MEMREFINE: LLM-Guided Compression for Long-Term Agent Memory

Anonymous ACL submission

Abstract

Large language model (LLM) agents are increasingly expected to operate over long-term interactions, where information from past dialogues must be preserved and recalled to support future tasks. However, as interactions accumulate, the memory store grows without bound and fills with redundant entries that inflate storage cost and degrade retrieval by crowding out the most useful evidence. Furthermore, this is especially limiting on resource-constrained platforms with hard memory budgets, motivating us to formulate *storage-budgeted memory management*, the task of keeping an already constructed memory store within a fixed budget while preserving information useful for future interactions. To this end, we then propose MEMREFINE, an LLM-guided framework that, since surface similarity poorly reflects factual value, uses similarity only to propose candidate pairs and defers delete, merge, and preserve decisions to an LLM judge based on factual content, iterating until the budget is met. Across multiple memory frameworks and long-term conversation benchmarks, MEMREFINE consistently meets target budgets while preserving downstream performance and outperforming rule-based baselines under tight budgets.

1 Introduction

Large language model (LLM) agents are increasingly expected to operate over long-term interactions, where useful information may be distributed across extended sequences of past conversations, sessions, or events (Zhong et al., 2024; Maharana et al., 2024; Xu et al., 2025). To support such interactions, recent agent frameworks equip LLMs with persistent memory stores that record interaction histories and supply them as evidence for future tasks (Packer et al., 2023; Zhong et al., 2024; Xu et al., 2025; Chhikara et al., 2025; Wu et al., 2025).

However, persistent memory creates a storage-efficiency problem that compounds over time. As the agent continues to interact, the memory store grows without bound and gradually fills with entries that overlap with or duplicate information already captured elsewhere (Zhong et al., 2024; Xu et al., 2025; Fang et al., 2025; Liu et al., 2026), which both inflates storage cost (a particularly acute issue on resource-constrained platforms such as on-device deployments, where memory budgets are hard-bounded) and degrades retrieval, since the agent may surface near-duplicate or low-value entries. Motivated by this, we introduce a new task of *storage-budgeted memory management*, where agent memory needs to be continuously kept within a fixed budget while preserving information that remains useful for future interactions.

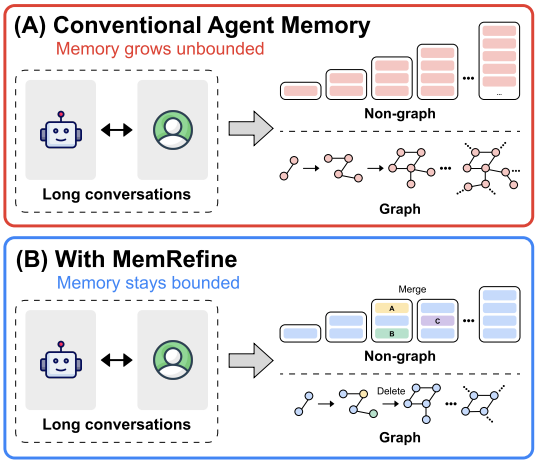

Figure 1: Conceptual overview of storage-budgeted memory management. Without compression, long-term interactions cause agent memory to grow without bound in both non-graph and graph memory stores. With MEMREFINE, the constructed memory store is refined under a fixed budget, keeping memory compact while preserving information useful for future retrieval.

Existing approaches to compression, however, do not target this setting. Session-level summarization (Wang et al., 2025; Chen et al., 2025) compresses raw conversation history before it enters

memory rather than the resulting store, prompt or retrieved-context compression (Jiang et al., 2023; Xu et al., 2024; Jiang et al., 2024) operates at inference time on the evidence provided to the LLM, and structure-based graph pruning (Brin and Page, 1998) relies on criteria such as centrality that do not capture whether entries are factually redundant, complementary, or distinct. As a result, none of these approaches jointly reduces an already constructed memory store to a target storage budget while reasoning about the factual content of entries.

To address this, we introduce *post-construction memory compression*, where an already constructed memory store is reduced to a target storage budget while retaining information useful for downstream tasks, and propose MEMREFINE, an LLM-guided framework that instantiates this task as a general module inserted after memory construction and before retrieval, without modifying the host memory pipeline. The central insight is that while surface-level similarity is useful for surfacing candidate pairs of related memories, similarity alone is too weak to decide their fate, since two textually similar entries may carry redundant facts that can be removed, complementary facts that should be merged, or distinct facts that both warrant preservation. Motivated by this, MEMREFINE uses similarity to propose candidate pairs but defers the compression decision itself to an LLM judge, which examines each pair and decides, based on factual content rather than surface wording, whether it should be deleted, merged, or preserved, until the memory store fits within the target storage budget.

We validate MEMREFINE on two representative memory frameworks with different representations, namely an A-MEM-style graph memory of linked structured notes (Xu et al., 2025) and the Mem0 pipeline that constructs entries through an ingestion-and-update process (Chhikara et al., 2025), evaluated across multiple LLM judges and conversation lengths on standard and scaled LoCoMo-style benchmarks (Maharana et al., 2024). Across settings, MEMREFINE consistently meets the target storage budget while preserving downstream performance under moderate compression, and degrades gracefully rather than collapsing under tighter budgets. Moreover, compared with rule-based baselines that replace the LLM judge with fixed similarity or graph-structure heuristics, MEMREFINE is consistently more robust, supporting our hypothesis that compression decisions benefit from LLM-guided factual judgment even when similarity alone is sufficient for candidate selection.

2 Related Work

Long-Term Memory for LLM Agents Long-term memory has become a central component for dialogue systems and LLM agents that operate over information accumulated across extended interactions. Early work on long-term and multi-session dialogue showed that conversational agents need to store, update, and retrieve information from previous sessions to maintain consistency over time (Xu et al., 2022; Bae et al., 2022; Jang et al., 2023). More recent work develops external memory systems for LLM agents, including user-level memory banks (Zhong et al., 2024), explicit read-write memory modules (Modarressi et al., 2023), virtual context management (Packer et al., 2023), structured note graphs (Xu et al., 2025), and general-purpose memory layers for agents (Chhikara et al., 2025), alongside benchmarks that evaluate whether such systems can recall information over extended histories (Maharana et al., 2024; Wu et al., 2025). However, these efforts predominantly focus on *growing* memory, namely how new information is added and recalled as interactions accumulate, leaving the inverse problem of *shrinking* an already constructed store under a storage budget largely unaddressed. In contrast, we treat post-construction storage reduction as a standalone task, compressing an existing memory store to fit a target budget while leaving the host memory framework unchanged.

Memory and Context Compression Reducing the cost of storing or processing long-term information has been studied in several related but distinct settings. Session-level memory summarization compresses dialogue histories into compact summaries before they enter memory, for instance through recursive summarization for long-term dialogue memory (Wang et al., 2025) or compressive memory-enhanced dialogue generation (Chen et al., 2025). In the meantime, inference-time context compression instead reduces the evidence or prompt fed to the LLM rather than the underlying store, including RECOMP for retrieved documents (Xu et al., 2024) and LLMLingua and LongLLMLingua for long-context prompts (Jiang et al., 2023, 2024). Structure-based reduction shrinks memory graphs offline through criteria such as centrality or connectivity (Brin and Page, 1998; Wang et al., 2026), while recent agent-memory frameworks improve memory efficiency through filtering, consoli-

dation, or lightweight updates interleaved with the construction pipeline itself (Fang et al., 2025; Liu et al., 2026).

None of these approaches, however, takes *reducing an already constructed memory store to a fixed storage budget through reasoning about the factual content of its entries* as the explicit objective: summarization preempts memory rather than shrinking it, context compression touches the inference-time prompt rather than the store, and structure- or update-based methods either rely on criteria insensitive to content or remain coupled to the construction pipeline. In contrast, we fill this gap by directly compressing an existing memory store toward a target storage budget, applying LLM-guided judgments over semantically related entries to decide whether each pair carries redundant facts to be removed, complementary facts to be merged, or distinct facts to be preserved, while remaining decoupled from the underlying memory construction and retrieval pipelines.

3 Method

We present MEMREFINE, an LLM-guided framework that compresses an already constructed agent memory store to a target storage budget while preserving information useful for future interactions. We first formalize storage-budgeted memory management as a budget-constrained max-min program over query-agnostic memory compression, and introduce MEMREFINE, an LLM-guided pairwise refinement procedure that operationalizes it.

3.1 Problem Formulation

Notation. Let $M_0 = \{m_i\}_{i=1}^{n}$ denote an already constructed memory store, where each entry m_i pairs textual content with an embedding vector and, depending on the host memory framework, may additionally carry links, timestamps, metadata, or other structured fields. We assume a downstream pipeline that, given a query q and a memory store M, retrieves a context $c = \texttt{Ret}(q; M)$ and produces an answer $a = \texttt{LLM}(q, c)$. Downstream utility is then measured by a task-specific score $U(q, M) \in [0, 1]$ that compares a against a reference answer (e.g., token-level F1 on factual recall).

Storage-Budgeted Memory Management. We formalize storage-budgeted memory management as the task of producing a compressed memory store M' from M_0 that fits within a fixed storage budget while remaining as useful as possible across an unknown future query distribution over a query space $\mathcal{Q}$. Formally, let $\mathcal{F}(M_0)$ denote the set of memory stores reachable from M_0 through edit operations, such as deletions and merges of entries, and let $\rho \in (0, 1]$ specify a storage budget ratio. Since the compressor cannot tailor M' to any particular query, we cast compression as a query-agnostic max-min program that hedges against the worst-case downstream query, defined as follows:

$$\begin{aligned} &\max_{M' \in \mathcal{F}(M_0)} \ \min_{q \in \mathcal{Q}} \ U(q, M') \\ &\text{s.t.} \quad \text{size}(M') \leq \rho \cdot \text{size}(M_0), \end{aligned} \tag{1}$$

where the inner minimization picks out the worst-case query under M', so that the outer maximization is forced to retain enough evidence for every plausible $q \in \mathcal{Q}$ rather than only the easy ones[1].

Query-Agnostic Compression. We note that directly optimizing Equation 1 is intractable: $\mathcal{Q}$ is unobserved at compression time, U requires running the full downstream pipeline for every candidate M', and $\mathcal{F}(M_0)$ is combinatorial in n. Yet the inner minimization itself implies a useful constraint on removable content: since the compressor should hedge against every plausible query, it can only safely discard entries that are either *redundant* (already covered by some other entry in M') or *non-factual* (carrying no information useful to any plausible query). This contrasts with inference-time prompt or context compressors (Jiang et al., 2023; Xu et al., 2024; Jiang et al., 2024), which use the incoming query as a selection signal to prune evidence at the token level. In contrast, we treat compression as a *preservation* problem and operationalize this view by replacing the intractable inner minimization with a tractable per-pair surrogate, namely a judge (powered by an LLM) that, for each pair of semantically related entries, decides whether their content is redundant, complementary, or distinct, and authorizes only those edits that preserve facts not redundantly stored elsewhere.

3.2 MEMREFINE

Pairwise Refinement. A direct alternative to the per-pair surrogate above would be to assign an importance score to each memory entry independently and remove low-scoring entries until the budget

[1] Equivalently, the dual form $\min_{M' \in \mathcal{F}(M_0)} \text{size}(M')$ subject to $U(q, M') \geq U(q, M_0) - \epsilon$ for all $q \in \mathcal{Q}$ expresses the same intent of reducing memory size while uniformly preserving downstream utility within a tolerance ϵ.

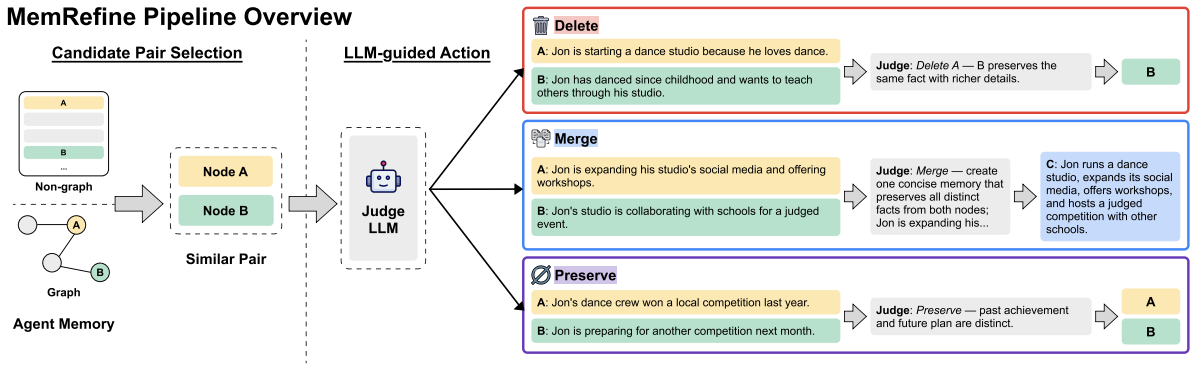

Figure 2: Overview of the MEMREFINE pipeline. MEMREFINE selects semantically similar memory pairs from an already constructed memory store, then uses an LLM judge to choose one of three actions: delete redundant memories, merge complementary memories, or preserve factually distinct memories. The resulting update refines the memory store under a storage budget while retaining information useful for future retrieval.

is met. However, this fails to capture the central property that motivates Equation 1: the value of a memory entry is *relational* rather than independent, since an entry may appear marginal in isolation yet become indispensable when combined with another; conversely, an entry may look high-value yet be safely removable because the same fact is already stored elsewhere in M'. These two failure modes look identical under any per-entry criterion (e.g., embedding norm, recency, or graph centrality), so single-entry scoring cannot tell when an edit removes a fact not redundantly covered elsewhere. To resolve this, MEMREFINE operates on *pairs* of semantically related memory entries: for any pair (u, v), the joint content decomposes exhaustively into a redundant case (the content of one is already covered by the other), a complementary case (each entry contributes content the other lacks), and a distinct case (the two entries assert unrelated facts), which directly induces a *delete–merge–preserve* action space matching the edits admitted by $\mathcal{F}(M_0)$. Furthermore, MEMREFINE realizes this by iterating over candidate pairs and selecting one action per pair until the budget is met or no unresolved candidate remains; Algorithm 1 summarizes the procedure, with each step detailed below.

Candidate Pair Selection. At each iteration, MEMREFINE selects the most similar pair of entries that has not yet been judged, denoted as follows: $(u, v) \leftarrow \arg\max_{(u,v)\notin\mathcal{S}} \cos(e_u, e_v)$, where e_u denotes the embedding of m_u and $\mathcal{S}$ caches pairs previously judged as PRESERVE to avoid revisiting them in subsequent iterations. Notably, similarity here serves only as a *proposal* mechanism: it surfaces pairs whose content is most likely

Algorithm 1 MEMREFINE compression procedure.

Require: Input memory store M_0, storage budget ratio ρ
Ensure: Compressed memory store M'

```
B ← ρ · size(M_0)
M' ← M_0
S ← ∅                                  ▷ Pairs judged as PRESERVE
while size(M') > B do
    (u, v) ← MostSimilarPair(M', S)
    if (u, v) = ∅ then
        break
    end if
    d ← Judge(u, v)
    if d.action = DELETE then
        Remove d.target from M'
        Prune dangling links if links exist
    else if d.action = MERGE then
        m ← Merge(u, v, d.instruction)
        Replace u and v with m in M'
        Inherit the union of links if links exist
        Recompute embedding for m
    else
        S ← S ∪ {(u, v)}
    end if
end while
return M'
```

to overlap, but does not determine whether either entry should be removed. This separation matters because high cosine similarity can equally indicate redundancy, complementarity, or merely topical overlap, all of which call for different actions.

LLM-Guided Action. The selected pair (u, v) is then passed to an LLM judge: $d \leftarrow \texttt{Judge}(u, v)$, which returns a structured decision d containing an action $d.\text{action} \in \{\text{DELETE}, \text{MERGE}, \text{PRESERVE}\}$, an optional deletion target, and (when the action is MERGE) a merge instruction specifying which facts should be retained. The judge is prompted to reason about the factual content of u and v, so that lexically similar entries with distinct facts are not collapsed, and lexically dissimilar entries that

nonetheless encode the same fact are recognized as redundant (See Appendix A.1 for the full prompt).

Store Update. Given the returned action, MEMREFINE triggers a corresponding update to M':

- DELETE: the judge-selected target is removed from M', and dangling edges incident to it are pruned if the memory framework contains links;
- MERGE: a separate merge LLM, conditioned on the judge instruction, synthesizes a single entry $m \leftarrow \texttt{Merge}(u, v, d.\text{instruction})$ that replaces both u and v in M', inheriting the union of their links (excluding duplicates and self-loops) along with a freshly recomputed embedding so that subsequent proposals reflect the merged content;
- PRESERVE: both entries remain in M' unchanged, and (u, v) is added to $\mathcal{S}$.

Termination. The loop continues until either the budget $\text{size}(M') \leq B$ is met or no unjudged candidate pair remains. This early-exit behavior reflects the design principle that factual preservation takes precedence over budget compliance: when all sufficiently similar pairs have already been judged PRESERVE, further compression would necessarily remove facts not redundantly stored elsewhere, violating the inner minimization in Equation 1.

Practical Implications. It is worth noting that the proposed MEMREFINE is designed as an offline, post-construction module that sits between memory construction and retrieval: all judge and merge calls are issued during memory maintenance rather than at query time, so compression reduces the persistent footprint of the store without inflating per-query retrieval or generation latency. It is also agnostic to the host memory representation, maintaining the link structure (pruning dangling edges on DELETE, inheriting the union of links on MERGE) for graph-structured frameworks (Xu et al., 2025) and operating directly on entries and their embeddings for non-graph stores (Chhikara et al., 2025); in both cases, the host construction and retrieval pipelines remain untouched.

4 Experimental Setup

Our evaluation tests whether an already constructed long-term memory store can be compressed under an explicit storage budget while preserving downstream utility. We vary three factors that directly stress this claim: the host memory framework, the compression budget, and the length or distribution of the interaction history.

Datasets. We primarily evaluate on LoCoMo, a long-term conversational memory benchmark whose questions require evidence from multi-session dialogues (Maharana et al., 2024). We use 10 LoCoMo samples containing 1,986 questions across five categories: single-hop factual recall, multi-hop reasoning, temporal reasoning, open-ended questions, and counterfactual questions. To stress-test longer histories, we also construct scaled LoCoMo-style datasets with approximately 3x and 10x longer conversations. These settings increase redundancy and topic recurrence, while also increasing the risk that rare facts are removed under compression. Finally, we evaluate on LongMemEval$_S$ (Wu et al., 2025), using 60 questions with 10 questions per type. We report accuracy for LongMemEval$_S$, with GPT-4o-mini used as the automatic judge.

Memory frameworks. We evaluate MEMREFINE on two memory frameworks with different representations. For A-MEM, we use an A-MEM-style factual graph representation that retains memory content, embeddings, and links, while excluding auxiliary context and tag fields (Xu et al., 2025). This isolates compression over factual graph memory; a supporting field analysis is provided in Appendix A. For Mem0, memory entries are constructed through its ingestion-and-update pipeline rather than preserved as raw dialogue nodes (Chhikara et al., 2025). In both settings, MEMREFINE is inserted after memory construction and before retrieval, leaving the host construction and retrieval pipelines unchanged.

Budgets and baselines. For each framework, the primary comparison is against the corresponding uncompressed Base Memory. We apply MEMREFINE under storage budgets of 70%, 60%, 50%, 40%, and 30% of the uncompressed store. On A-MEM-style graph memory, we additionally compare against rule-based baselines that remove the LLM judge: RULESIM, which uses embedding similarity as the action rule, and RULEPR, which incorporates graph structure through PageRank-style preservation. These baselines test whether similarity or graph centrality is sufficient to decide which facts should be deleted, merged, or preserved. Additional implementation details are provided in Appendix B, rule-based baseline details in Appendix C, threshold selection in Appendix D, prompts in Appendix A.1, and full model-family results in Appendix G.

Dataset	Framework	Metric	Base	70%	60%	50%	40%	30%
LoCoMo	A-MEM graph	F1	0.4013	**0.4014**	0.3977	0.3902	0.3844	0.3628
		EM	0.1712	0.1690	**0.1715**	0.1628	0.1615	0.1474
	Mem0	F1	0.2827	**0.2888**	0.2773	0.2761	0.2685	0.2510
		EM	0.1098	**0.1177**	0.1095	0.1132	0.1102	0.0994
3x LoCoMo	A-MEM graph	F1	0.2344	0.2307	**0.2324**	0.2163	0.2079	0.1900
		EM	0.0890	0.0893	**0.0923**	0.0879	0.0779	0.0702
	Mem0	F1	0.2204	0.2152	0.2065	**0.2085**	0.2013	0.1900
		EM	0.0588	0.0643	0.0621	**0.0651**	0.0599	0.0559
10x LoCoMo	A-MEM graph	F1	0.2053	**0.2033**	0.2006	0.1916	0.1776	0.1660
		EM	0.0786	0.0795	**0.0798**	0.0723	0.0659	0.0574
	Mem0	F1	0.2039	**0.1986**	0.1955	0.1899	0.1858	0.1800
		EM	0.0634	**0.0617**	0.0603	0.0600	0.0581	0.0564
LongMemEval$_S$	A-MEM graph	Acc.	0.5833	0.6000	0.6000	**0.6167**	0.5833	0.5500
	Mem0	Acc.	0.5167	0.4833	**0.5167**	0.4833	0.4500	0.4000

Table 1: Main storage–performance results across benchmarks, scaled conversation histories, and memory frameworks. LoCoMo-style datasets are evaluated with F1 and EM, while LongMemEval$_S$ is evaluated with accuracy. Bold values indicate the strongest compressed result for each dataset, framework, and metric. Full node counts, storage sizes, category-level results, and LongMemEval$_S$ type-level results are reported in the appendix.

5 Experimental Results and Analysis

We evaluate MEMREFINE by asking three questions. First, can an already constructed memory store be compressed without sacrificing downstream utility? Second, is factual judgment necessary for compression, or are similarity and graph structure sufficient? Third, does the same trade-off between storage and utility hold as conversations become longer or as the evaluation benchmark changes? We answer these questions with benchmark-level results, category-wise analysis, rule-based comparisons, model-family analysis, and scaled-history stress tests.

Overall results. Table 1 summarizes how storage budget affects performance across standard LoCoMo, scaled LoCoMo-style datasets, and LongMemEval$_S$. Across diverse settings, MEMREFINE reduces memory size with gradual changes in downstream performance as the storage budget becomes tighter, indicating a controllable balance between storage and utility.

Specifically, on standard LoCoMo, MEMREFINE reduces the size of both A-MEM-style graph memory and Mem0 while largely maintaining comparable downstream performance under moderate compression, suggesting that constructed memory stores contain redundant or fragmented entries. The scaled LoCoMo-style settings further test whether this pattern holds under more challenging conditions, as longer conversations introduce more redundancy while also increasing the risk of removing rare but useful facts. Even in these more challenging settings, performance generally changes gradually as the storage budget is reduced, suggesting that MEMREFINE can serve as a practical maintenance mechanism for growing interaction histories. On LongMemEval$_S$, a different long-term memory benchmark, A-MEM-style graph memory again benefits from moderate compression, while Mem0 is more sensitive to the chosen storage budget. This contrast suggests that the effective compression range depends on the host memory representation: A-MEM-style graph memory appears to contain more removable redundancy or fragmentation, whereas Mem0 may have already filtered or consolidated entries through its ingestion-and-update process. Importantly, MEMREFINE can still be applied flexibly across both memory stores, serving as a post-construction refinement layer rather than a framework-specific pruning rule.

Category-wise effects. The benchmark-level results show that moderate compression can maintain comparable overall performance, motivating a closer look at how this effect varies across question types shown. (Full numerical category results are reported in Appendix F.)

As shown in Figure 3, the effect of compression varies across question types. Specifically, in A-MEM-style graph memory, localized categories such as single-hop and temporal questions tend to improve after compression. This supports the intuition behind post-construction refinement: when repeated facts are removed or consolidated, retrieval

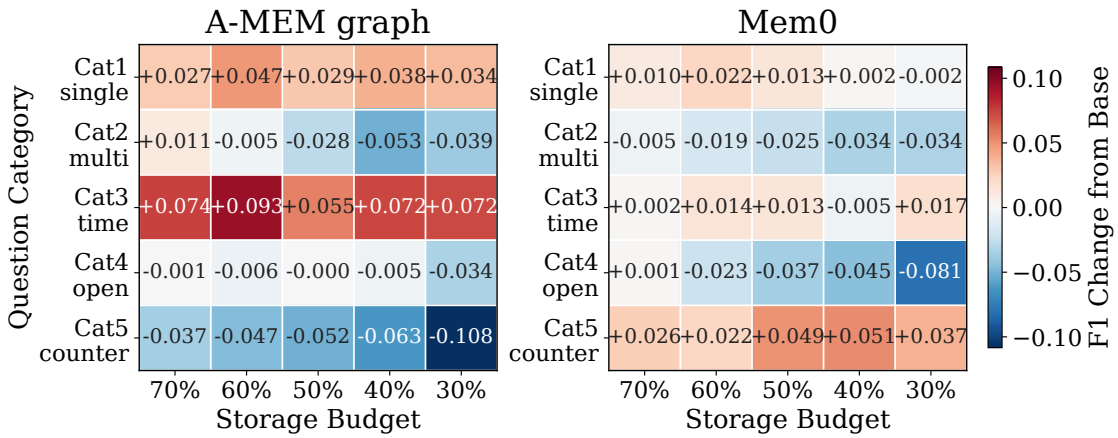

Figure 3: Category-wise F1 change under MEMREFINE compression on LoCoMo. Each value denotes the difference from the corresponding uncompressed Base Memory. Positive values indicate that compression improves category-level F1.

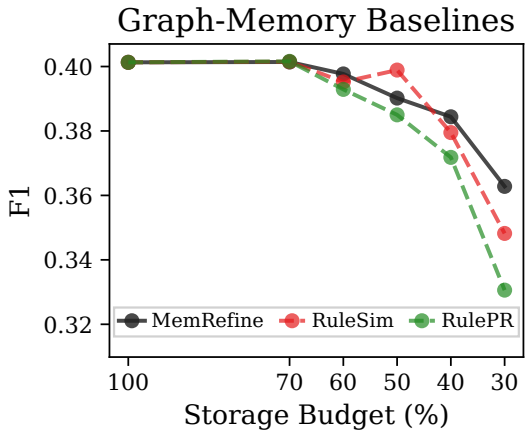

Figure 4: Comparison with rule-based baselines on A-MEM-style graph memory. The gap widens under tighter budgets, showing that similarity and graph-centrality heuristics are insufficient for factual compression decisions.

can become less noisy for questions whose answers depend on localized factual evidence. In contrast, open-ended and counterfactual questions are more vulnerable under tighter budgets because their evidence may be distributed across a wider portion of the memory store. Mem0 shows a related but distinct pattern across question types, likely because its ingestion-and-update pipeline transforms raw interactions into processed memory entries before MEMREFINE is applied. Together, these category-level results show that the effect of compression depends on both the question type and the memory representation produced by the host framework.

Comparison with rule-based graph-memory baselines. To isolate the role of the LLM judge, we compare MEMREFINE with rule-based graph-memory baselines that replace factual judgment with fixed heuristics. This comparison tests the central design choice of our method: semantic similarity is useful for proposing candidate pairs, but insufficient for deciding the compression action.

Figure 4 shows that rule-based methods can remain competitive when the budget is loose, where most compression decisions correspond to obvious redundancies. However, their performance degrades more sharply as the budget becomes tighter. This is expected: under strong compression, the difficult cases are no longer merely near-duplicate nodes, but semantically related memories whose factual relationship must be interpreted. A fixed similarity threshold cannot reliably distinguish redundant paraphrases from complementary facts, and graph centrality does not directly indicate whether a memory contains information needed for future questions. The advantage of MEMREFINE under tighter budgets therefore supports the need for LLM-guided factual action selection.

Effect of compression model. Because MEMREFINE relies on an LLM judge and merge model, we next test whether its behavior depends on a single model family. Figure 5 compares GPT-5-mini with Qwen3-8B while keeping the downstream retrieval and QA pipeline fixed. The two models show similar behavior under moderate compression, suggesting that MEMREFINE is not tied to a single proprietary judge model. However, the gap widens under tighter budgets, where compression requires finer factual distinctions between redundant, complementary, and distinct memories. This indicates that stronger judge models become more

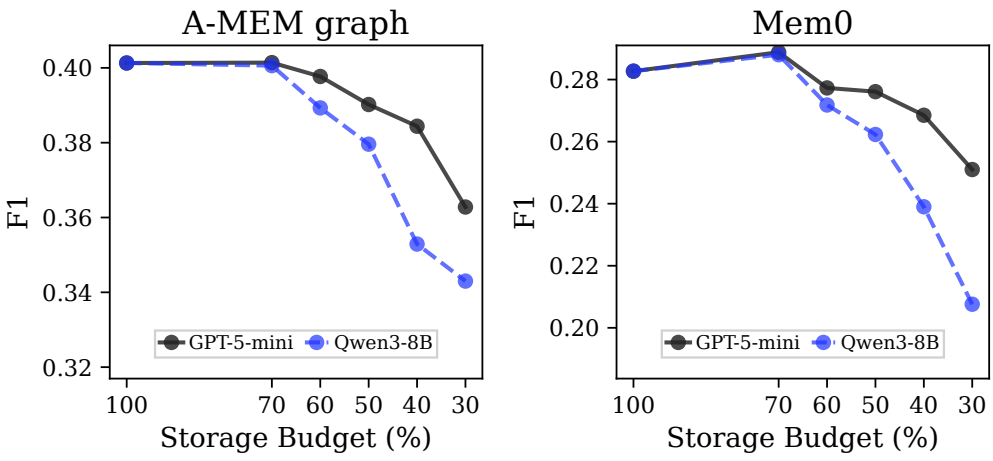

Figure 5: Effect of compression model. Qwen3-8B closely matches GPT-5-mini at 70% storage, while the gap increases under tighter storage budgets.

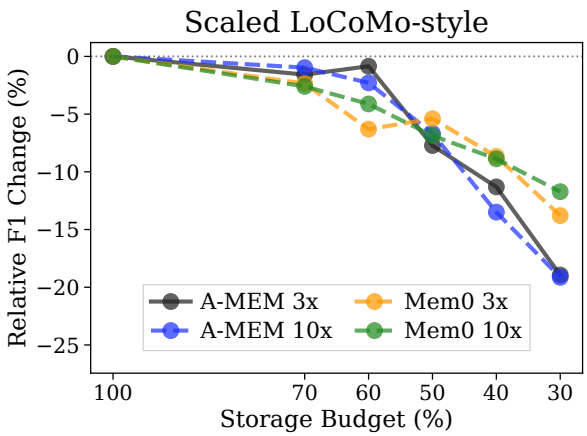

Figure 6: MEMREFINE on scaled LoCoMo-style datasets. The 3x and 10x settings contain substantially longer conversations than standard LoCoMo.

important as the storage constraint becomes severe. Full numerical results are reported in Appendix G.

Scaling to longer conversations. Figure 6 visualizes the degradation curve because the key question is whether compression fails abruptly as histories grow longer. The curve remains progressive across both 3x and 10x settings. This suggests that MEMREFINE can serve as a practical maintenance mechanism for long-running agents: reducing storage pressure exposes a smooth compactness–utility trade-off rather than an unpredictable failure point.

Takeaways. Together, the results show that accumulated memory contains compressible redundancy, but effective compression requires factual action selection rather than uniform pruning. MEMREFINE is most useful when compression is treated as memory refinement: removing redundant facts, consolidating complementary facts, and preserving distinct ones.

6 Conclusion

In this paper, we proposed MEMREFINE, a post-construction memory compression framework designed to keep an already constructed long-term agent memory store within a fixed storage budget while preserving information useful for future interactions. Through a pairwise refinement procedure that uses similarity only to surface candidate pairs and defers the delete, merge, and preserve decision to an LLM judge grounded in factual content, MEMREFINE can be inserted between memory construction and retrieval without modifying the host memory construction pipeline. Empirical evaluations across A-MEM-style graph memory, Mem0, scaled LoCoMo-style settings, and LongMemEval$_S$ demonstrate that moderate compression preserves downstream utility while tighter budgets expose a gradual storage-utility trade-off, with the gap over rule-based baselines widening precisely as the budget tightens. Also, our analyses highlight that the amount of safely removable content is shaped not only by the compressor but also by the host memory representation, with A-MEM-style graph memory exposing more removable redundancy and Mem0, having already filtered and consolidated entries during ingestion, leaving less room before utility is affected. We believe these findings establish post-construction compression as a practical maintenance layer for storage-budgeted long-term agent memory, suggesting that compression and construction should be considered jointly rather than as independent stages.

Limitations

The proposed MEMREFINE is designed as a post-construction module that compresses an already constructed long-term memory store to a target storage budget while preserving information useful for downstream tasks. It is worth noting that our evaluation centers on two representative memory frameworks (an A-MEM-style graph memory of linked structured notes and the Mem0 ingestion-and-update pipeline) and on long-term conversational benchmarks, which together cover both graph-structured and processed memory stores but do not exhaust the space of agent memory systems; extending post-construction compression to additional memory architectures and deployed multi-agent or tool-use settings would be a valuable direction for future work. Also, our scaled LoCoMo-style benchmarks serve as controlled stress tests for studying how compression behaves as conversation length and redundancy grow, and evaluating MEMREFINE on real longitudinal memory logs collected over extended deployments would be an exciting next step for its practical applicability.

Ethical Considerations

The proposed MEMREFINE edits an already constructed long-term memory store through DELETE and MERGE operations, which persistently remove or consolidate past interactions and, in deployed systems, may involve sensitive user information. While our experiments use public research benchmarks and synthetically generated scaled LoCoMo-style datasets without collecting or releasing any real user data, in practice, compression should be applied under appropriate privacy safeguards, access control, and data-retention policies, and we further recommend pairing it with mechanisms such as soft deletion or audit logs of applied edits so that maintenance decisions remain reviewable.

References

Sanghwan Bae, Dong-Hyun Kwak, Soyoung Kang, Min Young Lee, Sungdong Kim, Yuin Jeong, Hyeri Kim, Sang-Woo Lee, Woo-Myoung Park, and Nako Sung. 2022. Keep me updated! memory management in long-term conversations. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3769–3787, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117.

Nuo Chen, Hongguang Li, Jianhui Chang, Juhua Huang, Baoyuan Wang, and Jia Li. 2025. Compress to impress: Unleashing the potential of compressive memory in real-world long-term conversations. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 755–773, Abu Dhabi, UAE. Association for Computational Linguistics.

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

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. 2025. LightMem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*.

Jihyoung Jang, Minseong Boo, and Hyounghun Kim. 2023. 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*, pages 13584–13606, Singapore. Association for Computational Linguistics.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore. Association for Computational Linguistics.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677, Bangkok, Thailand. Association for Computational Linguistics.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. 2026. SimpleMem: Efficient lifelong memory for LLM agents. *arXiv preprint arXiv:2601.02553*.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. 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)*, pages 13851–13870, Bangkok, Thailand. Association for Computational Linguistics.

Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. 2023. RET-LLM: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322*.

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

Junxi Wang, Te Sun, Jiayi Zhu, Junxian Li, Haowen Xu, Zichen Wen, Xuming Hu, Zhiyu Li, and Linfeng Zhang. 2026. StreamMeCo: Long-term agent memory compression for efficient streaming video understanding. *arXiv preprint arXiv:2604.09000*.

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

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations*. OpenReview.net.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024. RECOMP: Improving retrieval-augmented LMs with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*.

Jing Xu, Arthur Szlam, and Jason Weston. 2022. 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)*, pages 5180–5197, Dublin, Ireland. Association for Computational Linguistics.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems 38*.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence*, pages 19724–19731. AAAI Press.

Memory Representation	Size	F1	EM
Original A-MEM	0.5430	0.3718	0.1385
Base Memory	0.2692	0.4013	0.1712

Table 2: Representative field-analysis result on LoCoMo. Base Memory retains memory content, embeddings, and links while excluding auxiliary context and tag fields. Size is reported in MB.

A A-MEM Input Representation

A-MEM represents each memory as a structured note with multiple fields, including content, contextual descriptions, keywords, tags, embeddings, and links. For the graph-based memory setting, we use a simplified A-MEM-style representation that retains memory content, embeddings, and links while excluding auxiliary context and tag fields. This design focuses the experiment on compressing factual memory entries and their graph structure, rather than evaluating A-MEM's memory evolution fields. All A-MEM-based compression methods therefore start from the same Base Memory representation.

This field analysis suggests that the auxiliary context and tag fields do not provide useful evidence for the downstream retrieval setting used in our experiments, and may introduce additional retrieval noise. We therefore use the simplified factual graph representation consistently across all A-MEM-style experiments.

A.1 Prompt Details

MEMREFINE uses two LLM calls during compression: a judge LLM and a merge LLM. For both calls, each memory content field is truncated to the first 2,000 characters before being inserted into the prompt. The judge returns one of `delete`, `merge`, or `nothing`; the implementation maps `nothing` to the PRESERVE action described in the main paper.

A.1.1 Judge Prompt

```
You are a memory compression assistant. Your goal is to
↪  reduce memory size while preserving information that is
↪  useful for answering future questions about the
↪  conversation.

Two memory nodes have cosine similarity {similarity}.

[Node A]
{content_a}

[Node B]
{content_b}

IMPORTANT -- Judge by FACTUAL INFORMATION VALUE, not by
↪  surface wording:
- Facts worth preserving: names, dates, events, locations,
↪  preferences, plans, opinions, personal experiences,
↪  relationships
- Low-value content: greetings ("Hey!", "Good to talk"),
↪  generic thanks ("Thanks!", "Appreciate it"), small
↪  talk, emotional filler without specific facts
- Two nodes that only differ in wording but carry the same
↪  facts -> delete one
- If both nodes are low-value (e.g. both are just greetings
↪  or generic thanks) -> delete one, don't merge

Choose one action:
1. "delete" -- One node is redundant OR both carry the same
↪  facts (even if worded differently). Delete the less
↪  informative one.
   -> Specify which to delete ("a" or "b"). Keep the one
   ↪  with more specific facts.
2. "merge" -- Both contain DISTINCT FACTS that would be
↪  lost if either were deleted. Combine into one.
   -> Write a merge instruction specifying which FACTS (not
   ↪  phrasings) from each node must be preserved.
3. "nothing" -- Both contain clearly distinct, important
↪  factual information. Keep both.

Bias toward "delete" when nodes only differ in wording.
↪  Only choose "merge" when there are genuinely different
↪  facts in each node.

Return JSON:
- action: "delete", "merge", or "nothing"
- target: "a" or "b" (which to delete; set "a" if not
↪  deleting)
- merge_instruction: instruction for merge LLM (set "" if
↪  not merging)
- reason: brief explanation
```

A.1.2 Merge Prompt

```
You are merging two memory nodes into one. Follow the
↪  instruction below.

[Merge Instruction]
{instruction}

[Node A]
{content_a}

[Node B]
{content_b}

Rules:
- Preserve all FACTS (names, dates, events, locations,
↪  preferences, plans, experiences) from both nodes
- Drop generic phrases, greetings, and emotional filler
↪  that don't carry factual content
- Keep it concise -- shorter is better as long as no facts
↪  are lost
- Maintain speaker attribution (who said what) when it
↪  matters for the fact

Return JSON:
- content: the merged memory content
- keywords: list of keywords for the merged memory
```

B Implementation Details

Memory construction and retrieval. Unless otherwise stated, we follow the original A-Mem memory construction pipeline and represent each memory as a graph node with its content and metadata. For candidate pair selection, we embed each memory entry using `all-MiniLM-L6-v2` and compute pairwise cosine similarities over the resulting embeddings. For question answering, we use the default retriever of the memory framework with top-k = `10` retrieved memories. To control the context length and cost of LLM-based decisions, we truncate each memory content field to the first 2,000 characters before passing it to the judge and merge models.

Memory size accounting. We measure memory size by serializing the stored memory entries into JSON with `ensure_ascii=False`. For our compressed variants, memory size is computed over the 8 fields used by our implementation. The original A-Mem output additionally contains `context`, `category`, `tags`, and `retrieval_count`, resulting in 12 serialized fields. For ratio-based compression, the target memory budget for each sample is set to r times the corresponding no-compression memory size, where $r \in \{0.3, 0.4, 0.5, 0.6, 0.7\}$.

Rule-based compression. For the rule-based delete-then-merge compressor, we first identify redundant memory pairs whose cosine similarity is at least a threshold τ. We evaluate $\tau \in \{0.70, 0.80, 0.90\}$ for ratio-based budgets and additionally use $\tau \in \{0.75, 0.85\}$ for absolute-budget experiments. In the PR-based setting, PageRank is computed on the memory graph and the higher-PageRank memory is preferentially preserved within redundant clusters. We use PR protection percentile 0.9 unless otherwise specified, which protects the top 10% PageRank nodes from being selected as merge anchors. If deletion alone does not satisfy the target memory budget, we fall back to the A-Mem-style merge operation, where low-PageRank anchor nodes and their neighbors are summarized into a single memory node. All compression operations preserve the original timestamps associated with the affected memories.

LLM-judged compression. For the LLM-judged compressor, we remove the fixed similarity threshold and instead sort candidate memory pairs by cosine similarity in descending order. For each pair, the judge model decides one of three actions: `delete`, `merge`, or `nothing`. For `delete`, the judge also selects which memory should be removed. For `merge`, the judge produces a merge instruction, and a separate merge model generates the merged memory following that instruction. For `nothing`, the pair is skipped and is not queried again. Compression stops when the target memory budget is reached or when no remaining pair receives a compressive action. Unless otherwise noted, both the judge and merge models are `gpt-5-mini`.

Datasets. For LoCoMo, we evaluate on 10 samples, containing 1,986 questions across five categories: single-hop, multi-hop, temporal, open-ended, and counterfactual questions. For LongMemEval$_S$, we evaluate on 60 questions across six types: single-user, single-assistant, single-preference, multi-session, temporal, and knowledge-update questions. To stress-test longer histories, we construct scaled LoCoMo-style datasets at two scales (3x and 10x): the 3x setting comprises 5 long-dialogue samples (80 sessions and 1,600 user-assistant turns each, 2,720 QA pairs in total), and the 10x setting comprises 5 samples (270 sessions and 5,400 turns each, 9,180 QA pairs in total); full statistics are reported in Appendix J. For both scales, personas and events are generated with `gpt-5-mini`, while dialogues are generated with `gpt-4o-mini`.

Evaluation. For LoCoMo and the scaled LoCoMo-style datasets, we report token-level F1 and exact match (EM) following the LoCoMo evaluation protocol (Maharana et al., 2024). We also report category-wise F1 for the five LoCoMo question categories. For LongMemEval$_S$, we report accuracy. Unless otherwise noted, answer evaluation for the LLM-judged compression experiments uses `gpt-5-mini`. For compression-model analysis, we use `Qwen3-8B`. All reported results are from a single compression and evaluation run for each setting due to the cost of API-based judge, merge, and evaluation calls; therefore, we do not report confidence intervals or error bars.

C Rule-based Graph-memory Baselines

We evaluate two rule-based baselines on A-MEM-style graph memory. These baselines are used only in the graph-memory setting because they rely on graph structure or fixed similarity rules over graph nodes.

C.1 RuleSim

RuleSim uses the same pair-selection loop as MemRefine but replaces the LLM judge with a fixed similarity rule. At each step, it selects the most similar unresolved memory pair (u, v) with cosine similarity $s(u, v)$. The compression action is determined as follows:

$$\begin{cases} \text{DELETE}, & s(u,v) \geq \tau_{\text{delete}}, \\ \text{MERGE}, & s(u,v) < \tau_{\text{delete}}. \end{cases}$$

If the selected action is DELETE, one node is removed according to a deterministic deletion rule. If the selected action is MERGE, the same merge LLM used by MemRefine generates the merged memory. Thus, RuleSim isolates the effect of

Method	Storage	Nodes	Size	F1
Base	100%	588.2	0.2692	0.4013
MEMREFINE	70%	386.4	0.1888	0.4014
	60%	318.9	0.1618	0.3977
	50%	257.8	0.1349	0.3902
	40%	197.5	0.1079	0.3844
	30%	141.3	0.0811	0.3628
RULESIM	70%	–	–	0.4015
	60%	–	–	0.3952
	50%	–	–	0.3989
	40%	–	–	0.3795
	30%	–	–	0.3482
RULEPR	70%	347.4	0.1882	0.4016
	60%	268.1	0.1604	0.3929
	50%	200.0	0.1340	0.3850
	40%	135.9	0.1069	0.3718
	30%	80.6	0.0793	0.3306

Table 3: Full graph-memory baseline comparison on LoCoMo. Size is reported in MB when available.

Storage	$\tau = 0.90$	$\tau = 0.80$	$\tau = 0.70$
70%	0.3880	0.4016	0.3763
60%	0.3687	0.3929	0.3740
50%	0.3634	0.3850	0.3828
40%	0.3484	0.3718	0.3633
30%	0.3347	0.3306	0.3330

Table 4: Threshold sensitivity for the PageRank-based rule baseline on A-MEM-style graph memory. We report F1 under each storage budget.

replacing the LLM judge with a fixed similarity decision rule.

C.2 RULEPR

RULEPR combines similarity-threshold deletion with PageRank-guided preservation and merge fallback. It first identifies near-duplicate memory nodes whose cosine similarity exceeds a threshold. Within each redundant group, PageRank is used as a graph-centrality proxy to decide which node should be preserved. If deletion alone is insufficient to satisfy the storage budget, RULEPR applies a merge fallback guided by the graph structure.

This baseline tests whether graph centrality is a useful substitute for LLM-guided factual compression decisions. Unlike MEMREFINE, RULEPR does not reason over factual redundancy or complementarity with an LLM judge.

D Threshold Selection for Rule-based Baselines

For rule-based graph-memory baselines, we use $\tau = 0.8$ in the main experiments. This value is selected based on threshold sweeps over the ratio-based graph-memory compression setting. Table 4 shows that $\tau = 0.8$ performs best at 70%, 60%, 50%, and 40% storage, while also providing a stable choice across storage budgets. Although $\tau = 0.9$ is slightly better at 30% storage, we use $\tau = 0.8$ as the main setting because it is the most consistently strong threshold across budgets.

Retention	Storage	Nodes	Size	F1
PageRank	70%	417.5	0.1857	0.3111
Random	70%	411.4	0.1814	0.3085
PageRank	50%	294.5	0.1314	0.2673
Random	50%	293.7	0.1302	0.2848
PageRank	30%	176.8	0.0785	0.2098
Random	30%	175.9	0.0778	0.2374

Table 5: Retention-only baselines on A-MEM-style graph memory. PageRank centrality is not a reliable criterion for selecting useful memories, and both retention-only baselines underperform compression-based methods. Size is reported in MB.

E Retention-only Baselines

We also compare retention-only baselines that keep a fixed percentage of existing graph nodes without performing delete–merge compression. The PageRank retention baseline keeps the top-ranked nodes according to graph centrality, while the random retention baseline keeps a random subset of nodes. Table 5 shows that PageRank retention does not provide a reliable advantage over random retention. Both retention-only baselines are substantially worse than compression-based methods, indicating that memory compression is not simply a node-selection problem.

F Category-wise LoCoMo Results

The main paper visualizes category-wise F1 changes under compression on LoCoMo. Here, Table 6 reports the full numerical values for each question category and storage budget.

G Compression Model Analysis

The main paper visualizes the effect of changing the judge and merge model. Here, Table 7 reports the full numerical results comparing GPT-5-mini with Qwen3-8B while keeping the downstream retrieval and QA pipeline fixed. The results show that smaller open models can match the main compression pattern under moderate budgets, but stronger

judge models are more robust when the storage budget becomes tight.

H LongMemEval$_S$ Full Results

The main paper summarizes LongMemEval$_S$ accuracy across storage budgets. Table 8 reports the full type-level results across all budgets. LongMemEval$_S$ contains 60 questions, with 10 questions per type. Accuracy is judged using GPT-4o-mini.

I Full Scaled LoCoMo-style Results

The main paper reports scaled LoCoMo-style results as a storage–performance curve. Table 9 provides the full numerical values for the 3x and 10x settings.

J Scaled LoCoMo-style Dataset Details

We construct scaled LoCoMo-style datasets to evaluate memory compression under longer conversational histories. Because the scaled datasets are synthetically generated, we verify that they preserve the broad structure of the original benchmark while increasing conversation length and redundancy. Table 10 summarizes their basic statistics, text-level diversity indicators, and QA category distributions. The 3x setting is a relatively faithful scaled version of LoCoMo, while the 10x setting serves as a stronger stress test with longer and more repetitive interaction histories.

K Artifact Licenses and Use

We use publicly available research artifacts only for research and evaluation. LoCoMo is released under a Creative Commons Attribution-NonCommercial license. LongMemEval is distributed as a public research benchmark through its official repository and dataset release. For memory frameworks, the A-MEM implementation is released under the MIT License, and Mem0 is released under the Apache License 2.0. Our use of these artifacts is limited to benchmark evaluation and method comparison, and we do not redistribute the original benchmark data or framework code as part of this paper. The scaled LoCoMo-style datasets are synthetic stress-test variants constructed for analysis of longer conversational histories.

Framework	Storage	Cat1	Cat2	Cat3	Cat4	Cat5
A-MEM graph	100%	0.2961	0.4776	0.1754	0.4612	0.3310
A-MEM graph	70%	0.3235	0.4882	0.2498	0.4607	0.2938
A-MEM graph	60%	0.3434	0.4727	0.2687	0.4550	0.2842
A-MEM graph	50%	0.3251	0.4492	0.2303	0.4611	0.2789
A-MEM graph	40%	0.3345	0.4251	0.2477	0.4560	0.2681
A-MEM graph	30%	0.3300	0.4383	0.2470	0.4269	0.2226
Mem0	100%	0.2686	0.1246	0.1180	0.4266	0.1703
Mem0	70%	0.2785	0.1194	0.1196	0.4276	0.1964
Mem0	60%	0.2908	0.1060	0.1316	0.4036	0.1927
Mem0	50%	0.2818	0.0996	0.1311	0.3897	0.2194
Mem0	40%	0.2707	0.0904	0.1125	0.3812	0.2217
Mem0	30%	0.2670	0.0908	0.1354	0.3456	0.2077

Table 6: Category-wise F1 results on LoCoMo. Cat1–Cat5 correspond to the LoCoMo question categories: single-hop, multi-hop, temporal, open-ended, and counterfactual questions.

Framework	Model	100%	70%	60%	50%	40%	30%
A-MEM graph	GPT-5-mini	0.4013	0.4014	0.3977	0.3902	0.3844	0.3628
A-MEM graph	Qwen3-8B	0.4013	0.4006	0.3893	0.3796	0.3529	0.3430
Mem0	GPT-5-mini	0.2827	0.2888	0.2773	0.2761	0.2685	0.2510
Mem0	Qwen3-8B	0.2827	0.2880	0.2718	0.2623	0.2390	0.2076

Table 7: Full compression-model comparison. Values indicate F1 under each storage budget.

System	Storage	Acc.	N	s-user	s-asst	s-pref	multi	temporal	k-update
A-MEM graph	Base	0.5833	60	9/10	10/10	3/10	4/10	3/10	6/10
A-MEM graph + MemRefine	70%	0.6000	60	9/10	9/10	5/10	4/10	3/10	6/10
A-MEM graph + MemRefine	60%	0.6000	60	10/10	9/10	5/10	5/10	1/10	6/10
A-MEM graph + MemRefine	50%	0.6167	60	10/10	8/10	5/10	6/10	2/10	6/10
A-MEM graph + MemRefine	40%	0.5833	60	10/10	10/10	5/10	4/10	1/10	5/10
A-MEM graph + MemRefine	30%	0.5500	60	8/10	7/10	3/10	6/10	3/10	6/10
Mem0	Base	0.5167	60	8/10	4/10	6/10	5/10	4/10	4/10
Mem0 + MemRefine	70%	0.4833	60	7/10	3/10	5/10	5/10	4/10	5/10
Mem0 + MemRefine	60%	0.5167	60	8/10	3/10	6/10	6/10	3/10	5/10
Mem0 + MemRefine	50%	0.4833	60	8/10	3/10	6/10	5/10	4/10	3/10
Mem0 + MemRefine	40%	0.4500	60	8/10	3/10	5/10	4/10	3/10	4/10
Mem0 + MemRefine	30%	0.4000	60	7/10	4/10	4/10	3/10	3/10	3/10

Table 8: Full LongMemEval$_S$ results. The benchmark contains 60 questions, with 10 questions per type. Accuracy is judged using GPT-4o-mini.

Framework	Dataset	Storage	Nodes/Entries	F1	EM	Rel. F1 Change
A-MEM graph	3x	100%	1600.0	0.2344	0.0890	–
		70%	1038.2	0.2307	0.0893	-1.6%
		60%	856.0	0.2324	0.0923	-0.9%
		50%	701.6	0.2163	0.0879	-7.7%
		40%	552.4	0.2079	0.0779	-11.3%
		30%	398.6	0.1900	0.0702	-18.9%
A-MEM graph	10x	100%	5400.0	0.2053	0.0786	–
		70%	3492.0	0.2033	0.0795	-1.0%
		60%	2908.0	0.2006	0.0798	-2.3%
		50%	2372.0	0.1916	0.0723	-6.7%
		40%	1878.0	0.1776	0.0659	-13.5%
		30%	1386.0	0.1660	0.0574	-19.1%
Mem0	3x	100%	1241.6	0.2204	0.0588	–
		70%	868.6	0.2152	0.0643	-2.4%
		60%	744.4	0.2065	0.0621	-6.3%
		50%	620.6	0.2085	0.0651	-5.4%
		40%	496.2	0.2013	0.0599	-8.7%
		30%	372.0	0.1900	0.0559	-13.8%
Mem0	10x	100%	3531.0	0.2039	0.0634	–
		70%	2471.4	0.1986	0.0617	-2.6%
		60%	2118.2	0.1955	0.0603	-4.1%
		50%	1765.2	0.1899	0.0600	-6.9%
		40%	1412.0	0.1858	0.0581	-8.9%
		30%	1058.6	0.1800	0.0564	-11.7%

Table 9: Full MEMREFINE results on scaled LoCoMo-style datasets. The 3x and 10x settings contain approximately three and ten times longer conversations than standard LoCoMo, respectively. Node or entry counts are reported as the average number of stored memory units after compression.

(a) Basic statistics

Dataset	Samples	Sessions	Turns	QAs	Avg. Turns/Sess.
LoCoMo	10	272	5,882	1,986	21.6
3x	5	400	8,000	2,720	20.0
10x	5	1,350	27,000	9,180	20.0

(b) QA category distribution

Category	LoCoMo	3x	10x
Cat1 single-hop	14.2%	14.0%	14.0%
Cat2 multi-hop	16.2%	16.0%	16.0%
Cat3 temporal	4.8%	5.0%	5.0%
Cat4 open-ended	42.3%	43.2%	43.1%
Cat5 counterfactual	22.5%	21.9%	21.9%

(c) Text and topic diversity indicators

Dataset	Turn Len.	Word TTR	Session Sim.	Adjacent Sim.	3-gram Repeat	QA TTR	Ref. Len.
LoCoMo	124	0.0386	0.3460	0.4537	0.316	0.1004	23
3x	137	0.0304	0.2931	0.4253	0.319	0.0558	83
10x	136	0.0128	0.3300	0.4496	0.451	0.0248	82

Table 10: Quality analysis of the scaled LoCoMo-style datasets. Panel (a) reports basic dataset statistics, panel (b) shows that the QA category distribution is preserved across the original and scaled datasets, and panel (c) reports text-level and topic-level diversity indicators. Turn length and reference length are measured in characters. Word TTR and QA TTR denote type-token ratios for dialogue turns and questions, respectively.