LOCALMEM: LEARNING TO ARCHIVE AND RETRIEVE LONG-TERM MEMORY FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-term interactive agents need external memory to retain information across interactions and recover information required by later queries. Existing systems commonly use fixed memory-management rules and general-purpose large language models (LLMs), e.g., GPT-4o, for memory operations. However, fixed rules often store or retrieve misaligned memory evidence, while reliance on cloud-hosted LLMs raises operating costs and privacy risks. We present LocalMem, a plug-and-play memory-management system with separate local language models for memory construction and retrieval. For memory construction, a 4B reinforced Memory Archiver is optimized with the proposed unified memory reward to learn what information to preserve. For memory retrieval, a 4B SFT-trained Memory Enquirer uses progressive retrieval to acquire additional evidence only when needed, improving the accuracy-efficiency trade-off. The proposed LocalMem achieves the state-of-the-art (SOTA) performance, i.e., 69.3 Overall$_5$ F1 on LoCoMo10 and 81.8% LLM-as-a-Judge accuracy on LongMemEval-S. Controlled ablations show that the trained 4B Archiver outperforms a prompted 122B Archiver by 4.1 F1, while progressive retrieval improves F1 by 1.5 over fixed top-k retrieval.

1 INTRODUCTION

Long-term interactive agents accumulate information across interactions, but large language models (LLMs) do not inherently maintain persistent memory and can only access information available in the current context window. External memory systems address this limitation by storing information beyond the active context window (Packer et al., 2023; Chhikara et al., 2025). Such systems face two key challenges: what information to preserve and how to retrieve enough evidence for later user queries. Existing systems commonly use fixed memory-management rules and general-purpose LLMs for these operations. However, fixed rules often store or retrieve misaligned memory evidence, while reliance on cloud-hosted LLMs raises operating costs and privacy risks. These limitations motivate learning memory construction and retrieval with local models without sacrificing downstream effectiveness.

The first challenge concerns **memory construction**: deciding what information to preserve for future queries. Many memory systems rely on fixed memory-management rules for extraction (Liu et al., 2026b; Kang et al., 2025), which are not directly optimized for downstream utility. Such rules may retain redundant details while omitting facts needed to answer future user queries. Recent approaches instead learn memory construction by repeatedly updating an evolving memory state through insertion, update, and deletion (Wang et al., 2025; Yan et al., 2026; Yue et al., 2026), which couples each construction step to the previously formed memory. In contrast, LocalMem formulates memory construction as streaming window-to-entry generation. Specifically, the 4B Memory Archiver reads only the current interaction window, produces structured entries, and does not edit previously stored memory. We further propose a unified memory reward that evaluates the memory jointly formed by all windows and assigns downstream QA utility to the corresponding window-level outputs.

The second challenge concerns **memory retrieval**: deciding how much evidence to retrieve for the current query. Many memory systems use a fixed top-k retrieval budget (Chhikara et al., 2025; Zhong et al., 2024; Gutiérrez et al., 2024), which may waste context on simple queries while missing

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.

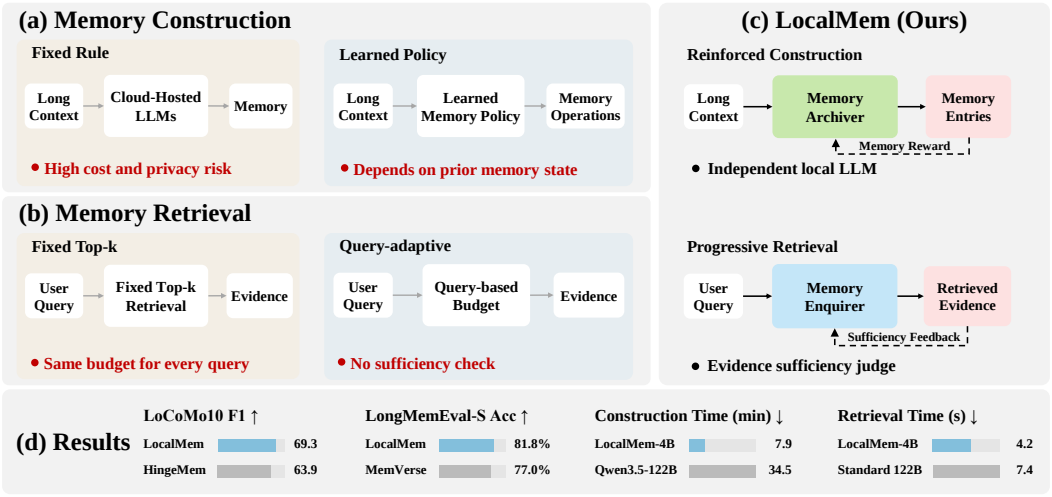


Figure 1: LocalMem in one view. **(a)** Existing memory construction commonly relies on fixed rules or learned sequential memory operations, often implemented with general-purpose LLMs. **(b)** Existing memory retrieval commonly uses fixed or query-adaptive budgets. **(c)** LocalMem uses separate local language models for memory construction and progressive retrieval, forming a plug-and-play memory-management system that is decoupled from the Answer Model. **(d)** LocalMem achieves SOTA performance on LoCoMo10 and LongMemEval-S.

required evidence for complex ones. Recent systems adapt retrieval depth or computation according to query type (Zhang et al., 2026b; Zhong et al., 2026). However, these methods mainly set the budget from query-side signals, rather than checking whether the evidence retrieved so far is sufficient. In contrast, LocalMem formulates memory retrieval as progressive evidence acquisition: it starts with a small set of retrieved entries, and a frozen sufficiency judge decides after each retrieval stage whether more evidence is needed. Only when the evidence is insufficient does the 4B Memory Enquirer expand retrieval across multiple memory views or target the information that is still missing.

Together, these two components form **LocalMem**, a plug-and-play memory-management system that addresses both challenges with separate local language models. The **Memory Archiver** is trained with Group Relative Policy Optimization (GRPO) (Shao et al., 2024) using the proposed unified memory reward, and the **Memory Enquirer** is trained with supervised fine-tuning (SFT). Although both memory models contain only 4B parameters, they outperform their prompted 122B counterparts in downstream QA. Running them locally also reduces the cost of processing growing interaction histories and avoids sending raw interactions to cloud-hosted models. The frozen LLM used for sufficiency judging and answer generation receives only the memory entries retrieved for each query, rather than the raw interaction history. LocalMem therefore serves as a plug-and-play memory system whose trained memory models can be paired with different Answer Models without retraining. With Qwen3.5-122B as the Answer Model, LocalMem achieves state-of-the-art (SOTA) performance, with **69.3 Overall$_5$ F1** on LoCoMo10 and **81.8% LLM-as-a-Judge accuracy** on LongMemEval-S.

Our contributions are summarized as follows:

1. We present LocalMem, a plug-and-play memory-management system that learns memory construction and retrieval with separate local language models, keeping raw interaction histories local and decoupling memory management from the Answer Model.
2. We introduce a unified memory reward for learning what information to preserve from downstream QA utility, together with progressive retrieval that acquires additional evidence according to evidence sufficiency.
3. We provide extensive experiments and controlled ablations showing that LocalMem achieves SOTA performance on LoCoMo10 and LongMemEval-S, i.e., **69.3 Overall$_5$ F1** on LoCoMo10 and **81.8%** LLM-as-a-Judge accuracy on LongMemEval-S.

2 Related Work

Long-term memory for LLM agents. Long-term memory enables LLM-based agents to retain and reuse information across extended interactions beyond the context window (Hu et al., 2025; Huang et al., 2026; Liu et al., 2025a). Existing systems organize interaction history into persistent memory, including summarized or fact-based memories (Zhong et al., 2024; Chhikara et al., 2025), structured knowledge units (Yang et al., 2026), associative or graph-based memory (Xu et al., 2025; Gutiérrez et al., 2024; Gutiérrez et al., 2025; Ji et al., 2026), and topic- or episode-oriented hierarchical structures (Fang et al., 2026; Wang et al., 2026a). SimpleMem (Liu et al., 2026b) compresses interactions into compact multi-view memory units and consolidates related memories for efficient long-term access. Recent work also considers the runtime overhead of continuous memory management. LightMem (Fang et al., 2026) reduces online cost through lightweight filtering, topic-aware consolidation, and offline updates. Other resource-constrained systems explore memory processing with small models (Wang et al., 2026b; Lee et al., 2026; Fu et al., 2025), while Zero-Mem (Xiao et al., 2026) removes LLM calls from memory operations.

Learnable memory management. Recent work learns memory construction, update, or utilization directly from downstream task feedback rather than relying on fixed rules. For persistent external memory, Mem-α (Wang et al., 2025) trains a memory-construction policy with reinforcement learning using downstream QA reward. Memory-R1 (Yan et al., 2026) learns structured memory operations together with an Answer Agent, while Mem-T (Yue et al., 2026) uses Memory-Operation-Tree-guided GRPO with step-wise credit assignment to jointly optimize memory construction and iterative retrieval. Related work also learns memory behavior in long-context or iterative reasoning settings. MemAgent (Yu et al., 2026) learns to maintain a fixed-length memory state while processing segmented long contexts, MEM1 (Zhou et al., 2026) jointly learns reasoning and memory consolidation through an evolving internal state, and MemSearcher (Yuan et al., 2025) jointly learns search and iterative memory integration through reinforcement learning. MemEvolve (Zhang et al., 2026a) further explores automatic evolution of the memory architecture. In contrast, LocalMem avoids sequential memory editing and decouples memory construction from query-time retrieval with independently trained 4B Memory Archiver and Memory Enquirer.

Adaptive memory retrieval. Recent systems move beyond fixed top-k retrieval by adapting retrieval scope, search depth, or computation according to query needs and the retrieval state. HingeMem (Zhong et al., 2026) adapts retrieval through query-dependent routing and query-type-specific stopping criteria, BudgetMem (Zhang et al., 2026b) learns query-aware budget allocation, and HiGMem (Cao et al., 2026) uses event-level summaries as semantic anchors to identify which underlying dialogue turns are worth inspecting. MRAgent (Ji et al., 2026) incorporates LLM reasoning into iterative Cue–Tag–Content graph traversal, allowing accumulated evidence to generate new retrieval cues, redirect subsequent search, and guide further exploration. In contrast, LocalMem introduces progressive retrieval that explicitly assesses the sufficiency of accumulated evidence and acquires additional evidence only when needed.

3 Method

3.1 Problem Formulation

As illustrated in Figure 2, LocalMem consists of three stages: Store, Retrieve, and Answer. Given a long-context input $\mathcal{H} = d_1, \ldots, d_T$, we partition it into N interaction windows $\{W_t\}_{t=1}^{N}$.

Store. The Memory Archiver f_ϕ independently converts each interaction window into memory items, which are stored in a persistent memory bank $\mathcal{M}$:

$$\mathcal{E}_t = f_\phi(W_t), \qquad \mathcal{M} = \mathrm{Store}(\mathcal{E}_1, \ldots, \mathcal{E}_N). \tag{1}$$

Retrieve. Given a query q, the Memory Enquirer progressively selects and accumulates multi-grained memory items from the persistent memory bank:

$$C^{(r)} = C^{(r-1)} \cup \mathrm{Retrieve}_r(\mathcal{M}, q, C^{(r-1)}), \qquad C^{(0)} = \varnothing. \tag{2}$$

After each tier, a frozen judge h_J determines whether $C^{(r)}$ is sufficient. Retrieval stops at the first sufficient tier τ, or at the final tier R

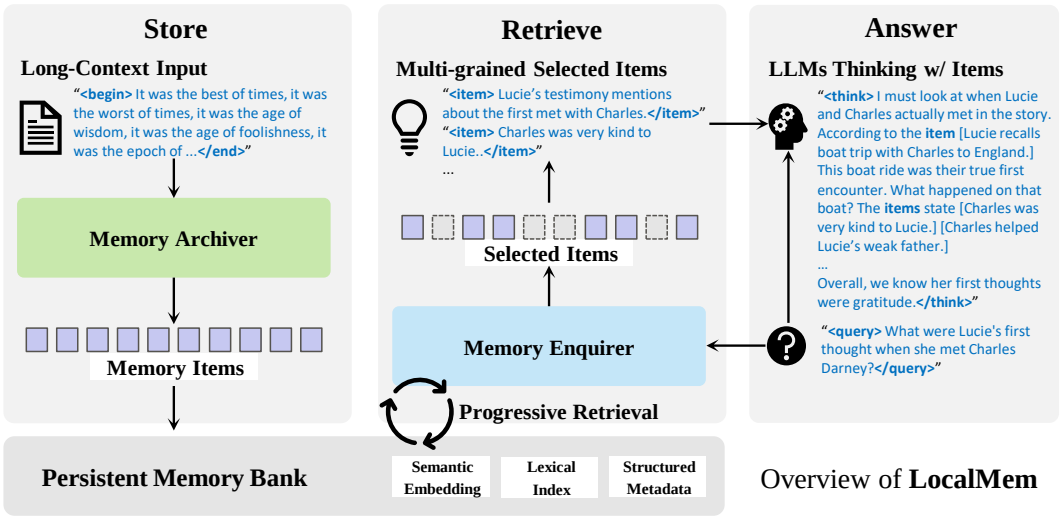


Figure 2: Overview of LocalMem. **Store:** the Memory Archiver converts long context input into structured memory items stored in a persistent memory bank. **Retrieve:** the Memory Enquirer progressively retrieves memory items through semantic, lexical, and structured views. **Answer:** a frozen Answer Model generates the final response.

Answer. The frozen Answer Model h_A generates the final response from the query q and the accumulated selected items $C^{(\tau)}$:

$$a = h_A(q, C^{(\tau)}). \tag{3}$$

3.2 Memory Archiver

Structured memory representation. We adopt a window-based processing strategy similar to SimpleMem (Liu et al., 2026b). For LoCoMo10, messages are grouped into overlapping windows of size W with overlap O. For LongMemEval-S and Mem-α, messages are grouped into windows according to a token budget B. Each window is independently processed by the 4B Memory Archiver to produce atomic memory entries.

Each entry represents a self-contained fact or event and contains a `lossless_restatement`, together with `keywords` and optional metadata including `timestamp`, `location`, `persons`, `entities`, and `topic`. The `lossless_restatement` preserves the source information while resolving coreferences and normalizing temporal expressions. The `keywords` capture salient lexical cues, while the metadata fields record temporal information (`timestamp`), spatial information (`location`), mentioned people (`persons`), other named entities such as organizations or products (`entities`), and the semantic subject of the entry (`topic`). An example of the resulting memory entries is shown in Figure 3.

Each memory entry is indexed through three retrieval views, $I(m_k) = s_k, l_k, u_k$, where s_k is a semantic embedding produced by the frozen Embedding model, l_k is a BM25-style lexical index over the extracted keywords, and u_k represents structured metadata attributes used for symbolic filtering.

3.3 Memory Enquirer

A fixed retrieval budget assigns the same amount of evidence to queries with different information requirements. We instead formulate memory retrieval as progressive evidence acquisition, starting with lightweight dense retrieval and acquiring additional evidence only when the accumulated evidence is insufficient. The first two tiers use small, fixed candidate sizes k_1 and k_2 to provide an efficient initial retrieval path. If further evidence is required, the Memory Enquirer is invoked at Tiers 3–4 to expand or refine retrieval. After each tier, a frozen LLM judge evaluates the accumulated evidence and determines whether retrieval should proceed.

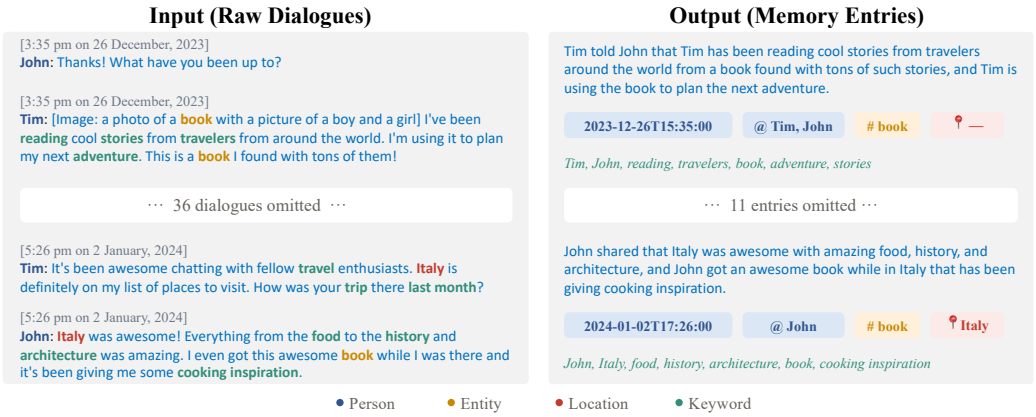


Figure 3: Example Memory Archiver output. **Left:** raw dialogue messages with persons, entities, locations, and keywords highlighted. **Right:** the resulting atomic memory entries, each containing a lossless restatement and structured metadata for multi-view indexing.

Tiers 1–2: Lightweight semantic retrieval. Given a query q, Tier 1 retrieves a small set of memory entries using dense similarity. If the accumulated evidence is insufficient, Tier 2 expands retrieval to a larger fixed candidate size:

$$C^{(1)} = \mathrm{TopK}_{\mathrm{dense}}(\mathcal{M}, q, k_1), \qquad C^{(2)} = \mathrm{TopK}_{\mathrm{dense}}(\mathcal{M}, q, k_2), \quad k_2 > k_1. \tag{4}$$

These two tiers provide an efficient initial retrieval path using only dense similarity.

Tier 3: Planner-guided multi-view retrieval. If the accumulated evidence remains insufficient after Tier 2, the Memory Enquirer generates a structured retrieval plan from the query:

$$P^{(3)} = g_\psi(q). \tag{5}$$

The plan includes `required_info` together with lexical (`keywords`), entity (`persons`, `entities`), spatial (`locations`), and temporal (`time_expression`) retrieval cues. The `required_info` field decomposes the query into independent information requirements, while the remaining fields provide retrieval signals for the dense, lexical, and symbolic memory views. The resulting plan is then executed through multi-view retrieval:

$$C^{(3)} = C^{(2)} \cup \mathrm{Retrieve}_{\mathrm{multi}}(\mathcal{M}, P^{(3)}, C^{(2)}). \tag{6}$$

Tier 4: Gap-aware refinement. If the accumulated evidence remains insufficient, the same Memory Enquirer, using a gap-refinement prompt, identifies information requirements that are not yet supported by the retrieved evidence:

$$G^{(4)} = g_\psi^{\mathrm{gap}}(q, \mathcal{R}, \mathcal{S}^{(3)}, C^{(3)}), \qquad C^{(4)} = C^{(3)} \cup \mathrm{Retrieve}_{\mathrm{gap}}(\mathcal{M}, G^{(4)}, C^{(3)}). \tag{7}$$

Here, $\mathcal{R}$ denotes the information requirements generated at Tier 3, and $\mathcal{S}^{(3)}$ denotes those already targeted by previous retrieval. The gap planner returns the remaining unsupported requirements, redirecting retrieval toward complementary memories rather than repeatedly expanding the same semantic neighborhood.

After every tier, the frozen judge evaluates the accumulated evidence, retrieval terminates once the evidence is judged sufficient or the final tier is reached.

3.4 Training the Memory Models

Memory Archiver (GRPO). The Memory Archiver is first trained with supervised fine-tuning on structured memory-extraction targets distilled from a large teacher LLM, and is subsequently optimized with GRPO. For each training conversation i, we sample multiple on-policy rollouts. Each rollout k generates memory entries for all interaction windows, which jointly form a persistent memory $\mathcal{M}_{i,k}$. We then evaluate $\mathcal{M}_{i,k}$ on the downstream question set $\mathcal{Q}_i$ using frozen retrieval and answer models, yielding the conversation-level QA reward

$$R_{i,k}^{\mathrm{QA}} = \frac{1}{|\mathcal{Q}_i|} \sum_{q \in \mathcal{Q}_i} \mathrm{F1}(\hat{a}_{i,k,q}, a_{i,q}^{\star}), \tag{8}$$

where $\hat{a}_{i,k,q}$ is the answer obtained from the constructed memory and $a^{\star}_{i,q}$ is the corresponding ground-truth answer. This directly optimizes memory construction for downstream answer utility, rather than relying solely on imitation of teacher-generated memory targets.

Unlike standard GRPO, a memory-construction rollout consists of multiple window-level generations that jointly determine the final persistent memory. We therefore introduce a unified memory reward that propagates the conversation-level QA reward to all parseable generations within the rollout while penalizing malformed outputs:

$$r_{i,k,j} = \begin{cases} R^{\mathrm{QA}}_{i,k}, & \text{if the output at window } j \text{ is parseable,} \\ -1, & \text{otherwise.} \end{cases} \tag{9}$$

Valid empty extractions are treated as parseable outputs. This formulation provides shared credit to the window-level decisions that collectively construct the memory, allowing the Archiver to learn which information is useful to preserve from its effect on downstream QA.

Group-relative normalization is performed across sibling rollouts at the same window position:

$$A_{i,k,j} = \frac{1}{N_i} \frac{r_{i,k,j} - \mu_{i,j}}{\sigma_{i,j} + \epsilon}, \tag{10}$$

where $\mu_{i,j}$ and $\sigma_{i,j}$ are computed across sibling rollouts and N_i is the number of interaction windows in conversation i. The factor $1/N_i$ scales each window-level advantage so that the aggregate gradient contribution of a conversation does not grow proportionally with its number of interaction windows.

Importantly, we impose no explicit memory-size regularization, optimizing the Archiver solely according to downstream QA utility rather than explicitly favoring shorter memories. Detailed GRPO training procedures are provided in Appendix A.1.

Memory Enquirer (SFT). The Memory Enquirer g_ψ is trained separately with supervised fine-tuning to generate the structured retrieval plans and gap refinements used at Tiers 3–4. The SFT targets are generated by a large teacher LLM and distilled into the 4B Enquirer. We further investigate Uni-GRPO and Multi-Teacher On-Policy Distillation (OPD) as alternative strategies for retrieval-policy optimization in Appendix A.3, while main experiments use the SFT-trained Enquirer.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate LocalMem on two long-term, multi-session memory benchmarks. LoCoMo10 (Maharana et al., 2024) contains 10 long conversations with about 27 sessions and 587 turns per conversation on average, and 1,986 questions across Single-Hop (SH), Multi-Hop (MH), Temporal (Temp), Open-Domain (OD), and Adversarial (Adv) categories. These categories test the ability to recall, combine, and temporally reason over information distributed across long interaction histories. LongMemEval-S (Wu et al., 2025) contains 500 manually curated questions covering information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention. We use the cleaned evaluation splits and report accuracy under the benchmark's LLM-as-a-Judge protocol.

Baselines. We compare with long-context baselines and representative memory systems spanning semantic, structured, graph-based, and adaptive memory management. These include MemGPT (Packer et al., 2023), A-MEM (Xu et al., 2025), MemoryBank (Zhong et al., 2024), Mem0 (Chhikara et al., 2025), MemVerse (Liu et al., 2025b), HippoRAG2 (Gutiérrez et al., 2025), EvolveMem (Liu et al., 2026c), SimpleMem (Liu et al., 2026b), Omni-SimpleMem (Liu et al., 2026a), LightMem (Fang et al., 2026), HingeMem (Zhong et al., 2026), Memora (Xia et al., 2026), E-mem (Wang et al., 2026a), Mem-α (Wang et al., 2025), and Mem-T (Yue et al., 2026). We report published results for cross-paper comparison and additionally reproduce representative baselines using the same Answer Model for matched-backbone evaluation.

Implementation details. Unless otherwise specified, LocalMem uses a Qwen3.5-4B SFT+GRPO Memory Archiver, a Qwen3.5-4B SFT Memory Enquirer, and a frozen Qwen3.5-122B-A10B-FP8 (Qwen3.5-122B for short) model as both the evidence-sufficiency judge and Answer Model. The

Memory Archiver and Memory Enquirer are trained separately on 372 synthetic conversations from the non-TTL portion of the Mem-α training set, yielding 4,147 memory-extraction and 13,821 retrieval-planning examples, respectively. DeepSeek-V4-Pro provides SFT supervision for both components, while Qwen3.5-35B-A3B serves as the answer generator during Memory Archiver GRPO rollouts. For memory construction, LoCoMo10 uses turn-count windows with $W = 40$ and overlap $O = 2$, averaging about 2.8k tokens per window, whereas LongMemEval-S and Mem-α use 1k token-based windows to bound the context processed at each step and control peak memory usage during training. For progressive retrieval, Tiers 1–2 use $k = 15$ and $k = 25$; Tier 3 uses $k = 25$ for semantic retrieval and $k = 5$ for keyword/structured retrieval; and Tier 4 increases them to $k = 50$ and $k = 25$. All results of LocalMem are averaged over three independent runs, and training details are provided in Table 7.

4.2 Main Results

Table 1: Comparison on LoCoMo10. Baseline results are grouped by the Answer Model and taken from the respective papers. Overall_4 and Overall_5 denote question-count-weighted F1 over the four non-adversarial categories and all five categories, respectively, using the official category counts of 282/321/96/841/446 for MH/Temp/OD/SH/Adv. Best and second-best results are shown in **bold** and underline, respectively.

Method	Memory	Answer	MH	Temp	OD	SH	Adv	Overall_4	Overall_5
GPT-4o (Long-Context)	–	GPT-4o	34.4	52.6	27.0	55.1	–	49.0	–
A-MEM	GPT-4o	GPT-4o	32.9	39.4	17.1	48.4	36.4	41.7	40.5
MemGPT	GPT-4o	GPT-4o	30.4	17.3	12.2	60.2	35.0	42.8	41.1
MemoryBank	GPT-4o	GPT-4o	23.3	23.3	20.9	39.9	86.3	32.2	44.4
EvolveMem	GPT-4o	GPT-4o	31.6	38.4	32.9	49.6	93.6	42.9	54.3
LangMem	GPT-4o	GPT-4o	45.6	46.8	21.5	54.1	87.5	49.0	57.6
HippoRAG2	GPT-4o	GPT-4o	38.0	44.9	21.2	59.2	87.7	50.0	58.4
Omni-SimpleMem	GPT-4o	GPT-4o	55.6	36.5	25.5	64.1	83.5	54.4	60.9
HingeMem	GPT-4o	GPT-4o	53.6	57.4	30.7	61.1	87.4	57.1	63.9
SimpleMem	GPT-4o	GPT-4o	35.9	56.7	18.2	45.4	–	44.3	–
E-mem	GPT-4o	GPT-4o	45.9	**64.8**	32.7	51.0	–	51.8	–
Memora	GPT-4.1-mini	GPT-4.1-mini	42.8	62.3	30.8	59.7	–	55.3	–
GPT-5.1 (Long-Context)	–	GPT-5.1	44.5	56.9	31.2	57.4	–	53.3	–
MemVerse	GPT-5.1	GPT-5.1	35.6	38.3	26.7	49.2	–	43.0	–
EvolveMem	GPT-5.1	GPT-5.1	54.1	35.2	**33.7**	51.9	90.2	47.7	57.2
Omni-SimpleMem	GPT-5.1	GPT-5.1	**59.8**	36.7	30.7	67.6	74.7	57.4	61.3
LocalMem (Ours)	Qwen3.5-4B	Qwen3.5-122B	48.5	63.3	30.8	**68.3**	**97.1**	**61.3**	**69.3**

LoCoMo10. Table 1 compares LocalMem with previously reported memory systems on LoCoMo10. We report Overall_4, the question-count-weighted F1 over the four non-adversarial categories, and Overall_5, the corresponding aggregate over all five categories. Using the open-source Qwen3.5-122B as the Answer Model, LocalMem achieves **69.3 Overall_5 F1**, outperforming all reported systems in Table 1, including HingeMem with GPT-4o (63.9) and Omni-SimpleMem with GPT-5.1 (61.3). The gains are broad across categories, with LocalMem achieving the best SH and Adv scores while remaining competitive on MH, Temp, and OD. Notably, when paired with GPT-4o without any retraining, the same learned memory system achieves **64.8 Overall_5 F1** (Table 9), outperforming all reported GPT-4o-based methods, including HingeMem (63.9).

We further conduct a matched-backbone comparison over the four non-adversarial categories to better isolate differences in memory management. As shown in Table 2, we replace the model of Mem0 and Omni-SimpleMem with the same Qwen3.5-122B-A10B-FP8 model used by LocalMem, while retaining the memory design of each method. Under this controlled setting, LocalMem achieves **61.3 Overall_4 F1**, outperforming Omni-SimpleMem by 9.7 points and Mem0 by 11.0 points. Compared with Omni-SimpleMem, LocalMem improves all four categories, with gains of +8.3 on MH, +5.8 on Temp, +8.3 on OD, and +11.8 on SH. Notably, Mem0 (New Memory Algorithm, April 2026) and Omni-SimpleMem use Qwen3.5-122B for memory management, whereas LocalMem uses separately trained 4B Memory Archiver and Memory Enquirer models. These results suggest that the improvement is not solely attributable to the Answer Model, but also reflects the learned memory-construction and progressive-retrieval policies.

LongMemEval-S. Table 3 reports LLM-as-a-Judge accuracy on LongMemEval-S. Using the same Memory Archiver and Memory Enquirer configuration as on LoCoMo10, LocalMem achieves

Table 2: Matched-backbone comparison on the four non-adversarial LoCoMo10 categories using the same Qwen3.5-122B-A10B-FP8 Answer Model for all methods. Each method retains its own memory-management design. Best results in each metric column are shown in **bold**.

Method	Memory Model	Answer Model	MH	Temp	OD	SH	Overall_4
Mem0	Qwen3.5-122B	Qwen3.5-122B	43.7	58.4	26.6	52.2	50.3
Omni-SimpleMem	Qwen3.5-122B	Qwen3.5-122B	40.2	57.5	22.5	56.5	51.6
LocalMem (Ours)	Qwen3.5-4B	Qwen3.5-122B	**48.5**	**63.3**	**30.8**	**68.3**	**61.3**

81.8% accuracy, exceeding the reported results of MemVerse (77.0%), LightMem (73.2%), Mem-T (65.8%), and Mem-α (52.0%). For our evaluation, we follow the official LongMemEval-S evaluation setup, using the official evaluation prompts and GPT-4o as the answer judge. Since LLM-as-a-Judge accuracy can be affected by both the judge model and evaluation prompt, cross-paper results should be interpreted with potential differences in evaluation settings in mind.

Table 3: Comparison on LongMemEval-S measured by LLM-as-a-Judge accuracy (Acc), following the official LongMemEval evaluation protocol. The best result is shown in **bold**.

Method	Memory Model	Answer Model	Acc
Mem-α	Qwen3-4B	Qwen3-32B	52.0
Mem-T	Qwen3-4B	–	65.8
LightMem	GLM-4.6	GLM-4.6	73.2
MemVerse	Qwen2.5-7B	GPT-5.1	77.0
LocalMem (Ours)	Qwen3.5-4B	Qwen3.5-122B	**81.8**

4.3 Ablation Studies

Table 4: Memory Archiver ablation on the four non-adversarial LoCoMo10 categories, with a Qwen3.5-122B judge/answer model and a fixed Qwen3.5-122B Memory Enquirer. We compare prompted baselines of different model sizes with our 4B Memory Archiver trained using SFT and SFT+GRPO. For DeepSeek-V4-Pro, the reported runtime (*) corresponds to API-call latency.

Memory Archiver	MH	Temp	OD	SH	Overall_4	Time (min)
DeepSeek-V4-Pro	45.5	**64.6**	29.1	66.9	60.1	16.1^*
Qwen3.5-4B	41.4	54.9	29.2	62.5	55.0	–
Qwen3.5-122B	47.0	62.2	**30.6**	61.4	57.0	34.5
LocalMem-4B SFT	46.7	62.0	29.1	68.2	60.5	–
LocalMem-4B SFT+GRPO	**48.1**	62.6	29.8	**68.3**	**61.1**	7.9

Ablation of the learned Memory Archiver. As shown in Table 4, with the Qwen3.5-122B Memory Enquirer and answer model fixed, the prompted Qwen3.5-4B Memory Archiver achieves an Overall F1 of 55.0. SFT improves the score to 60.5 ($+5.6$), and further GRPO training raises it to **61.1** ($+0.5$ over SFT). The trained 4B Memory Archiver also outperforms the prompted larger-model baselines, including DeepSeek-V4-Pro (60.1) and Qwen3.5-122B (57.0). Notably, SFT+GRPO achieves higher F1 than SFT while retaining fewer dialogue tokens (70.1% vs. 72.7%), suggesting that GRPO helps preserve more useful information rather than simply increasing memory length. In addition, replacing the 122B Memory Archiver with the 4B model reduces the total construction time over the ten LoCoMo10 conversations from 34.5 to 7.9 minutes, a $4.4\times$ speedup.

Ablation of the progressive Memory Enquirer.

For all configurations, the same Qwen3.5-122B model serves as the frozen sufficiency judge and answer model. The standard baseline follows the retrieval strategy of SimpleMem (Liu et al., 2026b) with a separate Qwen3.5-122B retriever, while our progressive Memory Enquirer is implemented with either Qwen3.5-122B or the SFT-trained Qwen3.5-4B. For a fair efficiency comparison, all retrieval models are evaluated on two NVIDIA A100-SXM4-80GB GPUs. For progressive retrieval, the reported retrieval latency includes the cost of sufficiency judging.

As shown in Table 5, replacing the SimpleMem-style fixed top-k strategy with progressive retrieval using Qwen3.5-122B improves Overall F1 from 59.7 to 61.2. Retrieval latency also decreases from 7.4 s to 6.7 s. This shows that adapting retrieval according to evidence sufficiency improves retrieval effectiveness without increasing cost. Directly replacing the 122B Enquirer with an off-the-shelf

Table 5: Retrieval-strategy ablation on the four non-adversarial LoCoMo10 categories. Progressive denotes the four-tier retrieval pipeline described in Section 3.3. **Retrieval (s)** reports per-question retrieval latency, and **Answer Token** reports the mean context size passed to the answer model.

Retrieval Strategy	MH	Temp	OD	SH	Overall$_4$	Retrieval (s)	Answer Token
Standard, Qwen3.5-122B	47.1	60.9	27.7	67.2	59.7	7.4	3.9k
Progressive, Qwen3.5-4B	46.4	61.3	26.2	67.4	59.7	–	2.8k
Progressive, Qwen3.5-122B	48.2	63.0	30.7	**68.3**	61.2	6.7	3.7k
Progressive, LocalMem-4B SFT	**48.5**	**63.3**	**30.8**	68.3	**61.3**	**4.2**	**3.5k**

Qwen3.5-4B yields an Overall F1 of 59.7, matching the 122B standard retriever but not preserving the full gain of progressive retrieval. After SFT, the 4B Memory Enquirer improves to 61.3 Overall F1 and reduces retrieval latency to 4.2 s. It therefore matches and slightly exceeds the 122B progressive Enquirer at substantially lower online cost. The answer context also increases from 2.8k tokens with the off-the-shelf 4B model to 3.5k after SFT. This suggests that the trained Enquirer learns when additional evidence is needed rather than simply retrieving less. Overall, SFT improves the retrieval decisions of the 4B Enquirer and allows it to use progressive retrieval more effectively.

Table 6: Ablation on the maximum retrieval tier of progressive retrieval on the four non-adversarial LoCoMo10 categories. We fix the 4B Memory Archiver and the Qwen3.5-122B answer model.

Retrieval Depth	MH	Temp	OD	SH	Overall$_4$	Retrieval (s)	Answer Token
Tier 1	40.8	57.5	27.6	62.7	55.4	1.7	1.8k
Tier 1–2	43.6	59.1	28.3	64.3	57.2	2.4	2.1k
Tier 1–3	44.7	61.1	30.1	66.7	59.2	5.0	2.5k
Tier 1–4	**48.2**	63.0	**30.7**	68.3	**61.2**	6.7	3.7k
Tier 1–5	46.7	**63.0**	30.2	**68.5**	61.0	15.5	4.7k

As shown in Table 6, increasing the maximum retrieval tier steadily improves performance up to Tier 4. Overall F1 rises from 55.4 with Tier 1 only to 57.2, 59.2, and 61.2 when extending retrieval to Tiers 2, 3, and 4, respectively. Extending retrieval to Tier 4 yields a further $+2.0$ F1 gain, with the largest improvement on MH, indicating that additional targeted retrieval is particularly useful when earlier evidence is insufficient. These gains come with increasing retrieval cost: latency grows from 1.7 s to 6.7 s, while the answer context increases from 1.8k to 3.7k tokens. Further extending gap refinement to Tier 5, with $k = 75$ for semantic retrieval and $k = 50$ for keyword/structured retrieval, increases latency to 15.5 s and the answer context to 4.7k tokens, while Overall F1 slightly decreases to 61.0. This suggests that deeper retrieval can provide complementary evidence when earlier tiers are insufficient, but excessive expansion introduces substantially higher cost without further performance gains. Overall, Tier 4 provides a favorable trade-off between retrieval effectiveness and efficiency, and is used as the maximum retrieval depth in our main configuration.

5 Conclusion

In this work, we introduced LocalMem, a plug-and-play memory-management system that learns memory construction and retrieval with separate 4B local language models. The Memory Archiver learns what information to preserve from downstream QA supervision, while the Memory Enquirer progressively retrieves additional evidence only when needed. Experiments on LoCoMo10 and LongMemEval-S show that LocalMem achieves state-of-the-art performance, while controlled comparisons demonstrate that the learned 4B memory models can outperform substantially larger prompted models with lower memory-management costs. These results provide a practical foundation for efficient and locally deployable long-term memory systems. We discuss the deployment feasibility of the 4B memory models in Appendix A.7. Looking ahead, we envision LocalMem as a step toward lifelong personal memory assistants running directly on users' mobile devices.

Limitations. The current progressive retrieval pipeline relies on a frozen evidence-sufficiency judge, whose errors may cause premature stopping or unnecessary retrieval. LocalMem also uses separate Memory Archiver and Memory Enquirer models, while our attempts to unify them remain less effective than separate training. Our evaluation focuses on text-based conversational memory, leaving multimodal and broader agentic settings unexplored. Future work will investigate more reliable sufficiency assessment, unified memory modeling, and extensions to multimodal memory.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the proposed method, training procedure, hyperparameters, datasets, and evaluation protocols to facilitate reproducibility. All datasets and benchmarks are publicly available. Implementation, training/evaluation scripts, configuration files, and model checkpoints will be released after review while preserving author anonymity.

AI USE STATEMENT

We used generative AI tools for drafting and polishing manuscript text and for code assistance. All AI-assisted work was reviewed; all experimental results, claims, and artifacts were produced and verified by the authors, who take responsibility for the final content, including AI-assisted text, claims, or artifacts.

REFERENCES

Shuqi Cao, Jingyi He, and Fei Tan. HiGMem: A hierarchical and LLM-guided memory system for long-term conversational agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 33853–33862, 2026.

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.

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*, 2026.

Yicheng Fu, Raviteja Anantha, and Jianpeng Cheng. CAMPHOR: Collaborative agents for multi-input planning and high-order reasoning on device. In *Proceedings of the 1st Workshop for Research on Agent Language Models (REALM 2025)*, pp. 33–51, 2025.

Weidong Guo, Dakai Wang, Zixuan Wang, Hui Liu, and Yu Xu. T-Mem: Memory that anticipates, not archives. *arXiv preprint arXiv:2606.15405*, 2026.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 59532–59569, 2024.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From RAG to memory: Non-parametric continual learning for large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

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. *arXiv preprint arXiv:2512.13564*, 2025.

Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Zhen Tan, Yu Wang, Tianxin Wei, Shanglin Wu, et al. A survey of agent memory in the second half: Towards self-evolving and long-horizon agents. *Transactions on Machine Learning Research*, 2026.

Shuo Ji, Yibo Li, and Bryan Hooi. Memory is reconstructed, not retrieved: Graph memory for LLM agents. *arXiv preprint arXiv:2606.06036*, 2026.

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. 25961–25970, 2025.

Changmin Lee, Jaemin Kim, and Taesik Gong. From volume to value: Preference-aligned memory construction for on-device RAG. In *Forty-third International Conference on Machine Learning*, 2026.

Jiahong Liu, Zexuan Qiu, Zhongyang Li, Quanyu Dai, Wenhao Yu, Jieming Zhu, Minda Hu, Menglin Yang, Tat-Seng Chua, and Irwin King. A survey of personalized large language models: Progress and future directions. *arXiv preprint arXiv:2502.11528*, 2025a.

Jiaqi Liu, Zipeng Ling, Shi Qiu, Yanqing Liu, Siwei Han, Peng Xia, Haoqin Tu, Zeyu Zheng, Cihang Xie, Charles Fleming, et al. Omni-SimpleMem: Autoresearch-guided discovery of lifelong multimodal agent memory. *arXiv preprint arXiv:2604.01007*, 2026a.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. SimpleMem: Efficient lifelong memory for LLM agents. In *Forty-third International Conference on Machine Learning*, 2026b.

Jiaqi Liu, Xinyu Ye, Peng Xia, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. EvolveMem: Self-evolving memory architecture via autoresearch for LLM agents. *arXiv preprint arXiv:2605.13941*, 2026c.

Junming Liu, Yifei Sun, Weihua Cheng, Haodong Lei, Yirong Chen, Licheng Wen, Xuemeng Yang, Daocheng Fu, Pinlong Cai, Nianchen Deng, et al. MemVerse: Multimodal memory for lifelong learning agents. *arXiv preprint arXiv:2512.03627*, 2025b.

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.

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.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Kaixiang Wang, Yidan Lin, Zihan Wang, Bunyod Suvonov, Zhaojiacheng Zhou, Yuxiang Zheng, Jiaxi Cao, Zhiheng Dong, Chentao Wu, Jiong Lou, and Jie Li. E-mem: Multi-agent based episodic context reconstruction for LLM agent memory. In *Forty-third International Conference on Machine Learning*, 2026a.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-α: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025.

Ziyi Wang, Haonan Jin, Zian Wang, Wendong Wang, and Lanshan Zhang. LightWM: Training-free hierarchical working memory for small language model agents. In *Forty-third International Conference on Machine Learning*, 2026b.

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

Menglin Xia, Xuchao Zhang, Shantanu Dixit, Paramaguru Harimurugan, Rujia Wang, Victor Ruhle, Robert Sim, Chetan Bansal, and Saravan Rajmohan. Memora: A harmonic memory representation balancing abstraction and specificity. *arXiv preprint arXiv:2602.03315*, 2026.

Yilin Xiao, Zhehan Zhu, Yujing Zhang, Jin Chen, Zijin Hong, Luyao Zhuang, Qinggang Zhang, Shengyuan Chen, Xiaocao Ouyang, Lingfei Ren, et al. Zero-Mem: Zero-token memory operations for LLM agents. *arXiv preprint arXiv:2607.29377*, 2026.

Buqiang Xu, Yijun Chen, Jizhan Fang, Ruobin Zhong, Yunzhi Yao, Yuqi Zhu, Lun Du, and Shumin Deng. StructMem: Structured memory for long-horizon behavior in LLMs. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 122–146, 2026.

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, 2025.

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.

Ke Yang, Zixi Chen, Xuan He, Jize Jiang, Michel Galley, Chenglong Wang, Jianfeng Gao, Jiawei Han, and ChengXiang Zhai. PlugMem: A task-agnostic plugin memory module for LLM agents. *arXiv preprint arXiv:2603.03296*, 2026.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. MemAgent: Reshaping long-context LLM with multi-conv RL-based memory agent. In *International Conference on Learning Representations*, 2026.

Qianhao Yuan, Jie Lou, Zichao Li, Jiawei Chen, Yaojie Lu, Hongyu Lin, Le Sun, Debing Zhang, and Xianpei Han. MemSearcher: Training LLMs to reason, search and manage memory via end-to-end reinforcement learning. *arXiv preprint arXiv:2511.02805*, 2025.

Yanwei Yue, Guibin Zhang, Boci Peng, Xuanbo Fan, Jiaxin Guo, Qiankun Li, and Yan Zhang. Mem-T: Densifying rewards for long-horizon memory agents. In *Forty-third International Conference on Machine Learning*, 2026.

Guibin Zhang, Haotian Ren, Chong Zhan, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. MemEvolve: Meta-evolution of agent memory systems. In *Forty-third International Conference on Machine Learning*, 2026a.

Haozhen Zhang, Haodong Yue, Tao Feng, Quanyu Long, Jianzhu Bao, Bowen Jin, Weizhi Zhang, Xiao Li, Jiaxuan You, Chengwei Qin, et al. Learning query-aware budget-tier routing for runtime agent memory. In *Forty-third International Conference on Machine Learning*, 2026b.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19724–19731, 2024.

Yijie Zhong, Yunfan Gao, and Haofen Wang. HingeMem: Boundary guided long-term memory with query adaptive retrieval for scalable dialogues. In *Proceedings of the ACM Web Conference 2026*, pp. 3576–3587, 2026.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, 2026.

A Appendix

A.1 Memory Archiver GRPO Training Details

Training uses the `slime` framework (Megatron + SGLang). Each rollout batch contains N training conversations, with K on-policy rollouts sampled for each conversation. We use i to index conversations, k to index sibling rollouts of the same conversation, and j to index interaction windows.

Rollout. Let conversation i contain N_i interaction windows $\{W_{i,j}\}_{j=1}^{N_i}$. For each rollout k, the Memory Archiver independently samples one output $y_{i,k,j}$ for every interaction window $W_{i,j}$. A parseable output is converted into a possibly empty set of memory entries $E_{i,k,j}$. The entries produced across all windows are then merged to form the persistent memory

$$\mathcal{M}_{i,k} = \mathrm{Store}\left(E_{i,k,1}, \ldots, E_{i,k,N_i}\right), \tag{11}$$

where malformed outputs contribute no memory entries. Importantly, each interaction window is processed independently by the Archiver; window-level generations do not condition on memory entries produced for earlier windows. The resulting memory $\mathcal{M}_{i,k}$ is evaluated using the frozen retrieval and answer models on every question q in the downstream question set $\mathcal{Q}_i$, producing predicted answers $\hat{a}_{i,k,q}$.

Conversation-level QA reward. We evaluate the downstream utility of each sampled persistent memory using the mean QA F1 over $\mathcal{Q}_i$:

$$R_{i,k}^{\mathrm{QA}} = \frac{1}{|\mathcal{Q}_i|} \sum_{q \in \mathcal{Q}_i} \mathrm{F1}\left(\hat{a}_{i,k,q}, a_{i,q}^{\star}\right), \tag{12}$$

where $a_{i,q}^{\star}$ denotes the corresponding ground-truth answer. Different sibling rollouts generally construct different persistent memories and therefore receive different conversation-level QA rewards.

Unified memory reward. Memory construction differs from standard GRPO in that each rollout contains multiple window-level generations that jointly determine the final persistent memory. We therefore assign each rollout's conversation-level QA reward to its parseable window-level generations and apply a hard format penalty to malformed outputs:

$$r_{i,k,j} = \begin{cases} R_{i,k}^{\mathrm{QA}}, & \text{if } y_{i,k,j} \text{ is parseable,} \\ -1, & \text{otherwise.} \end{cases} \tag{13}$$

A parseable empty extraction is treated as a valid memory decision and receives $R_{i,k}^{\mathrm{QA}}$ rather than the format penalty. Thus, the Archiver is not forced to extract memory from every interaction window. In our implementation, $r_{i,k,j}$ is rounded to three decimal places before group-relative normalization to suppress numerically negligible F1 differences.

Group-relative normalization. Rewards are normalized at each window position across the K sibling rollouts of the same conversation. Let $\mu_{i,j}$ and $\sigma_{i,j}$ denote the mean and standard deviation of $\{r_{i,k,j}\}_{k=1}^{K}$. We compute the window-level advantage as

$$A_{i,k,j} = \frac{1}{N_i} \frac{r_{i,k,j} - \mu_{i,j}}{\sigma_{i,j} + \epsilon}, \tag{14}$$

where $\epsilon = 10^{-6}$. Both QA-based rewards and format penalties participate in the same group-relative normalization. The factor $1/N_i$ normalizes the aggregate contribution of each rollout by the number of interaction windows in the conversation, preventing conversations with more windows from contributing disproportionately large gradients. Each generated token within window j of rollout k inherits the same window-level advantage $A_{i,k,j}$.

This formulation evaluates each sampled persistent memory using a rollout-level downstream QA reward and uses the resulting signal as shared credit for the window-level generations that jointly construct it. At the same time, malformed generations receive a lower reward through the format penalty and are compared against their sibling generations at the same window position.

Clipped policy-gradient objective. Let $o_{i,k,j,t}$ denote the t-th generated token in window j of rollout k, and let $T_{i,k,j}$ denote the number of generated tokens in that window. The token-level importance ratio is

$$\rho_{i,k,j,t} = \exp\left(\log \pi_{\phi}(o_{i,k,j,t}) - \log \pi_{\phi_{\mathrm{old}}}(o_{i,k,j,t})\right). \tag{15}$$

After completing a rollout batch, the generated window chunks are flattened into independent training samples and partitioned into optimization mini-batches. Let $\mathcal{B}$ denote one such mini-batch, the clipped policy-gradient objective is

$$\mathcal{L}_{\mathrm{PG}} = -\frac{1}{|\mathcal{B}|} \sum_{(i,k,j) \in \mathcal{B}} \frac{1}{T_{i,k,j}} \sum_{t=1}^{T_{i,k,j}} \min\left(\rho_{i,k,j,t} A_{i,k,j}, \mathrm{clip}\left(\rho_{i,k,j,t}, 1-\varepsilon_{\mathrm{low}}, 1+\varepsilon_{\mathrm{high}}\right) A_{i,k,j}\right), \tag{16}$$

where $|\mathcal{B}| = 48$ in our experiments, $\varepsilon_{\mathrm{low}} = 0.2$, and $\varepsilon_{\mathrm{high}} = 0.28$. The loss is first averaged over generated tokens within each chunk and then equally averaged across chunks in the optimization mini-batch.

Table 7: Training hyperparameters for the Memory Archiver and Memory Enquirer.

Hyperparameter	**Archiver SFT**	**Archiver GRPO**	**Enquirer SFT**
Training setup			
Training data	4,147 examples	1,584 examples	13,821 examples
Epochs / steps	2 epochs	500 steps	2 epochs
Batch size	16	48	128
Precision	bf16	bf16	bf16
Optimization			
Learning rate	$10^{-5} \to 10^{-6}$	10^{-6}	$10^{-5} \to 10^{-6}$
LR schedule	Cosine	Constant	Cosine
Weight decay	0.1	–	0.1
GRPO rollout			
Prompts per rollout batch	–	6	–
Generations per prompt	–	8	–
Sampling temperature	–	1.0	–
KL coefficient	–	0.01	–
Clip coefficients $(\varepsilon_{\text{low}}, \varepsilon_{\text{high}})$	–	$(0.2,\ 0.28)$	–

We additionally regularize the policy toward the SFT-initialized reference policy using a low-variance KL estimator:

$$\mathcal{L}_{\text{builder}} = \mathcal{L}_{\text{PG}} + \beta_{\text{KL}} \mathcal{L}_{\text{KL}}, \tag{17}$$

where $\beta_{\text{KL}} = 0.01$. No entropy bonus is used, i.e., $\beta_{\text{entropy}} = 0$.

A.2 Training Hyperparameters

Table 7 summarizes the training hyperparameters for the Memory Archiver and Memory Enquirer. All trainable models use the Qwen3.5-4B architecture and are trained on 8× NVIDIA A100-SXM4-80GB GPUs with the `slime` framework (Megatron + SGLang) in bf16. The Archiver and Enquirer SFT stages are initialized from the pretrained Qwen3.5-4B checkpoint and use cosine learning-rate decay from 10^{-5} to 10^{-6} with weight decay 0.1. The Archiver GRPO stage is initialized from the Archiver SFT checkpoint and uses the same checkpoint as the KL reference policy. Each GRPO rollout batch contains 6 conversations with 8 on-policy rollouts per conversation, yielding 48 rollout samples per batch. During GRPO rollouts, Qwen3.5-35B-A3B serves as the frozen judge/answer model, while Qwen3-Embedding-0.6B is used for memory retrieval, both are served via vLLM.

A.3 Consolidating the Memory Models via Multi-Teacher On-Policy Distillation

The main system uses separate 4B Memory Archiver and Memory Enquirer models for memory construction and progressive retrieval, respectively. As a preliminary study, we investigate consolidating these two specialized policies into a single 4B student π_θ using Multi-Teacher On-Policy Distillation (OPD) with a role-aware unified GRPO objective, termed *Uni-GRPO*. This unified model is not used for any main-text results.

A.3.1 Unified Rollout and Role-Aware Credit Assignment

The student π_θ is initialized from the Memory Archiver GRPO checkpoint and performs memory construction, Tier-3 retrieval planning, and Tier-4 gap refinement within a single on-policy rollout. For each training conversation i, rollout k first constructs a persistent memory and then performs progressive retrieval for every question $q \in \mathcal{Q}_i$ using the same student. The frozen sufficiency judge and answer model remain unchanged from the main pipeline.

The unified rollout contains three generation roles: `archiver`, `plan`, and `gap_plan`. Archiver chunks follow the reward assignment in Section A.1: parseable outputs receive the conversation-level QA reward $R^{\text{QA}}_{i,k}$, whereas malformed outputs receive the -1 format penalty. Each `plan` or `gap_plan` chunk affects only one downstream question and instead receives the corresponding

question-level QA score:

$$r^{\mathrm{E}}_{i,k,c} = \begin{cases} \mathrm{F1}\left(\hat{a}_{i,k,q(c)}, a^{\star}_{i,q(c)}\right), & \text{if chunk } c \text{ is parseable,} \\ -1, & \text{otherwise,} \end{cases} \tag{18}$$

where $q(c)$ denotes the question associated with retrieval chunk c. Thus, Archiver chunks are credited according to their effect on the shared persistent memory, whereas Enquirer chunks are credited according to the retrieval trajectory of their associated question.

A.3.2 UNIFIED GRPO

Because asynchronously generated retrieval chunks are not positionally aligned across sibling rollouts, we normalize rewards using a role-aware semantic key:

$$\kappa(c) = \begin{cases} (\mathrm{A}, j(c)), & c \text{ is an Archiver chunk,} \\ (\mathrm{P}, q(c)), & c \text{ is a Tier-3 plan chunk,} \\ (\mathrm{G}, q(c)), & c \text{ is a Tier-4 gap-plan chunk,} \end{cases} \tag{19}$$

where $j(c)$ denotes the interaction-window index. For each conversation i and semantic key κ, let $\mu_{i,\kappa}$ and $\sigma_{i,\kappa}$ denote the reward statistics across sibling rollouts. The normalized chunk-level advantage is

$$\widehat{A}_{i,k,c} = \frac{r_{i,k,c} - \mu_{i,\kappa(c)}}{\sigma_{i,\kappa(c)} + \epsilon}. \tag{20}$$

Unlike the standalone Archiver GRPO objective, Uni-GRPO does not apply the additional $1/N_i$ scaling to the normalized advantage.

After completing the unified rollouts, all generated chunks are flattened into independent training samples and partitioned into optimization mini-batches. Let $\mathcal{B}$ denote one such mini-batch and $T_{i,k,c}$ the number of generated tokens in chunk c. For token t, the importance ratio is

$$\rho_{i,k,c,t}(\theta) = \exp\left(\log \pi_\theta\left(o_{i,k,c,t} \mid s_{i,k,c,t}\right) - \log \pi_{\theta_{\mathrm{old}}}\left(o_{i,k,c,t} \mid s_{i,k,c,t}\right)\right), \tag{21}$$

where $\pi_{\theta_{\mathrm{old}}}$ denotes the rollout policy held fixed during the corresponding policy update.

We use asymmetric clipping and define the clipped importance ratio as

$$\bar{\rho}_{i,k,c,t}(\theta) = \mathrm{clip}\left(\rho_{i,k,c,t}(\theta), 1 - \varepsilon_{\mathrm{low}}, 1 + \varepsilon_{\mathrm{high}}\right), \tag{22}$$

where $\varepsilon_{\mathrm{low}} = 0.2$ and $\varepsilon_{\mathrm{high}} = 0.28$.

The Uni-GRPO policy-gradient objective is

$$\mathcal{L}_{\text{Uni-GRPO}} = -\frac{1}{|\mathcal{B}|} \sum_{(i,k,c)\in\mathcal{B}} \frac{1}{T_{i,k,c}} \sum_{t=1}^{T_{i,k,c}} \min\left(\rho_{i,k,c,t}(\theta)\widehat{A}_{i,k,c}, \bar{\rho}_{i,k,c,t}(\theta)\widehat{A}_{i,k,c}\right). \tag{23}$$

The loss is first averaged over generated tokens within each chunk and then equally averaged across chunks in the optimization mini-batch.

A.3.3 MULTI-TEACHER ON-POLICY DISTILLATION

To preserve the behaviors of the separately trained specialists, each student-generated chunk is routed to its corresponding frozen teacher. Archiver chunks use the GRPO-trained Memory Archiver π^{A}_T, while `plan` and `gap_plan` chunks use the SFT-trained Memory Enquirer π^{E}_T. Generation remains fully on-policy, and the teachers are used only to evaluate the likelihood of student-generated tokens.

For token t in chunk c, we define the routed teacher–student log-probability difference as

$$d_{i,k,c,t} = \log \pi^{\tau(c)}_T\left(o_{i,k,c,t} \mid s_{i,k,c,t}\right) - \log \pi_\theta\left(o_{i,k,c,t} \mid s_{i,k,c,t}\right), \tag{24}$$

where $\tau(c) \in \{\mathrm{A}, \mathrm{E}\}$ selects the corresponding specialist teacher. Because the tokens are sampled on-policy from the student, $-d_{i,k,c,t}$ is the $K1$ Monte Carlo estimator of $\mathrm{KL}\left(\pi_\theta \,\|\, \pi^{\tau(c)}_T\right)$. We therefore augment the chunk-level Uni-GRPO advantage with a token-level distillation signal:

$$\widetilde{A}_{i,k,c,t} = \widehat{A}_{i,k,c} + \beta_{\mathrm{OPD}}\, d_{i,k,c,t}. \tag{25}$$

Replacing the chunk-level advantage $\widehat{A}_{i,k,c}$ in equation 23 with the token-level shaped advantage $\widetilde{A}_{i,k,c,t}$ yields the OPD policy-gradient loss:

$$\mathcal{L}_{\text{OPD-PG}} = -\frac{1}{|\mathcal{B}|} \sum_{(i,k,c)\in\mathcal{B}} \frac{1}{T_{i,k,c}} \sum_{t=1}^{T_{i,k,c}} \min\left(\rho_{i,k,c,t}(\theta)\widetilde{A}_{i,k,c,t}, \bar{\rho}_{i,k,c,t}(\theta)\widetilde{A}_{i,k,c,t}\right). \tag{26}$$

The final student objective combines this policy-gradient loss with reference-policy KL regularization:

$$\mathcal{L}_{\text{student}} = \mathcal{L}_{\text{OPD-PG}} + \beta_{\text{KL}}\mathcal{L}_{\text{KL}}, \tag{27}$$

where $\beta_{\text{OPD}} = 0.01$ and $\beta_{\text{KL}} = 0.01$.

A.3.4 Preliminary Results

Table 8 compares the consolidated 4B OPD student with the corresponding two-model baseline using the checkpoints matched to this OPD study. The 68.9 Overall$_5$ F1 baseline reported here is distinct from the 69.3 main-paper result, which uses the final main-experiment checkpoints. This matched comparison therefore isolates the effect of consolidating the two specialized memory models.

Table 8: Preliminary consolidation of the Memory Archiver and Memory Enquirer with Multi-Teacher OPD. The two-model baseline uses checkpoints matched to this study and differs from the final main-paper configuration.

Configuration	LoCoMo10 Overall$_5$ F1	LongMemEval-S Acc.
4B Archiver + 4B Enquirer	68.9	80.9
4B OPD Model	68.8	80.0

The single-model student reaches 68.8 Overall$_5$ F1 on LoCoMo10 and 80.0% accuracy on LongMemEval-S, compared with 68.9 and 80.9% for the matched two-model baseline. These preliminary results indicate that most of the specialized memory-construction and retrieval behavior can be retained in a single 4B policy, although a small performance gap remains.

A.4 Cross-Backbone Analysis

The matched-backbone comparison in Section 4.3 evaluates different memory systems under the same Judge / Answer Model. We further examine whether the trained LocalMem memory layer can be reused with different downstream backbones without retraining.

Specifically, we keep the 4B GRPO Memory Archiver and 4B SFT Memory Enquirer fixed and change only the Judge / Answer Model. As shown in Table 9, the same trained memory layer can be directly used with either GPT-4o or Qwen3.5-122B.

A.5 Memory-Construction Cost

We further compare the memory-construction cost of different memory systems in Table 10. Across all 10 LoCoMo10 conversations, LocalMem requires only 158 Memory Archiver calls, substantially fewer than the reported construction calls of existing memory systems. Even compared with StructMem, which has the fewest construction calls among the baselines in the table, LocalMem uses approximately $6.7\times$ fewer LLM calls.

The low construction cost follows directly from our memory-construction design. Each interaction window is independently converted into structured memory entries and written to the persistent memory bank. Unlike systems that repeatedly retrieve, link, consolidate, or rewrite previously constructed memories during memory construction, our Memory Archiver does not require iterative maintenance of the existing memory state. This makes the construction cost scale primarily with the number of interaction windows rather than with the number or connectivity of previously stored memory entries.

In addition to requiring fewer model calls, LocalMem also processes substantially fewer tokens during construction. The reported 1.19M total tokens include JSON serialization tokens in the Memory

Table 9: Cross-backbone analysis on LoCoMo10 with the fixed 4B GRPO Memory Archiver and 4B SFT Memory Enquirer.

Judge / Answer Model	MH	Temp	OD	SH	Adv	Overall_4	Overall_5
GPT-4o	47.9	63.2	**31.0**	61.1	90.8	57.3	64.8
Qwen3.5-122B	**48.5**	**63.3**	30.8	**68.3**	**97.1**	**61.3**	**69.3**

Table 10: Memory-construction cost on LoCoMo10, aggregated over all 10 conversations. In, Out, and Sum denote input, output, and total LLM tokens in millions, respectively, and Calls denotes the number of LLM calls. Results for LightRAG, A-MEM, Mem0, MemoryOS, Mem0g, and StructMem are taken from StructMem (Xu et al., 2026), while T-Mem is taken from T-Mem (Guo et al., 2026). For LocalMem, the reported output tokens include JSON serialization tokens, excluding these formatting tokens gives 0.41M output tokens and 0.85M total tokens.

Method	In (M)	Out (M)	Sum (M)	Calls
LightRAG	10.01	1.92	11.93	13,576
A-MEM	9.13	2.37	11.49	11,754
Mem0	10.96	1.24	12.20	9,181
MemoryOS	1.89	0.94	2.87	5,534
Mem0g	33.51	2.31	35.83	53,514
StructMem	1.50	0.44	1.94	1,056
T-Mem	13.58	2.02	15.60	9,689
LocalMem	**0.44**	**0.75**	**1.19**	**158**

Archiver outputs. After removing these formatting tokens, the output volume decreases from 0.75M to 0.41M tokens, corresponding to only 0.85M total construction tokens. We include the unfiltered 1.19M value in Table 10 for a conservative comparison.

A.6 Per-Tier Routing and Fallback Analysis

We first analyze how progressive retrieval routes queries across retrieval tiers under the default Qwen3.5-122B Evidence Sufficiency Judge. We then examine how the choice of Evidence Sufficiency Judge affects routing behavior and final answer quality.

Table 11: Per-tier routing statistics and answer performance on the four non-adversarial LoCoMo10 categories under the default Qwen3.5-122B Evidence Sufficiency Judge. **Count** and **Ratio** denote the mean number and fraction of queries that terminate at each retrieval tier, averaged over three independent runs. **Fallback** denotes queries that exhaust all four retrieval tiers without triggering the sufficiency-based stopping criterion.

Exit stage	Count	Ratio	Overall_4
Tier 1	944.0	61.3%	68.1
Tier 2	68.7	4.5%	57.0
Tier 3	73.7	4.8%	61.0
Tier 4	37.7	2.4%	61.6
Fallback	416.0	27.0%	46.5
Overall	1540.0	100.0%	61.3

Per-tier routing. Table 11 shows where queries exit the progressive retrieval procedure. A majority of queries (61.3%) terminate at Tier 1, while Tiers 2–4 resolve an additional 11.7%. Overall, 73.0% of queries trigger the sufficiency-based stopping criterion, whereas the remaining 27.0% exhaust all four retrieval tiers and proceed to fallback answering. This distribution shows that progressive retrieval avoids unnecessary deeper retrieval for most queries while retaining the ability to acquire additional evidence when needed. Because each tier contains a different subset of queries selected by the stopping criterion, the per-tier F1 values should not be interpreted as a direct comparison of retrieval quality across tiers.

Reaching fallback does not necessarily imply that the final answer is incorrect. Under the default Judge, fallback queries achieve a mean F1 of 46.5; 49.3% achieve F1 ≥ 0.5, and 25.1% achieve F1 $= 1$. Thus, a substantial fraction of fallback queries remain answerable, suggesting that sufficiency estimation itself affects when retrieval terminates.

Table 12: Category-wise routing behavior under different Evidence Sufficiency Judges. Tier 2–4 denotes queries that terminate at one of the later retrieval tiers.

Evidence Sufficiency Judge	Query Type	Tier 1	Tier 2–4	Fallback
Qwen3.5-122B	MH	54.3%	15.5%	30.3%
	Temp	40.4%	12.0%	47.6%
	OD	22.6%	14.9%	62.5%
	SH	76.1%	9.9%	14.0%
Qwen3.5-4B	MH	56.0%	11.2%	32.7%
	Temp	34.0%	8.0%	58.0%
	OD	26.0%	9.0%	64.9%
	SH	72.3%	8.3%	19.4%
LocalMem-4B Enquirer	MH	63.1%	12.4%	24.5%
	Temp	46.1%	12.1%	41.7%
	OD	27.1%	12.5%	60.4%
	SH	78.1%	11.6%	10.3%

Category-wise routing. Table 12 shows a consistent relationship between query type and retrieval depth across all three Evidence Sufficiency Judges. Single-hop queries are predominantly resolved at Tier 1, with Tier 1 termination rates between 72.3% and 78.1%. In contrast, Open-domain queries consistently exhibit the highest fallback rates, ranging from 60.4% to 64.9%, while Temporal queries also frequently require the complete retrieval procedure. This consistent pattern across different Judges indicates that retrieval depth reflects the information requirements of the query rather than only the stopping preference of a particular Judge.

The choice of Judge nevertheless shifts where queries terminate. Compared with Qwen3.5-122B, Qwen3.5-4B increases the Temporal fallback rate from 47.6% to 58.0% and the Single-hop fallback rate from 14.0% to 19.4%. By contrast, the LocalMem-4B Enquirer reduces these rates to 41.7% and 10.3%, respectively, and increases Tier 1 termination for Multi-hop queries from 54.3% to 63.1%. These differences suggest that query difficulty determines the broad retrieval-depth pattern, while the Judge controls how aggressively the system terminates retrieval within that pattern.

Table 13: Evidence Sufficiency Judge ablation on the four non-adversarial LoCoMo10 categories. **Fallback F1** reports answer quality for queries that exhaust all four retrieval tiers without triggering the stopping criterion.

Evidence Sufficiency Judge	**Answer Performance**					**Routing Behavior**			
	MH	Temp	OD	SH	Overall_4	Tier 1	Tier 2–4	Fallback	Fallback F1
Qwen3.5-122B	**48.5**	**63.3**	**30.8**	**68.3**	**61.3**	61.3%	11.7%	27.0%	46.5
Qwen3.5-4B	46.7	62.8	29.7	67.3	60.2	58.4%	8.8%	32.7%	**51.0**
LocalMem-4B Enquirer	46.8	62.5	30.0	67.0	60.1	65.5%	11.9%	**22.6%**	44.7

Evidence Sufficiency Judge ablation. Table 13 compares the default Qwen3.5-122B Judge with two 4B alternatives. Qwen3.5-4B isolates the effect of reducing Judge capacity while retaining a general-purpose model. The LocalMem-4B Enquirer operates at the same 4B scale but is trained for retrieval planning, allowing us to test whether the Memory Enquirer can additionally perform sufficiency estimation without introducing another Judge model.

Replacing Qwen3.5-122B with Qwen3.5-4B reduces Overall F1 from 61.3 to 60.2, with modest decreases across all four categories. Reusing the LocalMem-4B Enquirer as the Judge achieves 60.1 F1, only 1.2 points below the default 122B Judge and close to the standalone Qwen3.5-4B Judge. This result shows that the trained Memory Enquirer can additionally serve as the Sufficiency Judge when a separate Judge model is undesirable.

However, the two 4B alternatives also reveal the limitation of using a smaller model for sufficiency estimation. Qwen3.5-4B terminates fewer queries at Tier 1 than the 122B Judge (58.4% vs. 61.3%) and sends more queries to fallback (32.7% vs. 27.0%). Its fallback subset achieves a higher F1 of 51.0, suggesting that some answerable queries are not recognized as sufficient and are unnecessarily routed through the complete retrieval procedure. In contrast, the LocalMem-4B Enquirer terminates 65.5% of queries at Tier 1 and reduces the fallback rate to 22.6%, while its fallback subset achieves a lower F1 of 44.7. This pattern is consistent with more permissive stopping, where some queries are terminated earlier.

Despite these different routing tendencies, both 4B variants remain below the 122B Judge in final answer performance. This indicates that sufficiency estimation benefits from a more capable model with broader external knowledge and stronger reasoning ability, particularly when determining whether the retrieved evidence is sufficient to support a correct answer. We therefore use the stronger large model as both the Sufficiency Judge and Answer Model in the default configuration. The LocalMem-4B Enquirer can optionally assume the judging role when reducing dependence on an additional large Judge is preferred, at the cost of a modest decrease in accuracy.

Overall, these results suggest a practical division of labor in LocalMem: the local Memory Enquirer is specialized for retrieval planning, while a stronger model with richer external knowledge and reasoning capability is better suited for evidence-sufficiency assessment and final answer generation.

A.7 Local Deployment Discussion

The 4B scale of the Memory Archiver and Memory Enquirer makes local deployment of the memory-management layer practical with existing quantization and mobile inference runtimes. Public GGUF releases of Qwen3.5-4B reduce the model from 8.42 GB in BF16 to approximately 2.5–3 GB with 4-bit quantization, e.g., Q4_K_M occupies 2.78 GB.[1] Quantized Qwen3.5-4B models can be executed with `llama.cpp`, which supports Android deployment and ARM acceleration.[2] These results indicate that a single 4B memory model can fit within the resource range of recent mobile devices, although actual latency and peak memory depend on the device, quantization scheme, context length, and inference backend. Local GPU-equipped desktops or workstations provide even more memory and compute resources for deployment.

The Memory Archiver and Memory Enquirer are used for offline memory construction and online query-time retrieval, respectively. Therefore, the two models do not need to remain resident in memory simultaneously, reducing the peak memory requirement of the memory-management layer.

We emphasize that our experiments do not benchmark inference latency on mobile hardware, and the current default system still uses a large model for evidence-sufficiency assessment and answer generation. Therefore, this discussion applies specifically to the learned memory-management layer rather than to a fully on-device end-to-end long-term interactive agent.

A.8 Prompt Templates

This appendix provides the prompt templates used for answer generation, memory construction, retrieval planning, retrieval-gap planning, and evidence-sufficiency judge. Variables enclosed in braces are instantiated with the corresponding inputs at runtime.

A.8.1 Answer Generation

System Prompt.

```
You are a professional Q&A assistant. Extract concise answers from context. You must output
valid JSON format.
```

User Prompt.

```
Answer the user's question based on the provided context.
User Question: {query}
Relevant Context:
{context_str}
Requirements:
1. First, think through the reasoning process.
2. Then provide a very CONCISE answer (short phrase about core information).
3. Answer must be based ONLY on the provided context.
4. All dates in the response must be formatted as "DD Month YYYY", but more or fewer details
     may be included if needed.
5. Return the response in JSON format.
Output Format:
{
  "reasoning": "Brief explanation of your thought process",
```

[1] https://huggingface.co/unsloth/Qwen3.5-4B-GGUF

[2] https://github.com/ggml-org/llama.cpp/blob/master/docs/android.md

```
  "answer": "Concise answer in a short phrase"
}
Example:
Question: "When will they meet?"
Context: "Alice suggested meeting Bob at 2025-11-16T14:00:00..."
Output:
{
  "reasoning": "The context explicitly states the meeting time as 2025-11-16T14:00:00",
  "answer": "16 November 2025 at 2:00 PM"
}

Now answer the question. Return ONLY the JSON, no other text.
```

A.8.2 MEMORY CONSTRUCTION

System Prompt.

```
You are a professional information extraction assistant, skilled at extracting structured,
unambiguous information from conversations.  You must output valid JSON format.
```

User Prompt.

```
Your task is to extract all valuable information from the following dialogues and convert
     them into structured memory entries.
[Current Window Dialogues]
{dialogue_text}
[Requirements]
1. Complete Coverage: Generate enough memory entries to ensure ALL information in the
     dialogues is captured.
2. Force Disambiguation: Absolutely PROHIBIT using pronouns (he, she, it, they, this, that)
     and relative time (yesterday, today, last week, tomorrow).
3. Lossless Information: Each entry's lossless_restatement must be a complete, independent,
     understandable sentence.
4. Precise Extraction:
- keywords: Core keywords (names, places, entities, topic words)
- timestamp: Absolute time in ISO 8601 format (if explicit time is mentioned)
- location: Specific location name (if mentioned)
- persons: All person names mentioned
- entities: Companies, products, organizations, etc.
- topic: The topic of this information
[Output Format]
Return a JSON array, each element is a memory entry:
[
  {
    "lossless_restatement": "Complete unambiguous restatement (must include all subjects,
     objects, time, location, etc.)",
    "keywords": ["keyword1", "keyword2", "..."],
    "timestamp": "YYYY-MM-DDTHH:MM:SS or null",
    "location": "location name or null",
    "persons": ["name1", "name2", "..."],
    "entities": ["entity1", "entity2", "..."],
    "topic": "topic phrase"
  }
]
[Example]
Dialogues:
[2025-11-15T14:30:00] Alice: Bob, let's meet at Starbucks tomorrow at 2pm to discuss the new
      product
[2025-11-15T14:31:00] Bob: Okay, I'll prepare the materials
Output:
[
  {
    "lossless_restatement": "Alice suggested at 2025-11-15T14:30:00 to meet with Bob at
     Starbucks on 2025-11-16T14:00:00 to discuss the new product.",
    "keywords": ["Alice", "Bob", "Starbucks", "new product", "meeting"],
    "timestamp": "2025-11-16T14:00:00",
    "location": "Starbucks",
    "persons": ["Alice", "Bob"],
    "entities": ["new product"],
    "topic": "Product discussion meeting arrangement"
  },
  {
    "lossless_restatement": "Bob agreed to attend the meeting and committed to prepare
     relevant materials.",
    "keywords": ["Bob", "prepare materials", "agree"],
    "timestamp": null,
    "location": null,
```

```
    "persons": ["Bob"],
    "entities": [],
    "topic": "Meeting preparation confirmation"
  }
]

Now process the above dialogues. Return ONLY the JSON array, no other explanations.
```

A.8.3 RETRIEVAL PLANNING

System Prompt.

```
You are an intelligent retrieval planner.  You must output valid JSON format.
```

User Prompt.

```
You are a retrieval planner.
Analyze the user's question and produce a retrieval plan. Do NOT answer the question.
The plan drives a three-layer memory retriever. The memory store contains self-contained
     statements in which every pronoun is resolved to an explicit name and every time is an
     absolute date. Your outputs are consumed as follows:
- required_info -> dense semantic queries (matched against those statements)
- keywords -> BM25 lexical search over the statement text
- persons / locations / entities / time_expression -> exact metadata filters
Question:
{query}
Think step by step:
1. Identify the minimal information required to answer the question correctly.
2. Decompose the question into independent information requirements.
3. Extract the retrieval signals for the semantic, keyword, and structured layers.
Guidelines
### required_info
- List only the information explicitly required by the question.
- Each item is ONE independent information requirement.
- Make each item self-contained: use explicit entity/person names from the question and
     NEVER use pronouns (he, she, it, they, this).
- Describe the information need as the fact you expect to find, not a keyword list.
- Keep each item to one short phrase; produce at most 4 items.
- Do NOT include information that is merely helpful but unnecessary.
- Do NOT infer facts not mentioned in the question.
Examples:
Question:
When and why did Alice move to London?
required_info:
- The date when Alice moved to London
- The reason why Alice moved to London
----------------------------
Question:
Did Tom graduate after transferring universities?
required_info:
- Whether Tom transferred universities
- Whether Tom graduated
- The time order between Tom's university transfer and Tom's graduation
### keywords
Lexical terms for BM25 search over the memory statements.
- Include the important nouns, verbs, and domain terms from the question.
- Prefer the original wording likely to appear verbatim in the memory.
- Do NOT include stop words.
- Do NOT generate synonyms or rephrasings.
### persons
All person names explicitly mentioned, written as they appear in the question (used for
     exact substring metadata matching). Empty list if none.
### locations
All locations explicitly mentioned. Empty list if none. Only the first location is applied
     as a filter, so list the most relevant location first.
### time_expression
A single natural-language temporal expression from the question (a downstream parser
     converts it to an absolute date range). Return null if none exists.
### entities
All explicitly mentioned named entities (organizations, products, events, etc.), written as
     they appear in the question. Do NOT infer entities that are not explicitly mentioned.
     Empty list if none.

Return ONLY valid JSON:
{
  "required_info": ["..."],
  "keywords": ["..."],
```

```
  "persons": ["..."],
  "locations": ["..."],
  "time_expression": null,
  "entities": ["..."]
}
```

A.8.4 RETRIEVAL GAP PLANNING

System Prompt.

```
You are a retrieval gap planner.  You must output valid JSON format.
```

User Prompt.

```
You are a retrieval gap planner.
The previous retrieval did NOT fully answer the user's question. Your job is to identify
     ONLY the information that is still missing, so it can be re-queried.
Each missing item you output is embedded as a dense semantic query against a memory store of
      self-contained statements (pronouns resolved, absolute dates).
Question:
{query}
Do NOT answer the question.
Do NOT repeat information already supported by the retrieved memory.
Do NOT use external knowledge.
All information requirements for this question:
{required_info}
Information needs already attempted in earlier retrieval:
{searched_required_info}
Retrieved Memory ({result_count} entries):
{context_str}
Your task:
Determine which required information is NOT sufficiently supported by the retrieved memory.
Guidelines:
- Compare each required information item against the retrieved memory.
- Include an item ONLY if it is still missing or unsupported by the memory.
- Do NOT include information that is already sufficiently supported.
- Make each item self-contained: use explicit entity/person names and NO pronouns, so it
     works as a standalone semantic query.
- Each item is ONE independent information requirement, phrased as the fact you expect to
     find (a short phrase, not a keyword list).
- Do NOT output synonyms or rephrasings of the already-attempted needs listed above; if a
     need was already searched and still has no support, the memory does not contain it.
- Do NOT infer facts beyond the retrieved memory.
- If all required information is already supported, return an empty list.

Return ONLY valid JSON:
{
  "missing_information": ["..."]
}
```

A.8.5 EVIDENCE SUFFICIENCY JUDGE

System Prompt.

```
You are an information sufficiency evaluator.  You must output valid JSON format.
```

User Prompt.

```
You are a strict information sufficiency evaluator.
Determine whether the retrieved memory contains sufficient information to answer the
     question correctly and with the level of detail requested.
Question:
{query}
Retrieved Memory Entries ({result_count}):
{context_str}
Evaluation process:
1. Identify the essential information required by the question.
2. Check whether each required information element is explicitly supported by the retrieved
     entries.
3. Determine whether a correct answer can be produced using only the retrieved entries.
Decision rules:
Return "complete" ONLY if:
- All essential information required by the question is present in the retrieved entries.
- The answer can be derived directly from the retrieved entries.
```

```
- No key entity, attribute, relationship, time, quantity, explanation, or other required
    detail is missing.
Return "incomplete" if:
- Any essential information is missing.
- The retrieved entries provide only partial coverage of the question.
- Important details are ambiguous, vague, or unsupported.
- Answering would require assumptions, inference beyond the evidence, or filling information
     gaps.
Guidelines:
- Judge information sufficiency, not relevance.
- Partial answers are incomplete.
- Evidence must cover the scope requested by the question.
- Do not assume missing facts are true.
- Do NOT require more detail than the question asks for.
- Minor missing details that would not affect the correctness of the answer may be ignored.
- When uncertain, choose "incomplete".

Return ONLY valid JSON:
{
  "assessment": "complete" | "incomplete",
  "reasoning": "<brief explanation>"
}
```