EVALUATING MEMORY STRUCTURE IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern LLM-based agents and chat assistants rely on long-term memory frameworks to store reusable knowledge, recall user preferences, and augment reasoning. As researchers create more complex memory architectures, it becomes increasingly difficult to analyze their capabilities and guide future memory designs. Most long-term memory benchmarks focus on simple fact retention, multi-hop recall, and time-based changes. While undoubtedly important, these capabilities can often be achieved with simple retrieval-augmented LLMs and do not test complex memory hierarchies. To bridge this gap, we propose StructMemEval — a benchmark that tests the agent's ability to organize its long-term memory, not just factual recall. We gather a suite of tasks that humans solve by organizing their knowledge in a specific structure: transaction ledgers, to-do lists, trees and others. Our initial experiments show that simple retrieval-augmented LLMs struggle with these tasks, whereas memory agents can reliably solve them if prompted how to organize their memory. However, we also find that modern LLMs do not always recognize the memory structure when not prompted to do so. This highlights an important direction for future improvements in both LLM training and memory frameworks.

1 INTRODUCTION

As Large Language Models (LLMs) keep improving, we trust them with increasingly longer and more difficult tasks (Kwa et al., 2025). Frontier models assist users over long stretches of time (Xu et al., 2022; Jang et al., 2023; Wang et al., 2025a), work with large documents (Packer et al., 2023; Chevalier et al., 2023), perform long-form reasoning (ARC Prize Foundation, 2024; Phan et al., 2025) and agentic tasks (Erdogan et al., 2025; Merrill et al., 2026). To solve these tasks, a model needs to process vast amounts of information, both input sources and its own workings. As LLMs deal with harder problems, it becomes infeasible to keep all that information in their working memory.

Recent works circumvent this by equipping the LLM with long-term memory: allowing the agent to offload its knowledge from working memory to an external database and retrieve it only when necessary (Packer et al., 2023; OpenAI, 2024). Memory designs vary between methods: from simple look-up dictionaries (Packer et al., 2023; Wang et al., 2024b) to complex frameworks (Chhikara et al., 2025; Rasmussen et al., 2025) and entire operating systems (Li et al., 2025; Kang et al., 2025).

To compare between memory architectures, researchers and practitioners evaluate them on specialized benchmarks such as LOCOMO (Maharana et al., 2024) and LongMemEval (Wu et al., 2024). These benchmarks typically let the LLM process a scenario (e.g. personal assistant, corporate, agent) and form memories, then ask questions to test its recall: single-hop, multi-hop, time-based queries, and others. However, recent works have shown that LLMs do not necessarily need complex memory architecture to answer these kinds of questions (Zhou & Han, 2025; Ai et al., 2026). In one such work, Zhou & Han (2025) demonstrate that a simple retrieval system (EMem) can outperform sophisticated memory architectures on both LOCOMO and LongMemEval. This leads to a question: what kinds of problems *do* require complex memory hierarchies? What capabilities do they add?

In this work, we focus on one answer: that *memory allows LLM agents to organize their knowledge*. Instead of retrieving messages as-is, memory-augmented LLMs can extract knowledge and assemble it into a structure that best fits their task, such as arranging knowledge into graphs, maintaining todo lists, sorting events by category, calculating running statistics, and more. If a problem requires knowledge organization, simple message retrieval will not be able to solve it past certain size, but a more flexible agentic memory may succeed if the controlling LLM is capable enough.

To test this hypothesis, we gather a suite of tasks that a "notepad-augmented human" would solve by maintaining ledgers, drawing graphs, tracking states and similar. We find that even simple tasks of this kind prove difficult for retrieval-augmented agents, but proper memory-augmented agents fare significantly better. When we explicitly prompt LLM agents to organize their memory, they can reliably solve tasks on a scale that retrieval-augmented LLMs struggle

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

with, but perform less reliably without "hints". Informally, LLMs that can solve abstract algorithmic tasks often fail to recognize the same structure patterns in the wild. Our main contributions can be summarized as follows:

1. We gather StructMemEval, a suite of memory evaluation scenarios that focus on knowledge organization: tracking financial settlements, reasoning about family trees and organizations, tracking changing states from indirect observations, analyzing user preferences from events.
2. We analyze the effectiveness of different retrieval and memory-based LLM agents on memory organization tasks. Our findings suggest that retrieval-only systems fail after a certain problem size, whereas memory agents can maintain effectiveness longer, depending on the model.
3. We systematically analyze the failure modes of memory agents on StructMemEval tasks and report main failure modes: incorrect structures, memory hallucinations, record duplication, and others.
4. We release the main set of 51 scenarios and $\geq$200 additional tasks for analysis. Our scenarios decouple memory organization from other LLM capabilities to help inform future memory designs.

2 BACKGROUND

Transformer Working Memory. Transformer LLMs normally store task-specific information in the form of a Key-Value (KV) cache. This cache contains token-level vector representations used by attention layers on every inference step (Vaswani et al., 2017). For modern LLMs, this cache is limited to few $\times 10^{4-5}$ tokens with a few extreme cases going over a million tokens (Glm et al., 2024; Yang et al., 2025). While this limit can be extended (Peng et al., 2024; Li et al., 2023; Pekelis et al., 2024), a large KV cache fills accelerator memory[1] and slows down inference.

If the LLM needs to process more information than its KV cache allows, there are two main strategies for achieving this: compressing the KV cache or using external long-term memory. The former includes KV quantization (Liu et al., 2024; Hooper et al., 2024; Shutova et al., 2025), pruning (Zhang et al., 2023b; Xiao et al., 2024), offloading techniques (Lee et al., 2024) and other techniques that reduce the overhead from having a large KV cache. The latter bypasses the need for large KV cache by allowing the LLM to offload its knowledge to an external database and retrieve it later.

Memory-Augmented LLMs. Even before the LLMs, NLP researchers have allowed their models to retrieve relevant knowledge from corpora to give it access to additional knowledge and improve factuality (Guu et al., 2020; Karpukhin et al., 2020; Lewis et al., 2020). However, not all problems can be solved with a static database: for instance, when dealing with user preferences or working memory, the model must be able to update its knowledge to maintain relevant information (Wu et al., 2022). To address this, Packer et al. (2023) propose MemGPT — an external memory system that equips LLM agents with tools to create and update memory entries for later retrieval.

Since then, multiple lines of work propose extensions to this idea using knowledge graphs (Rasmussen et al., 2025; Chhikara et al., 2025; Wang et al., 2024a), hierarchy (Sun & Zeng, 2025; Kirmayr et al., 2025; Zhang et al., 2026b; Yu et al., 2026) insights from cognitive science (Nan et al., 2025), note taking (Xu et al., 2026; Tekparmak & Kaya, 2025), and others (Gao et al., 2025; Fang et al., 2026; MemMachine Contributors, 2025; Bhardwaj, 2026; Zhang et al., 2026d; Zhao et al., 2026). Aside from storing user preferences, agentic memory is also used for knowledge updates (Wang et al., 2024b; 2025b), iterative "evolutionary" problem solving (Agrawal et al., 2026; Zhang et al., 2026c), programming (Gandhi et al., 2025; Shen & Joung, 2026), embodied and virtual agents (Glocker et al., 2025; Lei et al., 2025; Wang et al., 2025c; Qi et al., 2025). Recent works propose even more complex memory frameworks and operating systems (Li et al., 2025; Kang et al., 2025) or fine-tuning the LLM to a specific memory type (Modarressi et al., 2024; Yan et al., 2026; Zhang et al., 2026a).

Benchmarking Long-term Memory frameworks is an inherently difficult task. Two of the most popular benchmarks for memory agents, LOCOMO (Maharana et al., 2024) and LongMemEval (Wu et al., 2024), focus on a chat assistant use case where the agent is expected to recall individual facts, do multi-hop look-ups and update its knowledge over time as the context changes. Other benchmarking scenarios are based on interactive conversations with user feedback (Kim et al., 2024; Wan & Ma, 2025; Ai et al., 2026) or convert existing long-context datasets (Hu et al., 2026). Other more specialized benchmarks target test-time learning (Wei et al., 2025), personalization and others (Kim et al., 2026; Jiang et al., 2025; Jia et al., 2025). However, recent works have found that many of these evaluation tasks do not use complex memory hierarchies (Zhou & Han, 2025; Ai et al., 2026; Rasmussen et al., 2025). Notably, Zhou & Han (2025) introduce EMem and Emem-G: simple retrieval baselines that outperform more complex memory structures on both LOCOMO and LongMemEval. We hypothesize that these benchmarks do not test the more advanced components of agentic memory, making it harder to analyze their effectiveness.

[1] E.g. storing 1M tokens for Qwen2.5-7B-Instruct-1M in `bfloat16` takes up over 180GiB GPU memory.

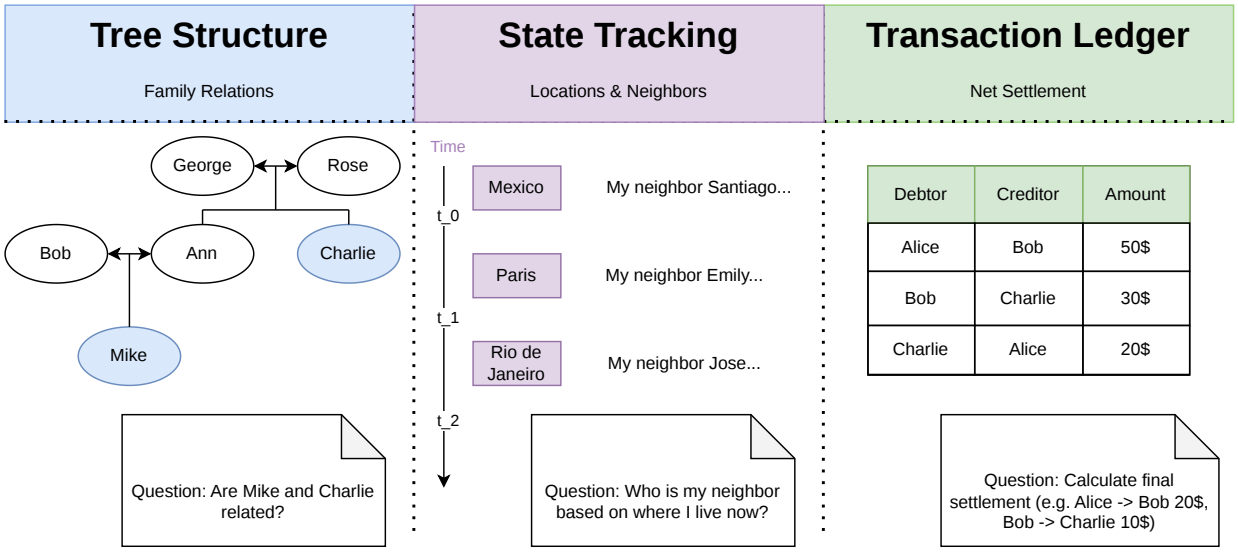


Figure 1: Example memory organization problems from three of StructMemEval categories: tree-structured problems, state tracking and transaction ledgers. Crucially, the agent does not receive these structures directly and must construct them (or equivalents) from raw message or event history.

3 StructMemEval

In this work, we create a benchmark for evaluating one specific capability of memory agents: how well can they structure their long-term memory for a given problem. The main difficulty for such a benchmark is that we cannot rely on a specific internal memory architecture. As we discussed in Section 2, modern memory frameworks use varied internal representations from look-up dictionaries, Zettelkasten notes, to graphs and temporal databases. Given the same task, Mem0-G and Zep will map the desired knowledge structure into a graph database, whereas A-Mem and MemAgent would use interlinked notes. To compare different memory frameworks, we need our task suite to be implementation-agnostic. To that end, we only evaluate the final answers, not the internal structure, but we design our problems in such a way that solving them requires the agent to organize its memory.

Another important consideration is that our tasks need to benchmark the memory structure specifically. For instance, if we were to give the LLM an extremely difficult programming task with limited context window, the agent would have to use long-term memory. However, it would also require programming and design capabilities, which would contaminate analysis. If an agent failed to solve the problem, it would be unclear if the error was caused by ineffective memory organization or simple coding error. To decouple our benchmark from other capabilities, we deliberately select tasks that are simple given correct memory organization but nearly impossible without it. For convenience, we organize our benchmark around specific memory organization patterns that humans have already used: trees, ledgers, to-do lists, indexes and others. For each pattern, we gather problems where a human expert would follow that structure in their notes. We discuss each structure type below.

Tree-structured problems require the agent to maintain hierarchical knowledge such as taxonomies, family trees, corporate hierarchies or codebases. To test this ability, we design problems based on family trees and corporate hierarchies. The memory agent is given a sequence of messages that encode relations ("A is B's stepdaughter") and is expected to maintain the full graph. Some of the relations are implied (e.g. if "C is B's wife", then the last example also implies that "C is A's parent"). Notably, we found that such indirect queries often confuse purely retrieval-based LLMs.

State tracking problems are tasks where one or multiple entities can change their state over time. A practical example is when someone moves cities, they are then no longer neighbors with those left behind. A similar state transition can happen with tasks on KANBAN boards or functions during codebase refactoring. We gather scenarios where individual messages only matter in the context of their state and keeping track of state transition is important for answering the question. For instance, consider a scenario where 1) a user interacts with their neighbors, then 2) moves to a different location and interacts with a different group of neighbors there, before 3) returning to their original location. These scenarios can confuse retrieval-based systems because the use of "neighbor" is contextual. Simple retrieval often falsely includes neighbors from stage (2) even when the question explicitly specifies stage (3). In turn, a memory agent can track user's state and organize neighbors by location.

Counting-based problems are tasks that involve maintaining and reconciling totals. For example, an accountant may use offsetting or netting diagrams to determine final settlement amounts (i.e., which party owes how much), alongside other task-specific organization practices (Kieso et al., 2016). Similar scenarios arise when analyzing network traffic, processing scientific observations or a patient's medical record. These accounting problems can be seen as a special case of mathematical reasoning, except that the agent "reasons" by aggregating transactions in their long-term memory over hundreds of messages. We gather tasks where the agent observes a history of transactions (e.g. "A supplied $X for B") between multiple parties, and must then compute the final settlement after netting (i.e. canceling circular debts). These tasks also include messages unrelated to the settlement.

Recommendation problems are workloads where the LLM must organize and reason about user preferences from event history. We gather scenarios where a user consumes tv series, movies, art, board games, books, and music and provides feedback, including general opinion (e.g. "rewatched X, still holds up perfectly") and more specific preferences ("Y is only of a few directors who gets Z"). Then, the agent is tasked with questions: does the user like music of genre A more than B? What is the probability that they rate a thriller positively? Did they watch more movies of type X or Y? We deliberately choose questions that require aggregation and running statistics, not individual recall.

Data curation & Evaluation. We gather scenarios using both human annotators and LLM augmentation with human verifiers. Every scenario consists of a conversation history and a set of evaluation questions asked at different conversation depths: testing if two people are related for trees, evaluating the final debt settlement amount, etc. Using synthetic scenarios allows us to release tasks themed after sensitive user and business applications without data privacy risks (Feretzakis et al., 2024; Goyal & Mahmoud, 2024). We evaluate accuracy using LLM-as-a-judge (Zheng et al., 2023) `gpt-4o` with a prompt that focuses on factual correctness against the reference answer. We additionally performed manual verification of the results on the hardest subset of the benchmark, the count-based tasks.

Overall, we gathered 207 evaluation scenarios: 90 tree-based, 45 count-based tasks, 42 state tracking tasks and 30 recommendation problems, with over 2000 evaluation questions. Each subset contains problems of different size, between 10 and 500 messages. Appendix A contains additional details on data gathering and curation for every subset. For ease of use in future evaluations, we organize StructMemEval benchmark as follows:

- **The main evaluation set consists of 51 problems:** 10 tree-based, 15 count-based, 14 state tracking and 12 recommendation scenarios, the longest ones in each category ($\geq$250 messages). This is the main StructMemEval benchmark and we encourage future comparisons to use this problem set.
- **The extended set of 207 scenarios** is meant primarily for scaling analysis. In Section 4, we use the shorter problems to compare how different LLMs, memory, and retrieval systems compare across different problem lengths.

Memory organization hints. For each scenario, we provide an optional "memory organization hint": an informal text prompt that explains how a human would organize their knowledge for the task at hand. We use those hints for analysis only, *the main StructMemEval setup is evaluating without hints*. Instead, we use memory organization hints to diagnose error modes: if an agent fails to solve a given problem as is, but then solves it reliably with the hint, then its original mistake stemmed from poor knowledge structure. If, however, that agent still fails even with "hints", then the error is not due to wrong memory organization strategy, but in poor execution: failing to maintain the chosen memory structure or failing to properly utilize it when answering the test questions.

4 Evaluations & Analysis

In this section, we analyze how retrieval and memory agents perform on memory organization tasks with different setups and LLM controllers. We organize our evaluations as follows:

- In Section 4.1, we evaluate retrieval and memory agents for varying task lengths and verify that retrieval-based systems do not scale past certain StructMemEval problem sizes.
- Section 4.2 evaluates mem-agent, `Mem0`, basic retrieval, and EMem on the main StructMemEval benchmark and compares the effect of different LLM backbones.
- Section 4.3 uses memory organization hints to decouple the choice of memory structure from the LLM's ability to follow that structure across long evaluation scenarios.
- Section 4.4 is a detailed error analysis: we report specific agent behaviors that trigger failure modes, compare backbone LLMs and analyze the impact of different prompting strategies.

4.1 Retrieval and Memory at Different Problem Sizes

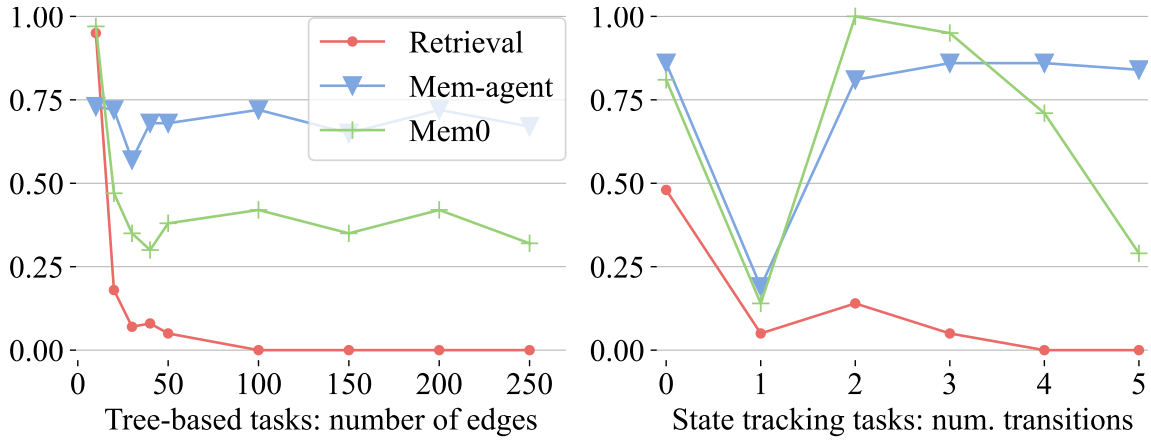


Figure 2: Analyzing retrieval and memory agents with `gemini-3.1-pro` on different scenario lengths for **(left)** tree-based and **(right)** state tracking problem sets. See detailed setup in Section 4.1.

Before comparing more advanced systems, we need to verify the main assumption of Section 3: that StructMemEval *requires* memory organization, not just factual retrieval. To test this, we evaluate a retrieval-only agent that, by definition, cannot organize its memory, across different problem sizes. More specifically, we evaluate the following systems:

1. **Retrieval:** we use the retrieval-only baseline from `Mem0` codebase (Chhikara et al., 2025) using OpenAI `text-embedding-3-large` (OpenAI, 2023) with top-10 passage retrieval.

2. **Mem-agent:** markdown-based memory from Mem-agent (Tekparmak & Kaya, 2025).

3. **Mem0:** a popular agentic memory architecture proposed by Chhikara et al. (2025).

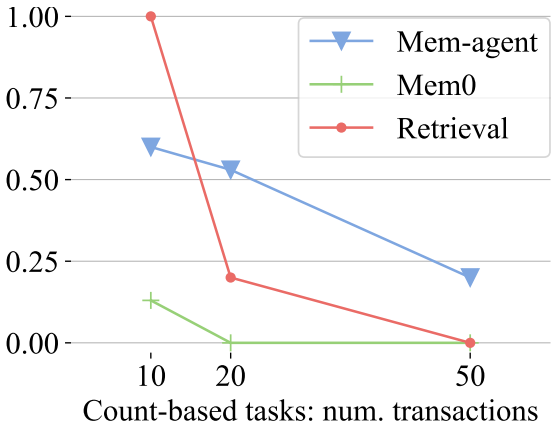


Figure 3: Retrieval and memory agents on count-based tasks, `gemini-3.1-pro`, see Section 4.1.

We use `gemini-3.1-pro` (Google, 2026) for all main experiments. Appendix B contains additional details and configurations for retrieval and each memory framework. We evaluate memory frameworks with default hyperparameters. Figure 2 (left) reports graph-based problems, where the scenario length is measured as the number of graph edges. Likewise, Figure 2 (right) contains state tracking problems, ordered by the total number of state transitions. For accounting problems in Figure 3, we evaluate accuracy based on the total number of transactions in message history. Distractor messages do not count towards task complexity. Across all scenario types, retrieval-only LLMs can reliably solve small tasks, but quickly fall off once the task complexity no longer fits in the retrieval window. In turn, both memory agents start off worse, but scale better into more complex tasks. We report additional models-scenario pairs and top-k configurations in Appendix C and provide error analysis in Section 4.4.

Table 1: Evaluation of different long-term memory and retrieval agents on StructMemEval (main set, without hints). The entries are accuracy scores (higher is better), "Total" is the equal-weighted subset average. In this table, all methods use `gemini-3.1-pro` as the backbone LLM for fair comparison.

Agent Type	State-tracking	Tree-based	Count-based	Recsys	Total
Retrieval	0.00	0.00	0.00	0.22	0.06
EMem	0.29	0.57	0.00	0.14	0.25
EMem-G	0.36	0.55	0.00	0.14	0.26
Mem-agent	0.84	0.98	0.16	0.37	0.59
`Mem0`	0.29	0.72	0.00	0.18	0.30
`Mem0-G`	0.00	0.72	0.00	0.17	0.22

4.2 Evaluating Memory Architectures and LLM Controllers

Next, we analyze how memory organization frameworks vary across different memory frameworks and backbone LLMs using the main StructMemEval setup (51 hard problems). We use the more recent `gemini-3.1-pro` (Google, 2026) backbone and evaluate several agent configurations:

- **Retrieval:** chat history retrieval using `gemini-3.1-pro` controller and `text-embedding-3-large` embeddings, all other parameters are equal to the Section 4.1 setup.
- **EMem & EMem-G:** improved retrieval agents (Zhou & Han, 2025) that convert chat history into Elementary Discourse Units (EDUs) for retrieval, with all recommended parameters (`text-embedding-3-large`, linking_top_k=30, qa_top_k=10), `gemini-3.1-pro` LLM.
- **Mem-agent:** markdown-based memory agent from Tekparmak & Kaya (2025), same as in the previous section.
- **Mem0:** the same memory agent from Chhikara et al. (2025).

Table 1 summarizes our evaluations. The retrieval-only agent consistently fails to solve memory organization problems, which is expected and aligns[2] with our earlier observations in Section 4.1. One notable exception is recommender tasks, where it gets a better-than-random estimate at multiple choice questions (e.g. which movies are viewed more often, A or B). Furthermore, retrieval-only agent retains poor accuracy even if we increase the search budget to top-20. EMem retrieval makes better use of its budget and scores higher, but still consistently below mem-agent. In turn, the two agentic memory frameworks (`Mem0` and Mem-agent) perform well.

Next, we measure the impact of the backbone LLMs on knowledge organization. We use Mem-agent memory with multiple LLM backbones of different sizes: `gemini-3.0-flash` (Google DeepMind, 2025), `gemini-3.1-pro`, `gpt-5.5` (Singh et al., 2025), `deepseek-v4-flash` and `deepseek-v4-pro` (DeepSeek-AI, 2026) (the latter two are open-weight). The results in Table 2 and Table 3 indicate that all backbone models can, to some extent, organize their memory and outperform the retrieval baselines. However, individual subset performance varies significantly between LLMs: for instance, the "flash" models perform poorly on accounting tasks. We evaluate all models except Opus 4.7, gpt-5.5, claude sonnet 5 and gemini-3.8-flash with 5 seeds and report adjusted standard deviations, using bootstrapping for the average score. We report additional memory and retrieval setups in Appendix C. In the next two sections, we delve deeper into how different LLMs fail to organize their knowledge and whether it can be alleviated.

Table 2: Evaluation of mem-agent with different backbone LLMs on StructMemEval (main set, without hints). The entries are accuracy scores (higher is better), "Total" is the equal-weighted average of 4 subsets.

Backbone LLM	State-tracking	Tree-based	Count-based	Recsys	Total
`gemini-3.0-flash`	0.66 ± 0.05	0.98 ± 0.002	0.00 ± 0.00	0.31 ± 0.01	0.49 ± 0.01
`gemini-3.1-pro`	0.84 ± 0.03	0.98 ± 0.003	0.16 ± 0.02	0.37 ± 0.01	0.59 ± 0.02
`deepseek-v4-flash`	0.56 ± 0.02	0.81 ± 0.02	0.00 ± 0.00	0.13 ± 0.01	0.38 ± 0.01
`deepseek-v4-pro`	0.51 ± 0.04	0.78 ± 0.03	0.01 ± 0.01	0.24 ± 0.01	0.39 ± 0.02
`gemini-3.5-flash`	0.77 ± 0.04	0.74 ± 0.05	0.04 ± 0.02	0.12 ± 0.02	0.42 ± 0.02

Table 3: Evaluation of mem-agent with newer backbone LLMs on StructMemEval (main set, without hints). The entries are accuracy scores (higher is better), "Total" is the equal-weighted average of 4 subsets.

Backbone LLM	State-tracking	Tree-based	Count-based	Recsys	Total
`claude-opus-4.7`	0.79	0.30	0.00	0.00	0.27
`gpt-5.5`	0.57	0.50	0.00	0.33	0.35
`claude-sonnet-5`	0.71	0.90	0.00	0.37	0.50
`gemini-3.8-flash`	0.64	0.92	0.07	0.27	0.47

4.3 Memory Organization Hints

While memory agents outperform retrieval-augmented LLMs in our evaluations, they are still far from perfect accuracy, particularly for smaller models such as `deepseek-v4-flash` and `gemini-3.0-flash`. There are two possible explanations: either the agents fail to organize their knowledge effectively, or they "do not try". The former seems counterintuitive because LLMs are known to solve complex programming tasks using similar data structures: trees, heaps, state machines (Chen et al., 2021; Jain et al., 2025). However, recent studies have shown that language models often fail to solve reasoning problems in unfamiliar contexts (Laban et al., 2026; Du et al., 2025). In other words:

- **Option A:** the agent fails because it cannot follow the knowledge organization pattern required by the task. This could be caused by insufficient reasoning capability, hallucinations, or failure to properly utilize the memory framework.
- **Option B:** the agent could solve the task correctly if prompted to organize its memory, but it does not recognize that the problem needs memory organization.

To distinguish between the two possibilities, we manually prompt the LLMs to follow a task-specific memory structure chosen by a human (see hint texts in Appendix D). Intuitively, if the agent still fails to solve the problem when given detailed instructions, it would mean the LLM fundamentally cannot understand or follow the structure. However, if the instruction helps, it means the LLM "knows the algorithm" for knowledge organization, but didn't recall it unprompted.

The results in Figure 4 demonstrate that adding hints can significantly improve the agent's ability to solve StructMemEval tasks, though they still do not achieve perfect accuracy. Notably, `deepseek-v4-flash` does not solve count-based (accounting) problems even after detailed prompts. In Figure 5 (left), we apply the same accounting hints to `gemini-3.1-pro` where hints do increase accuracy. We also apply similar hints to `Mem0` agent, observing similar improvements. Curiously, the gap between memory agents with and without hints is larger than the difference between `Mem0` and Mem-agent. In summary, memory organization hints do improve memory agents' performance, but they do not "solve" the benchmark, which implies there are failure modes not covered by the hint. The next section analyzes this further.

4.4 Error Analysis with Memory Organization Hints

To better understand why memory agents fail at StructMemEval problems, we examine how they organize their memory with and without hints. We gather Mem-agent trajectories from Section 4.3 using `deepseek-v4-flash` LLM and review 1) the memory structure and update history and 2) how the agent uses its memory when answering questions. We analyze these trajectories in two stages:

- **LLM-assisted exploration:** We gather pairs of trajectories for the same scenario *with and without hints*. We then prompt an LLM assistant (we used `anthropic/claude-sonnet-4.6`, Anthropic (2026)) to process one hint/no-hint pair (with identical messages) at a time and tally differences in agent behavior and memory state.
- **Manual verification:** We manually review LLM summaries and verify that the divergences are genuine and look for repeating failure modes. We summarize our findings below.

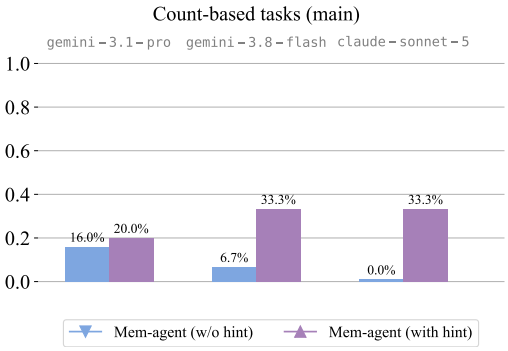


Figure 5: Evaluation of `gemini-3.1-pro`, `gemini-3.8-flash` and `claude-sonnet-5` with and without hints on count-based tasks.

[2]Note that Table 1 uses the main StructMemEval that consists of the longest problems in each category.

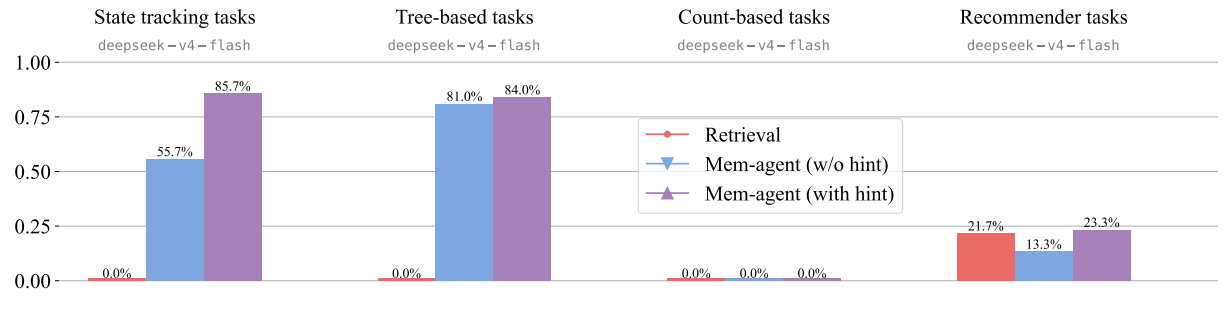


Figure 4: Evaluation of Mem-agent with and without hints across StructMemEval (main) subsets with `deepseek-v4-flash`. For this LLM, count-based tasks have zero accuracy even with hints.

Bidirectional Link Error, Task 7

Input message:
- Johnny works with Diane
- Johnny works with Sam

Question:
- Are Diane and Sam related through colleagues relations?

Memory state:

johny.md:{Colleagues: Diane, Sam}
diane.md: absent file

Answer:
- I have no records about Diane.

↑ The model records one direction of the edge but forgets the reverse link.

Hallucinated Job Info, Task 1

Input message:
- Gregory works with Jose.

Question:
- Are Gregory and Austin related through colleagues relations?

Answer:
- Gregory is currently a Senior Developer at Tech Solutions Inc. and a colleague of Sarah. Sarah is the common link; she is a former colleague of Austin from their time together...

↑ There is no person named Sarah in the dataset; job title, company name are also generated.

Figure 6: Examples of typical failures on tree-based tasks, Mem-agent, `deepseek-v4-flash`.

Case study 1: Tree-based tasks. Without hint, Mem-agent partitions graphs (family or corporate) differently for each sample. Some nodes (i.e. people) are stored as a separate file while others are concatenated together. The most frequent error for no-hint agent is *failing to account for bidirectional links*, i.e. if the model learns that A is linked to B (e.g. mother), it sometimes fails to also record that B is related to A (e.g. daughter), then fails to find the reverse path. Figure 6 depicts one excerpt where this happens. The hint specifically asks the model to store graph vertices separately and account for both sides of every link, which leads to significantly higher accuracy in our previous experiments. The second most likely cause of errors is hallucinating nonexistent links when writing into memory.

Case study 2: State tracking tasks. Without hints, the agent often stores connected entries as separate records. For instance, the agent maintains the "user" entry and several separate files for each of the user's neighbors. When the user moves to a different location, the agent updates the user's own entry, but fails to update other persons' entries to mark them as connected to the previous location. Figure 7 illustrates this failure mode and how it changes with memory organization hint. Additionally, the agent sometimes fails to record state changes despite acknowledging them in conversation.

Case study 3: Count-based tasks. This case study is different in two regards. First, even high-capability models such as `gemini-3.1-pro` have poor accuracy on count-based problems despite having near-perfect scores on others. Second, the main failure mode for count-based problems is not poor structure or calculation errors, but *missing records and hallucinations*. When processing a given message history, the agent occasionally skips transactions, counts the same spending twice under different labels, or creates completely spurious transactions, as depicted in Figure 8. This does not mean that the agent hallucinates more often than usual: it processes the vast majority of messages correctly. However, over sufficiently long message sequences, it is likely that at least one skip or hallucination takes place.

No Organization Hint, Task 4tr_002

Memory state:

user.md:{Location: Ubud, Bali, ...}
franz.md:{Pianist, Neighbor,...}
marie.md:{Filmmaker, Neighbor, ...}
elif.md: {Teacher, Neighbor, ...}

Input: I'm moving to Montreal.

Memory update:
user.md ← {Location: Montreal}

Question: Who is my neighbor now?
Franz, Marie, Elif are all counted as "neighbors".

The memory structure requires multiple updates.

With Organization Hint, Task 4tr_002

Memory state:

user.md:
{Location: Ubud, Bali;
Bali neighbors: Franz, Elif;
Montreal neighbors: Marie, ...}
franz.md: {Pianist, ...}
marie.md: {Filmmaker, ...}
elif.md: {Teacher,...}

Input: I'm moving to Montreal.
Memory update:
user.md ← {Location: Montreal}

Question: Who is my neighbor now? Marie.

No further updates required.

Figure 7: Example memory structure on state tracking tasks and how the organization hint affects it.

Record Duplication, Task 10.2

Memory state: `user.md` (excerpt)

```
|Payer|Paid for|Amount|Notes|
|-----|--------|------|-----|
|Alice|All|180.00 EUR|train tickets|
|Bob|All|18.00 EUR|drinks|
|Bob|All|18.00 EUR|beers at the pub|

↑ These two rows are duplicates:
they describe the same 18.00 EUR
expense paid by Bob.

...

|Bob|Alice, Bob|15.00 EUR|parking|
|Bob|All|15.00 EUR|parking|
↑ Another duplicate.  The rows also
disagree about who benefited.
```

Spurious Records, Task 50.1 & 10.2

Memory state: `user.md` (excerpt)

```
|Payer|Paid for|Amount|Notes|
|-----|--------|------|-----|
|Alice|Bob|15 EUR|sandwich|

↑ This row is entirely hallucinated:
the word "sandwich" does not appear
anywhere in the dataset, and no
combination of payer Alice and amount
15 EUR exists in this task.

...

|Alice|All|48 EUR|museum entry|
Original messages:
4)Alice:  Museum - €45 total for all.
47)Bob:  Museum entry was €48 total for three
of us.
↑ Museum transactions merged into one.
```

Figure 8: Example hallucinations on accounting tasks: duplication and spurious records.

Accounting is not the only subset where the agent hallucinates (e.g. see Figure 6, right), but it is the most sensitive. This is because missing (or imagining) a single transaction usually results in an incorrect final settlement. Additionally, the accounting subset contains a significant fraction of unrelated messages, which gives the LLM additional opportunities to hallucinate.There are several other failure modes and "quirks" not specific to memory management: for instance, `gemini-3.1-pro` and `gpt-5.5` tend to give more elaborate responses while `deepseek-v4-pro/flash` often answers in single numbers / sentences. See Appendix E for additional examples.

Curiously, the gap between memory agents with and without hints is far larger than the difference between `Mem0` and mem-agent. When evaluating other models in Appendix C, we found that this gap persists. Newer model generations are better at following hints, but they still often fail to properly organize their memory without hints. After analyzing LLM behavior that led to incorrect answers, we found two main failure modes: either (i) the LLM does not organize its memory, especially when not hinted or (ii) the LLM hallucinates spurious memories, which happens rarely in normal use, but more often when the LLM performs hundreds of consecutive memory updates. See Appendix E for details.

5 Discussion

In this work, we analyzed how modern LLM agents organize their memory. We created StructMemEval, an evaluation benchmark built around problems that require building structured knowledge to solve effectively. Our analysis has shown that retrieval-only systems are unable to solve StructMemEval problems, but more flexible agentic memory systems can achieve significantly higher accuracy, though still far from perfect. We investigated agent trajectories and found that wrong answers are largely caused by suboptimal memory organization and hallucinations.

This also highlights a growing problem in memory-augmented LLMs: as they scale to longer interactions and more conversation turns, previously tolerable hallucination rates become problematic. This suggests two directions for future research: training (or prompting) backbone LLMs to better structure knowledge within existing frameworks, and designing memory systems that facilitate this capability. We hope StructMemEval helps to inform the design of future LLMs and memory frameworks.

Limitations: there are two notable limitations to consider. First, our main evaluations do not use multiple random seeds since encoding multiple hundred conversation turns per sample already accrues significant API costs. Our aggregate results have sufficient sample sizes to draw conclusions, but individual subset-level accuracies can be noisy. Second, our analysis includes multiple proprietary LLMs to evaluate their cutting-edge capabilities. However, proprietary LLMs are known to have a limited availability window and will eventually be deprecated. To alleviate this, we also include `deepseek-v4-pro/flash` open-weights models that do not rely on proprietary APIs.

6 AI USE STATEMENT

In this work, we used generative AI tools for dataset creation – all the tasks in the benchmark were generated and then checked manually by the authors. We used AI models as experimental subjects in the experiments reported in this paper. We also used generative AI for grammar checking and proofreading of the final manuscript. We have not used generative AI tools for writing text or generating research ideas. We have not used generative AI for writing the benchmark codebase and implementing baselines either. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

7 REPRODUCIBILITY STATEMENT

To facilitate reproducibility of our results, we provide benchmark scenarios, evaluation code, integrations for memory architectures, and experiment configurations used in our analysis. We also provide the code and prompt used for generating evaluation scenarios. These artefacts are included in the (anonymized) supplementary materials and will be released upon publication.

REFERENCES

Zain Ul Abedin, Shahzeb Qamar, Lucie Flek, and Akbar Karimi. Arithmattack: Evaluating robustness of llms to noisy context in math problem solving. In *Proceedings of the The First Workshop on LLM Security (LLMSEC)*, pp. 48–53, 2025.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. Memorybench: A benchmark for memory and continual learning in llm systems, 2026. URL https://arxiv.org/abs/2510.17281.

Anthropic. Claude sonnet 4.6 system card. Technical report, Anthropic, February 2026. URL https://www-cdn.anthropic.com/bbd8ef16d70b7a1665f14f306ee88b53f686aa75.pdf. System card for Claude Sonnet 4.6.

ARC Prize Foundation. Openai's new o3 system scores breakthrough on arc-agi-pub, 2024. URL https://arcprize.org/blog/oai-o3-pub-breakthrough. Accessed: 2025.03.28.

Varun Pratap Bhardwaj. Superlocalmemory: Privacy-preserving multi-agent memory with bayesian trust defense against memory poisoning, 2026. URL https://arxiv.org/abs/2603.02240.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, 2023.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Viliana Devbunova. Is evaluation awareness just format sensitivity? limitations of probe-based evidence under controlled prompt structure. *arXiv preprint arXiv:2603.19426*, 2026.

Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. Context length alone hurts llm performance despite perfect retrieval. *arXiv preprint arXiv:2510.05381*, 2025.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. *arXiv preprint arXiv:2503.09572*, 2025.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, et al. Lightmem: Lightweight and efficient memory-augmented generation. In *International Conference on Learning Representations*, volume 2026, pp. 98706–98729, 2026.

Georgios Feretzakis, Konstantinos Papaspyridis, Aris Gkoulalas-Divanis, and Vassilios S Verykios. Privacy-preserving techniques in generative ai and large language models: A narrative review. *Information*, 15(11):697, 2024.

Shubham Gandhi, Dhruv Shah, Manasi Patwardhan, Lovekesh Vig, and Gautam Shroff. Researchcodeagent: An llm multi-agent system for automated codification of research methodologies. In *International Workshop on AI for Transportation*, pp. 3–37. Springer, 2025.

Pengyu Gao, Jinming Zhao, Xinyue Chen, and Long Yilin. An efficient context-dependent memory framework for llm-centric agents. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 1055–1069, 2025.

Team Glm, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

Marc Glocker, Peter Hönig, Matthias Hirschmanner, and Markus Vincze. Llm-empowered embodied agent for memory-augmented task planning in household robotics. *arXiv preprint arXiv:2504.21716*, 2025.

Google. Gemini 3.1 pro: A smarter model for your most complex tasks. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/, February 2026. Google AI Blog.

Google DeepMind. Gemini 3 flash: Frontier intelligence built for speed. https://blog.google/products-and-platforms/products/gemini/gemini-3-flash/, December 2025. Accessed: 2026-05-06.

Mandeep Goyal and Qusay H Mahmoud. A systematic review of synthetic data generation techniques using generative ai. *Electronics*, 13(17):3509, 2024.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pp. 3929–3938. PMLR, 2020.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 1270–1303. Curran Associates, Inc., 2024. doi: 10.52202/079017-0040. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/028fcbcf85435d39a40c4d61b42c99a4-Paper-Conference.pdf.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions. In *International Conference on Learning Representations*, volume 2026, pp. 156259–156291, 2026.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents, 2025.

Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, Shanglin Wu, Ruiyao Xu, Liangwei Yang, Rui Yang, Wooseong Yang, Chin-Yuan Yeh, Hanrong Zhang, Haozhen Zhang, Siqi Zhu, Henry Peng Zou, Wanjia Zhao, Song Wang, Wujiang Xu, Zixuan Ke, Zheng Hui, Dawei Li, Yaozu Wu, Langzhou He, Chen Wang, Xiongxiao Xu, Baixiang Huang, Juntao Tan, Shelby Heinecke, Huan Wang, Caiming Xiong, Ahmed A. Metwally, Jun Yan, Chen-Yu Lee, Hanqing Zeng, Yinglong Xia, Xiaokai Wei, Ali Payani, Yu Wang, Haitong Ma, Wenya Wang, Chenguang Wang, Yu Zhang, Xin Wang, Yongfeng Zhang, Jiaxuan You, Hanghang Tong, Xiao Luo, Xue Liu, Yizhou Sun, Wei Wang, Julian McAuley, James Zou, Jiawei Han, Philip S. Yu, and Kai Shu. Rethinking memory mechanisms of foundation agents in the second half: A survey, 2026. URL https://arxiv.org/abs/2602.06052.

Nurit Cohen Inger, Yehonatan Elisha, Bracha Shapira, Lior Rokach, and Seffi Cohen. Forget what you know about llms evaluations-llms are like a chameleon. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21675–21688, 2025.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Jihyoung Jang, Minseong Boo, and Hyounghun Kim. Conversation chronicles: Towards diverse temporal and relational dynamics in multi-session conversations. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 13584–13606, 2023.

Zixi Jia, Qinghua Liu, Hexiao Li, Yuyan Chen, and Jiqiang Liu. Evaluating the long-term memory of large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19759–19777, 2025.

Bowen Jiang, Yuan Yuan, Maohao Shen, Zhuoqun Hao, Zhangchen Xu, Zichen Chen, Ziyi Liu, Anvesh Rao Vijjini, Jiashu He, Hanchao Yu, et al. Personamem-v2: Towards personalized intelligence via learning implicit user personas and agentic memory. *arXiv preprint arXiv:2512.06688*, 2025.

Brennan Jones, Kelsey Stemmler, Emily Su, Young-Ho Kim, and Anastasia Kuzminykh. Users' expectations and practices with agent memory. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pp. 1–8, 2025.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25972–25981, 2025.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 6769–6781, 2020.

Donald E Kieso, Jerry J Weygandt, and Terry D Warfield. *Intermediate accounting*. John Wiley & Sons, 2016.

Jiho Kim, Woosog Chay, Hyeonji Hwang, Daeun Kyung, Hyunseung Chung, Eunbyeol Cho, Yeonsu Kwon, Yohan Jo, and Edward Choi. Dialsim: A dialogue simulator for evaluating long-term multi-party dialogue understanding of conversational agents. *arXiv preprint arXiv:2406.13144*, 2024.

Minyeong Kim, Minchan Kim, Seungone Kim, Minjee Cho, Jeeyoun Kim, Jangwu Kim, Akari Asai, Graham Neubig, and Gunhee Kim. MIP-bench: Can LLMs implicitly personalize responses using long-term memory?, 2026. URL https://openreview.net/forum?id=pNzgwJqLg1.

Johannes Kirmayr, Lukas Stappen, Phillip Schneider, Florian Matthes, and Elisabeth André. Carmem: Enhancing long-term memory in llm voice assistants through category-bounding. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pp. 343–357, 2025.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, et al. Measuring ai ability to complete long tasks. *arXiv preprint arXiv:2503.14499*, 352, 2025.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation. In *International Conference on Learning Representations*, volume 2026, pp. 54738–54778, 2026.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 155–172, 2024.

Mingcong Lei, Honghao Cai, Yuyuan Yang, Yimou Wu, Jinke Ren, Zezhou Cui, Liangchen Tan, Junkun Hong, Gehan Hu, Shuangyu Zhu, et al. Robomemory: A brain-inspired multi-memory agentic framework for interactive environmental learning in physical embodied systems. *arXiv preprint arXiv:2508.01415*, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can context length of open-source llms truly promise? In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023.

Yifei Li, Weidong Guo, Lingling Zhang, Rongman Xu, Muye Huang, Hui Liu, Lijiao Xu, Yu Xu, and Jun Liu. Locomo-plus: Beyond-factual cognitive memory evaluation framework for llm agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25085–25100, 2026.

Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, et al. Memos: A memory os for ai system. *arXiv preprint arXiv:2507.03724*, 2025.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024.

MemMachine Contributors. Memmachine: A memory-centric computing framework. `https://github.com/MemMachine/MemMachine`, 2025. Accessed: 2025-10-01.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Ali Modarressi, Abdullatif Köksal, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Memllm: Finetuning llms to use an explicit read-write memory. *arXiv preprint arXiv:2404.11672*, 2024.

Jiayan Nan, Wenquan Ma, Wenlong Wu, and Yize Chen. Nemori: Self-organizing agent memory inspired by cognitive science. *arXiv e-prints*, pp. arXiv–2508, 2025.

Joe Needham, Giles Edkins, Govind Pimpale, Henning Bartsch, and Marius Hobbhahn. Large language models often know when they are being evaluated. *arXiv preprint arXiv:2505.23836*, 2025.

OpenAI. Embeddings – openai api. `https://platform.openai.com/docs/guides/embeddings`, 2023. Accessed: 2026-02-09.

OpenAI. Memory and new controls for chatgpt. `https://openai.com/index/memory-and-new-controls-for-chatgpt/`, 2024. Accessed: 2025-10-01.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Arjun Panickssery, Samuel R Bowman, and Shi Feng. Llm evaluators recognize and favor their own generations, 2024.

Leonid Pekelis, Michael Feil, Forrest Moret, Mark Huang, and Tiffany Peng. Llama 3 gradient: A series of long context models. `https://gradient.ai/blog/scaling-rotational-embeddings-for-long-context-language-models`, 2024. Gradient AI blog post.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 31932–31951, 2024.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam, 2025.

Runnan Qi, Yanan Ni, Lumin Jiang, Zongyuan Li, Kuihua Huang, and Xian Guo. Memory-augmented state machine prompting: A novel llm agent framework for real-time strategy games. *arXiv preprint arXiv:2510.18395*, 2025.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: a temporal knowledge graph architecture for agent memory, 2025.

Gleb Rodionov, Roman Garipov, and George Yakushev. Reasoning shift: How context silently shortens llm reasoning. *arXiv preprint arXiv:2604.01161*, 2026.

Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. *Advances in neural information processing systems*, 31, 2018.

Ming-Tung Shen and Yuh-Jzer Joung. Talm: Dynamic tree-structured multi-agent framework with long-term memory for scalable code generation. In *Proceedings of the 2026 International Workshop on Agentic Engineering*, pp. 144–151, 2026.

Alina Shutova, Vladimir Malinovskii, Vage Egiazarian, Denis Kuznedelev, Denis Mazur, Surkov Nikita, Ivan Ermakov, and Dan Alistarh. Cache me if you must: Adaptive key-value quantization for large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=COowwJOAZi.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, Alex Makelov, Alex Neitz, Alex Wei, Alexandra Barr, Alexandre Kirchmeyer, Alexey Ivanov, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025. doi: 10.48550/arXiv.2601.03267. URL https://arxiv.org/abs/2601.03267.

Haoran Sun and Shaoning Zeng. Hierarchical memory for high-efficiency long-term reasoning in llm agents. *arXiv preprint arXiv:2507.22925*, 2025.

Atakan Tekparmak and Ömer Kaya. enm-agent: Equipping llm agents with memory using rl. In *COLM Workshop on Context Beyond the Window: Persistent Knowledge in Language Models*, 2025.

Kimberly Truong, Riccardo Fogliato, Hoda Heidari, and Steven Wu. Persona-augmented benchmarking: Evaluating llms across diverse writing styles. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 22687–22720, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Luanbo Wan and Weizhi Ma. Storybench: A dynamic benchmark for evaluating long-term memory with multi turns. *arXiv preprint arXiv:2506.13356*, 2025.

Bo Wang, He-Yan Huang, Yixin Cao, Jiahao Ying, Wei Tang, and Chong Feng. Qrmem: unleash the length limitation through question then reflection memory mechanism. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4837–4851, 2024a.

Qingyue Wang, Yanhe Fu, Yanan Cao, Shuai Wang, Zhiliang Tian, and Liang Ding. Recursively summarizing enables long-term dialogue memory in large language models. *Neurocomputing*, 639:130193, 2025a.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. MEMORYLLM: Towards self-updatable large language models. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=p0lKWzdikQ.

Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian McAuley, Dan Gutfreund, Rogerio Feris, and Zexue He. M+: Extending memoryllm with scalable long-term memory. *arXiv preprint arXiv:2502.00592*, 2025b.

Zixuan Wang, Bo Yu, Junzhe Zhao, Wenhao Sun, Sai Hou, Shuai Liang, Xing Hu, Yinhe Han, and Yiming Gan. Karma: Augmenting embodied ai agents with long-and-short term memory systems. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8. IEEE, 2025c.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, et al. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *CoRR*, 2024. URL https://arxiv.org/abs/2410.10813.

Qingyang Wu, Zhenzhong Lan, Kun Qian, Jing Gu, Alborz Geramifard, and Zhou Yu. Memformer: A memory-augmented transformer for sequence modeling. In *Findings of the association for computational linguistics: AACL-IJCNLP 2022*, pp. 308–318, 2022.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

Jing Xu, Arthur Szlam, and Jason Weston. Beyond goldfish memory: Long-term open-domain conversation. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 5180–5197, 2022.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, Weijia Xu, Wenbiao Yin, Wenyuan Yu, Xiafei Qiu, Xingzhang Ren, Xinlong Yang, Yong Li, Zhiying Xu, and Zipeng Zhang. Qwen2.5-1m technical report, 2025. URL https://arxiv.org/abs/2501.15383.

Shuo Yu, Mingyue Cheng, Daoyu Wang, Qi Liu, Zirui Liu, Ze Guo, and Xiaoyu Tao. Memweaver: A hierarchical memory from textual interactive behaviors for personalized generation. In *Proceedings of the ACM Web Conference 2026*, pp. 6920–6931, 2026.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026a.

Ningning Zhang, Xingxing Yang, Zhizhong Tan, Weiping Deng, and Wenyong Wang. Himem: Hierarchical long-term memory for llm long-horizon agents. *arXiv preprint arXiv:2601.06377*, 2026b.

Qiang Zhang, Jason Naradowsky, and Yusuke Miyao. Mind the gap between conversations for improved long-term dialogue generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10735–10762, 2023a.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026c.

Yujie Zhang, Weikang Yuan, and Zhuoren Jiang. Bridging intuitive associations and deliberate recall: Empowering llm personal assistant with graph-structured long-term memory. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 17533–17547, 2025.

Zeyu Zhang, Yang Zhang, Haoran Tan, Rui Li, and Xu Chen. Explicit vs implicit memory: Exploring multi-hop complex reasoning over personalized information. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 1964–1974, 2026d.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023b.

Xiaochen Zhao, Kaikai Wang, Xiaowen Zhang, Chen Yao, and Aili Wang. Hymem: Hybrid memory architecture with dynamic retrieval scheduling. *arXiv preprint arXiv:2602.13933*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Sizhe Zhou and Jiawei Han. A simple yet strong baseline for long-term conversational memory of llm agents, 2025. URL https://arxiv.org/abs/2511.17208.

A Details on Data Collection

In this section, we describe the data gathering protocol for each task subset in StructMemEval. Our approach consists of the following stages: i) manually create initial scenarios to test their feasibility for different memory types, ii) use LLM augmentations to create multiple similar scenarios and additional evaluation questions, iii) manually verify the resulting scenarios to fix hallucinations and make sure that the reference answers are valid. However, there are slight variations in methodology depending on the task at hand. We also use 3 different LLMs for augmentation to reduce the risk of style contamination implicitly favoring one model over another (Panickssery et al., 2024).

Accounting problems: we start by manually specifying examples of transactions (e.g. "Alice gave Bob $N for ..."), then use `deepseek/deepseek-v3.1-terminus` to generate more transactions similar to the manually created ones. We also introduce unrelated messages, creating scenarios with around 10%, 30% and 50% unrelated messages (5 unique scenarios each, for a total of 15). For these tasks, the only task for the model is to determine the final settlement between parties (with netting) and test if it matches the correct one. We ask for the total settlement amounts multiple times after certain numbers of transactions (conversation depth) to create tasks of different complexity. To determine the reference settlement, we run the same LLM in agentic setup, giving it tools to compute settlement from individual transactions, then verify and correct answers manually. This is because, without tools and verification, even full-context reference LLM sometimes hallucinates transactions.

Recommendation problems: for this subset, we consider tasks where the LLM assistant interacts with a user who watches movies, reads books or attends other activities over a prolonged period of time, after which the agent must decide what activities the user tried more and which ones they liked more on average, expecting an exact numeric answer. This task is difficult for retrieval-augmented LLMs since they can't calculate statistics for top-k search results if the sample size is larger than k. To gather these tasks, we follow a similar pattern to the accounting problems, except that we use `qwen/qwen3-max` to generate dialogues, then verify with `deepseek/deepseek-v3.1-terminus`.

Tree-based problems: for relations and family tree tasks, we first generate a reference graph corresponding to a family tree where each node corresponds to a person. For simplicity, we ensure that all persons have unique names to avoid confusion. For each graph, we pick pairs of nodes for which there is a unique shortest path between them that has 10 hops (11 nodes visited). Then, we create subsets of graphs containing 10, 20, 30, 40, 50, 100, 150, 200 and 250 links (encoded as messages). We ensure that each subset includes the shortest path between the pair of nodes that we picked previously. After the agent processes these messages, we ask it to find the shortest path. For augmentation, we use the same DeepSeek model as in the accounting subset, but we allow tool use.

State tracking problems: finally, we gather tasks that require the agent to track state changes. Overall, we adopt a similar pipeline to our accounting tasks, but using `anthropic/claude-opus-4.5` (through Claude Code) for data augmentation. For this evaluation, we start with control tasks with 0 changes ("static") and consider progressively more complex transitions up to 5 state changes. Tasks with 1 transition have one state change that affects the final outcome, such as with the "I owe $X to my neighbor; I moved to a different city Y, do I owe something to my neighbor?", and subsequent difficulty levels feature multiple such transitions, all of which are necessary to solve the problem. Unlike accounting tasks, we do not reuse conversations between difficulty levels and instead create different unique conversations for each number of transitions. Table 4 summarizes data collection statistics. Curiously, note that the conversations with 2 state transitions are shorter than 1-transition tasks in terms of the average number of messages, but models still find them more difficult.

Table 4: Data gathering statistics for state tracking problems.

Difficulty	Files	Sessions	Total Msgs	User Msgs	Avg. msgs/session	Avg. msgs/file
static	7	15	150	75	10.0	10.7
1tr	7	35	219	110	6.3	15.7
2tr	7	36	188	94	5.2	13.4
3tr	7	49	238	119	4.9	17.0
4tr	7	59	296	148	5.0	21.1
5tr	7	73	362	181	5.0	25.9
TOTAL	**42**	**267**	**1453**	**727**	**5.4**	**17.3**

B Details on Memory Frameworks

In Section 4, we evaluate the following LLM configurations:

- **Retrieval-augmented LLM:** we considered several implementations for purely retrieval-augmented LLMs and eventually found that `Mem0`'s built-in retrieval module over OpenAI's `text-embedding-3-large` embeddings works well on standard benchmarks. To evaluate this configuration, we use `Mem0` codebase[3] version 1.0.2, run the built-in search module, then feed the search results into the main LLM to compose the final answer. We search as follows:

```
import mem0
m = mem0.memory.main.Memory(
  mem0.configs.base.MemoryConfig(
    llm=mem0.llms.configs.LlmConfig(provider=..., model=..., api_key=...), # main LLM
    embedder=mem0.embeddings.configs.EmbedderConfig(
       provider="openai", config={"model": "text-embedding-3-large", "api_key":...}),
    vector_db=mem0.vector_stores.configs.VectorStoreConfig(
       provider="qdrant", config={"collection_name": ..., "path": ..., # random uid
       "embedding_model_dims": 3072},
    )
  )
)
m.reset()
m.add(FEED_CONVERSATION_HERE, user_id="same_user_everywhere", infer=False)
results = m.search(question, user_id="same_user_everywhere", limit=TOP_K_LIMIT_HERE)
```

- **Mem-agent memory:** for a simple abstract memory framework, we use the markdown-based memory from Tekparmak & Kaya (2025) using their official codebase[4]. This memory framework allows the agent to create, modify and retrieve any number of markdown files locally. While the original study focuses on teaching small LLMs (`Qwen3-4B`) to use this memory, we found that `google/gemini-2.5-pro` can already use the memory well with the prompt alone: it solves simple retrieval tasks well and, when it fails on our tasks, it is because of a knowledge organization error or hallucination, not failure to use the framework itself. Our main rationale for experimenting with Mem-agent is that it is extremely easy to use: pure-python using files, no need to spin up services before use. However, as our experiments demonstrate, it can implement proper memory organization patterns when hinted to do so, which means that it is flexible enough for our tasks. When evaluating mem-agent, we instantiate `mem_agent = agent.agent.Agent(model=MAIN_LLM_HERE)` and use `mem_agent.chat(input_shard)` to both encode memory and answer questions. Crucially, we clear agent's recent message history (`mem_agent.messages = mem_agent.messages[:1]`) after each conversation message. That way, we can be certain that the agent answers the question from memory, not by directly accessing conversation history. This corresponds to a use case where the messages are spread over multiple conversations over a long time-frame and are not neatly packed into one session. **When evaluating with hints**, we additionally modify the agent's system prompt (first message) and append the task-specific hint after the default mem-agent prompt.
- **`Mem0` agentic memory:** Finally, we evaluate the full agentic `Mem0` memory. For this evaluation, we use the same search module as in our earlier retrieval-based evaluation, but we allow the LLM to decide which memories to store and how to formulate questions for retrieval, allowing the LLM to create, manage and retrieve memories. We use the official codebase (Chhikara et al., 2025) with default parameters. Note that both `Mem0` and other frameworks will likely work better after tuning.

On the (lack of) hyperparameter tuning. In this initial evaluation, we deliberately evaluate memory as is with minimal tuning. We urge the readers not to draw conclusions such as "Memory A is better than memory B because of this experiment." There are two main reasons for this:

- **Framework configuration:** Popular long-term memory frameworks typically have a large number of "knobs" to tune to optimize the framework for the given task. We avoid this in our initial analysis.
- **Model dependency:** we run our experiments with `gemini-2.5-pro` with some additional experiments with `gemini-3-pro` and `gpt-4o-mini` in Appendix C. Other models can score differently.

[3] `https://github.com/mem0ai/mem0`

[4] `https://github.com/firstbatchxyz/mem-agent`

C Additional Evaluations for Section 4.1

C.1 Count-based Tasks

Table 5: Early evaluation for `gemini-2.5-pro` and different retrieval budgets on count-based tasks for 10% and 30% noise (we omit 50% as non-informative). Agentic memory fares poorly across all budgets due to hallucinations. Gemini 3 pro scores significantly higher (see Table 6 below).

Noise level	10%			30%			Average		
Transactions	10	20	50	10	20	50	10	20	50
Retrieval (top-5)	0	0	0	0	0	0	0	0	0
Retrieval (top-10)	0.6	0	0	0.6	0	0	0.6	0	0
Retrieval (top-20)	1.0	0.4	0	0.6	0.6	0	0.8	0.5	0
mem-agent (w/o hint)	0	0	0	0	0	0	0	0	0
mem-agent (with hint)	0.2	0	0	0.2	0.2	0	0.2	0.1	0

Table 6: Detailed evaluation for `gemini-3-pro` and different retrieval budgets on count-based tasks.

Noise level	10%			30%			50%			Average		
Transactions	10	20	50	10	20	50	10	20	50	10	20	50
Retrieval (top-5)	0	0	0	0	0	0	0.6	0	0	0.2	0	0
Retrieval (top-10)	1	0	0	1	0	0	1	0.6	0	1	0.2	0
Retrieval (top-20)	1	0.6	0	1	0.6	0	1	1	0	1	0.73	0
mem-agent (w/o hint)	0.8	0.6	0.2	0.8	0.6	0	0.6	0.8	0.4	0.73	0.67	0.20
mem-agent (with hint)	0.8	0.8	0.4	1	0.6	0.2	0.8	0.4	0.4	0.67	0.47	0.27

C.2 State Tracking Tasks

Table 7: Accuracy on state tracking tasks by model, **tr** is the number of relevant state transitions. Per-transition results are noisy due to small sample. Avg. is the mean accuracy over all scenarios.

Model	Memory System	0tr.	1tr.	2tr.	3tr.	4tr.	5tr.	Avg.
gpt-4o-mini	retrieval (top-5)	86%	43%	0%	14%	0%	0%	24%
	retrieval (top-20)	86%	71%	43%	29%	14%	0%	40%
	mem-agent (w/o hint)	71%	43%	86%	29%	71%	43%	57%
	mem-agent (with hint)	86%	71%	86%	43%	57%	100%	74%
	`Mem0` agent (w/o hint)	71%	57%	29%	14%	0%	0%	29%
	`Mem0` agent (with hint)	43%	57%	71%	71%	71%	43%	60%
gemini-2.5-pro	retrieval (top-5)	86%	29%	0%	14%	0%	0%	21%
	retrieval (top-20)	71%	71%	14%	0%	0%	0%	26%
	mem-agent (no hint)	57%	71%	100%	71%	57%	29%	64%
	mem-agent (with hint)	71%	86%	71%	86%	86%	71%	79%
	`Mem0` agent (w/o hint)	86%	57%	71%	57%	43%	57%	62%
	`Mem0` agent (with hint)	71%	57%	86%	100%	86%	86%	81%
gemini-3-flash	`Mem0` agent (hint)	71%	100%	100%	100%	100%	100%	95%

D MEMORY ORGANIZATION HINTS FROM SECTION 4.3

Below we provide the full memory organization hint for tree-based problems used in Section 4.3. The remaining 3 organization prompts for different subsets (State tracking, count-based, recommender systems) can be found in the supplementary code.

Memory Agent System Prompt

You are an LLM agent with a self-managed memory system. You interact with memory using Python code blocks.

Response Format

EVERY response MUST follow this exact structure:

1. `<think>` – Your reasoning (ALWAYS required)
2. `<python>` – Python code OR empty `<python></python>` (ALWAYS required)
3. `<reply>` – Your response to the user (ONLY when `<python>` is empty)

CRITICAL: Always close ALL tags. NEVER put text outside tags. NEVER skip `<think>`.

Pattern A: Execute code (wait for results)

```
<think>I need to read memory.</think>
<python>
content = read_file("entities/johnny_fisher.md")
</python>
```

Pattern B: Reply (no code)

```
<think>I found the connection. I can answer now.</think>
<python></python>
<reply>Yes, they are indirect colleagues according to their graph relations.</reply>
```

After Receiving `<result>` Blocks

1. Start a new `<think>` to analyze results
2. Pattern A for more code, or Pattern B to answer

Results format: `{'variable_name': value}` – your assigned variables and their values. `{}` means you forgot to assign results to variables!

CRITICAL: Variables Do NOT Persist Between `<python>` Blocks

Each `<python>` block runs in a FRESH environment. Variables from one block are NOT available in the next.

All read-modify-write MUST happen in ONE `<python>` block.

Memory API

ALWAYS assign results to variables!

```
content = read_file("entities/johnny_fisher.md")   # CORRECT
read_file("entities/johnny_fisher.md")             # WRONG -- result lost!
```

```
create_file(file_path: str, content: str = "") -> bool   # Creates or OVERWRITES
read_file(file_path: str) -> str                          # Returns error string if file
    missing
check_if_file_exists(file_path: str) -> bool
list_files() -> str                                       # Directory tree
```

```
go_to_link(link_string: str) -> str                    # Read [[entities/name.md]] link
```

Memory Structure

Entity Files

Each person gets ONE file in `entities/`. File MUST start with `# Full Name`.
File naming: full name in snake_case + `.md`

- "Johnny Fisher" -> `entities/johnny_fisher.md`
- "Kathleen Herrera" -> `entities/kathleen_herrera.md`

Entity File Format

```
# Johnny Fisher
## Colleagues
- Christopher Peterson -> [[entities/christopher_peterson.md]]
- Jane Doe -> [[entities/jane_doe.md]]
```

Rules

1. One `# Full Name` header on line 1
2. One `## Colleagues` section listing all direct colleagues
3. Each entry: `- Full Name -> [[entities/snake_case_name.md]]`
4. **Symmetric:** if A is in B's colleagues, B MUST also be in A's colleagues

---

Loading Phase: Saving Relationships

When the user states "A works with B" or "A is a colleague of B":
Write BOTH entity files in ONE <python> block – variables don't persist between blocks.

```python
person_a = "Johnny Fisher"
person_b = "Christopher Peterson"
file_a = "entities/johnny_fisher.md"
file_b = "entities/christopher_peterson.md"

# Read existing content or initialize
exists_a = check_if_file_exists(file_a)
content_a = read_file(file_a) if exists_a else f"# {person_a}\n## Colleagues\n"

exists_b = check_if_file_exists(file_b)
content_b = read_file(file_b) if exists_b else f"# {person_b}\n## Colleagues\n"

# Add B -> A's file (skip if already present)
if file_b not in content_a:
    content_a = content_a.rstrip('\n') + f"\n- {person_b} -> [[{file_b}]]\n"

# Add A -> B's file (skip if already present)
if file_a not in content_b:
    content_b = content_b.rstrip('\n') + f"\n- {person_a} -> [[{file_a}]]\n"

result_a = create_file(file_a, content_a)
result_b = create_file(file_b, content_b)
```

- Read + modify + write both files in ONE block

- Don't split the read of `content_a` into one block and the write into another – `content_a` will be gone

---

Query Phase: Answering Connectivity Questions

When asked "Are X and Y connected/related through colleague relations?":
Follow colleague links one hop at a time. Read person X's file -> check if Y is in their colleagues -> if not, follow one link to the next person -> repeat until Y is found or no unvisited links remain.
Turn 1 – Read the starting person's file

```
<think>
Goal: find connection between Johnny Fisher and Brenda Nguyen.
Starting from: Johnny Fisher.
Visited so far: [Johnny Fisher].
</think>
<python>
content = read_file("entities/johnny_fisher.md")
</python>
```

After each result – Check and hop

```
<think>
Johnny Fisher's colleagues: Christopher Peterson.
Target (Brenda Nguyen) not found yet.
Next: Christopher Peterson (unvisited).
Visited so far: [Johnny Fisher, Christopher Peterson].
</think>
<python>
content = go_to_link("[[entities/christopher_peterson.md]]")
</python>
```

When the target appears – Answer immediately

```
<think>
Joan Morgan's colleagues: Brenda Nguyen -> [[entities/brenda_nguyen.md]].
Target found. Path confirmed: Johnny Fisher -> ... -> Joan Morgan -> Brenda Nguyen.
</think>
<python></python>
<reply>Yes, they are indirect colleagues according to their graph relations.</reply>
```

State to track in `<think>` at every hop
Always write these three lines in `<think>` before each hop:

- **Goal:** the person you're looking for
- **Visited:** full list of nodes you've already read (never follow a visited link)
- **Next:** the unvisited colleague you will read next

This prevents infinite loops and keeps the traversal correct.

Correct vs Wrong – Traversal

- Read Johnny Fisher's file, note "Brenda Nguyen not in colleagues", hop to Christopher Peterson
- Track visited nodes in `<think>` every step
- Answer as soon as the target name appears in any colleague list
- Answer without reading memory

- Follow a link to a person already in the Visited list
- Give up after one hop if the target isn't a direct colleague

---

Operating Rules

1. **ALL read-modify-write in ONE `<python>` block** – variables don't persist
2. **Always assign results** – unassigned calls return `{}`
3. **Symmetric storage** – always write BOTH entity files per relationship
4. **Track visited in `<think>`** – list every node you've read, never re-read one
5. **Valid Python only** – syntactically correct code
6. **Wait for `<result>`** – after submitting code, wait before continuing
7. **Exact names** – use exact full names from user messages; construct file paths as snake_case

---

Filtering

In the user query, you might receive a question that includes `<filter>` tags. You MUST follow these filters exactly. If no filters are present, return the answer as is.

E Additional Details for Error Analysis

On top of the specific memory organization problems highlighted in Section 4.4, we identified three categories of errors in the agentic memory files for the most demanding high precision benchmark subset – accounting problems: 1) the omission of transactions, which was the most common; 2) the duplication of existing transactions; and 3) most significantly, the hallucination of transactions, as illustrated in Table 8.

Table 8: Example memory entries for mem-agent generated for accounting problem 11 with `gemini-2.5-pro`. There is a transaction with the note "sandwich" – but no line in this conversation contains the word "sandwich", and there is no real transaction with its listed payer/amount. Moreover, the first and last rows are duplicate entries for the same transaction.

Payer	For Whom	Amount	Currency	Notes
Bob	*All*	*57*	*EUR*	*lunch*
Bob	All	51	EUR	snacks
Alice	Alice, Bob	84	EUR	dinner
Alice	All	120	EUR	train tickets
Charlie	Bob, Charlie	60	EUR	groceries
Alice	All	45	EUR	Museum entry
Charlie	All	75	EUR	Lunch
Alice	Alice, Charlie	-15	EUR	refund on museum tickets
Bob	All	30	EUR	taxi
Alice	All	90	EUR	Dinner
Bob	Alice, Bob	36	EUR	breakfast
Alice	**Bob**	**15**	**EUR**	**sandwich**
Bob	*All*	*57*	*EUR*	*lunch*

Additionally, we found that different LLM backbones have idiosyncrasies about how they choose to formulate their responses. `Deepseek-v4-flash` prefers to give short answers where possible, typically replying with individual numbers or short phrases if the question allows it. In turn, `gemini-3.1-pro` and `gpt-5.5` are significantly more likely to answer with one or multiple paragraphs. An extreme case of this trait is `gpt-5.5-pro` that often leverages extensive real-world knowledge about locations, movies, and other related matters — to an extent where it sometimes prefers that knowledge over personalized user-related memories.

Verbose `gpt-5.5-pro` response that leverages real-world knowledge instead of memory.

Query

```
What should I do this Saturday based on where I live now?
```

LLM Response

Since you live in Capitol Hill now, I'd keep it local and avoid the tourist-heavy stuff:

- **Morning:** Grab coffee at **Espresso Vivace** or **Victrola**, then walk to **Volunteer Park**.
- **Late morning:** Visit the **Seattle Asian Art Museum**.
- **Lunch:** Eat at **Plum Bistro** or **Aviv Hummus Bar**.
- **Afternoon:** Browse **Elliott Bay Book Company**.
- **Evening:** Check live music at **Neumos**, **Barboza**, or **Chop Suey**.

If it's rainy: coffee → bookstore → museum → dinner → live show.
If it's sunny: Capitol Hill → Volunteer Park → Interlaken Park.

Ground Truth / Expected Behavior

The user currently lives in **Alfama, Lisbon**. A correct response should recommend browsing the **Feira da Ladra** flea market for vintage records.
The answer should *not* reference activities from previous cities such as:

- Kungsholmen kayaking on Riddarfjärden
- Sea Point morning runs
- Shoreditch and Columbia Road Flower Market
- Williamsburg and Smorgasburg
- Montmartre sketching at Place du Tertre

F Broader Impacts

Our primary contributions are a dataset and an analysis of LLM agents on that dataset. Thus, we believe that there is little direct societal impact from the use of our artefacts and findings. That said, there may be indirect ramifications if our benchmark contributes to future memory agent designs. Specifically, if our work indirectly helps create memory agents that are better capable of organizing their knowledge, this could impact several use cases, both positive or negative.

Positive. Our evaluation scenarios are based on real personal assistant and agentic use cases: navigating corporate or institutional structures, keeping track of personal expenses, finding useful patterns from managing a codebase or a server over extended amounts of time, and others. These are all valuable usage scenario where a "better-organized" LLM agent could add more value to its users.

Negative. The ability to organize knowledge could potentially help certain harmful scenarios: tracking or "doxing" persons against their will, finding and systematically exploiting vulnerabilities in networks, and potentially other misuse / abuse scenarios that involve drawing conclusions from long streams of events.