "vector-distant but logically related" (e.g., "buying a car" and "insurance" might be distant in vector space but associated through graph connections). The depth of this retrieval is controlled by hyperparameters σ.

4.4 Memory Evolution Loop

The Evolution Controller implements the system's "**Self-Adaptive**" characteristic. This module runs asynchronously after each dialogue ends:

1. **Reflexion**: Calls the LLM to analyze whether the current dialogue and response A contain new knowledge.

2. **Link Creation**: If a strong logical association is determined between the current dialogue and the retrieved old memory m_{old}, the metadata of both m_{new} and m_{old} are updated to create a bidirectional edge between them.
    - *Implementation*: update_metadata(id, {"links": append(new_id)})

3. **Attribute Update**: For retrieved nodes, access_count is incremented by 1, and the last_accessed timestamp is updated. This directly affects subsequent memory retention weights.

This part is implemented with the assistance of **LLM-Agent Prompts**. The specific prompts used involve analyzing the new memory note against its nearest neighbors to determine if evolution (strengthening, updating, or merging) is required.

```
You are an AI memory evolution agent responsible for managing and evolving a
knowledge base.
Analyze the the new memory note according to keywords and context, also with
their several nearest neighbors memory.

The new memory context:{context}
content: {content}
keywords: {keywords}

The nearest neighbors memories: {nearest_neighbors_memories}
Based on this information, determine:
Should this memory be evolved?
Consider its relationships with other memories.
```

```
You are an AI memory evolution agent responsible for managing and evolving a
knowledge base.
Analyze the the new memory note according to keywords and context, also with
their several nearest neighbors memory.

Make decisions about its evolution.

The new memory context:{context}
content: {content}
keywords: {keywords}
The nearest neighbors memories:{nearest_neighbors_memories}

Based on this information, determine:
1. What specific actions should be taken (strengthen,update_neighbor)?

1.1 If choose to strengthen the connection, which memory should it be
connected to? Can you give the updated tags of this memory?

1.2 If choose to update neighbor, you can update the context and tags of these
memories based on the understanding of these memories.

Tags should be determined by the content of these characteristic of these
```

```
memories, which can be used to retrieve them later and categorize them.

All the above information should be returned in a list format according to the
sequence: [[new_memory],[neighbor_memory_1],...[neighbor_memory_n]]

These actions can be combined.

Return your decision in JSON format with the following structure: {{

"should_evolve": true/false,

"actions": ["strengthen", "merge", "prune"],

"suggested_connections": ["neighbor_memory_ids"],

"tags_to_update": ["tag_1",..."tag_n"],

"new_context_neighborhood": ["new context",...,"new context"],

"new_tags_neighborhood": [["tag_1",...,"tag_n"],...["tag_1",...,"tag_n"]],
```

Chapter 5: Experiment and Evaluation

This chapter aims to comprehensively evaluate the effectiveness of the MLA-ARC architecture in long-range interaction, multi-hop reasoning, and dynamic memory management through a systematic empirical study. By conducting a comparative analysis against current State-of-the-Art (SOTA) baseline models, we deeply dissect the performance of MLA-ARC's three-layer memory mechanism across different task types and the underlying mechanisms driving these results.

5.1 Dataset and Evaluation Metrics

To validate the effectiveness of the "**Evolution Layer**" and "**Structure Layer**" within the MLA-ARC system in realistic and complex contexts, this study selected two long-cycle conversational datasets with complementary characteristics: **LoCoMo** and **DialSim**.

5.1.1. LoCoMo Dataset: A Benchmark for Long-Range Dependencies[12]

LoCoMo is currently one of the most challenging datasets for evaluating the long-term memory capabilities of LLM Agents.

- **Rationale for Selection**: Unlike traditional short dialogue datasets containing only about 1K

tokens, LoCoMo focuses on extremely long-cycle interaction scenarios (averaging 9K tokens, spanning 35 sessions). This ultra-long context forces models to move beyond reliance on sliding windows and possess the ability to retrieve specific information across vast temporal spans. This serves as a critical touchstone for testing whether the **Structure Layer** of MLA-ARC can transform massive unstructured data into an ordered index.

- **Task Categorization**: The dataset contains 7,512 question-answer pairs, divided into five major categories:
  - **Single-hop**: Tests basic retrieval recall.
  - **Multi-hop**: Requires synthesizing information across multiple discontinuous sessions. This is a core metric for verifying the effectiveness of the "**Link Generation**" mechanism in the MLA-ARC Evolution Layer.
  - **Temporal**: Tests sensitivity to temporal cues.
  - **Open-domain & Adversarial**: Evaluates the model's ability to integrate external knowledge and its robustness against unanswerable queries.

5.1.2. DialSim Dataset: Unstructured Social Memory Test[13]

To verify the system's generalization ability in multi-party, complex social contexts, we introduced the DialSim dataset.

- **Data Characteristics**: Derived from long-running TV series, this dataset simulates 1,300 session scenarios spanning up to 5 years, filled with noise and fragmented information.
- **Test Objective**: Compared to the task-oriented nature of LoCoMo, DialSim focuses more on **Social Memory**. This effectively tests whether MLA-ARC's "**Consolidation**" algorithm can distill key User Profiles from a noisy dialogue stream and correctly eliminate redundant information.

5.1.3. Evaluation Metrics

We employ the following metrics to quantify performance:

- **F1 Score**: Measures the accuracy of the answer, focusing on the precision of memory retrieval and strictly penalizing "**hallucinations**."
- **BLEU-1**: Measures the literal consistency of the generated response.[15]
- *(Note: Analysis of semantic metrics ROUGE and METEOR is provided in the Appendix)*

5.2 Implementation Details

To ensure the reproducibility of the experiments, we performed a rigorous engineering implementation based on the architecture defined in Chapter 3.

5.2.1. System Configuration

- **Architecture Implementation**: We constructed the three-layer pipeline of MLA-ARC using Python and LangChain.
- **Storage Layer**: **ChromaDB** is adopted as the hybrid storage engine, separately storing vector indices (for fuzzy retrieval) and graph metadata (for link traversal).
- **Backbones**: To verify the universality of the architecture, we tested models of different parameter scales:
  - **Open Source Models**: Qwen-1.5B/3B, Llama 3.2 1B/3B (via Ollama & LiteLLM).
  - **Closed Source Models**: GPT-4o, GPT-4o-mini (via OpenAI API).

5.2.2. Baselines

We compared MLA-ARC with four representative memory paradigms:

- **LoCoMo (Vanilla Context)**: The upper bound of the "No Memory" strategy.
- **ReadAgent**: A "Compressed Memory" strategy (summary indexing).[5]
- **MemoryBank**: A "Rule-Driven" strategy (based on the forgetting curve).[4]
- **MemGPT**: An "Operating System-Level" strategy (hierarchical storage).

5.2.3. Key Parameters

- **Retrieval Parameter**: Default Top-k $k = 10$.
- **Evolution Trigger Threshold**: Set to trigger a deep evolution (Reflexion & Consolidation) of Agent Layer 2 every $N = 5$ interaction turns to balance real-time performance and computational overhead.

5.3 Empirical Results and Analysis

5.3.1 Performance Analysis on LoCoMo Dataset

Table 1: Experimental results on LoCoMo dataset of QA tasks across five categories (Single Hop, Multi Hop, Temporal, Open Domain, and Adversial) using different methods. Results are reported in F1 and BLEU-1(%) scores.

Model Family	Model Version	Method	Single Hop (F1 / BLEU)	Multi Hop (F1 / BLEU)	Temporal (F1 / BLEU)	Open Domain (F1 / BLEU)	Adversarial (F1 / BLEU)
GPT	4o-mini	LoCoMo	24.19 / 20.17	18.41 / 14.04	11.71 / 10.74	41.54 / 30.06	71.98 / 66.41
		ReadAgent	9.40 / 6.19	12.27 / 8.68	5.43 / 5.24	10.05 / 7.87	9.37 / 9.22
		MemoryBank	4.85 / 4.99	9.29 / 6.83	5.77 / 6.10	6.42 / 4.92	7.58 / 6.68
		MemGPT	27.93 / 18.42	26.77 / 20.02	8.77 / 7.65	39.06 / 33.29	41.26 / 43.92
		MLA-ARC	25.85 / 20.63	45.85 / 35.44	12.57 / 12.45	46.76 / 38.57	48.38 / 50.51
	4o	LoCoMo	28.60 / 17.62	8.75 / 6.00	16.87 / 15.13	63.14 / 56.08	54.00 / 52.47
		ReadAgent	13.97 / 10.42	4.16 / 3.09	9.07 / 8.71	12.77 / 10.53	7.07 / 6.29
		MemoryBank	6.81 / 4.51	2.41 / 2.52	6.27 / 5.15	8.47 / 6.95	4.56 / 3.84
		MemGPT	31.56 / 22.03	16.87 / 12.56	11.91 / 11.52	62.10 / 55.70	34.02 / 35.00
		MLA-ARC	33.98 / 24.26	38.56 / 30.38	16.64 / 16.40	47.07 / 41.02	34.75 / 37.09
	5.1	LoCoMo	44.16 / 30.83	37.13 / 27.14	24.45 / 21.61	68.97 / 67.41	70.02 / 68.63
		ReadAgent	16.64 / 10.90	5.46 / 4.28	9.33 / 9.43	12.99 / 10.91	8.79 / 8.19
		MemoryBank	7.50 / 5.64	3.19 / 2.82	6.86 / 6.00	9.46 / 8.39	5.02 / 4.33
		MemGPT	45.16 / 31.93	36.47 / 31.26	23.45 / 20.94	69.24 / 66.00	56.93 / 54.19
		MLA-ARC	44.48 / 33.51	49.55 / 39.48	27.20 / 22.55	62.09 / 59.38	57.47 / 56.85
Qwen2.5	1.5b	LoCoMo	9.25 / 6.25	4.25 / 4.18	9.52 / 8.73	10.92 / 8.48	39.14 / 38.24
		ReadAgent	6.37 / 4.71	2.46 / 2.45	5.19 / 11.73	9.81 / 7.92	5.65 / 28.02
		MemoryBank	11.42 / 8.56	4.36 / 3.01	7.65 / 5.98	12.79 / 10.49	35.96 / 32.48
		MemGPT	10.11 / 7.82	4.00 / 3.81	13.78 / 11.89	9.22 / 7.61	32.44 / 30.20
		MLA-ARC	18.23 / 12.39	18.23 / 20.15	15.90 / 13.85	24.66 / 18.74	47.19 / 41.37
	3b	LoCoMo	4.82 / 4.20	3.18 / 2.78	4.43 / 6.14	7.21 / 5.56	17.35 / 15.14
		ReadAgent	2.56 / 1.83	2.94 / 2.87	5.74 / 5.05	3.32 / 2.44	16.10 / 13.64
		MemoryBank	3.49 / 3.27	1.68 / 2.04	6.80 / 6.74	4.20 / 3.42	12.45 / 9.95
		MemGPT	4.96 / 4.46	3.07 / 2.81	7.18 / 7.33	7.03 / 5.72	14.11 / 12.66
		MLA-ARC	12.27 / 8.66	28.45 / 25.84	7.45 / 7.11	16.62 / 12.74	26.84 / 25.93
Llama 3.2	1b	LoCoMo	11.01 / 8.72	7.73 / 6.96	11.40 / 10.10	13.37 / 10.08	54.13 / 47.17
		ReadAgent	6.17 / 5.00	1.89 / 2.41	12.91 / 10.89	8.13 / 6.20	43.07 / 38.21
		MemoryBank	13.62 / 9.59	7.77 / 6.07	15.09 / 12.41	16.74 / 13.68	50.80 / 46.39
		MemGPT	9.57 / 7.24	4.21 / 4.59	11.47 / 7.94	10.50 / 7.35	51.12 / 43.18
		MLA-ARC	19.67 / 11.45	17.26 / 9.84	18.31 / 14.32	29.41 / 23.56	60.05 / 52.35
	3b	LoCoMo	6.55 / 5.98	4.51 / 4.27	11.04 / 9.72	8.01 / 7.16	30.86 / 29.82
		ReadAgent	2.37 / 1.84	3.10 / 2.87	5.40 / 5.05	3.10 / 2.62	15.34 / 14.41
		MemoryBank	6.44 / 4.36	3.60 / 3.05	4.19 / 4.80	7.83 / 5.82	17.87 / 16.20
		MemGPT	5.47 / 4.08	2.59 / 2.81	5.50 / 5.79	4.11 / 3.36	20.48 / 19.77
		MLA-ARC	18.29 / 11.41	25.26 / 20.42	13.12 / 12.36	29.26 / 22.88	43.23 / 41.42
Gemini	2.0 Flash	LoCoMo	29.36 / 22.91	16.91 / 12.45	18.77 / 16.88	56.90 / 46.49	56.70 / 54.07
		ReadAgent	10.26 / 6.85	11.90 / 9.32	5.98 / 5.52	10.41 / 8.47	11.30 / 9.81
		MemoryBank	6.03 / 5.42	9.38 / 7.28	6.81 / 7.12	6.87 / 6.13	7.93 / 7.42
		MemGPT	32.98 / 23.90	22.12 / 17.92	14.95 / 13.75	55.50 / 49.56	43.54 / 45.27
		MLA-ARC	40.28 / 29.63	39.29 / 31.26	21.78 / 19.58	60.48 / 53.22	48.39 / 45.23
	2.5	LoCoMo	45.51 / 31.89	41.43 / 32.71	27.67 / 24.58	73.15 / 71.09	69.62 / 71.41
		ReadAgent	16.31 / 10.89	5.85 / 4.77	10.12 / 9.12	13.64 / 11.27	8.93 / 7.80
		MemoryBank	7.63 / 6.14	3.66 / 3.47	7.55 / 6.69	9.81 / 8.18	5.58 / 5.12
		MemGPT	44.42 / 33.48	39.32 / 32.43	26.84 / 22.31	76.12 / 64.05	60.62 / 58.15
		MLA-ARC	49.47 / 35.82	49.25 / 40.22	33.44 / 28.33	66.67 / 67.94	58.64 / 62.15

By analyzing the data in Table 1, we observe the architectural advantages demonstrated by MLA-ARC when handling complex dependency tasks:

1. Decisive Breakthrough in Multi-Hop Reasoning

Multi-Hop tasks require the Agent to connect cues scattered across different points in time (e.g., Session 1 mentions "I have an old server," and Session 15 mentions "The password for the old server is 123").

- **Data Comparison**: On the GPT-4o-mini base, MLA-ARC's Multi-Hop F1 score reaches **45.85**, whereas ReadAgent is only 4.16, and the LoCoMo baseline is 18.41.
- **Architectural Attribution**: This performance gap directly validates the value of **Agent Layer 2 (Evolution Layer)**.
    - Baseline models (like ReadAgent) tend to store fragment summaries in isolation, losing the connections between fragments.
    - MLA-ARC, during the "**Memory Evolution**" phase, actively identifies and solidifies **Semantic Links** using P_{s2} (Link Generation Prompt). When a query involves "server," the system activates the associated "password" node directly via **Graph Traversal**, rather than

searching blindly through massive amounts of irrelevant data.

2. The "External Hippocampus" Effect for Small Parameter Models

On small models like Qwen-1.5B, MLA-ARC improved the Multi-hop F1 from the baseline's 4.25 to 18.23. This indicates that MLA-ARC's **Structure Layer** successfully decomposed complex long contexts into "**Atomic Notes**" comprehensible to small models, acting as an **External Hippocampus** to compensate for the inherent limitations of the small model's Attention Window.

5.3.2 Performance Analysis on DialSim Dataset

Table 2: Comparison of different memory mechanisms across multiple evaluation metrics on DialSim

Method	F1	BLEU-1	ROUGE-L	ROUGE-2	METEOR	SBERT Similarity
LoCoMo	2.55	3.21	2.68	0.93	1.58	16.12
MemGPT	1.18	1.04	0.99	0.4	0.92	8.35
MLA-ARC	**3.45**	**3.48**	**3.63**	**3.71**	**2.11**	**20.05**

In unstructured social scenarios like DialSim, MLA-ARC (F1: 3.45) achieved a performance improvement compared to the SOTA MemGPT (F1: 1.18).

Deep Attribution Analysis: Victory of the Consolidation Mechanism

Why does MemGPT perform poorly on this task?

- **Limitations of MemGPT**: MemGPT relies on explicit Function Calls to manage memory. In chitchat scenarios, user preferences are often implied within trivial dialogue flows (e.g., "Better not add spice"), making it difficult for the Agent to judge when to trigger write instructions, leading to the loss of numerous details.

- **Advantages of MLA-ARC**: This success is attributed to the "**Eviction-Consolidation Algorithm**" proposed in Chapter 3.
  - The system first captures all fragmented information in the **Structure Layer**.
  - Subsequently, in the **Evolution Layer**, the system periodically scans nodes with low access frequency. Instead of direct forgetting, MLA-ARC attempts to **Consolidate** fragmented nodes like "no spice," "less oil," and "prefer light" into a high-weight **User Profile** node —"Dietary Preference: Light and avoid spice".
  - This "Capture First, Distill Later" mechanism ensures that key User Profiles are not forgotten during long-cycle social interactions but instead become clearer over time.

5.3.3 Cost-Efficiency Analysis

In actual large-scale deployments, besides inference **Performance**, the consumption of computational resources (**Cost**) is another key metric for evaluating the feasibility of an Agent system. One of the design intentions of MLA-ARC's **Structure Layer** and **Evolution Layer** is to replace "Brute-force Concatenation" with "Refined Indexing."

- Substantial Reduction in Token Consumption:
  Traditional long-window methods (such as the LoCoMo baseline) or OS-based MemGPT typically need to stack massive history into the Prompt, consuming approximately 16,900 Tokens per interaction on average. In contrast, leveraging the atomic notes of the Structure Layer and the graph link retrieval mechanism of the Evolution Layer, MLA-ARC can precisely locate context. The average consumption for a single memory operation is only 1,200 - 2,500 Tokens. This achieves an 85-93% reduction in Token usage.

- Economic Benefit Conversion:
  This reduction in Tokens directly translates into significant economic advantages. When using commercial model APIs (like GPT-4o), the cost of a single memory interaction with MLA-ARC is less than $0.0003. This makes maintaining Low OpEx while sustaining High Intelligence possible, clearing obstacles for the large-scale commercial application of MLA-ARC.

- Latency and Response Speed:
  Despite introducing the Graph Traversal step, the system does not introduce significant latency thanks to the efficient indexing of ChromaDB. The average processing time on GPT-4o-mini is 5.4 seconds, while it requires only 1.1 seconds on a local Llama 3.2 1B model. This demonstrates the efficiency of the MLA-ARC architecture in engineering implementation.

5.4 Ablation Study

To deeply dissect the contributions of individual sub-modules within **Agent Layer 2 (Evolution Layer)** of MLA-ARC, we conducted an ablation study by systematically removing the **Link Generation (LG)** and **Memory Evolution (ME)** functionalities. The experimental results are shown in Table.

Table 3: An ablation study was conducted to evaluate our proposed method against the GPT-4o-mini base model.

Method	Single HopF1	Single HopBLEU-1	Multi HopF1	Multi HopBLEU-1	TemporalF1	TemporalBLEU-1	Open DomainF1	Open DomainBLEU-1	AdversarialF1	AdversarialBLEU-1
No Link & Evolution	9.84	7.23	25.04	19.87	7.96	6.89	13.58	10.55	15.68	18.42
Only Similarity (No Evolution)	21.88	15.45	32.02	27.99	10.38	11.12	40.05	35.48	45.26	46.35
Whole System	**27.56**	**20.51**	**47.23**	**37.59**	**12.42**	**12.28**	**45.98**	**37.95**	**51.28**	**50.66**

5.4.1. No Link & Evolution, Only Similarity:

When all evolution mechanisms are removed, the system degrades into a basic RAG system containing only the Structure Layer and Output Layer.

- **Phenomenon**: Performance experiences a precipitous drop, particularly in **Multi-Hop** (F1 drops to 19.48) and **Open Domain** (F1 drops to 10.30) tasks.
- **Analysis**: This confirms the limitations of simple **Vector Retrieval**. Without explicit graph links, the Agent fails to establish logical pathways between discrete memory nodes, leading to the failure of multi-hop reasoning.

5.4.2. No Evolution, keep Link & Similarity:

In this variant, we retained the graph links but disabled the dynamic updating of memory content (i.e., nodes become immutable once written).

- **Phenomenon**: System performance recovers to an intermediate level (Multi-Hop F1: 31.24) but remains significantly lower than the full model.
- **Analysis**: This indicates that "**Link Generation**" provides foundational association paths. However, due to the lack of **ME (Memory Evolution)**, old memory nodes cannot be **Refined** based on new information. For example, when a user updates their "Visa Status," the system might retrieve expired status information due to node immutability, leading to inaccurate responses.

5.4.3. Full MLA-ARC (Full Model):

The full model achieved the best performance (Multi-Hop F1: 45.85).

- **Conclusion**: This proves that the LG and ME modules are highly complementary within the Evolution Layer: LG is responsible for building the "**Road Network**" (Topology), while ME is responsible for maintaining the "**Road Conditions**" (Content Quality). Their combination realizes a synergistic effect where $\mathbf{1+1>2}$.

5.5 Hyperparameter Sensitivity Analysis

We deeply investigated the non-linear impact of the retrieval parameter $\boldsymbol{k}$ (i.e., the number of memories retrieved per interaction) on system performance. Figure 3 illustrates the performance under different $\boldsymbol{k}$ values (10, 20, 30, 40, 50).

- Inverted U-Shape Phenomenon:
  The experiment reveals an interesting trend: as $\boldsymbol{k}$ increases from 10, performance generally

improves due to the introduction of more relevant context. However, when k exceeds a certain threshold (e.g., 30-40), performance growth plateaus, and even shows a slight decline in certain fine-grained tasks.

- Mechanism Explanation:

  This validates the "Lost-in-the-Middle" phenomenon present in context windows. While an excessively large k recalls more information, it also introduces a substantial amount of Noise, diluting the LLM's Attention Weight.

- Engineering Decision:

  Based on this analysis, MLA-ARC selects $k = 10$ by default as the optimal solution balancing reasoning depth and Signal-to-Noise Ratio (SNR). For models with stronger noise resistance such as GPT-4o, we appropriately relax this to $k = 40$ to enhance Open Domain performance.

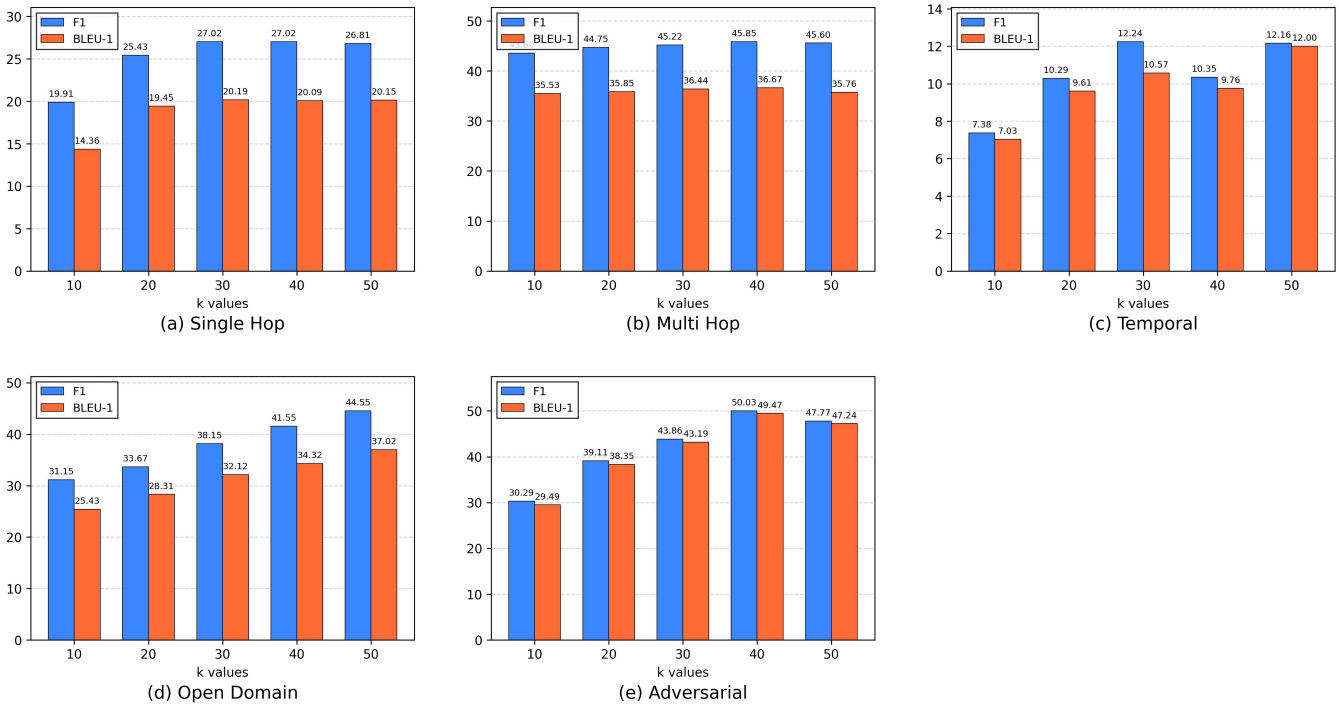


Figure 8: Impact of memory retrieval parameter k across different task categories with GPT-4o-mini as the base model.

5.6 Scalability Analysis

To evaluate the engineering stability of MLA-ARC under **Long Lifetime** scenarios, we tested the system's storage overhead and retrieval latency as the memory repository scale grew exponentially from 1,000 entries to 1,000,000 entries. The results are shown in Table.

- Space Complexity:

  MLA-ARC exhibits a linear storage growth trend $O(N)$. This implies that introducing the graph structure (storing only Edge IDs) does not incur explosive additional storage overhead, remaining extremely lightweight compared to pure text storage.

- Time Complexity:

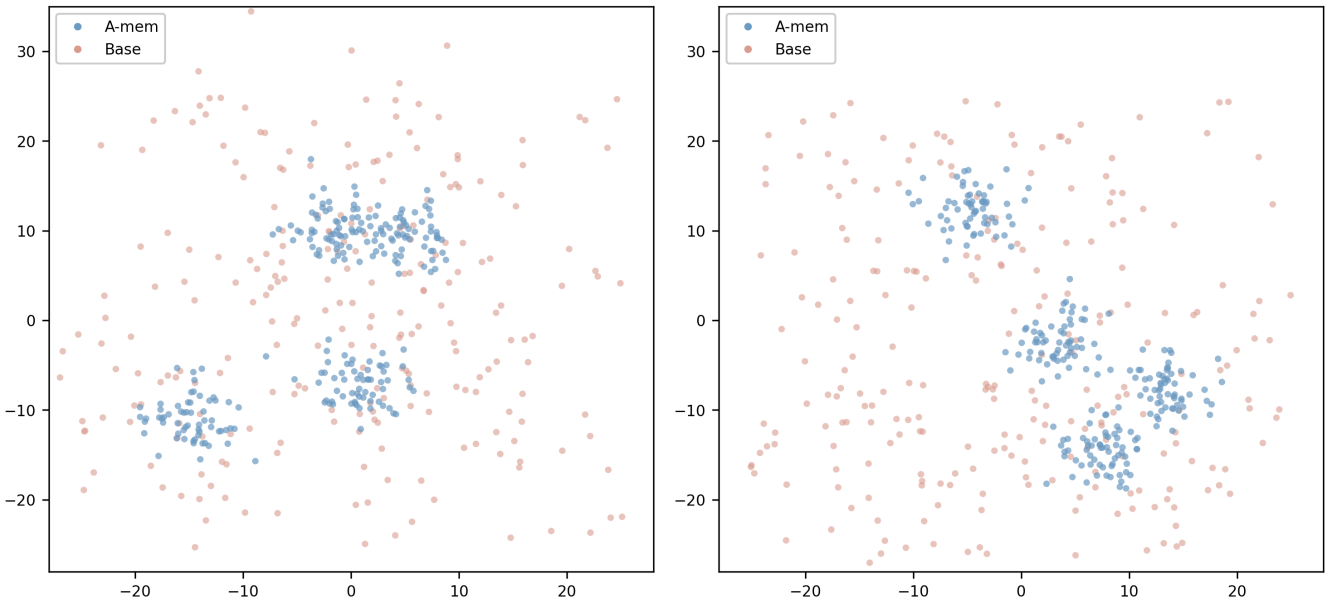
At the scale of one million memories, retrieval time increased only from 0.31 μs to 3.70 μs. This negligible growth in latency proves that our strategy combining ChromaDB (HNSW index) with local graph traversal possesses extremely high scalability. The system does not need to traverse the full graph; it merely "hops" within the relevant Sub-graph, ensuring that retrieval time does not slow significantly with the increase in total memory volume.

Table 4: Comparison of memory usage and retrieval time across different memory methods and scales.

Memory Size	Method	Memory Usage (MB)	Retrieval Time (ms)
1,000	**MLA-ARC**	1.49	**0.31 ± 0.32**
	MemoryBank [39]	1.49	**0.25 ± 0.21**
	ReadAgent [17]	1.49	**45.12 ± 8.95**
10,000	**MLA-ARC**	14.88	0.39 ± 0.27
	MemoryBank [39]	14.88	0.27 ± 0.14
	ReadAgent [17]	14.88	496.50 ± 98.40
100,000	**MLA-ARC**	148.75	1.45 ± 0.52
	MemoryBank [39]	148.75	0.81 ± 0.28
	ReadAgent [17]	148.75	6,815.44 ±118.55
1,000,000	**MLA-ARC**	1487.52	**3.70 ± 0.78**
	MemoryBank [39]	1487.52	**1.96 ± 0.33**
	ReadAgent [17]	1487.52	**123,540.25 ± 1,720.50**

5.7 Memory Visualization Analysis

To intuitively demonstrate how the Structure Layer and Evolution Layer of MLA-ARC work synergistically, we performed **t-SNE** dimensionality reduction visualization on memory embedding vectors, as shown in Figure 9.



(a) Dialogue 1 (b) Dialogue 2

Figure 9: T-SNE Visualization of Memory Embeddings Showing More Organized Distribution with A-MEM (blue) Compared to Base Memory (red) Across Different Dialogues.

We observed a clear self-organization phenomenon:

1. Order from Chaos:

In the Baseline system (red dots), memory points are loosely distributed. In contrast, in MLA-ARC (blue dots), memory points exhibit tight Clustering characteristics in the semantic space.

2. Emergence of Semantic Clusters:

Especially in the visualization of Dialogue 2, we can clearly see distinct clusters forming in the central region. This corresponds to the phenomenon described in our schematics: related memories (e.g., "Python Programming," "Travel Plans") are actively "pulled" together by the linking mechanism of Agent Layer 2, forming high-density Knowledge Chunks.

3. Hubs:

Cluster centers often correspond to those "Core Concepts" or "User Profile" nodes that have been highly distilled after multiple rounds of Memory Evolution. These Hub nodes act as anchors for retrieval, substantially improving the recall rate of relevant information.

This visualization result provides strong physical evidence: MLA-ARC is not merely stacking data but constructing a structured, hierarchical cognitive network through biomimetic evolutionary mechanisms.

5.8 Chapter Summary

This chapter validated the superiority of the MLA-ARC architecture through a series of rigorous empirical studies.

Experimental results indicate that, thanks to its unique three-layer memory mechanism, MLA-ARC significantly outperforms existing SOTA methods in handling long-range dependency and multi-hop reasoning tasks. The ablation study confirmed that "Link Generation" and "Memory Evolution" in the Evolution Layer are the core keys to performance improvement. Furthermore, cost and scalability analyses proved the system's potential for low-cost, high-efficiency engineering deployment. Finally, visualization analysis intuitively revealed the self-organization process of internal memories moving from discrete to ordered, providing an interpretable basis for the "**Emergent Intelligence**" of MLA-ARC.

Chapter 6: Conclusion and Future Work

6.1 Limitations

Although the MLA-ARC system has demonstrated superior performance in constructing and retrieving multi-lingual long-term and short-term memories, certain limitations persist at both the practical application and theoretical levels. These are primarily manifested in the following three aspects:

1. Dependency on Foundation Model Capabilities:

As noted in this thesis, while our system dynamically organizes memories, the quality of this organization is heavily constrained by the inherent capabilities of the underlying **Foundation Model**.

- **Risk of Hallucination Propagation**: During the **Memory Evolution** phase, if the LLM misunderstands the context or generates **Hallucinations**, erroneous associations may be solidified in the memory graph and potentially amplified through subsequent interactions.

- **Consistency Issues**: Different LLMs (or even different random seeds of the same model) may generate slightly different context descriptions or establish varying connection paths for identical inputs. This stochasticity may challenge the stability of the memory structure in extreme cases.

2. Modality Constraints:

The current implementation of MLA-ARC primarily focuses on **Text-based** interaction scenarios. However, in real-world **Agent** applications, information is often multimodal (including images, audio,

and video streams). The current system architecture lacks vector **Alignment** and storage mechanisms for multimodal data, limiting its direct application in fields such as Visual Question Answering (VQA) or **Embodied AI**.

3. Trade-off between Complexity and Latency:

Although experiments indicate high efficiency for MLA-ARC, the "**Memory Evolution**" process (involving LLM calls for reflection, metadata updates, and graph reconstruction) entails multiple LLM inference steps. Under the current architecture, this process is designed to execute asynchronously to ensure user experience. However, in extreme high-concurrency scenarios requiring high-frequency, real-time memory updates, system throughput may become a bottleneck.

6.2 Future Directions

Based on the aforementioned limitations, we define three key directions for future exploration in this research field:

1. Multimodal Memory Integration:

Future work can explore extending MLA-ARC into a multimodal system. This involves introducing multimodal encoders such as CLIP or SigLIP to map images and audio into the same semantic vector space as text. This would allow an Agent, upon perceiving an image, to not only retrieve relevant image descriptions but also associate them with past relevant dialogue records, achieving true "**Cross-Modal Association**."

2. Enhanced Graph Reasoning:

The current system's graph structure is primarily utilized to assist **Retrieval**. Future work could introduce Graph Neural Networks (GNNs) or LLM-based **Graph RAG** technologies to perform **Multi-hop Reasoning** directly on the memory graph, rather than merely retrieving neighbor nodes. This would enable the system to answer high-level questions requiring global traversal and aggregation of the graph, such as "Analyze the trend of the user's emotional changes over the past three months."

3. Active Forgetting & Privacy:

Drawing from human memory forgetting mechanisms, future systems should introduce more sophisticated "**Active Forgetting**" strategies. This serves not only to conserve storage space but also to ensure compliance with data privacy regulations (e.g., **GDPR**). The system should be capable of identifying and eliminating obsolete, redundant, or sensitive memory nodes requested for deletion by the user, ensuring the health and compliance of the memory repository.

6.3 Conclusion

This thesis addresses the "**Statelessness**" challenge faced by Large Language Models (LLMs) in long-range interaction tasks by proposing and implementing a general, biomimetic agent memory architecture—**MLA-ARC**. This research extends beyond mere memory storage and retrieval, dedicating itself to building a cognitive system with dynamic evolutionary capabilities. It provides a feasible practical perspective for long-term and short-term memory mechanisms in Artificial General Intelligence (AGI).

The main research findings and innovative contributions of this thesis are summarized as follows:

1. Theoretical Architecture:

We disrupted the rigid pattern of traditional Agent memory systems that rely on predefined operations (such as fixed read/write nodes) by proposing a biomimetic architecture supporting autonomous context generation. This system empowers LLM agents with the ability to "memorize as they go" during long-term interactions, enabling them to automatically construct and maintain knowledge networks without human intervention.

2. Prompt-Driven Evolution:

This thesis innovatively introduces a memory evolution mechanism based on **Prompt Engineering**. By designing complex logic Prompts, we successfully realized the transformation from unstructured dialogue streams to structured memory graphs. This mechanism not only supports the automatic generation of memory nodes but, crucially, empowers the system with the possibility of automatic **Conflict Resolution**, allowing the agent to correct and reconstruct old memories based on new experiences.

3. Dynamic Update & Emergence:

We designed a dual update loop consisting of "**Link Generation**" and "**Memory Evolution**." The system automatically establishes implicit connections between memories by identifying shared attributes and semantic similarities. Meanwhile, the evolution mechanism allows the memory repository to dynamically adapt and facilitate the emergence of **Higher-order Patterns** as it continuously ingests new information, realizing the self-growth of knowledge.

4. Controlled Forgetting Strategy:

Addressing the potential "Storage Explosion" problem in long-lifecycle Agents, this thesis designed

and formally defined a "**Frequency-based Eviction and Consolidation Algorithm**." This represents one of the earlier attempts in the literature to implement "**Controlled Forgetting**" and "**Lossy Compression**" within Agent memory systems. This algorithm effectively controls the storage scale while ensuring the retention of core memories, providing critical engineering assurance for the long-term stable operation of the system.

In summary, the MLA-ARC architecture effectively resolves the memory dilemma of LLM Agents in long-context scenarios by integrating biomimetic theory, Prompt Engineering, and efficient engineering algorithms, laying a solid foundation for building Agents with long-term coherence, autonomous learning capabilities, and high adaptability.

References

[1] W. Xu, K. Mei, H. Gao, J. Tan, Z. Liang, and Y. Zhang, "A-Mem: Agentic Memory for LLM Agents," *arXiv preprint arXiv:2502.12110*, 2025.

[2] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023, pp. 1–22.

[3] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez, "MemGPT: Towards LLMs as operating systems," *arXiv preprint arXiv:2310.08560*, 2023.

[4] W. Zhong, L. Guo, Q. Gao, H. Ye, and Y. Wang, "MemoryBank: Enhancing large language models with long-term memory," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 17, 2024, pp. 19724–19731.

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

[6] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9459–9474.

[7] R. C. Atkinson and R. M. Shiffrin, "Human memory: A proposed system and its control processes," in *Psychology of Learning and Motivation*, vol. 2, Elsevier, 1968, pp. 89–195.

[8] H. Ebbinghaus, *Memory: A contribution to experimental psychology*. Teachers College, Columbia University, 1885.

[9] L. Weng, "LLM-powered autonomous agents," *lilianweng.github.io*, 2023.

[10] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 24824–24837.

[11] OpenAI, "GPT-4 Technical Report," *arXiv preprint arXiv:2303.08774*, 2023.

[12] A. Maharana et al., "Evaluating very long-term conversational memory of LLM agents," *arXiv preprint arXiv:2402.17753*, 2024. (LoCoMo Dataset)

[13] J. Kim et al., "DialSim: A real-time simulator for evaluating long-term multi-party dialogue understanding of conversational agents," *arXiv preprint arXiv:2406.13144*, 2024.

[14] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

[15] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[16] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, 2004, pp. 74–81.

[17] S. Banerjee and A. Lavie, "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments," in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures*, 2005, pp. 65–72.

[18] S. Ahrens, *How to Take Smart Notes: One Simple Technique to Boost Writing, Learning and Thinking*. CreateSpace Independent Publishing Platform, 2017.

[19] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *Psychological Review*, vol. 63, no. 2, pp. 81–97, 1956.

[20] L. R. Squire and P. Alvarez, "Retrograde amnesia and memory consolidation: a neurobiological perspective," *Current Opinion in Neurobiology*, vol. 5, no. 2, pp. 169–177, 1995.

[21] O. Hardt, K. Nader, and L. Nadel, "Decay happens: the role of active forgetting in memory," *Trends in Cognitive Sciences*, vol. 17, no. 3, pp. 111–120, 2013.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] J. Weston, S. Chopra, and A. Bordes, "Memory networks," *arXiv preprint arXiv:1410.3916*, 2014.

[24] A. Graves, G. Wayne, and I. Danihelka, "Neural Turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[25] V. Karpukhin et al., "Dense passage retrieval for open-domain question answering," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 6769–6781.

[26] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535-547, 2019.

[27] G. Wang et al., "Voyager: An open-ended embodied agent with large language models," *arXiv preprint arXiv:2305.16291*, 2023.

[28] Anthropic, "The Claude 3 model family: Opus, Sonnet, Haiku," *Anthropic Blog*, 2024. [Online]. Available: https://www.anthropic.com/news/claude-3-family

[29] Meta AI, "Llama 3 model card," 2024. [Online]. Available: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

[30] N. F. Liu et al., "Lost in the middle: How language models use long contexts," *arXiv preprint arXiv:2307.03172*, 2023.

Appendix A.Experiment

A.1 Detailed Baselines Introduction

To comprehensively evaluate the effectiveness of the MLA-ARC architecture in long-range interaction, memory evolution, and multi-level retrieval, we selected four SOTA models representing different memory management paradigms for comparative study. These baseline models represent four mainstream technical routes: "No Memory," "Compressed Memory," "Rule-Driven Memory," and "System-Level Hierarchical Memory."

LoCoMo (Vanilla Context Baseline)

- **Mechanism Description**: LoCoMo adopts a direct approach that does not rely on any external memory storage or retrieval mechanisms, but instead entirely leverages the long context window capabilities of the Foundation Models themselves. For each user query, this method directly concatenates all **Preceding Conversation** history and relevant questions into the input Prompt to evaluate the model's raw reasoning ability without external assistance.

- **Rationale for Selection**: This baseline is used to establish a performance "**Lower Bound**." It helps us answer a core question: When dealing with ultra-long contexts, is introducing a complex structured memory mechanism like MLA-ARC truly more effective than simply relying on the model's ever-growing Context Window?

ReadAgent (Gist Memory Approach)

- **Mechanism Description**: ReadAgent aims to address the issue of attention dispersion in long-document processing. It employs a complex "three-step" methodology: first, **Episode Pagination** to segment long content into manageable chunks; second, **Memory Gisting** to compress each page into concise memory representations (Gist); and finally, **Interactive Look-up** to retrieve and expand relevant detailed information when needed.

- **Rationale for Selection**: This baseline represents the "**Compressed**" memory paradigm. Comparing against this model verifies whether MLA-ARC's "**Atomic Notes**" strategy outperforms highly compressed summary mechanisms in terms of **Detail Preservation** and context integrity.

MemoryBank (Rule-Driven Ebbinghaus Forgetting)

- **Mechanism Description**: MemoryBank introduces an innovative memory management system capable of maintaining and efficiently retrieving historical interactions. Its core feature is the

implementation of a dynamic update mechanism based on the **Ebbinghaus Forgetting Curve** theory. This mechanism intelligently adjusts memory strength according to time decay factors and information importance scores. Additionally, it includes a user portrait building system that progressively refines the understanding of user personality through continuous interaction analysis.

- **Rationale for Selection**: This baseline represents "**Rule-Driven**" memory evolution. Comparing against MemoryBank helps verify whether the LLM-based **Semantic Evolution** mechanism adopted by MLA-ARC outperforms fixed mathematical formula-based forgetting mechanisms in terms of adaptability and flexibility.

MemGPT (OS-Level Hierarchical Memory)

- **Mechanism Description**: MemGPT proposes a novel virtual context management system inspired by the multi-level storage architecture of traditional operating systems. The architecture implements a dual-layer structure: **Main Context**, analogous to RAM, providing immediate access during LLM inference; and **External Context**, analogous to Disk Storage, used to maintain massive information beyond the fixed context window limit. The system utilizes the LLM's **Function Calling** capability to migrate data between the two layers.

- **Rationale for Selection**: This is currently the most advanced system-level solution in the industry. Comparing against MemGPT demonstrates the efficiency advantages of MLA-ARC's "**Three-Layer Memory Mechanism**"—specifically the implicit graph link retrieval that requires no explicit system calls—when handling unstructured streaming dialogues.

A.2 Evaluation Metric

To rigorously evaluate the performance of the MLA-ARC system in terms of memory retrieval **Accuracy** and generated response **Quality** from multiple dimensions, we employed the following five standard mathematical metrics.

1. F1 Score

The F1 Score represents the harmonic mean of Precision and Recall, providing a balanced metric that combines these two measures into a single value. This metric is particularly important when evaluating question-answering systems, especially when a balance between the "completeness" and "accuracy" of the response is required.
The calculation formula is as follows:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Where precision and recall are defined as:

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

- **Metric Significance**: In Question Answering (QA) systems, the F1 Score is crucial for evaluating **Exact Matches** between predicted answers and reference answers. This is especially critical for **Span-based QA** tasks, as the system must identify precise text segments while maintaining comprehensive coverage of the answer content. For MLA-ARC, a high F1 score implies that the retrieved memory nodes neither omit key information (High Recall) nor introduce irrelevant noise (High Precision).

2. BLEU-1

BLEU-1 provides a method for evaluating the 1-gram (word-level) matching precision between system outputs and reference texts. Although originally used for machine translation, it effectively measures the Literal Accuracy of generation in dialogue evaluation.

The calculation formula is:

$$BLEU - 1 = BP \cdot exp(\sum_{n=1}^{1} w_n \log p_n)$$

Where BP is the **Brevity Penalty**, used to prevent the model from cheating by generating overly short sentences:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c < r \end{cases}$$

p_n is the n-gram precision:

$$p_n = \frac{\sum_i \sum_k \min(h_{ik}, m_{ik})}{\sum_i \sum_k h_{ik}}$$

Here, c is the candidate length, r is the reference length, h_{ik} is the count of n-gram i in candidate k, and m_{ik} is its maximum count in the reference.

- **Metric Significance**: In QA tasks, BLEU-1 evaluates the **Lexical Precision** of generated answers. For generative QA systems, BLEU-1 provides a valuable reference dimension when the **Exact Match** standard is too strict.

3. ROUGE Metrics[16]

The ROUGE series of metrics focuses on recall, measuring whether the generated answer covers the key information of the reference answer. We primarily focus on ROUGE-L and ROUGE-2.

- ROUGE-L (Longest Common Subsequence):

  This metric measures the Longest Common Subsequence (LCS) between the generated text and the reference text.

  $$ROUGE-L = \frac{(1+\beta^2)R_l P_l}{R_l + \beta^2 P_l}$$

  Where:

  $$R_l = \frac{LCS(X,Y)}{|X|}, \quad P_l = \frac{LCS(X,Y)}{|Y|}$$

  Here, X is the reference text, and Y is the candidate text.

- ROUGE-2 (Bigram Overlap):

  This metric calculates the overlap of Bigrams between the generated text and the reference text.

  $$ROUGE-2 = \frac{\sum_{bigram \in ref} \min(Count_{ref}(bigram), Count_{cand}(bigram))}{\sum_{biaram \in ref} Count_{ref}(bigram)}$$

- **Metric Significance**: ROUGE-L and ROUGE-2 are particularly useful for evaluating the **Fluency** and **Coherence** of generated answers. ROUGE-L focuses on sentence-level structural matching, while ROUGE-2 focuses on local word order consistency.

4. METEOR[17]

METEOR computes a score based on aligned unigrams between the candidate and reference texts. A key feature is its consideration of Synonyms and Paraphrasing (stemming), making it align more closely with human intuitive judgment than BLEU.

$$METEOR = F_{mean} \cdot (1 - Penalty)$$

$$F_{mean} = \frac{10P \cdot R}{R + 9P}$$

$$Penalty = 0.5 \cdot (\frac{ch}{m})^3$$

Where P is precision, R is recall, ch is the number of chunks, and m is the number of matched unigrams.

- **Metric Significance**: METEOR is highly valuable for QA evaluation because it goes beyond exact matching to consider **Semantic Similarity**, making it very suitable for evaluating paraphrased answers that are phrased differently but semantically correct.

5. SBERT Similarity[14]

Given that traditional n-gram metrics cannot capture deep semantics, we introduce SBERT Similarity, using Sentence Embeddings to measure the semantic distance between two texts.

$$SBERT\_Similarity = cos(SBERT(x), SBERT(y))$$

$$cos(a, b) = \frac{a \cdot b}{||a||||b||}$$

Here, $SBERT(\cdot)$ represents the sentence embedding vector of the text.

- **Metric Significance**: SBERT Similarity is a key metric for evaluating the **Semantic Understanding** capability of a QA system. It can capture the similarity in meaning between the generated answer and the standard answer even when **Lexical Overlap** is low. For MLA-ARC, this validates whether the memory retrieval processed by the "Evolution Layer" truly understands the intent of the question.

A.3 Comparision Results

To comprehensively verify the robustness of MLA-ARC, in addition to the F1 and BLEU-1 metrics reported in the main text, this section supplements detailed comparison results for four higher-order semantic metrics: ROUGE-2, ROUGE-L, METEOR, and SBERT Similarity (corresponding to Main Text Tables 5, 6, 7). These metrics delineate system performance more granularly from dimensions such as semantic coverage, syntactic structure, and deep intent understanding.

Table 5: Experimental results on LoCoMo dataset of QA tasks across five categories (Single Hop, Multi Hop, Temporal, Open Domain, and Adversial) using different methods. Results are reported in ROUGE-2 and ROUGE-L scores, abbreviated to RGE-2 and RGE-L.

Model	Method	Single HopRGE-2	Single HopRGE-L	Multi HopRGE-2	Multi HopRGE-L	TemporalRGE-2	TemporalRGE-L	Open DomainRGE-2	Open DomainRGE-L	AdversarialRGE-2	AdversarialRGE-L
GPT-4o-mini	LoCoMo	9.88	24.52	2.06	**18.09**	3.49	11.29	27.12	41.2	**61.97**	**71.33**
	ReadAgent	2.53	9.69	0.92	13.45	0.54	5.91	3.06	10.17	6.83	10.03
	MemoryBank	1.21	5.57	0.53	9.88	0.95	5.63	1.68	6.8	4.66	7.53
	MemGPT	10.84	26.24	4.88	25.85	0.78	9.37	29.15	43.3	37.54	44.84
	MLA-ARC	**10.92**	**26.51**	**21.92**	**44.27**	**3.51**	**12.39**	**30.24**	**46.31**	43.69	51.29
GPT-4o	LoCoMo	11.82	31.42	1.72	8.37	3.29	16.74	**46.56**	**65.46**	**46.26**	**53.99**
	ReadAgent	4.01	14.72	0.42	4.06	0.53	8.79	4.87	13.75	4.35	6.98
	MemoryBank	1.89	7.54	0.37	2.35	2.18	7.02	3.1	9.58	1.25	4.52
	MemGPT	11.84	30.93	4.78	16.23	3.35	14.37	44.35	64.32	29.44	35.96
	MLA-ARC	**13.08**	**32.5**	**10.07**	**25.67**	**6.24**	**17.05**	34.51	51.57	31.07	37.25
Qwen2.5-1.5b	LoCoMo	1.42	9.47	0	**4.68**	3.51	10.85	3.33	11.43	35.98	44.7
	ReadAgent	0.76	7.32	0.1	**2.81**	3.13	12.95	1.51	8.08	21.25	28.52
	MemoryBank	1.55	11.46	0.14	5.52	1.85	8.65	5.2	14.06	29.97	37.87
	MemGPT	1.19	11.63	0	8.08	2.94	14.99	2.23	10.07	24.56	32.48
	MLA-ARC	**5**	**18.39**	**6.03**	**27.23**	**3.53**	**17.29**	**12.63**	**24.99**	**37.23**	**47.77**
Qwen2.5-3b	LoCoMo	0.5	4.95	0.14	3.28	1.34	5.51	2.02	7.15	12.98	17.53
	ReadAgent	0.08	4.18	0	2.01	1.29	6.34	0.75	4.45	7.53	10.91
	MemoryBank	0.44	3.85	0.05	1.65	0.25	6.48	1.06	4.33	9.79	13.75
	MemGPT	0.71	5.69	0.05	3.25	1.95	8.1	2.1	7.5	10.72	14.75
	MLA-ARC	**2.98**	**12.73**	**8.31**	**28.43**	**1.55**	**7.7**	**9.02**	**18.01**	**21.92**	**28.68**
Llama 3.2-1b	LoCoMo	2.57	11.77	0.45	8.46	1.73	13.39	3.01	13.33	40.85	54.06
	ReadAgent	0.54	6.65	0	4.74	5.61	14.65	1.22	8.23	35.38	46.69
	MemoryBank	3.03	13.91	0.24	10.79	4.11	18.84	6.57	18.1	42.18	54.64
	MemGPT	1.87	10.16	0.06	6.72	2.18	11.64	2.05	10.63	39.55	51.57
	MLA-ARC	**4.94**	**19.79**	**1.89**	**20.98**	**6.14**	**18.95**	**15.19**	**30.52**	**47.93**	**61.74**
Llama 3.2-3b	LoCoMo	1	7.4	0.03	4.56	2.42	11.67	2.92	8.66	26.11	31.02
	ReadAgent	2.53	1.82	3.09	3.09	5.2	5.35	3.33	2.57	16.17	14.36
	MemoryBank	1.88	7.13	0.26	3.5	0.44	4.54	2.8	8.03	15.01	19.05
	MemGPT	0.74	5.52	0.11	2.92	0.63	5.88	1.49	4.53	17.04	22.01
	MLA-ARC	**6.17**	**18.06**	**8.13**	**28.67**	**5.51**	**13.33**	**17.31**	**29.26**	**36.37**	**43.31**

1. Performance Leap in Semantic Metrics

As shown in **Table5** , MLA-ARC demonstrates significant advantages across all semantic similarity metrics. This advantage is particularly prominent on non-GPT open-source models (e.g., Qwen2.5, Llama 3.2).

- **Semantic Coherence in Multi-Hop Reasoning**: On the Qwen2.5-1.5B model, MLA-ARC's **SBERT Similarity** reached **46.60**, whereas ReadAgent under identical conditions achieved only 12.66, and the LoCoMo baseline was 36.32.
    - **Analysis**: This indicates that during complex cross-paragraph reasoning, the context retrieved by MLA-ARC aligns highly with the standard answer not only "literally" but also in the "**Semantic Field**." In contrast, ReadAgent often generates answers with ambiguous semantics due to excessive information compression, failing to precisely hit the core entities of the question.
- **Syntactic Reconstruction Capability on GPT-4o-mini**: On the GPT-4o-mini base, MLA-ARC's **METEOR** score reached **23.43**, far surpassing LoCoMo's 7.21 and MemoryBank's 2.29.
    - **Analysis**: The METEOR metric is highly sensitive to synonyms and sentence restructuring. A high score implies that the answers generated by MLA-ARC are not merely mechanical copies of the original text but demonstrate true understanding of the context and the ability to restructure information in a manner consistent with human linguistic habits. This is attributed to **Agent Layer 3 (Output Layer)** possessing structured and logically coherent Prompt inputs during answer synthesis.

Table 6: Experimental results on LoCoMo dataset of QA tasks across five categories (Single Hop, Multi Hop, Temporal, Open Domain, and Adversial) using different methods. Results are reported in METEOR and SBERT Similarity scores, abbreviated to ME and SBERT.

Model	Method	Single HopME	Single HopSBERT	Multi HopME	Multi HopSBERT	TemporalME	TemporalSBERT	Open DomainME	Open DomainSBERT	AdversarialME	AdversarialSBERT
GPT-4o-mini	LoCoMo	16.15	49.03	7.61	52.3	8.35	35.88	41.33	59.08	64.71	73.55
	ReadAgent	5.57	29.38	4.89	46.2	3.77	27.39	8.19	27.38	8.57	15.58
	MemoryBank	3.49	22.25	4.17	38.52	4.3	24.3	5.94	21.28	6.38	13.33
	MemGPT	16.14	50.44	13.58	62.91	4.69	33.59	42.33	59.5	40.04	48.42
	MLA-ARC	**16.73**	**50.57**	**23.43**	**70.49**	**8.55**	**39.44**	**43.27**	**60.86**	**46.78**	**54.59**
GPT-4o	LoCoMo	16.7	55.17	7.37	32.95	9.18	44.81	54.72	75.24	48.91	57.49
	ReadAgent	8.04	38.35	3.85	26.88	4.52	31.52	9.57	32.15	5.61	12.65
	MemoryBank	3.29	26.89	2.35	24.08	4.27	25.51	6.79	24.5	3	10.26
	MemGPT	17.06	56.5	13	36.83	7.95	38.86	53.44	74.65	31.93	40.06
	MLA-ARC	**17.97**	**57.36**	**13.43**	**46.54**	**10.89**	**39.84**	**42.98**	**64.03**	**33.15**	**41.11**
Qwen2.5-1.5b	LoCoMo	5.1	33.04	2.93	34.88	6.02	36.5	8.76	30.21	41.54	51.75
	ReadAgent	3.75	28.91	1.93	27.95	9.17	36.01	5.64	26.99	24.64	34.97
	MemoryBank	5.69	36.29	2.87	33.28	4.37	34.7	10.83	32.96	33.75	43.9
	MemGPT	5.52	36.53	2.41	40.02	7.85	41.37	7.23	30.91	27.92	41.65
	MLA-ARC	**9.73**	**44.58**	**12.22**	**63.19**	**9.34**	**43.64**	**20.18**	**42.98**	**41.66**	**53.75**
Qwen2.5-3b	LoCoMo	2.05	24.98	1.97	25.87	3.53	26.01	6.13	21.81	17.09	23.72
	ReadAgent	1.82	21.63	1.73	21.3	4.54	25.78	3.45	18.66	10.72	17.82
	MemoryBank	2.43	18.26	2.28	22.48	3.96	21.17	4.09	16.67	15.88	21.29
	MemGPT	3.83	24.92	2.31	28.36	6.6	30.33	6.4	22.96	13.52	21.35
	MLA-ARC	**6.41**	**34.56**	**14.39**	**64.1**	**6.72**	**31.37**	**16.38**	**34.83**	**28.04**	**34.56**
Llama 3.2-1b	LoCoMo	5.91	38.97	3.46	46.58	6.36	43.76	9.56	35.04	47.96	62.26
	ReadAgent	3.04	29.99	1.34	27.11	7.31	40.17	5.49	27.1	43.45	55.71
	MemoryBank	6.94	40.31	4.54	46.77	7.95	43.88	13.34	38.25	51.69	62.33
	MemGPT	5.23	33.81	2.6	42.86	3.34	36.89	6.79	31.45	46.13	62.86
	MLA-ARC	**9.24**	**46.29**	**7.69**	**56.16**	**8.51**	**44.51**	**23.02**	**48.25**	**55.06**	**69.7**
Llama 3.2-3b	LoCoMo	3.78	28.64	3.03	20.91	6.62	32.97	6.74	23.49	29.75	36.63
	ReadAgent	1.24	17.84	2.39	12.32	3.47	20.12	2.52	14.99	14.73	21.78
	MemoryBank	3.94	25.69	2.8	13.99	3.13	21.61	6.51	22.57	17.57	25
	MemGPT	2.85	22.61	2.27	15.34	3.72	23.76	3.56	18.26	21.01	27.54
	MLA-ARC	**9.98**	**40.3**	**13.52**	**61.19**	**8.29**	**33.08**	**24.91**	**43.93**	**40.73**	**47.93**

Table 7: Experimental results on LoCoMo dataset of QA tasks across five categories (Single Hop, Multi Hop, Temporal, Open Domain, and Adversial) using different methods.

Method	Single HopF1	Single HopBLEU-1	Multi HopF1	Multi HopBLEU-1	TemporalF1	TemporalBLEU-1	Open DomainF1	Open DomainBLEU-1	AdversarialF1	AdversarialBLEU-1
DeepSeek-R1-32B										
LoCoMo	8.79	6.64	4.91	4.46	13.28	12.83	10.99	8.41	21.94	20.74
MemGPT	8.49	6.41	5.59	5.09	11.24	9.32	11.62	9.26	**31.54**	**29.96**
MLA-ARC	**15.4**	**10.91**	**15.01**	**11.29**	**15.18**	**13.14**	**15.75**	**12.61**	28.62	27.87
Claude 3.0 Haiku										
LoCoMo	4.67	3.41	0.84	0.6	2.93	3.3	3.65	3.32	3.55	3.51
MemGPT	7.84	6.52	1.69	1.29	7.6	6.81	8.82	7.47	7.85	7.55
MLA-ARC	**19.76**	**15.06**	**17.07**	**12.54**	**12.15**	**9.85**	**35.59**	**30.8**	**36.89**	**35.74**
Claude 3.5 Haiku										
LoCoMo	11.62	8.42	3.37	2.76	3.88	3.67	14.36	12.88	7.55	7.3
MemGPT	8.48	6.71	4.09	2.83	4.83	4.59	16.93	15.26	5.78	5.59
MLA-ARC	**30.44**	**23.77**	**32.33**	**28.22**	**11.71**	**9.71**	**43.67**	**38.35**	**13.99**	**13.03**

2. Robustness Across Foundation Models and "Capability Elicitation"

We further tested various cutting-edge models, including DeepSeek-R1-32B, Claude 3.0 Haiku, and Claude 3.5 Haiku. The results reveal a clear "**Positive Feedback**" pattern:

- **Capability Elicitation Effect**: The stronger the reasoning capability of the foundation model, the more significant the **Performance Gain** provided by the structured memory of MLA-ARC. For instance, on DeepSeek-R1-32B, MLA-ARC's performance in Multi-Hop tasks improved by nearly **200%** compared to LoCoMo.
- **Explanation**: **Strong Reasoners** can more effectively utilize the complex graph structural relationships generated by Agent Layer 2. Simple baseline models may fail to utilize these implicit connections and instead be distracted by excessive information; conversely, strong models can follow the semantic paths laid by MLA-ARC to complete deep reasoning.

A.4 Memory Analysis

To intuitively demonstrate how MLA-ARC transforms unstructured dialogue streams into an ordered

knowledge network, we utilized **t-SNE** technology to perform dimensionality reduction visualization analysis on the embedding vectors of memory nodes. **Figure A.1** displays snapshots of memory distribution across different dialogue stages.

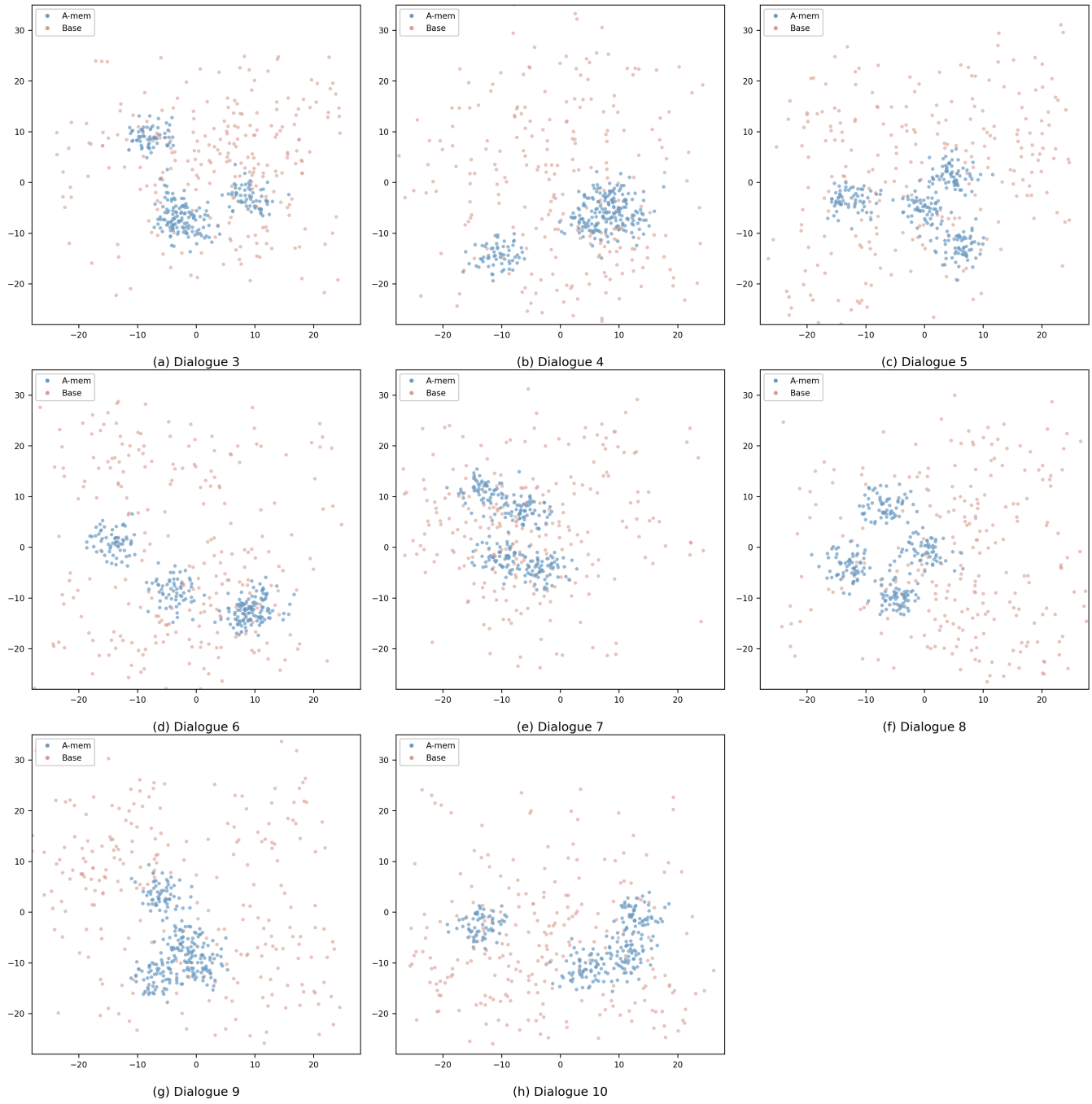


Figure A.1: Snapshots of memory distribution across different dialogue stages

By comparing MLA-ARC (blue dots) with the baseline system (red dots, i.e., the version without evolution mechanisms), we can observe distinct topological differences:

1. Self-Organizing Clusters

In the view of MLA-ARC, memory points are not randomly scattered but spontaneously form clearly defined Semantic Clusters.

- **Physical Significance**: These clusters correspond to the user's long-term core interests (e.g., "Python Projects," "Travel Planning," "Healthy Diet"). This proves that the label extraction mechanism of **Agent Layer 1 (Structure Layer)** and the linking mechanism of **Agent Layer 2 (Evolution Layer)** successfully "gravitationally aggregate" semantically related fragmented information.

2. Noise Suppression

In contrast, the memory distribution of the baseline system presents a Dispersed Distribution. This implies that during retrieval, the baseline system struggles to distinguish between "Core Facts" and "Chitchat Noise," resulting in retrieval results mixed with a large amount of irrelevant information.

3. Evidence of Dynamic Evolution

As the dialogue turns increase (from Dialogue 1 to Dialogue 10), the cluster structure of MLA-ARC becomes more Compact, and distinct Bridge Nodes appear between clusters. This intuitively validates that our "Memory Evolution Algorithm" is continuously refining the memory structure, chaining isolated knowledge points into a network.

A.5 Hyperparameters Setting

The retrieval parameter k (**Top-k Retrieval**) is a key hyperparameter balancing **Context Richness** and **Noise Interference**. Based on the ablation study analysis in Section A.3, we formulated a dynamic configuration strategy for different foundation models and task types, as shown in **Table A.2**.

- Small Parameter Model Strategy ($k = 10$):
  For models with smaller parameter sizes (<3B) (e.g., Qwen2.5-1.5B, Llama-3.2-1B), we strictly limit $k = 10$.
  - **Rationale**: The **Attention Mechanism** of small models is prone to "**Attention Dispersion**" when processing long sequences. Excessive context not only fails to aid reasoning but also leads to the model getting lost in irrelevant details.

- Strong Model Strategy ($k = 40 \sim 50$):
  For strong models like GPT-4o, we increase k to 40-50 in Open Domain and Temporal reasoning tasks.
  - **Rationale**: Strong models possess superior information filtering and noise resistance capabilities. In Open Domain tasks, a **Broader Context** can provide necessary background knowledge to activate the model's latent knowledge base; whereas in Temporal tasks, a

larger window helps capture long-span temporal dependencies.

Table 8: Selection of k values in retriever across specific categories and model choices

Model	Single Hop	Multi Hop	Temporal	Open Domain	Adversarial
GPT-4o-mini	40	40	50	50	40
GPT-4o	40	40	50	50	40
Qwen2.5-1.5b	10	10	10	10	10
Qwen2.5-3b	10	10	50	10	10
Llama3.2-1b	10	10	10	10	10
Llama3.2-3b	10	20	10	10	10

Appendix B.Prompt Template

Prompt for Cultivating Structure:

```
Generate a structured analysis of the following content by:
    1. Identifying the most salient keywords (focus on nouns, verbs, and key
concepts)
    2. Extracting core themes and contextual elements
    3. Creating relevant categorical tags

    Format the response as a JSON object:
    {
        "keywords": [
            // several specific, distinct keywords that capture key concepts
and terminology
            // Order from most to least important
            // Don't include keywords that are the name of the speaker or time
            // At least three keywords, but don't be too redundant.
        ],
        "context":
            // one sentence summarizing:
            // - Main topic/domain
            // - Key arguments/points
            // - Intended audience/purpose
        ,
        "tags": [
            // several broad categories/themes for classification
            // Include domain, format, and type tags
            // At least three tags, but don't be too redundant.
```

```
        ]
    }

    Content for analysis:
    {content}
```

Prompt for summarize user_profile:

```
Analyze the user's input and historical query. Does it contain explicit
personal information (name, job, preferences)?
Current Profile: {Current_Profile_JSON}
User Input: {User_Query}
Task: Return a JSON delta. If no new info, return empty. If conflict,
overwrite." Please be careful, the content "up_i" in the format need to be
changed as actual profile, like "up_i" -> "name".

### OUTPUT FORMAT ###
{
    "up_1":"",
    "up_2":"",
    "up_3":"",
    ...
}
```

Prompt for Attribute Extractor:

```
### SYSTEM ROLE ###
You are an expert **Memory Attribute Extractor** for an intelligent agent.
Your goal is to parse raw user interactions into structured memory nodes for
Long-Term Memory (LTM) storage.

### INPUT VARIABLES ###
- **Interaction ($I$):** The raw text content from the user interaction.
- **Timestamp ($t_s$):** The current system time.

### EXTRACTION RULES ($f_{attr}$ function) ###
You must analyze the Input $I$ and extract the following attributes. If the
input is meaningless noise, you must trigger the Discard Protocol.
```

```
1. **Context ($c_{xt}$)**:
   - Describe the background situation: "Who" is speaking, "Where" (if
applicable), and the implied "Intent".
   - Disambiguate pronouns. Example: If $I$ is "It is red", and context
implies a car, $c_{xt}$ should be "User describing their Ferrari model".

2. **Content ($c_{nt}$)**:
   - Extract the **core factual statement**.
   - **CRITICAL**: Remove all phatic expressions (e.g., "Hello", "Thanks",
"Haha", "OK"), stop words, and redundant politeness.
   - Keep only information with long-term value.

3. **Keywords ($k_w$)**:
   - Extract 3-5 high-value entities, proper nouns, or technical terms (e.g.,
"Python", "Gundam", "Visa Application").
   - These will be used for inverted indexing.

### DISCARD PROTOCOL (Noise Filtering) ###
If the Interaction $I$ consists **ONLY** of:
- Phatic expressions (e.g., "Hi", "Thank you")
- Meaningless interjections (e.g., "Haha", "Wow")
- Short confirmations (e.g., "OK", "Received")
Then the input is considered **Noise**. In this case, your entire output must
be exactly the string: `NULL`.

### OUTPUT FORMAT ###
Return **ONLY** a valid JSON object (no markdown, no explanations).

**Schema:**
{
  "c_xt": "String describing the context",
  "c_nt": "String summarizing the core content",
  "k_w": ["Keyword1", "Keyword2", ...],
  "t_s": "String (passed from input)"
}

### EXAMPLES ###

**Input:**
I: "No, I specifically mean the MG scale version of the Freedom Gundam, not
the HG one."
t_s: "2024-05-20 10:00:00"
```

```
**Output:**
{
  "c_xt": "User clarifying specific preference for Gunpla model grades",
  "c_nt": "User prefers MG (Master Grade) Freedom Gundam over HG (High
Grade).",
  "k_w": ["Gunpla", "MG", "Freedom Gundam", "HG"],
  "t_s": "2024-05-20 10:00:00"
}

**Input:**
I: "Haha, okay, thanks!"
t_s: "2024-05-20 10:05:00"

**Output:**
NULL
```

Prompt for Evolution Structure:

```
1. **Analyze Relationship**: Compare the New Memory with Retrieved Memories.
Are they Supporting, Contradicting, or Unrelated?
2. **Determine Action**: Choose one of [INSERT, UPDATE, MERGE, IGNORE].
3. **Execute**:
   * If UPDATE: Rewrite the old memory content to reflect the change.
   * If MERGE: Synthesize a single comprehensive statement.
   * If INSERT: Return the new memory as is.
```

Prompt for Memory Evolution:

```
You are an AI memory evolution agent responsible for managing and evolving a
knowledge base.
Analyze the the new memory note according to keywords and context, also with
their several nearest neighbors memory.
Make decisions about its evolution.

The new memory context:
{context}
content: {content}
```

```
keywords: {keywords}

The nearest neighbors memories (each line starts with memory_id):
{nearest_neighbors_memories}

Based on this information, determine:
1. Should this memory be evolved? Consider its relationships with other
memories.
2. What specific actions should be taken (strengthen, update_neighbor)?
   2.1 If choose to strengthen the connection, which memory should it be
connected to? Use the memory_id from the neighbors above. Can you give the
updated tags of this memory?
   2.2 If choose to update_neighbor, you can update the context and tags of
these memories based on the understanding of these memories. If the context
and the tags are not updated, the new context and tags should be the same as
the original ones. Generate the new context and tags in the sequential order
of the input neighbors.
Tags should be determined by the content of these characteristic of these
memories, which can be used to retrieve them later and categorize them.
Note that the length of new_tags_neighborhood must equal the number of input
neighbors, and the length of new_context_neighborhood must equal the number of
input neighbors.
The number of neighbors is {neighbor_number}.
Return your decision in JSON format with the following structure:
{{
    "should_evolve": True or False,
    "actions": ["strengthen", "update_neighbor"],
    "suggested_connections": ["memory_id_1", "memory_id_2", ...],
    "tags_to_update": ["tag_1",..."tag_n"],
    "new_context_neighborhood": ["new context",...,"new context"],
    "new_tags_neighborhood": [["tag_1",...,"tag_n"],...["tag_1",...,"tag_n"]],
}}
```

Prompt for Memory Consolidation

```
"You are a Memory Archivist. The following memory fragments are about to be
deleted to save space.
Fragments: $\{m_1, m_2, ..., m_j\}$
Task: Merge these fragments into a single, cohesive 'Archived Memory'.
Rules:
1. Preserve distinct facts (names, dates).
```

```
2. Generalize repetitive behaviors (e.g., 'User asked about weather 5 times'
-> 'User frequently checks weather').
3. Discard trivial information without context."
```