

Memory Design in LLM-based Multi-Agent Agenda Assistants: A Comparative Study

Elia Capiluppi¹, Angelo Ferrando^{1,*}, Vadim Malvone² and Vladana Perlic^{2,3}

¹Department of Physics, Informatics and Mathematics, University of Modena and Reggio Emilia, Italy

²Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France

³STMicroelectronics, Grenoble, France

Abstract

Long-term memory is one of the main design challenges in LLM-based multi-agent systems, especially when agents must manage information that evolves during a long interaction. This paper studies an agenda-management assistant implemented with graph-based orchestration and compares three memory strategies within the same application setting: short-term conversational context with periodic summarisation, a structured database accessed through LLM-controlled tools, and an external semantic memory based on Mem0. The comparison is performed on a benchmark of simulated user interactions evaluated after 5, 10, 15, 20, and 30 turns across four dimensions: factuality, semantic understanding, time awareness, and update/forgetting. The results show that no single memory design dominates all requirements. Context-only memory is simple and cheap, but degrades as the dialogue grows. The structured database is the most reliable choice for temporal information such as appointments and time slots, but struggles when user information must be retrieved semantically. Mem0 offers the best behaviour for semantic retrieval and memory updates, yet it is weaker when exact temporal structure must be preserved. These results suggest that agenda assistants benefit from hybrid memory designs combining structured temporal storage with more flexible semantic memory.

Keywords

LLM-based multi-agent systems, agent memory, long-term memory, agenda assistant, Mem0

1. Introduction

In recent years, agent-based systems have increasingly relied on large language models (LLMs), not only for text generation but also for deciding when to retrieve information, invoke tools, or modify external state. In real-world settings, however, generating responses alone is not sufficient. When an assistant interacts with a user over time, it must be able to retain relevant information, such as preferences, past conversations, and personal details, while also determining when such information should be updated or forgotten. As a result, memory emerges as a central challenge in LLM-based multi-agent systems [1, 2, 3].

To examine this issue more concretely, we consider the case of an assistant for agenda management. Although such a task may appear straightforward, it in fact combines several distinct categories of requirements. In this work, we compare three different approaches to memory management. The first keeps all information within the conversation context, periodically summarising older interactions to maintain tractability. The second relies on a structured database, where appointments and notes are stored and retrieved through external tools. The third adopts Mem0 [4], which functions as a semantic memory layer, enabling more flexible storage and retrieval of information [5].

Rather than arguing that a single approach is universally superior, our objective is more pragmatic: to understand which type of memory is most effective under different conditions. An agenda-management assistant must balance natural interaction, reliable long-term memory, and accurate temporal reasoning.

WOA 2026: 27th Workshop "From Objects to Agents", June 15–17, 2026, Salerno, Italy

*Corresponding author.

✉ 302710@studenti.unimore.it (E. Capiluppi); angelo.ferrando@unimore.it (A. Ferrando); vadim.malvone@telecom-paris.fr (V. Malvone); vladana.perlic@st.com (V. Perlic)

id 0000-0002-8711-4670 (A. Ferrando); 0000-0001-6138-4229 (V. Malvone); 0009-0002-6972-9062 (V. Perlic)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

This motivates our main research question: *how do different memory strategies affect both the quality and the cost of an agenda-management assistant?*

This paper makes three main contributions. First, it presents a direct comparison of three memory architectures within a common application setting, making their differences easier to identify. Second, it evaluates these architectures along four practically relevant dimensions: factual recall, semantic understanding, temporal reasoning, and the ability to update or forget information. Finally, it highlights a broader implication of the results: memory should not be treated as a single interchangeable component. Different forms of memory support different kinds of reasoning, and system design should reflect this distinction.

2. Background and problem framing

2.1. LLM-based multi-agent systems and tools

In this paper, we analyse assistants through the lens of LLM-based multi-agent systems (LaMAS). Rather than relying on a single monolithic assistant to perform all tasks, these systems are composed of multiple components, each driven by an LLM and responsible for a more specific function. This decomposition generally makes the overall system easier to manage, *i.e.*, it reduces complexity, keeps individual model calls more focused, and enables explicit coordination between reasoning, retrieval, and action.

A key element of this architecture is the use of *tools*. In this context, a tool can be understood as a callable function, with the important distinction that the LLM decides when and how to use it. The model is provided with descriptions of the available tools, together with guidance on when they may be appropriate, and it then determines whether to invoke them, which arguments to supply, and how to incorporate the returned results. This is what allows the assistant to go beyond text generation and interact with structured data or external services.

The system considered in this work is implemented as a graph-based workflow. Nodes correspond to Python functions, while edges determine how control flows between them. Within this framework, a shared *state* acts as a form of short-term execution memory, carrying messages and other information across nodes. Any information that must persist beyond a single interaction, however, must be stored separately. This distinction between temporary execution state and persistent memory is central to the discussion that follows.

2.2. Working memory, long-term memory, and RAG

It is useful to distinguish between working memory and long-term memory in these systems. Working memory refers to the current conversation, *i.e.*, all information that remains within the prompt. This is effective for short interactions, since the relevant context is immediately available to the model. As the conversation grows, however, its limitations become apparent. More messages imply more tokens, higher cost, and eventually some form of compression or loss of detail.

Long-term memory serves a different function. It stores information across interactions and makes it available later through retrieval or explicit queries. Depending on the system design, this memory may be private to a single agent or shared across multiple agents, but its defining property is persistence over time.

It is also important to distinguish long-term memory from retrieval-augmented generation (RAG). RAG is typically used when the assistant must retrieve information from a relatively stable external knowledge base. Memory, by contrast, is required when the system must continuously learn from the user¹. This distinction is especially important in the agenda-management setting, where the assistant is not merely querying static data, but maintaining an evolving representation of a single user’s schedule and context.

¹Storing new information, updating outdated facts, and forgetting what is no longer relevant [6].

2.3. Token-level memory and memory evolution

In this work, we focus on what has been termed *token-level memory*, meaning that memory is represented as explicit units that can be accessed, modified, and reorganised over time. Within this perspective, memory can be structured in several different ways [6].

The simplest form is flat memory, in which individual memory items are stored without explicit relations between them. This makes storage and maintenance relatively straightforward, but it also makes semantic retrieval more difficult, since connections among pieces of information are not directly represented. Planar memory introduces additional structure, typically through graph-based representations, allowing relationships between entities to be encoded explicitly. Hierarchical memory extends this idea further by organising information across multiple levels of abstraction, thereby supporting more complex forms of reasoning.

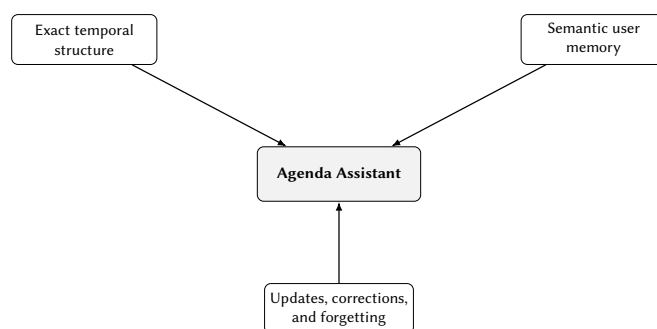
Equally important is the way memory evolves over time. Three operations are particularly relevant: *consolidation*, in which new information is integrated with what is already stored; *update*, in which existing knowledge is revised when changes occur or conflicts emerge; and *forgetting*, in which outdated or irrelevant information is removed or deactivated. Forgetting itself may be driven by different criteria, such as time, frequency of use, or perceived importance. In the case of an agenda assistant, these processes are highly concrete: the system must add new events, correct previous inaccuracies, and ensure that obsolete commitments do not remain treated as current.

3. Agenda-assistant case study

3.1. Task and memory requirements

The case study considers a conversational assistant designed to help a user manage an agenda. The assistant must support several types of interaction, including adding appointments, retrieving upcoming commitments, checking time availability, remembering personal notes that may influence future exchanges, and handling corrections to previously stated information.

Although the application appears simple, it combines several distinct memory requirements. Figure 1 summarises the three main pressures imposed by this setting. First, the assistant must preserve precise temporal structure for agenda items, including dates, times, and relationships between events. Second, it must retrieve more diffuse personal information, such as user habits, preferences, or contextual notes, even when subsequent queries do not closely match the original wording. Third, it must handle change: a user may reschedule an appointment, correct earlier statements, or introduce contradictory information. These requirements motivate the comparison of different memory substrates, as they highlight complementary demands that are not equally well supported by a single memory design.



Memory requirements in the agenda-assistant scenario.

Figure 1: Main memory requirements emerging in the agenda-assistant case study.

3.2. Common implementation substrate

All three systems share the same high-level orchestration strategy. They are implemented as graph-based workflows in which current messages and auxiliary control variables are propagated through a shared state. Models may invoke tools when needed, and the overall execution flow is governed by conditional transitions between nodes.

Figure 2 summarises this shared implementation substrate. The three systems are not independent assistants built from unrelated design choices; rather, they represent three variants of the same underlying application architecture. They share identical workflow logic and interaction patterns, differing primarily in how long-term information is represented and accessed. This ensures that the comparison remains focused on memory design rather than on broader differences in orchestration or user interaction.

The three variants are:

- a **context-only** assistant that relies on the current prompt, complemented by periodic summaries of older interactions;
- a **structured-database** assistant that uses explicit tables and tool-mediated CRUD operations;
- a **Mem0-based** assistant that retrieves and updates semantic memories extracted from conversational windows.

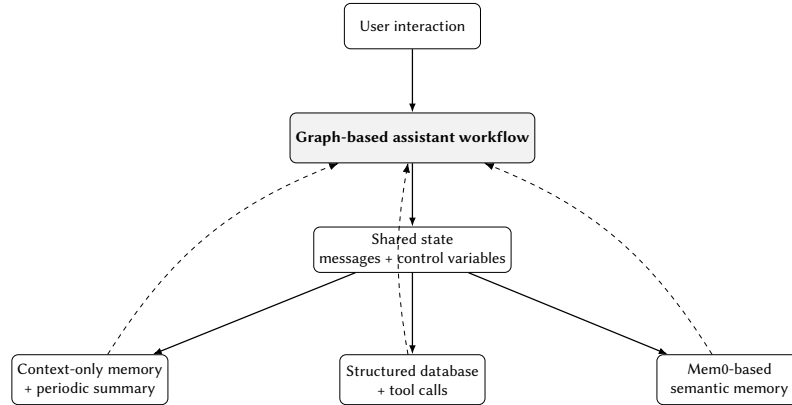


Figure 2: Shared orchestration substrate of the three assistant variants. The workflow and state management remain the same, while the long-term memory layer changes.

3.3. Operational design choices

Beyond the shared orchestration substrate, the three prototypes differ in a set of concrete operational choices that are important for interpreting the results. All variants are implemented in LangGraph and follow the same general execution pattern: a shared state carries the current dialogue and architecture-specific auxiliary information between nodes, while conditional edges determine whether the workflow continues, accesses external memory, or performs maintenance operations.

In the context-only variant, the state maintains both recent messages and a rolling summary of earlier interactions. The summarisation node is not executed at every turn, but only when the message history becomes sufficiently long to justify compression. In the implementation used for this case study, summarisation is triggered periodically, extending the summary while retaining only the most recent messages explicitly in context. This design keeps the workflow simple and limits token growth, but compression is irreversible: any detail not preserved in the summary is no longer directly accessible in subsequent turns.

In the structured-database variant, the workflow is divided into two distinct phases. An *info_user* node handles personal notes and preferences, while the chatbot node manages the agenda-related dialogue. The *info_user* node produces a structured output rather than free-form text, including search

Table 1

Operational choices of the three prototypes.

Variant	Retrieval and update strategy	Main operational consequence
Context only	Recent messages are kept in the prompt, while older ones are periodically summarised into a rolling textual memory	Minimal infrastructure and low complexity, but progressive compression of earlier detail
Structured DB	User notes are handled through a dedicated <i>info_user</i> step; agenda operations are delegated to validated tools over explicit tables	Strong control over appointments and safer execution, but more LLM-mediated steps and weaker semantic access to notes
Mem0	Relevant memories are retrieved from the latest user message; recent interaction windows are periodically committed to Mem0	Improved semantic recall and better handling of contradictions, but delayed availability of new memories and weaker temporal structure

keywords, optional memory to be stored, and associated indexing terms. The chatbot node then receives retrieved notes together with additional contextual information, such as the current date, time, and scheduled agenda. This design reduces ambiguity in memory access and clearly separates the two roles of the database: a structured calendar store and a lighter note repository.

In the Mem0-based variant, the workflow is simpler at the graph level because semantic memory management is delegated to an external framework. At each turn, the assistant queries Mem0 using the latest user message and injects the retrieved memories into the prompt. New information is not added after every individual message; instead, recent interaction windows are grouped and periodically committed to Mem0. This batching strategy reflects the fact that calendar information is often completed over multiple turns, for instance when a user first mentions an event and only later specifies its date or time. Updates are therefore applied to short interaction windows rather than isolated utterances. The implementation also relies on asynchronous updates and introduces a short delay before newly stored memories are used, which has an impact on observed latency.

Table 1 summarises the main implementation differences across the three variants.

All three prototypes were implemented in Python using LangGraph as the orchestration framework. The same OpenAI GPT nano model was used across the three variants, so that the comparison focused on the memory substrate rather than on differences between LLM backends. The model was used for response generation and, when required, for summarisation, tool selection, and structured-output generation.

In the context-only variant, the graph state stores the recent dialogue together with a rolling textual summary. The summarisation node is not called at every turn. Instead, it is triggered when the message history exceeds ten messages. When this happens, the summary is extended using the previous summary and the most recent dialogue, while only the two latest messages are retained explicitly in the state. In the implementation, the summary was constrained to a short textual representation in order to limit context growth and reduce token consumption.

In the structured-database variant, the external memory is implemented as a table-based store with two logical components: an appointments table and a user-notes table. The appointments table stores calendar entries through explicit fields such as title, date, start time, end time, and optional description. The user-notes table stores free-form contextual information together with keywords generated by the *info_user* node. The assistant does not generate database queries directly. Instead, it accesses the database through predefined tools for retrieval, insertion, update, deletion, and availability checking. These tools use schema-validated arguments, which constrains the operations available to the LLM and reduces the risk of malformed or unsafe database access.

In the Mem0-based variant, semantic memory management is delegated to the Mem0 MemoryClient. At each turn, the latest user message is used to retrieve potentially relevant memories, which are then injected into the assistant prompt. New information is committed to Mem0 periodically from

Table 2
Schema of the structured-memory prototype.

Table	Main fields	Role in the assistant
Appointments	title, date, time_start, time_end, description	Stores calendar events in an explicit temporal format, enabling reliable retrieval, update, deletion, and availability checks.
User notes	memory, keywords	Stores user preferences and contextual notes extracted from the dialogue. Retrieval is mainly keyword-based, which makes this component less effective for semantic queries.

recent interaction windows rather than from isolated messages. This choice reflects the fact that agenda-related information is often completed over multiple turns, for example when the user first mentions an appointment and only later specifies its date or time. Since Mem0 updates are performed asynchronously, the implementation also introduces a short delay after memory updates before newly committed memories are expected to be available for retrieval.

A further consequence of these design choices concerns memory lifecycle management. In a structured database, removing obsolete appointments or maintaining a limited time horizon is straightforward, as dates are explicit fields and maintenance can be performed through direct queries. In contrast, this operation is less immediate in the Mem0 setting. Two possible solutions are to assign an *expiration_date* when creating a memory or to introduce a dedicated cleaning routine that periodically identifies and removes stale entries. Both approaches increase system complexity and cost, and neither is as natural as operating on explicitly structured calendar records. This issue is particularly important for agenda assistants, where temporal data must not only be stored accurately but also evolve correctly over time.

4. Compared memory architectures

4.1. Context-only memory with summarisation

The simplest system does not employ any external long-term memory. Instead, it relies on the current conversation together with a summary of older messages. The workflow consists of two LLM-based components: a chatbot agent and a summariser. At regular intervals, the summariser compresses the oldest portion of the dialogue into a shorter textual representation, while the most recent messages remain explicitly available in the prompt.

This architecture is appealing due to its simplicity. It requires no external storage, no retrieval mechanism, and no schema design, making it the lightest baseline in implementation terms. In short interactions, it can perform well, since all relevant information is directly accessible to the model.

Its main limitation is the progressive loss of information. While summarisation reduces token consumption, it inevitably compresses detail. As the dialogue grows, precise facts and subtle distinctions may be lost. In the agenda domain, this affects not only general recall but also the assistant’s ability to answer temporally precise queries that depend on details introduced in earlier turns.

4.2. Structured database memory

The second system introduces explicit long-term memory through a structured database composed of two tables: one for appointments and one for user notes. The architecture includes an *info_user* phase for handling user-related notes and a chatbot phase responsible for managing calendar interactions through tool invocation.

More specifically, the structured memory is organised around two table-based stores. The first stores appointments, which are represented through explicit temporal fields. The second stores user-related notes, which are less rigidly structured and are indexed through keywords generated by the *info_user* node. Table 2 summarises the schema used in the prototype.

Table 3

Functional roles of the structured database in the case study.

Stored area	Main operations	Purpose in the assistant
Appointments table	Add, retrieve, update, delete, schedule checks	Handles explicitly structured temporal information such as events, free slots, and commitment queries
User notes table	Store and retrieve notes through the <i>info_user</i> phase	Preserves user-dependent contextual information, primarily through lexical rather than semantic access

A key design decision is that the LLM does not directly generate database queries. Instead, it interacts with predefined tools whose arguments are validated against structured schemas. The tool layer supports controlled operations such as adding, retrieving, updating, deleting, and checking scheduled commitments. This constrains how the model accesses and modifies stored data, reducing the likelihood of malformed operations and making the behaviour of the database-based assistant easier to inspect.

This approach is particularly well suited to the agenda domain, where appointments naturally conform to a structured representation (e.g., title, date, start time, end time, and optional description). By storing these fields explicitly, the assistant can reason about temporal relationships more reliably than when all information is embedded in natural language.

The main limitation concerns the handling of user notes. Unlike appointments, personal notes are heterogeneous and difficult to represent in a fixed schema. As a result, the notes table behaves largely as a lexical store. While exact or near-exact matches can be retrieved efficiently, the system struggles when semantic retrieval is required and queries do not closely match the original wording.

Table 3 summarises the functional roles of the two database components.

4.3. Mem0-based semantic memory

The third system replaces the structured database with Mem0, an external memory framework designed to extract, store, retrieve, and update salient conversational information [4]. In this setup, the assistant retrieves relevant memories from Mem0 when responding to a user query and periodically sends recent interaction windows for processing and integration into long-term memory.

Mem0 should not be interpreted simply as a vector database. It implements a memory-management pipeline in which conversational windows are first analysed to extract candidate memories. These candidate memories are then compared with semantically similar memories already stored in the system. Based on this comparison, the framework determines whether each candidate should be added as a new memory, used to update an existing one, used to remove contradictory information, or ignored when no modification is needed. This mechanism is particularly relevant for the agenda-assistant case study, because user information may evolve over time through corrections, rescheduling, or contradictory statements.

This design simplifies development in two main ways. First, it removes the need to define rigid schemas for different types of user information. Second, it delegates part of the complexity of memory extraction and updating to the external framework. This is particularly beneficial for personal notes and preferences, which are semantically rich but not easily captured in structured formats.

Mem0 also provides more advanced update behaviour compared to the structured database. During memory updates, newly extracted information is compared with semantically similar existing entries, allowing the system to decide whether to add, modify, delete, or ignore candidate memories [4]. This improves the system’s ability to handle contradictions and revisions.

However, this approach struggles with preserving exact temporal structure. Appointments are not merely semantic facts but structured entities defined by dates, times, and relationships. Free-form semantic representations may fail to retain such details precisely. For example, a memory such as “the user has the dentist on Tuesday” may not preserve the exact calendar date corresponding to that reference, which is critical for reliable agenda management.

Table 4

Compared memory strategies in the agenda-assistant case study.

Approach	Memory substrate	Main strength	Main weakness
Context only	Prompt context plus periodic summary	Simple architecture; low implementation overhead; strong performance in short interactions	Information is progressively compressed or lost as interactions grow
Structured DB	Appointment table plus notes table accessed via tools	Robust handling of structured temporal data	Limited semantic retrieval for user information stored as notes
Mem0	External semantic memory updated from interaction windows	Strong semantic retrieval and improved update/forgetting behaviour	Weaker preservation of precise temporal structure for appointments

4.4. Expected trade-offs

Before presenting the experimental results, it is useful to summarise the intended trade-offs among the three approaches. The context-only system prioritises simplicity and minimal infrastructure but risks information loss as interactions grow. The structured database sacrifices simplicity to achieve stronger control over temporal data. Mem0, in contrast, sacrifices strict structure to enable better semantic retrieval and more flexible memory evolution.

Table 4 summarises these trade-offs.

5. Evaluation

The three assistants are evaluated through a benchmark of simulated user interactions designed to stress their memory-related capabilities in a controlled manner. Each benchmark consists of 30 messages and includes five checkpoints, placed after 5, 10, 15, 20, and 30 turns. At each checkpoint, the assistant is asked a question aimed at verifying whether the relevant information is still handled correctly. The goal is not to construct a large-scale benchmark, but to characterise the behavioural profile of the three memory architectures under identical interaction conditions.

The evaluation focuses on four complementary aspects of memory behaviour: *factuality*, namely the accurate recall of explicitly stated information; *semantic understanding*, i.e., the ability to retrieve and use information beyond simple lexical overlap; *time awareness*, which concerns appointments, recurring commitments, free time slots, and more generally reasoning over dates and times; and *update and forgetting*, namely the ability to revise, replace, or discard stored information when subsequent interaction introduces corrections or contradictions. At each checkpoint, the assistant response is marked as either successful or unsuccessful. Although coarse, this binary scoring is sufficient for the purpose of this study, which is to compare the three approaches in a controlled case-study setting rather than to establish fine-grained benchmark rankings.

Table 5 summarises the role of each evaluation dimension.

Table 6 summarises the overall behaviour of the three memory strategies across the four evaluation dimensions, while Tables 7–10 report detailed checkpoint outcomes. Taken together, the results reveal a clear specialisation of the compared approaches: no single strategy dominates across all dimensions, and suitability depends strongly on the type of information being handled.

A first pattern highlights the contrast between the context-only baseline and the two external-memory solutions. The context-based assistant remains reasonably effective in shorter interactions and performs adequately up to intermediate checkpoints. However, its performance deteriorates as the dialogue grows. This degradation is particularly visible in factual recall and semantic understanding at the longest interaction length, where summarisation and context compression remove distinctions that

Table 5

Evaluation dimensions used in the agenda-assistant benchmark.

Dimension	What is stressed	Why it matters in the case study
Factuality	Retention of explicitly stated information	Long-term memory should preserve user-provided facts without ambiguity
Semantic understanding	Retrieval of relevant information not recoverable by lexical overlap	Preferences and contextual notes are often queried in semantically indirect ways
Time awareness	Management of dates, times, recurring events, and availability	Agenda assistants require precise temporal reasoning beyond generic recall
Update and forgetting	Revision or removal of obsolete or contradictory information	User data evolve over time and must not accumulate without correction

Table 6

Successful checkpoints out of five for each evaluation category.

Category	Context only	Structured DB	Mem0
Factuality	4/5	5/5	5/5
Semantic understanding	4/5	2/5	5/5
Time awareness	3/5	5/5	3/5
Update and forgetting	1/5	2/5	5/5
Total	12/20	14/20	18/20

were previously available. In this sense, context-only memory functions as a lightweight approximation, but does not provide a robust foundation for sustained interaction.

The comparison between the structured database and Mem0 is more nuanced, as the two approaches exhibit complementary strengths. The structured database is the most reliable system for time-aware behaviour. This is expected: appointments are inherently structured objects, and fields such as date, start time, and end time are more effectively handled when explicitly represented rather than embedded in free-form text. Accordingly, the database-based assistant maintains correct behaviour across all temporal checkpoints, whereas both the context-only and Mem0 systems degrade once temporal reasoning becomes more demanding.

Conversely, the structured database is weaker in semantic understanding and in update/forgetting. This indicates that structured storage alone is insufficient when retrieval depends on meaning rather than lexical overlap. User notes and preferences are heterogeneous and do not map cleanly to rigid schemas; when stored in largely lexical form, retrieval becomes brittle. Mem0 performs best in this regime. It consistently retrieves relevant information in the semantic benchmark and demonstrates strong performance in handling updates, contradictions, and revisions. This suggests that semantic memory is particularly valuable when managing evolving personal information rather than strictly structured data.

The detailed checkpoint tables further illustrate these trends. In factuality, both external-memory systems remain stable, whereas the context-only approach fails at the final checkpoint. In semantic understanding, Mem0 is the only system to maintain correctness throughout. In time awareness, the structured database dominates consistently. Finally, in update and forgetting, Mem0 clearly outperforms the alternatives, maintaining correct behaviour across all checkpoints.

The cost perspective complements these qualitative differences. Table 11 and Figure 3 show that the structured-database solution is significantly more expensive in terms of token consumption. This overhead arises from the architecture itself: each operation requires tool-oriented reasoning, structured prompting, and additional model calls. By contrast, the context-only and Mem0-based assistants exhibit similar token usage. From this perspective, the database approach trades computational efficiency for temporal precision.

These figures should, however, be interpreted with caution. The reported Mem0 token counts do

Table 7
Detailed checkpoint results for factuality.

Turn	Context only	Structured DB	Mem0
5	✓	✓	✓
10	✓	✓	✓
15	✓	✓	✓
20	✓	✓	✓
30	✗	✓	✓

Table 8
Detailed checkpoint results for semantic understanding.

Turn	Context only	Structured DB	Mem0
5	✓	✓	✓
10	✓	✗	✓
15	✓	✗	✓
20	✓	✓	✓
30	✗	✗	✓

Table 9
Detailed checkpoint results for time awareness.

Turn	Context only	Structured DB	Mem0
5	✓	✓	✓
10	✓	✓	✓
15	✓	✓	✓
20	✗	✓	✗
30	✗	✓	✗

Table 10
Detailed checkpoint results for update and forgetting.

Turn	Context only	Structured DB	Mem0
5	✗	✓	✓
10	✓	✓	✓
15	✗	✗	✓
20	✗	✗	✓
30	✗	✗	✓

Table 11
Token consumption across the four benchmarks.

Benchmark	Context only	Structured DB	Mem0
Factual	13,712	73,512	15,000
Temporal	15,700	82,000	14,000
Semantic	14,952	64,000	14,900
Update	15,700	55,000	14,600
Total	60,064	274,512	58,500

not include the internal models used by the framework itself, and therefore underestimate its full operational cost. Latency results in Figure 4 show a similar pattern. The context-only system is the most efficient overall, while both the structured database and Mem0 introduce additional delays. In the database case, this is mainly due to repeated tool invocations; in the Mem0 case, delays are also caused by the need to wait for memory updates to become available before retrieval. Consequently, efficiency and effectiveness do not align perfectly: the most accurate approach in one dimension is not necessarily the most efficient.

Overall, the experiments support a key conclusion: memory in LLM-based assistants should not be treated as a single interchangeable component. Different memory substrates support different forms of reasoning. Context-only memory is attractive due to its simplicity and low cost, but does not scale well to long interactions. Structured storage is highly effective for precise temporal data, but comparatively weak in semantic retrieval and dynamic revision. Mem0 provides strong support for semantic recall and memory evolution, but is less reliable when exact temporal structure must be preserved.

The evaluation is intentionally limited in scope, and each checkpoint is scored in a binary manner. As such, the study should be interpreted as a comparative case study rather than a definitive benchmark. Furthermore, the advantage of the structured database is partly tied to the agenda domain itself, where temporal information naturally fits structured representations. Different application contexts may shift the balance between the compared approaches.

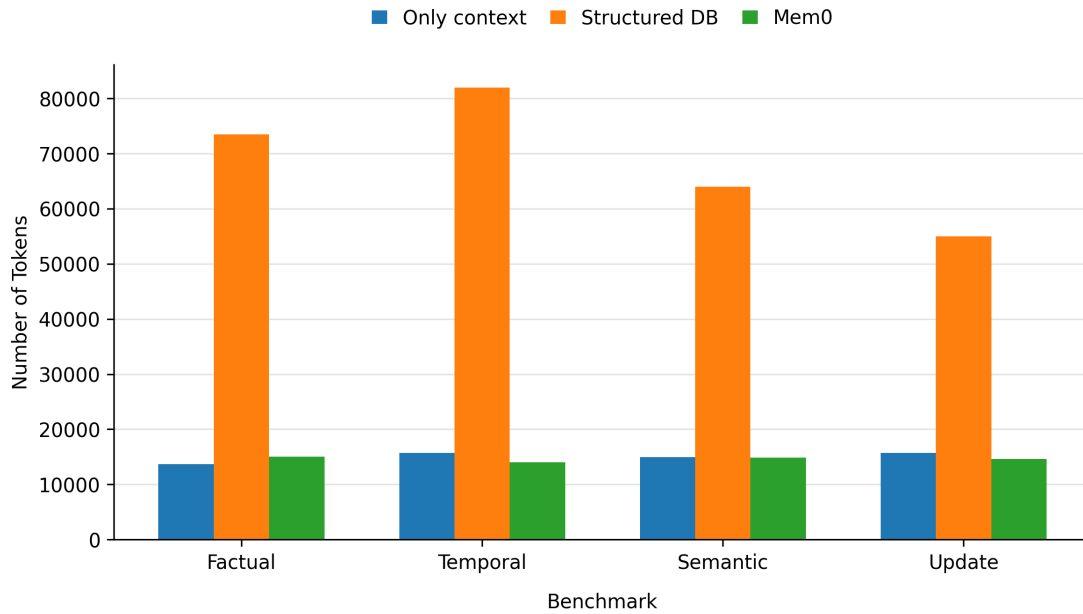


Figure 3: Token consumption per benchmark.

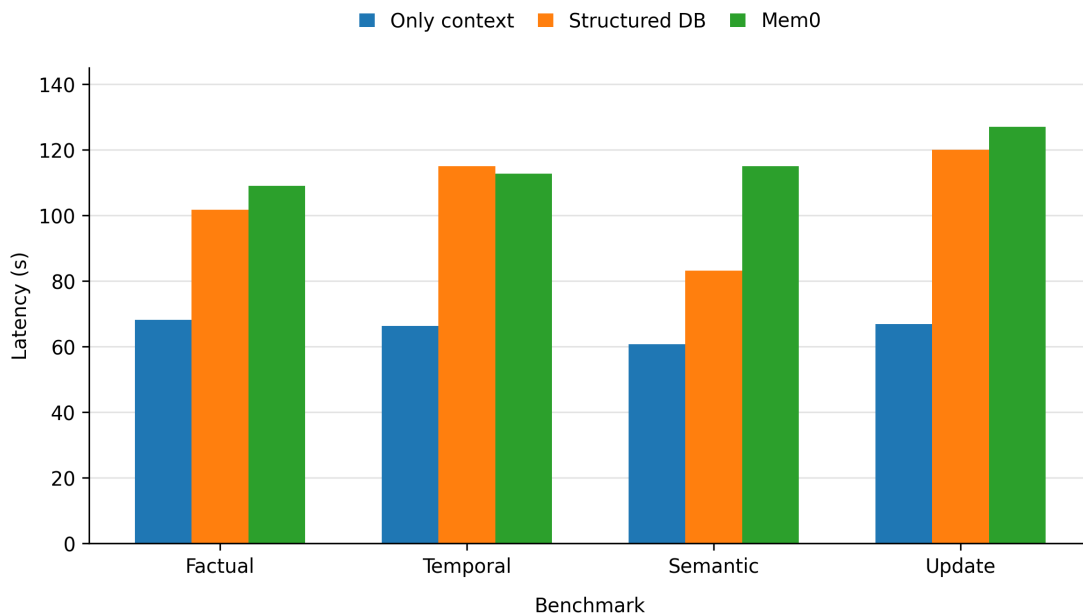


Figure 4: Latency comparison across benchmarks.

6. Practical implications for assistant design

The case study suggests that memory design in assistant systems should be guided by the structure of the underlying task rather than by a preference for any single framework. Appointments, deadlines, recurring commitments, and available time slots are naturally represented through explicit fields and therefore benefit from structured storage. Preferences, habits, and personal contextual notes are much less regular and are better suited to semantic memory. The comparison therefore supports decomposing memory by information type rather than adopting a single global storage solution.

A second implication concerns the relationship between working memory and long-term memory. Even when external memory is available, prompt-level context remains essential. In the structured-

database variant, the model benefits from having the current date, the current time, today’s agenda, and notes retrieved by the previous node already available when generating a reply. In the Mem0-based variant, local context is also necessary to bridge the interval between the production of new interaction and the point at which newly committed memories become reliably retrievable. Long-term memory, therefore, does not replace working memory; it complements it.

A third implication is that lifecycle management should be treated as a first-class design concern. Calendar assistants are not merely required to remember; they must remember at the appropriate granularity and for the appropriate duration. Structured databases make this comparatively easy, since obsolete events can be selected, updated, visualised, or deleted through explicit queries. In semantic memory systems, by contrast, achieving the same behaviour requires more application-specific logic. This is one reason why Mem0 performs well as a personal semantic memory layer but is more difficult to use as the sole representation of an agenda.

Taken together, these observations point toward a hybrid design. A practical assistant of the kind studied here would likely benefit from at least three complementary layers: a short-term conversational layer for immediate dialogue coherence, a structured calendar layer for appointments and other time-sensitive objects, and a semantic user-memory layer for preferences, habits, and contextual notes. Such an architecture would also align naturally with future interface requirements, since a calendar front-end can be rendered directly from structured events, while the assistant can still exploit semantic memory when the interaction depends on softer forms of user context. Although the full hybrid architecture is not implemented in this paper, the results provide a clear empirical motivation for it.

7. Related Work

Recent research on LLM-based agents has increasingly recognised that effective agent behaviour depends not only on reasoning and tool use, but also on mechanisms for storing, retrieving, and updating information across interactions. This has led to a growing body of work on long-term memory for LLM-based systems, aimed at overcoming the limitations of relying solely on the context window. Unlike short-term conversational context, long-term memory enables agents to retain salient facts, preferences, past events, and derived knowledge across sessions, supporting more consistent and personalised behaviour [7, 8].

A first line of work distinguishes between extending the effective context of a language model and equipping an agent with persistent memory. Architectures such as Transformer-XL improve long-range dependency modelling within the model’s processing pipeline, but do not provide an explicit, persistent, and updatable memory store across separate interactions [9]. More recent long-context models, such as LongT5, further extend this direction through efficient attention mechanisms that enable processing of longer input sequences [10]. However, these approaches still do not address core requirements of agent memory, such as persistence, update, and controlled forgetting across sessions.

A second line of work focuses on memory access as retrieval over an external store. Dense Passage Retrieval (DPR) demonstrates that semantically relevant information can be recovered from large collections using vector representations rather than sparse lexical matching [11]. This paradigm underlies many contemporary memory systems, in which past facts and events are embedded and retrieved on demand. Extensions such as Fusion-in-Decoder (FiD) show how retrieved information can be integrated within the generation process, effectively treating retrieval as an external memory mechanism rather than a standalone component [12].

More recent work moves beyond generic retrieval toward explicit long-term memory mechanisms for conversational agents. MemoryBank proposes a long-term memory architecture for sustained interaction, emphasising retrieval, updating, and forgetting-inspired dynamics to maintain user-relevant information over time [7]. Similarly, Reflective Memory Management introduces prospective and retrospective reflection to refine personalised memories across multiple granularities [8]. Complementary approaches, such as Mem0, conceptualise memory as a continuous pipeline of extraction, consolidation, and retrieval, highlighting the need to maintain coherence across long-running interactions rather than

treating memory as a passive store [4].

A key challenge concerns how memory evolves over time. Several works emphasise that long-term memory must support update and forgetting, not only accumulation. *Keep Me Updated!* addresses the problem of stale information by introducing mechanisms for selective memory invalidation [13], while *Long Time No See!* focuses on dynamic persona memory that is continuously updated during multi-session dialogue [14]. These contributions underline that maintaining accurate and up-to-date knowledge is as important as retrieving relevant past information.

Another important dimension concerns the structure of memory. While many systems rely on flat vector stores, recent work explores more structured representations. Timeline-based memory models capture the temporal evolution of user information [15], while graph-based representations encode relationships between entities and events to support richer retrieval [16]. Hierarchical approaches such as H-MEM organise memories across multiple levels of abstraction and use structured routing for efficient access, addressing scalability limitations of flat similarity search [4]. These works indicate that memory organisation is a central design choice rather than a mere implementation detail.

In parallel, evaluation of long-term memory has emerged as a distinct research problem. Benchmarks such as LoCoMo introduce long multi-session dialogues to assess persistence and consistency over time [17], while LOCCO studies temporal aspects such as memory decay and the effects of rehearsal [18]. MemBench proposes a multi-dimensional evaluation framework that considers effectiveness, efficiency, and capacity of memory systems [19]. These works demonstrate that memory quality cannot be fully captured by downstream task performance alone, but requires dedicated evaluation methodologies.

Finally, recent survey work has attempted to systematise this rapidly evolving design space. The ACM TOIS survey on memory mechanisms in LLM-based agents provides a taxonomy covering memory representation, storage, update policies, and evaluation strategies, offering a unifying perspective across otherwise heterogeneous approaches [20].

Overall, the literature suggests three main conclusions. First, enlarging the effective context of a model is not equivalent to equipping an agent with long-term memory [9, 7]. Second, retrieval-based memory improves scalability, but its effectiveness depends on how memories are represented, updated, and structured over time [11, 8, 15, 16]. Third, memory is not only a storage mechanism but a dynamic process involving consolidation, update, and forgetting, which becomes particularly important in long-running and multi-agent settings [13, 17, 19].

Against this background, the present paper compares alternative memory mechanisms for LLM-based multi-agent assistants and analyses their strengths and limitations in an agenda-management scenario.

8. Conclusions

This paper has investigated how different long-term memory strategies shape the behaviour of an LLM-based multi-agent agenda assistant. The comparison focused on three alternatives implemented within the same application setting: prompt-based conversational memory with periodic summarisation, a structured database accessed through tool calls, and an external semantic memory based on Mem0. By evaluating these architectures with respect to factual recall, semantic understanding, time awareness, and update/forgetting, the paper has shown that memory design is not a secondary implementation detail, but a central factor in determining the assistant’s overall capabilities.

The results reveal a clear specialisation among the three approaches. Context-only memory provides the simplest and lightest solution and remains effective for relatively short interactions, but its performance deteriorates as the dialogue grows and earlier information is compressed. The structured database is the most reliable option for exact temporal management, confirming that appointments and related calendar objects benefit from explicit structured representation. Mem0, by contrast, offers the strongest support for semantic retrieval and for handling corrections or evolving user information, but proves less dependable when the task requires precise temporal structure.

Taken together, these findings suggest that the key question is not which single memory mechanism is universally best, but which type of memory is best suited to which type of information. In agenda-

oriented assistants, temporal commitments, free time slots, and scheduling constraints are fundamentally different from preferences, habits, and contextual user notes. Treating them as if they could all be managed through a single storage model leads to avoidable weaknesses, either in semantic flexibility or in temporal precision.

The main design implication is therefore the need for hybrid memory architectures. A practical assistant should combine at least a short-term conversational layer, a structured temporal layer for agenda items, and a semantic memory layer for user-specific contextual information. Such an architecture would better reflect the heterogeneous nature of the assistant's knowledge and would support both robust scheduling behaviour and richer, personalised interaction.

This study remains intentionally limited in scope and should be read as a comparative case study rather than as a definitive benchmark of memory systems. Even so, it provides a clear empirical basis for future work. A natural next step is to implement and evaluate a fully hybrid agenda assistant that combines structured calendar storage with semantic user memory, together with explicit mechanisms for lifecycle management and controlled forgetting of obsolete events.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT to support the drafting and revision of the manuscript, including rephrasing, language polishing, and improving clarity and style. All generated content was reviewed and edited by the authors, who take full responsibility for the content of the publication.

References

- [1] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, J. Wen, A survey on large language model based autonomous agents, *Frontiers Comput. Sci.* 18 (2024) 186345. URL: <https://doi.org/10.1007/s11704-024-40231-1>. doi:10.1007/s11704-024-40231-1.
- [2] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, X. Zhang, Large language model based multi-agents: A survey of progress and challenges, in: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, ijcai.org, 2024, pp. 8048–8057. URL: <https://www.ijcai.org/proceedings/2024/890>.
- [3] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, R. Zheng, X. Fan, X. Wang, L. Xiong, Y. Zhou, W. Wang, C. Jiang, Y. Zou, X. Liu, Z. Yin, S. Dou, R. Weng, W. Qin, Y. Zheng, X. Qiu, X. Huang, Q. Zhang, T. Gui, The rise and potential of large language model based agents: a survey, *Sci. China Inf. Sci.* 68 (2025). URL: <https://doi.org/10.1007/s11432-024-4222-0>. doi:10.1007/s11432-024-4222-0.
- [4] P. Chhikara, D. Khant, S. Aryan, T. Singh, D. Yadav, Mem0: Building production-ready AI agents with scalable long-term memory, in: I. Lynce, N. Murano, M. Vallati, S. Villata, F. Chesani, M. Milano, A. Omicini, M. Dastani (Eds.), *ECAI 2025 - 28th European Conference on Artificial Intelligence, 25-30 October 2025, Bologna, Italy - Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025)*, *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2025, pp. 2993–3000. URL: <https://doi.org/10.3233/FAIA251160>. doi:10.3233/FAIA251160.
- [5] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, M. S. Bernstein, Generative agents: Interactive simulacra of human behavior, in: S. Follmer, J. Han, J. Steimle, N. H. Riche (Eds.), *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST 2023, San Francisco, CA, USA, 29 October 2023- 1 November 2023*, ACM, 2023, pp. 2:1–2:22. URL: <https://doi.org/10.1145/3586183.3606763>. doi:10.1145/3586183.3606763.
- [6] Y. Hu, S. Liu, Y. Yue, G. Zhang, B. Liu, F. Zhu, J. Lin, H. Guo, S. Dou, Z. Xi, S. Jin, J. Tan, Y. Yin, J. Liu, Z. Zhang, Z. Sun, Y. Zhu, H. Sun, B. Peng, Z. Cheng, X. Fan, J. Guo, X. Yu, Z. Zhou, Z. Hu, J. Huo, J. Wang, Y. Niu, Y. Wang, Z. Yin, X. Hu, Y. Liao, Q. Li, K. Wang, W. Zhou, Y. Liu, D. Cheng,

- Q. Zhang, T. Gui, S. Pan, Y. Zhang, P. Torr, Z. Dou, J. Wen, X. Huang, Y. Jiang, S. Yan, Memory in the age of AI agents, CoRR abs/2512.13564 (2025). URL: <https://doi.org/10.48550/arXiv.2512.13564>. doi:10.48550/ARXIV.2512.13564. arXiv:2512.13564.
- [7] W. Zhong, L. Guo, Q. Gao, H. Ye, Y. Wang, Memorybank: Enhancing large language models with long-term memory, in: M. J. Wooldridge, J. G. Dy, S. Natarajan (Eds.), Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada, AAAI Press, 2024, pp. 19724–19731. URL: <https://doi.org/10.1609/aaai.v38i17.29946>. doi:10.1609/AAAI.V38I17.29946.
 - [8] Z. Tan, J. Yan, I. Hsu, R. Han, Z. Wang, L. T. Le, Y. Song, Y. Chen, H. Palangi, G. Lee, A. R. Iyer, T. Chen, H. Liu, C. Lee, T. Pfister, In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents, in: W. Che, J. Nabende, E. Shutova, M. T. Pilehvar (Eds.), Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025, Association for Computational Linguistics, 2025, pp. 8416–8439. URL: <https://aclanthology.org/2025.acl-long.413/>.
 - [9] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, R. Salakhutdinov, Transformer-xl: Attentive language models beyond a fixed-length context, in: A. Korhonen, D. R. Traum, L. Màrquez (Eds.), Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers, Association for Computational Linguistics, 2019, pp. 2978–2988. URL: <https://doi.org/10.18653/v1/p19-1285>. doi:10.18653/V1/P19-1285.
 - [10] M. Guo, J. Ainslie, D. C. Uthus, S. Ontañón, J. Ni, Y. Sung, Y. Yang, Longt5: Efficient text-to-text transformer for long sequences, in: M. Carpuat, M. de Marneffe, I. V. M. Ruíz (Eds.), Findings of the Association for Computational Linguistics: NAACL 2022, Seattle, WA, United States, July 10-15, 2022, Findings of ACL, Association for Computational Linguistics, 2022, pp. 724–736. URL: <https://doi.org/10.18653/v1/2022.findings-naacl.55>. doi:10.18653/V1/2022.FINDINGS-NAACL.55.
 - [11] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, W. Yih, Dense passage retrieval for open-domain question answering, in: B. Webber, T. Cohn, Y. He, Y. Liu (Eds.), Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020, Association for Computational Linguistics, 2020, pp. 6769–6781. URL: <https://doi.org/10.18653/v1/2020.emnlp-main.550>. doi:10.18653/V1/2020.EMNLP-MAIN.550.
 - [12] G. Izacard, E. Grave, Leveraging passage retrieval with generative models for open domain question answering, in: P. Merlo, J. Tiedemann, R. Tsarfaty (Eds.), Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021, Association for Computational Linguistics, 2021, pp. 874–880. URL: <https://doi.org/10.18653/v1/2021.eacl-main.74>. doi:10.18653/V1/2021.EACL-MAIN.74.
 - [13] S. Bae, D. Kwak, S. Kang, M. Y. Lee, S. Kim, Y. Jeong, H. Kim, S. Lee, W. Park, N. Sung, Keep me updated! memory management in long-term conversations, in: Y. Goldberg, Z. Kozareva, Y. Zhang (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022, Findings of ACL, Association for Computational Linguistics, 2022, pp. 3769–3787. URL: <https://doi.org/10.18653/v1/2022.findings-emnlp.276>. doi:10.18653/V1/2022.FINDINGS-EMNLP.276.
 - [14] X. Xu, Z. Gou, W. Wu, Z. Niu, H. Wu, H. Wang, S. Wang, Long time no see! open-domain conversation with long-term persona memory, in: S. Muresan, P. Nakov, A. Villavicencio (Eds.), Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022, Findings of ACL, Association for Computational Linguistics, 2022, pp. 2639–2650. URL: <https://doi.org/10.18653/v1/2022.findings-acl.207>. doi:10.18653/V1/2022.FINDINGS-ACL.207.
 - [15] K. T. Ong, N. Kim, M. Gwak, H. Chae, T. Kwon, Y. Jo, S. Hwang, D. Lee, J. Yeo, Towards lifelong dialogue agents via timeline-based memory management, in: L. Chiruzzo, A. Ritter, L. Wang (Eds.), Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025, Association for Computational

Linguistics, 2025, pp. 8631–8661. URL: <https://doi.org/10.18653/v1/2025.naacl-long.435>. doi:10.18653/V1/2025.NAACL-LONG.435.

- [16] Y. Zhang, W. Yuan, Z. Jiang, Bridging intuitive associations and deliberate recall: Empowering LLM personal assistant with graph-structured long-term memory, in: W. Che, J. Nabende, E. Shutova, M. T. Pilehvar (Eds.), Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025, Findings of ACL, Association for Computational Linguistics, 2025, pp. 17533–17547. URL: <https://doi.org/10.18653/v1/2025.findings-acl.901>. doi:10.18653/V1/2025.FINDINGS-ACL.901.
- [17] A. Maharana, D. Lee, S. Tulyakov, M. Bansal, F. Barbieri, Y. Fang, Evaluating very long-term conversational memory of LLM agents, in: L. Ku, A. Martins, V. Srikumar (Eds.), Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024, Association for Computational Linguistics, 2024, pp. 13851–13870. URL: <https://doi.org/10.18653/v1/2024.acl-long.747>. doi:10.18653/V1/2024.ACL-LONG.747.
- [18] Z. Jia, Q. Liu, H. Li, Y. Chen, J. Liu, Evaluating the long-term memory of large language models, in: W. Che, J. Nabende, E. Shutova, M. T. Pilehvar (Eds.), Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025, Findings of ACL, Association for Computational Linguistics, 2025, pp. 19759–19777. URL: <https://aclanthology.org/2025.findings-acl.1014/>.
- [19] H. Tan, Z. Zhang, C. Ma, X. Chen, Q. Dai, Z. Dong, Membench: Towards more comprehensive evaluation on the memory of llm-based agents, in: W. Che, J. Nabende, E. Shutova, M. T. Pilehvar (Eds.), Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025, Findings of ACL, Association for Computational Linguistics, 2025, pp. 19336–19352. URL: <https://aclanthology.org/2025.findings-acl.989/>.
- [20] Z. Zhang, Q. Dai, X. Bo, C. Ma, R. Li, X. Chen, J. Zhu, Z. Dong, J. Wen, A survey on the memory mechanism of large language model-based agents, ACM Trans. Inf. Syst. 43 (2025) 155:1–155:47. URL: <https://doi.org/10.1145/3748302>. doi:10.1145/3748302.