JUST-IN-TIME AGENT MEMORY WITH RUNTIME AGENTIC RESEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory is critical for AI agents. Many existing agent-memory systems follow an Ahead-of-Time (AOT) design, constructing memory before a specific request arrives. While this reduces online serving cost, such request-agnostic memory construction can discard fine-grained information that later becomes important. To address this limitation, we propose Just-In-Time Agent Memory (JAM), a trainable framework for query-conditioned context construction at runtime. A Memorizer preserves complete raw histories in a hierarchical page-store with compact navigational summaries, while a Researcher iteratively retrieves, inspects, and integrates evidence for each request. To train these memory-use behaviors, we introduce Memory-Gym, an evidence-grounded data synthesis pipeline covering nine task types across six domains, and optimize the Researcher through verified-trajectory supervised fine-tuning followed by Hint-guided Group Relative Policy Optimization. We demonstrate the effectiveness of JAM across a variety of benchmarks on agent memory and long-context processing, where it achieves stronger task performance than AOT-style memory systems while remaining substantially more efficient than prior trained agentic memory approaches. To support reproducibility and future research, we release our anonymized source code at `https://anonymous.4open.science/r/Just-in-time-Agent-Memory-87D6`.

1 INTRODUCTION

Recent progress in large language models (LLMs) has accelerated the development of AI agents capable of carrying out increasingly complex workflows, such as deep research, software engineering, and other multi-stage applications Huang et al. (2025); He et al. (2025). Such workflows continuously produce long and evolving histories of observations, reasoning traces, and interactions, making agent memory essential for constructing the context needed by downstream tasks Hu et al. (2025).

Existing agent memory systems often follow an Ahead-of-Time (AOT) design philosophy. They compress or organize raw histories before any concrete request arrives, producing pre-constructed memories such as summaries or memory notes Xu et al. (2025); Kang et al. (2025); Fang et al. (2025). Although this request-agnostic preprocessing reduces online serving cost, it can discard fine-grained details and cross-session dependencies that may later become crucial. Such information loss is particularly problematic when task-relevant evidence is dispersed across large, evolving, and temporally interdependent histories, rather than contained in a single retrievable passage Maharana et al. (2024); Wu et al. (2024). Consequently, fixed retrieval or summarization pipelines struggle to assemble the query-specific context needed for downstream reasoning.

These limitations suggest that agent memory should not be viewed merely as storing past information, but as constructing query-relevant context at runtime. We therefore propose **Just-In-Time Agent Memory (JAM)**, a dual-component framework that follows a Just-In-Time (JIT) design philosophy. Instead of relying on fully pre-compiled memory, JAM separates offline memory organization from runtime memory construction. During the offline stage, a *Memorizer* preserves complete historical records in a page-store while maintaining lightweight memory as navigational guidance. At runtime, a *Researcher* conducts runtime agentic research guided by the lightweight memory, retrieving, inspecting, and integrating relevant information to assemble an optimized context for each request. By shifting memory from request-agnostic compression to query-conditioned context construction, JAM reduces information loss and improves adaptability to dynamic information needs.

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.

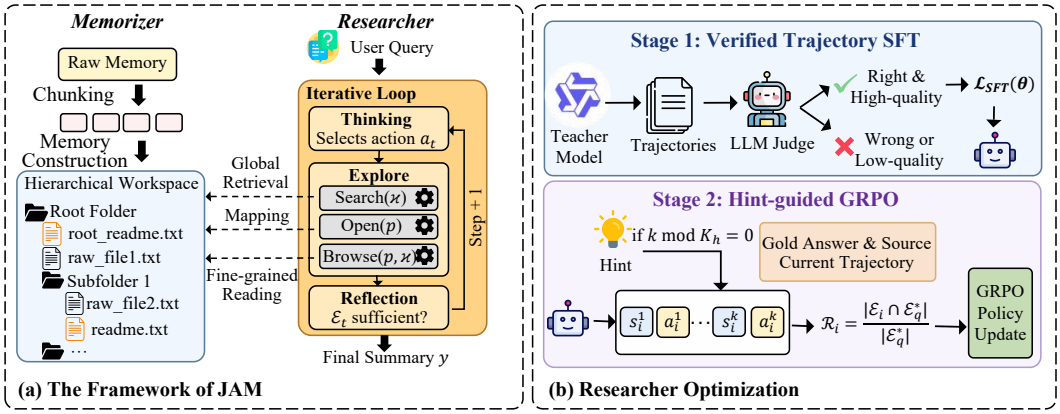


Figure 1: Overview of JAM. (a) The dual-agent JAM framework, where the Memorizer builds a hierarchical workspace and the Researcher performs iterative memory exploration. (b) The two-stage Researcher optimization pipeline, consisting of verified-trajectory SFT and Hint-guided GRPO.

JAM exposes runtime memory construction as an explicit agentic process, making the Researcher's thinking, tool-use, and sufficiency-assessment behaviors traceable and trainable. Although recent training-based memory agents have explored supervision or reinforcement learning for memory use Yu et al. (2025); Yan et al. (2025); Wang et al. (2025); Shi et al. (2025); Yue et al. (2026), their training data are often derived from conventional multi-hop QA datasets or benchmark-specific splits, offering limited coverage of realistic long-history memory scenarios. To address this data scarcity, we introduce **Memory-Gym**, a scalable data synthesis pipeline that provides evidence-grounded training scenarios for memory agents. Memory-Gym covers single-hop localization, multi-hop evidence connection, and multi-session summarization across heterogeneous domains, producing queries, answers, and supporting evidence over accumulated histories.

Building on Memory-Gym, we optimize JAM through a cascaded SFT-RL workflow. Supervised fine-tuning (SFT) on verified teacher-generated trajectories first initializes the Researcher's thinking, tool-use, and sufficiency-assessment behaviors. Hint-guided Group Relative Policy Optimization (GRPO) then further improves multi-step evidence exploration under sparse feedback, enabling JAM to construct more accurate query-relevant contexts at runtime.

Our contributions are summarized as follows: (1) We introduce a Just-In-Time paradigm, which shifts agent memory from static Ahead-of-Time compression to runtime query-conditioned context construction. (2) We propose JAM, a dual-component framework that instantiates this paradigm with a Memorizer for lightweight offline organization and a Researcher for runtime agentic research over complete historical records. (3) We introduce Memory-Gym, a scalable data synthesis pipeline that provides various training scenarios for memory agents, and develop a cascaded optimization workflow that combines verified-trajectory SFT with Hint-guided GRPO to improve JAM's runtime agentic research. (4) Extensive experiments demonstrate that JAM consistently improves task performance over AOT-style memory systems while providing a favorable quality–cost trade-off relative to trained agentic memory approaches.

2 Just-In-Time Agent Memory

JAM instantiates JIT memory with two components: a Memorizer that preserves and organizes accumulated histories offline, and a Researcher that constructs query-conditioned context at runtime.

2.1 Problem Formulation

AI agents often accumulate long histories while performing complex tasks such as software engineering and deep research. We represent such a history as a sequence of temporally ordered sessions $H = \{s_1, s_2, \ldots, s_T\}$, where each session may contain observations, reasoning traces, or interactions. As H grows, directly feeding the full history to the agent becomes inefficient and error-prone, motivating a memory system that produces a compact yet sufficient context for each online request.

Given an online request q and an accumulated history H, the memory system produces a context for the client agent: $c^* \leftarrow \text{Memory}(q, H)$. Ideally, this context should be both useful and compact: it should maximize the client agent's downstream performance while minimizing context size or

serving cost. We formalize this cost-effectiveness objective as:

$$\mathcal{C}^*(q, H) = \arg\max_{c \in \mathcal{C}(H)} \mathrm{Perf}_{\mathcal{A}}(q, c), \qquad c^* = \arg\min_{c \in \mathcal{C}^*(q,H)} |c|. \tag{1}$$

Here, $\mathcal{C}(H)$ denotes the space of contexts constructible from history H, $\mathcal{C}^*(q, H)$ denotes the set of performance-optimal contexts for request q, $\mathrm{Perf}_{\mathcal{A}}(q, c)$ measures the downstream performance of the client agent $\mathcal{A}$ given context c, and $|c|$ denotes context size or serving cost. Under this formulation, AOT-style memory approximates c^* through request-agnostic pre-compression, whereas JIT memory constructs c^* at runtime conditioned on the specific request q.

2.2 Memorizer: Hierarchical Workspace Construction

The Memorizer operates during the offline stage, converting accumulated agent histories into a hierarchical workspace that serves as JAM's persistent page-store. Given a history $H = \{s_1, \ldots, s_T\}$, it preserves raw sessions while constructing compact navigational summaries through two lightweight operations: *memorizing* and *workspace writing*. The same procedure supports both batch construction from existing histories and incremental updates when new sessions are appended.

First, *memorizing* produces a concise memo μ_i for each session s_i, summarizing its key information and serving as a lightweight descriptor for later organization and navigation:

$$\mathrm{Memorizer.memorize}(s_i, \mathcal{W}_{<i}) \to \mu_i, \tag{2}$$

where $\mathcal{W}_{<i}$ denotes the workspace context before incorporating s_i.

Second, *workspace writing* stores the raw content of s_i as a source file and uses its memo μ_i to organize the file within the hierarchical workspace:

$$\mathrm{Memorizer.write}(s_i, \mu_i, \mathcal{W}_{<i}) \to \mathcal{W}_{\leq i}. \tag{3}$$

During this process, semantically related sessions are grouped into coherent directories according to their memos. Each directory maintains a README constructed from the memos of its child files and subfolders, providing a compact navigational summary of the corresponding workspace region.

This design keeps raw files as complete, path-traceable evidence, while memos and READMEs guide the Researcher toward relevant sessions for fine-grained inspection.

2.3 Researcher: Runtime Agentic Research

During online serving, given a request q and the hierarchical workspace $\mathcal{W}$, the Researcher iteratively explores the workspace, collects useful information, and assesses sufficiency to construct an optimized context for the request.

The Researcher is equipped with three workspace-access tools: $\texttt{open}(p)$ reads the README and directory listing under path p; $\texttt{search}(\kappa)$ performs hybrid retrieval on candidate paths using BM25 and dense embeddings; and $\texttt{browse}(p, \kappa)$ reads a specific path and extracts query-relevant information. These tools define the action space for runtime exploration.

We formulate the Researcher as an iterative thinking–exploration–reflection process. At step i, the Researcher observes the request q and the previous interaction trajectory h_{i-1}. It first performs *thinking* to identify the current information needs and generate a set of memory-access actions:

$$A_i = \{a_i^1, \ldots, a_i^{K_i}\} \sim \pi_\theta(\cdot \mid q, h_{i-1}), \qquad a_i^k = (t_i^k, \rho_i^k), \quad t_i^k \in \mathcal{T}, \tag{4}$$

where $\mathcal{T} = \{\texttt{open}, \texttt{search}, \texttt{browse}\}$, K_i is the number of tool calls generated at step i, and ρ_i^k denotes the parameters of the selected tool.

The *exploration* step executes all tool calls in A_i over the workspace $\mathcal{W}$ and returns a set of observations: $O_i = \{o_i^1, \ldots, o_i^{K_i}\}$, $\quad o_i^k = \mathrm{Exec}(a_i^k, \mathcal{W})$. The interaction trajectory is then updated by appending the tool calls and their observations: $h_i = h_{i-1} \oplus (A_i, O_i)$.

After exploration, the Researcher implicitly assesses information sufficiency over the accumulated trajectory: $y_i = \mathrm{Reflect}_\theta(q, h_i)$. It stops when y_i is true or the configured action budget is reached, and continues otherwise. For either stopping condition, the Researcher constructs the final context from the accumulated trajectory: $c^* = \mathrm{Finalize}_\theta(q, h_T)$, retaining request-relevant information and source provenance for the client agent. If the budget is exhausted without a positive sufficiency assessment, the same finalization step summarizes the evidence collected so far for best-effort answering, without further exploration.

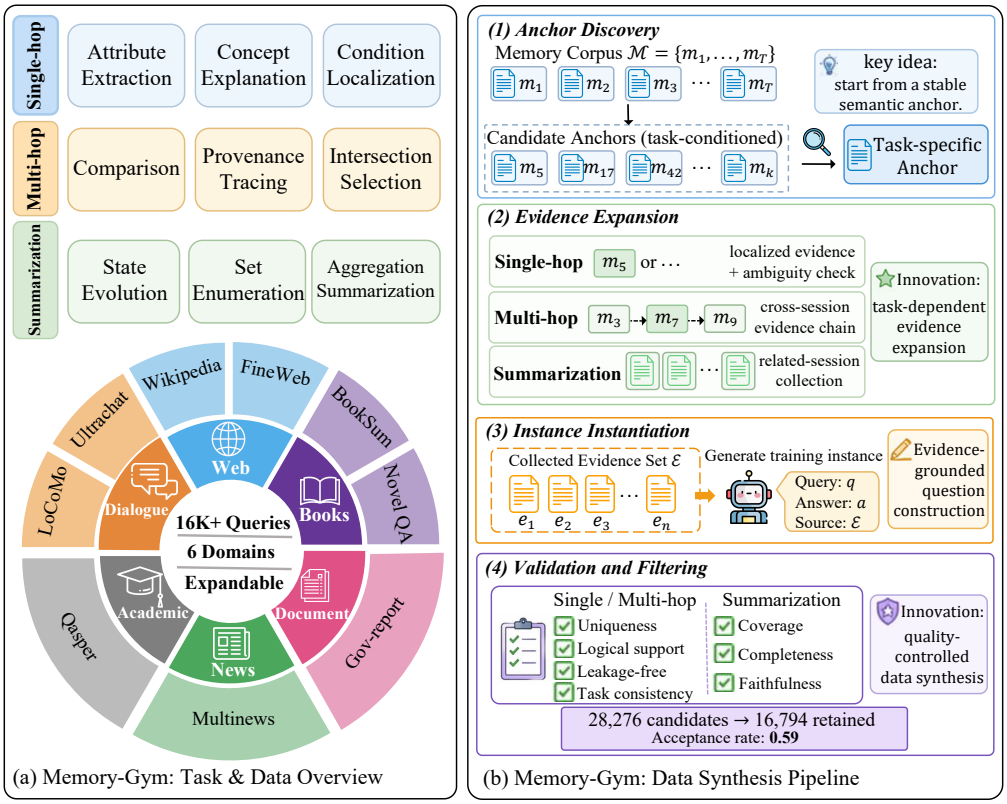


Figure 2: Overview of Memory-Gym. (a) Memory-Gym organizes realistic memory use into three task families covering nine task types across six heterogeneous domains. (b) Memory-Gym constructs evidence-grounded training instances through a four-stage pipeline.

3 Training JAM with Memory-Gym

To train the Researcher, we introduce Memory-Gym for evidence-grounded supervision and a cascaded optimization procedure combining verified-trajectory SFT with Hint-guided GRPO.

3.1 Memory-Gym: Data Synthesis for Memory Agents

Effectively training the Researcher requires suitable data that captures diverse memory-use scenarios over accumulated histories, which existing QA-style or benchmark-specific training sources often fail to provide. To address this data scarcity, we introduce **Memory-Gym**, a scalable data synthesis pipeline designed for JAM's optimization, as shown in Figure 2.

Given a memory corpus $\mathcal{M} = \{s_1, \ldots, s_T\}$ consisting of accumulated sessions and a target task type z, Memory-Gym constructs an instance $x = (q, a, \mathcal{E}, z)$. Here, q is a synthesized request, a is the reference answer, and $\mathcal{E} \subseteq \mathcal{M}$ records the supporting sessions used for construction and validation. Below, we introduce the task taxonomy, source domains, and construction pipeline.

3.1.1 Task Taxonomy and Source Domains

Memory-Gym organizes realistic memory use over accumulated histories into three task families: single-hop, multi-hop, and multi-session. These families focus on the memory operations required by different requests: locating a specific session-level fact, connecting information across sessions, and aggregating observations over broader histories.

Single-hop localization. Single-hop tasks require the agent to identify a memory session that directly supports the answer. Although the required evidence is local once found, the challenge lies in locating it from a large memory corpus containing irrelevant or semantically similar sessions. This family includes attribute extraction, concept explanation, and condition localization.

Multi-hop evidence connection. Multi-hop tasks require the agent to combine partial information from multiple sessions. The answer cannot be obtained from any single session alone, but must be

derived by linking entities, constraints, events, or temporal relations across memory. This family includes comparison, provenance tracing, and intersection selection.

Multi-session summarization. Summarization tasks require the agent to synthesize a structured answer from a broader collection of related sessions. The target answer is often an abstraction over multiple observations, such as temporal changes, repeated mentions, or partially overlapping information. This family includes state evolution, set enumeration, and aggregation summarization.

To improve domain coverage, we instantiate these task families over six heterogeneous domains, including web, books, dialogue, news, academic papers, and long-form documents. Together, these domains expose memory agents to diverse accumulated histories and provide broad training scenarios for learning realistic memory use. Detailed data sources are provided in Appendix A.1.

3.1.2 Task Construction Pipeline

Memory-Gym constructs each training instance through a four-stage pipeline: anchor discovery, task-dependent session expansion, instance instantiation, and validation. The pipeline first builds the session set required by the target task type, and then generates a query-answer pair.

Stage 1: Anchor discovery. Memory-Gym first selects a task-specific anchor, such as an entity, event, concept, condition, or evolving state, as the semantic starting point of an instance. For each sampled source session, it extracts multiple candidate anchors and filters out those that are underspecified, weakly grounded, or unsuitable for the target task type.

Stage 2: Task-dependent session expansion. Starting from the anchor, Memory-Gym expands a supporting session set $\mathcal{E}$ according to the memory-use pattern required by the target task family. For single-hop tasks, it identifies the local session that directly supports the instance and performs corpus-level ambiguity checking. For multi-hop tasks, it retrieves and links related sessions through shared entities, events, constraints, or temporal relations. For summarization tasks, it collects a broader set of sessions associated with the same theme, state, or evolving process, so that the instance requires aggregation rather than isolated fact retrieval.

Stage 3: Instance instantiation. Given the expanded session set $\mathcal{E}$, Memory-Gym generates a query-answer pair (q, a) consistent with the target task type z. The query is constructed to require the target memory-use pattern, while the reference answer is derived only from the selected sessions. The provenance of supporting sessions is also recorded for validation and downstream training.

Stage 4: Validation and filtering. Memory-Gym filters candidates using task-specific quality checks. For single- and multi-hop tasks, we verify answer uniqueness, evidence support, and the absence of answer leakage; for summarization, we additionally assess coverage, completeness, and faithfulness to the source sessions. Of $28{,}276$ candidates, $16{,}794$ passed validation (59.4%). A manual audit of 200 retained instances found that 95.0% met all applicable quality criteria. See Appendices A.3 and A.4 for construction and human-validation details, respectively.

3.2 Cascaded Optimization of JAM

JAM exposes runtime agentic research as a sequential and trainable process. We therefore focus on optimizing the Researcher, while keeping the Memorizer fixed to provide a stable hierarchical workspace. As shown in Figure 1, Memory-Gym supports a cascaded optimization workflow: verified-trajectory SFT initializes the Researcher's tool-use behavior, and Hint-guided GRPO further improves context construction under sparse task-level feedback.

Verified-trajectory SFT. To initialize the Researcher with effective thinking, tool-use, and sufficiency-assessment behaviors, we first construct a supervised corpus of verified Researcher trajectories. Given a Memory-Gym instance and the workspace constructed by the Memorizer, a strong teacher model solves the task under the JAM workflow and produces an interaction trajectory τ.

We retain trajectories whose final answers are correct and whose exploration processes are judged to be high-quality, using an LLM-as-a-judge to filter unsupported or shortcut behaviors such as shallow search without sufficient inspection. The remaining trajectories form a supervised dataset $\mathcal{D}_{\mathrm{SFT}}$. We fine-tune the Researcher policy by maximizing the likelihood of demonstrated action sets:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\sum_{\tau \in \mathcal{D}_{\mathrm{SFT}}} \sum_{i=1}^{|\tau|} \log \pi_\theta(A_i \mid q, h_{i-1}). \tag{5}$$

This stage provides a stable behavioral initialization before policy optimization.

Hint-guided GRPO. SFT provides a useful initialization, but multi-step exploration remains unstable under sparse task-level feedback. We therefore apply Hint-guided GRPO from the SFT checkpoint, injecting lightweight hints during training rollouts to guide search and tool-use decisions.

Specifically, every K_h steps, a hint generator produces a task-specific hint η_k conditioned on the current interaction trajectory and the reference answer: $\tilde{s}_k = s_k \oplus \eta_k$, if $k \bmod K_h = 0$. The hint provides high-level guidance about missing information, unexplored search directions, or potentially useful tool choices. It does not directly reveal the final answer or prescribe a complete search path. Hints are used only during training rollouts and are removed at inference time.

We use source recall as the primary reward, since this stage aims to improve the Researcher's ability to locate task-relevant information rather than merely imitate teacher trajectories. For a sampled trajectory τ, let $\mathcal{E}_\tau$ denote the set of source sessions collected or cited by the Researcher, and let $\mathcal{E}_q^*$ denote the gold supporting session set provided by Memory-Gym. The reward is defined as $R(\tau) = \mathrm{Recall}(\mathcal{E}_\tau, \mathcal{E}_q^*) = \frac{|\mathcal{E}_\tau \cap \mathcal{E}_q^*|}{|\mathcal{E}_q^*|}$. For each query, we sample a group of G trajectories $\{\tau_j\}_{j=1}^G$ and compute their rewards $\{R(\tau_j)\}_{j=1}^G$. The advantage A_j is obtained by normalizing rewards within the group. We then optimize the Researcher with the GRPO objective:

$$J_{\mathrm{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{j=1}^{G} \min(\rho_j A_j,\ \mathrm{clip}(\rho_j, 1-\epsilon, 1+\epsilon)A_j) - \beta D_{\mathrm{KL}}(\pi_\theta \| \pi_{\mathrm{ref}})\right], \tag{6}$$

where ρ_j denotes the policy ratio for τ_j, π_{ref} is the reference policy, and β controls the KL penalty.

4 Experiments

We conduct extensive experiments to evaluate JAM and validate the effectiveness of Memory-Gym as a training source. Specifically, our evaluation aims to answer the following research questions: **RQ1**: How does JAM perform compared with existing AOT-style memory systems and trained memory agents? **RQ2**: Does Memory-Gym provide effective and transferable supervision? **RQ3**: How do the key components of JAM contribute to the overall performance? **RQ4**: How does JAM benefit from increased test-time computation, and what is its effectiveness–efficiency trade-off?

4.1 Experiment Setting

Datasets & Metrics. We evaluate JAM on four long-context and memory-intensive benchmarks: LoCoMo Maharana et al. (2024), LongMemEval (LME) Wu et al. (2024), NarrativeQA (NAQA) Kočiskỳ et al. (2018), and HotpotQA Yu et al. (2025). These benchmarks cover long-term conversational memory, interactive memory, long-document reasoning, and multi-hop reasoning over dispersed evidence. We use the official evaluation metric for each benchmark: F1 for LoCoMo, NarrativeQA, and HotpotQA, and accuracy for LongMemEval.

Baselines. We compare JAM against both training-free and training-based baselines. Training-free baselines include Vanilla LLM and RAG Jiang et al. (2023), as well as AOT-style memory systems A-MEM Xu et al. (2025), Mem0 Chhikara et al. (2025), MemoryOS Kang et al. (2025), and LightMem Fang et al. (2025). Training-based baselines include MEM1 Zhou et al. (2026), MemAgent Yu et al. (2025), and Memory-R1 Yan et al. (2025).

Implementation Details. We use Qwen3.5-4B Qwen Team (2026) as the backbone model for JAM and all AOT-style baselines. For training-based baselines, we use their released checkpoints or official reported settings. JAM uses BM25 and BGE-M3 Chen et al. (2024) as its default lexical and dense retrievers. Further implementation and training details are provided in Appendix B.1.

4.2 Main Results

Table 1 reports the performance of JAM and ten competitive baselines across four benchmarks, demonstrating the overall effectiveness of our framework for memory-intensive tasks.

First, JAM exhibits strong generalization across heterogeneous settings. While several AOT-style memory systems remain competitive on dialogue-centric benchmarks such as LoCoMo, their performance often degrades on long-document and multi-hop reasoning benchmarks. In contrast, JAM

Table 1: Results of JAM and baselines on benchmarks. Best results are in **bold**; second-best results are underlined. Single-hop(SH); Multi-hop(MH); Temporal(TE); Open Domain(OD); Overall(OA). †: As Memory-R1 is not publicly released, we faithfully reported the results from the original paper.

Method	Base LLM	LoCoMo					LME	NAQA	HotpotQA		
		SH	MH	TE	OD	OA			56k	112k	224k
Training-free											
VANILLA	Qwen3.5-4B	48.15	32.98	42.66	19.63	42.45	54.80	31.23	63.56	53.04	36.97
RAG	Qwen3.5-4B	49.04	34.39	44.11	17.68	43.37	56.20	32.86	49.97	44.83	49.87
A-MEM	Qwen3.5-4B	46.99	29.88	42.56	18.65	41.17	56.80	31.30	27.08	25.39	28.32
Mem0	Qwen3.5-4B	34.84	26.45	34.47	16.83	32.10	34.40	26.72	30.19	27.57	24.53
MemoryOS	Qwen3.5-4B	32.75	28.95	36.36	14.50	31.67	36.00	27.21	20.27	22.49	19.61
LightMem	Qwen3.5-4B	35.35	27.88	35.47	18.68	32.97	52.40	28.24	30.73	27.12	26.51
Trained											
MEM1	MEM1-7B	28.56	20.48	32.46	14.73	27.11	32.40	23.86	33.94	32.47	27.32
MemAgent	MemAgent-14B	49.19	36.33	53.77	22.50	46.13	55.20	28.24	56.26	50.70	46.88
Memory-R1-PPO†	Mem-R1-8B†	32.52	26.86	41.57	45.30	41.05	–	–	–	–	–
Memory-R1-GRPO†	Mem-R1-8B†	35.73	35.65	49.86	**47.42**	45.02	–	–	–	–	–
Our Method JAM with training	Qwen3.5-4B	**55.14**	**40.21**	**62.59**	25.12	**52.09**	**65.20**	**45.72**	**64.56**	**59.28**	**58.69**

remains consistently strong across different task types, suggesting that its hierarchical workspace and runtime agentic research enable more robust memory use across domains.

Second, JAM performs strongly even against substantially larger trained memory agents. Despite using a Qwen3.5-4B backbone, JAM outperforms 7B–14B trained baselines across most settings. The results highlight the importance of effective training supervision: Memory-Gym exposes the Researcher to diverse memory-use scenarios, enabling it to learn exploration behaviors that generalize across heterogeneous benchmarks. We examine the contribution of Memory-Gym supervision more directly in Section 4.3.

Third, JAM remains stable as context length increases. On HotpotQA with increasing context sizes, prior methods often degrade when relevant information is buried among longer distractor contexts, while JAM sustains comparatively strong performance. This suggests that JAM can better handle dispersed information because its offline workspace organization and trained Researcher support progressive, query-conditioned exploration at runtime.

Complementary analyses further show that these gains are stable across inference seeds and persist under an LLM-judge evaluation (Appendices B.5 and B.6).

4.3 Effectiveness and Transferability of Memory-Gym

To evaluate Memory-Gym, we first compare different supervision sources and task families under a fixed JAM framework and backbone, with all trained variants using SFT only. We then examine whether the learned memory-use behaviors transfer beyond Memory-Gym's source domains on LongCodeQA Rando et al. (2025).

Table 2: Memory-Gym supervision analysis. All trained variants use SFT only.

Training data	LoCoMo	LME	NAQA	HotpotQA
w/o SFT	48.59	48.80	34.98	45.44
MemAgent data	48.60	55.20	31.98	**54.53**
Memory-R1 data	48.90	57.20	32.42	49.81
SH only	50.59	58.80	34.40	50.45
MH only	48.97	57.60	37.42	51.96
SU only	49.43	59.80	40.94	50.86
Memory-Gym	**50.70**	**60.20**	**41.59**	52.08

Supervision sources and task diversity. Table 2 shows that alternative supervision sources yield uneven gains across benchmarks. MemAgent-derived supervision performs best on HotpotQA, whereas both MemAgent- and Memory-R1-derived supervision underperform the untrained Researcher on NarrativeQA. In contrast, Memory-Gym improves over the untrained Researcher on all four benchmarks and achieves the strongest source-comparison results on three. The task-family analysis shows a complementary pattern: among single-family variants, SH performs best on LoCoMo, MH on HotpotQA, and SU on LongMemEval and NarrativeQA, while the full mixture outperforms every single-family variant across all four benchmarks. These

results are consistent with complementary benefits from evidence localization, cross-session connection, and broader aggregation, motivating the use of diverse memory-use tasks for supervision.

Cross-domain transfer. We further evaluate the Researcher on LongCodeQA, a code-comprehension benchmark from a domain not represented in Memory-Gym, without using any LongCodeQA examples for training or adaptation. As shown in Table 3, overall accuracy increases from 54.08% for the untrained Researcher to 65.67% after SFT and 75.54% after the full SFT+RL pipeline. Improvements are consistent across the 64K, 128K, and 256K subsets. These results indicate that the memory-use behaviors learned from Memory-Gym transfer beyond its source domains, with policy optimization providing further gains.

Table 3: Cross-domain transfer on LongCodeQA.

Researcher	64K	128K	256K	Overall
Untrained	56.58	55.43	49.23	54.08
SFT only	68.42	64.13	64.62	65.67
SFT+RL	**76.32**	**76.09**	**73.85**	**75.54**

4.4 Ablation Study

To evaluate the contribution of each core component, we conduct systematic ablation studies by removing individual modules. The results are summarized in Table 4. First, Researcher optimization consistently improves performance. The untrained variant already benefits from the JAM framework, but remains limited on more challenging memory-intensive tasks. SFT improves the Researcher by providing high-quality trajectory supervision for thinking, tool use, and workspace exploration. GRPO without hints brings further gains over SFT-only, showing that reinforcement learning helps the Researcher improve source discovery beyond imitation. Adding hint guidance achieves the best performance, especially on NAQA and HotpotQA, indicating that lightweight hints stabilize multi-step exploration under sparse rewards. Second, both the Memorizer and the Researcher contribute substantially to JAM. Removing the Memorizer weakens performance by eliminating the hierarchical organization and navigational summaries of the workspace. Removing the Researcher causes an even larger degradation, highlighting the importance of iterative, query-conditioned exploration at runtime. A controlled comparison against a flat raw-history store further shows that hierarchical workspace organization improves both effectiveness and online efficiency: it improves LoCoMo F1 from 49.06 to 52.09 while reducing online latency from 17.11 to 13.81 seconds per query and average research rounds from 3.57 to 3.16 (Appendix B.3). We further examine the sensitivity to the Memorizer backbone in Appendix B.7, where scaling to larger backbone models yields no consistent downstream improvement. Finally, removing either BM25 or dense retrieval also degrades performance, indicating that lexical and semantic retrieval are complementary and jointly provide more informative candidates for downstream exploration.

Table 4: Ablation results on different benchmarks. LoCoMo denotes the overall score. HotpotQA denotes the average over 56k, 112k, and 224k settings.

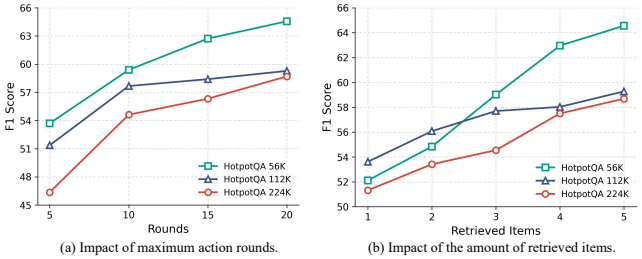
Method	**LoCoMo**	**LME**	**NAQA**	**HotpotQA**
w/o Training	48.59	48.80	34.98	45.44
SFT Only	50.70	60.20	41.59	52.08
GRPO *w/o* Hint	51.42	62.40	43.62	55.64
w/o Memorizer	49.06	64.00	38.87	57.72
w/o Researcher	43.30	58.40	32.51	39.10
w/o BM25	45.90	62.40	38.88	52.14
w/o Embedding	50.86	61.80	39.45	58.13
Full	52.09	65.20	45.72	60.84

4.5 Test-Time Scalability and Efficiency

Scaling test-time computation. Figure 3 examines two forms of inference-time budget: the maximum number of action rounds and the number of retrieved items per search. Increasing the action budget from 5 to 20 consistently improves performance, providing the Researcher with more opportunities to refine its search directions and inspect candidate workspace regions, with diminishing gains at larger budgets. Increasing the number of retrieved items from 1 to 5 shows a similar trend by broadening the candidate set available at each search step.

Importantly, the maximum action budget provides exploration headroom rather than imposing a fixed inference cost. Under the default 20-round limit, the median number of rounds is only 3 on LoCoMo, LongMemEval, and NarrativeQA and 6 on HotpotQA, while budget exhaustion remains below 5% across all benchmarks. Thus, JAM varies its computation across queries and terminates well before the maximum budget in most cases. Detailed runtime statistics are in Appendix B.4.

Runtime efficiency. Figure 4 reports wall-clock latency on LoCoMo, separating one-time offline workspace construction from online query serving. JAM requires 79.53 s to construct a workspace,



(a) Impact of maximum action rounds. (b) Impact of the amount of retrieved items.

Figure 3: Test-time scaling with increasing action rounds (left) and retrieved items (right).

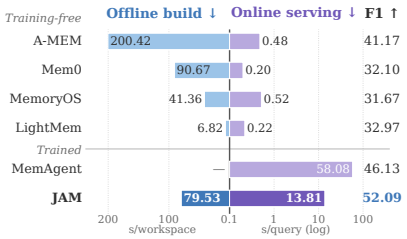


Figure 4: Wall-clock latency and overall F1 on LoCoMo.

lower than A-MEM (200.42 s) and Mem0 (90.67 s), but higher than MemoryOS (41.36 s) and LightMem (6.82 s). This cost is incurred once, after which the resulting workspace is reused across queries. The iterative Researcher introduces additional query-time latency relative to one-shot memory systems. However, this additional computation is accompanied by substantially higher answer quality: JAM achieves an overall LoCoMo F1 of 52.09, outperforming all AOT-style baselines despite their lower serving latency. Compared with the trained agentic baseline MemAgent, JAM reduces query latency from 58.08 s to 13.81 s (76.2%) while improving F1 from 46.13 to 52.09. Together, these results indicate a favorable quality–latency trade-off: JAM spends more computation than lightweight one-shot memory systems to obtain markedly higher answer quality, while remaining substantially faster than MemAgent. Complementary token-consumption results show a similar quality–cost pattern, with JAM achieving the highest F1 while using substantially fewer tokens per task than MemAgent (Appendix B.3).

5 Related Work

5.1 AOT-style Agent Memory Systems

Recent studies on agent memory systems mainly focus on preserving and organizing accumulated contexts for AI agents Packer et al. (2023). A large body of work follows an AOT design philosophy, where historical observations are compressed or structured into pre-constructed memories before specific online requests arrive. Representative methods organize observations into linked memory notes, hierarchical memory stores, or lightweight summarization and consolidation pipelines Xu et al. (2025); Kang et al. (2025); Fang et al. (2025). Nevertheless, their request-agnostic preprocessing can discard fine-grained details or cross-session dependencies that later become important. Moreover, stored memories are often utilized through relatively simple retrieval or summarization procedures, which are less effective for complex memory-intensive tasks that require iterative inspection and query-conditioned context construction.

5.2 Training and Data for Memory Agents

Recent work has trained memory agents with SFT or RL to improve memory use in long-context scenarios Wang et al. (2025); Shi et al. (2025). These studies demonstrate the promise of optimization-based memory agents, but their supervision is often tied to constrained multi-hop QA datasets such as HotpotQA Yu et al. (2025) or benchmark-specific splits Yu et al. (2025); Yan et al. (2025); Yue et al. (2026). This limits the diversity of memory-use scenarios and makes it difficult to train agents for realistic settings that require localization, cross-session connection, temporal tracking, and aggregation over accumulated long contexts.

6 Conclusion

In this paper, we propose JAM, a Just-In-Time agent memory framework that shifts memory from request-agnostic pre-construction to runtime query-conditioned context construction. JAM combines a Memorizer that preserves and organizes accumulated histories in a hierarchical workspace with a Researcher that iteratively explores the workspace to construct task-relevant context. To optimize this process, we introduce Memory-Gym and a cascaded training workflow combining verified-trajectory SFT with Hint-guided GRPO. Experiments across diverse long-context and memory-intensive benchmarks show that JAM consistently improves task performance over AOT-style memory systems, transfers across domains, and offers a favorable quality–cost trade-off for runtime memory construction.

AI USE STATEMENT

Generative AI tools were used in this work as part of the synthetic data construction pipeline for Memory-Gym. Language models assisted with the generation of evidence-grounded training instances and the automatic validation and filtering of generated data, as described in Section 3.1 and Appendix A. The resulting data were subject to explicit quality-control procedures, including human audits of generated and filtered instances to assess data quality and the reliability of automatic validation. We will release the generated tasks and training data, together with the corresponding construction details, to support reproducibility.

Beyond this data-construction pipeline, generative AI tools were used only for language polishing and to improve the organization and readability of the manuscript. All AI-assisted content was reviewed by the authors. The authors take full responsibility for the final content of this work, including its text, experimental results, claims, and conclusions.

ETHICS STATEMENT

This work studies memory systems for AI agents and uses existing research datasets together with synthetically generated training instances. Synthetic data may inherit biases or inaccuracies from the underlying sources and generative models. We mitigate these risks through explicit validation and filtering procedures and human audits, as detailed in Appendix A. Any released data and artifacts will follow the applicable licenses and redistribution requirements of their underlying sources.

JAM preserves raw historical information to support query-conditioned memory construction. In real-world deployments, such histories may contain private or sensitive information, and preserving them can introduce privacy, security, and access-control risks. Appropriate deployment therefore requires safeguards such as access control, data minimization, retention policies, and compliance with relevant privacy requirements. Our experiments focus on research benchmarks and do not evaluate deployment on private user data. The authors have reviewed the ICLR Code of Ethics and take responsibility for the ethical conduct and presentation of this work.

REPRODUCIBILITY STATEMENT

We provide an anonymized implementation of JAM through the code repository linked in the abstract. The paper and appendix document the components required to reproduce our experiments: Appendix A describes the construction, validation, and data sources of Memory-Gym; Appendix B provides training, inference, and evaluation details; and Appendix D reports the Researcher prompt used in our experiments. Upon acceptance, we will publicly release the Memory-Gym tasks and training data, together with the trained model checkpoints, to facilitate reproduction and future research. The authors have reviewed the released materials and made every effort to provide the information and artifacts necessary to reproduce the reported experiments.

REFERENCES

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*, 4(5), 2024.

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.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, 2021.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversa-

tions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051, 2023.

Alexander Richard Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 1074–1084, 2019.

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. *arXiv preprint arXiv:2510.18866*, 2025.

Wikimedia Foundation. Wikimedia downloads. `https://dumps.wikimedia.org`, 2023. English Wikipedia dump, version 20231101.en, accessed via the Hugging Face `wikimedia/wikipedia` dataset.

Google DeepMind. Gemini 3 Flash: Model card. Model card, December 2025. URL `https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Flash-Model-Card.pdf`.

Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–30, 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.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 conference of the north American chapter of the association for computational linguistics: Human language technologies*, pp. 1419–1436, 2021.

Yuxuan Huang, Yihang Chen, Haozheng Zhang, Kang Li, Huichi Zhou, Meng Fang, Linyi Yang, Xiaoguang Li, Lifeng Shang, Songcen Xu, et al. Deep research agents: A systematic examination and roadmap. *arXiv preprint arXiv:2506.18096*, 2025.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 7969–7992, 2023.

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

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

Wojciech Kryściński, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. Booksum: A collection of datasets for long-form narrative summarization. In *Findings of the association for computational linguistics: EMNLP 2022*, pp. 6536–6558, 2022.

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.

MiniMax. Minimax m2.5: Built for real-world productivity. `https://www.minimax.io/news/minimax-m25`, 2026. Accessed: 2026-04-28.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Stefano Rando, Luca Romani, Alessio Sampieri, Luca Franco, John Yang, Yuta Kyuragi, Fabio Galasso, and Tatsunori Hashimoto. Longcodebench: Evaluating coding llms at 1m context windows. *arXiv preprint arXiv:2505.07897*, 2025.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Yaorui Shi, Yuxin Chen, Siyuan Wang, Sihang Li, Hengxing Cai, Qi Gu, Xiang Wang, and An Zhang. Look back to reason forward: Revisitable memory for long-context llm agents. *arXiv preprint arXiv:2509.23040*, 2025.

Cunxiang Wang, Ruoxi Ning, Boqi Pan, Tonghui Wu, Qipeng Guo, Cheng Deng, Guangsheng Bao, Xiangkun Hu, Zheng Zhang, Qian Wang, and Yue Zhang. Novelqa: Benchmarking question answering on documents exceeding 200k tokens, 2024. URL https://arxiv.org/abs/2403.12766.

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

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 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. *arXiv preprint arXiv:2508.19828*, 2025.

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. *arXiv preprint arXiv:2507.02259*, 2025.

Yanwei Yue, Boci Peng, Xuanbo Fan, Jiaxin Guo, Qiankun Li, and Yan Zhang. Mem-t: Densifying rewards for long-horizon memory agents. *arXiv preprint arXiv:2601.23014*, 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*, volume 2026, pp. 58413–58438, 2026.

A Details of Memory-Gym

A.1 Data Sources and Corpus Construction

Memory-Gym is built from six heterogeneous domains: web, books, dialogue, news, academic papers, and long-form documents. These domains are selected to cover diverse forms of accumulated long-context memory, including factual lookup, narrative reasoning, conversational memory, event evolution, scholarly evidence tracing, and structured document understanding. We normalize all raw inputs into memory corpora $\mathcal{M} = \{s_1, \ldots, s_T\}$, where each s_i denotes a memory session. The following paragraphs describe the domain-specific data sources and session construction procedures.

Web. The web domain is constructed from Wikipedia Foundation (2023) and FineWeb Penedo et al. (2024), covering both structured encyclopedic knowledge and diverse open-domain web content. FineWeb provides cleaned general web documents with broad topics and heterogeneous writing styles. We retain FineWeb documents whose token count falls within $[30{,}000, 150{,}000]$ and process each retained document with the JAM Memorizer to obtain memory sessions. Wikipedia provides relatively clean, structured, and entity-centric encyclopedic articles. Since individual articles are usually short, we construct long-context memory corpora through random composition: each session is formed by concatenating 5–8 randomly sampled articles, and each memory corpus contains 30–50 such sessions. This creates web memory corpora with many factual units and distractor articles for evidence localization across sessions.

Books. The books domain is constructed from BookSum Kryściński et al. (2022) and NovelQA Wang et al. (2024), both of which provide long-form narrative texts involving extended plots, characters, events, and state changes. BookSumBooks contains book-length literary texts, while NovelQA consists of full-length novels designed for long-context narrative understanding. For both sources, we filter candidate books by length and retain full texts whose token count falls within $[20{,}000, 500{,}000]$. Each retained book is treated as a long-form raw input and processed by the JAM Memorizer, which segments it into memory sessions and organizes them into the hierarchical workspace. This preserves the original narrative order while converting each book into a session-based memory corpus.

Dialogue. The dialogue domain is constructed from UltraChat Ding et al. (2023) and LoCoMo Maharana et al. (2024), which provide multi-turn conversational data for simulating accumulated conversational contexts. UltraChat is a large-scale open-domain dialogue dataset covering diverse user intents, topics, and conversational styles. For UltraChat, we retain conversations with at least 5 turns and treat each retained conversation as one memory session. For each memory corpus, we randomly sample 25–50 such sessions, resulting in an accumulated dialogue memory corpus composed of multiple conversational episodes. LoCoMo is a long-term conversational memory benchmark designed to evaluate reasoning over information accumulated across extended interactions. We use the official scripts to generate dialogue data. Each generated session contains more than 15 dialogue turns, and each memory corpus contains more than 30 sessions. Compared with UltraChat, LoCoMo provides longer and more memory-intensive conversational contexts for cross-session evidence localization and temporal tracking.

News. The news domain is constructed from MultiNews Fabbri et al. (2019), which contains articles about diverse real-world events, topics, and social issues. Since related news articles often describe evolving events from different perspectives, this domain supports tasks involving event tracking, provenance tracing, comparison, and aggregation. We retain articles with at least 1,000 tokens, encode them using BGE-M3, and cluster them based on cosine similarity. For each memory corpus, we randomly sample 30–35 articles from a target cluster, with each article treated as one memory session. This yields long-context news memory corpora with distributed evidence and realistic distractor reports.

Academic. The academic domain is constructed from Qasper Dasigi et al. (2021), a question-answering dataset grounded in academic papers. Since academic papers contain structured information about research problems, methods, experiments, and results, this domain supports tasks such as concept explanation, method comparison, and evidence tracing. We use the original section structure of each paper for corpus construction, treating each section as one memory session. Papers with fewer than 10 sections are filtered out to ensure sufficient cross-session structure for long-context reasoning.

Document. The document domain is constructed from Gov-report Huang et al. (2021), which contains long-form government reports with hierarchical section structures. These reports describe policies, programs, institutional plans, outcomes, and conditions, making them suitable for condition localization, state tracking, set enumeration, and aggregation tasks. For corpus construction, we preserve the original document hierarchy by recursively traversing the report tree. Each node is treated as one memory session, and its child nodes are further expanded into additional sessions. We retain reports with at least 28 recursively expanded sessions to ensure sufficient depth and breadth for evidence localization and reasoning over structured long documents.

Evaluation contamination control. To avoid evaluation contamination, we ensure that all memory corpora used for Memory-Gym construction are disjoint from the evaluation data used in our experiments. We exclude all released evaluation instances, including their questions, answers, supporting evidence, and source sessions, from Memory-Gym construction. In particular, although LoCoMo is included as a dialogue-domain source, we do not use any released LoCoMo evaluation conversations, questions, answers, or supporting evidence. Instead, we generate new LoCoMo-style dialogue corpora using the official data-generation pipeline, with independently sampled scenarios and sessions. Therefore, the Memory-Gym training corpora have no overlap with the evaluation benchmarks in terms of users, conversations, sessions, queries, answers, or supporting evidence.

A.2 Implementation Details of the Construction Pipeline

We use MiniMax-2.5 MiniMax (2026) as the LLM-based generator for anchor extraction and evidence-grounded question-answer instantiation. Given a memory corpus and a target task type, the generator identifies candidate anchors from the available sessions and constructs query-answer pairs grounded in the selected evidence. We also use MiniMax-2.5 as the judge model to validate generated instances according to task-specific criteria. We use separate prompts for generation and validation to reduce the chance that the same generation procedure accepts invalid instances. For retrieval and clustering, we use BGE-M3 Chen et al. (2024) as the embedding model. Candidate documents or sessions are encoded into dense vectors, and cosine similarity is used to measure semantic relatedness. The resulting similarity scores are used for clustering related documents during corpus construction and for retrieving candidate evidence sessions during task construction.

A.3 Task-specific Construction and Validation

Although Memory-Gym follows a unified construction pipeline, different task types require different evidence structures and validation criteria. We therefore define task-specific rules to ensure that each generated query reflects the intended memory-use pattern, including single-session localization, cross-session reasoning, and multi-session aggregation. Across all tasks, valid instances must be evidence-supported, leakage-free, and consistent with the target task type.

Single-hop tasks. Single-hop tasks require the answer to be supported by a single memory session, while the challenge is to locate the correct session from a large memory corpus. We construct three single-hop task types: attribute extraction, concept explanation, and condition localization.

Attribute Extraction. For attribute extraction, we randomly sample a memory session and ask the generator to identify salient attributes of entities, objects, events, or concepts in the session. We then generate a query about one selected attribute, with the attribute value used as the reference answer. For validation, we retrieve passages using the generated query and discard instances if alternative plausible answers are found. The judge model further checks whether the query-answer pair is reasonable, evidence-supported, and consistent with the attribute extraction task.

Concept Explanation. For concept explanation, we randomly sample a session and ask the generator to identify key concepts or objects that have explicit definitions or property descriptions. The query asks for the definition or properties of the selected concept, and the answer is derived from the corresponding description. During validation, we retrieve relevant passages to check whether other sessions contain competing definitions or descriptions, and further verify that the concept is uniquely grounded in the target session. The judge model then checks evidence support and task consistency.

Condition Localization. For condition localization, we randomly sample a session and ask the generator to identify causal, conditional, or prerequisite relations. We extract the resulting event or phenomenon as the query target and generate a question asking what condition or cause leads to

it. The answer is the corresponding condition or cause in the evidence session. For validation, the judge model checks whether the answer has a valid causal or prerequisite relation with the queried outcome, and we discard instances if other sessions provide conflicting causes or conditions.

Multi-hop tasks. Multi-hop tasks require the answer to be derived by connecting evidence from multiple memory sessions. We construct three types of multi-hop tasks: comparison, provenance tracing, and intersection selection.

Comparison. For comparison, we randomly sample several sessions and ask the generator to identify a feature dimension suitable for entity comparison, such as location, scale, membership, or outcome. We then use this feature as a query to retrieve relevant sessions and ask the generator to identify two to three entities with corresponding evidence. A comparison question is generated based on the selected entities and feature dimension. For validation, the judge checks whether the answer is supported by the retrieved evidence, correctly addresses the comparison, and remains unique after searching each compared entity in the memory corpus.

Provenance Tracing. For provenance tracing, we construct cross-session entity-linking tasks. We first select a target entity that appears in at least two different sessions. One session provides a unique abstract description of the entity without mentioning its name, while another session provides an event or fact associated with the entity. The question is generated using the abstract description and event details, but the entity name is strictly removed. For validation, we apply name-leakage checks, verify that the abstract description uniquely identifies the target entity, and ask the judge to confirm that the full reasoning chain from identity resolution to event grounding is valid.

Intersection Selection. For intersection selection, we construct questions whose target can only be identified by combining multiple weak constraints. Starting from a target entity, we retrieve several sessions containing it and ask the generator to extract broad descriptions from each session. Each individual description should be ambiguous, while their intersection should uniquely identify the target. The generated question combines these constraints and asks about a specific fact or event without revealing the entity name. The judge validates individual ambiguity, intersection-level uniqueness, and answer correctness.

Summarization tasks. Summarization tasks require the agent to aggregate information over multiple memory sessions rather than locating a single evidence unit. We construct three types of summarization tasks: state evolution, set enumeration, and aggregation summarization.

State Evolution. For state evolution, we first sample five consecutive sessions and ask the generator to identify entities whose states change over time. After selecting a target entity, we scan the full memory corpus in chronological order by batches of sessions and ask the generator to identify all sessions that describe the entity's state changes. The collected sessions are then used to generate a question about the evolution of the target entity, with the answer summarizing the complete state transition process. For validation, the judge model checks whether the question is reasonable and whether the answer faithfully reflects the state changes supported by the selected sessions.

Set Enumeration. For set enumeration, we first cluster sessions using embeddings and FAISS to identify semantically related contexts. From a sampled cluster, the generator identifies an extraction criterion defined by a category and a constraint, such as entities of a certain type mentioned under a specific condition. We then scan the full memory corpus to incrementally collect all unique entities satisfying the criterion, together with their source sessions. The final query is generated from the extraction criterion, and the answer consists of the collected entity set. For validation, the judge model checks both correctness and completeness, ensuring that all listed items are supported by evidence and that no supported items are omitted.

Aggregation Summarization. For aggregation summarization, we cluster sessions based on embedding similarity and ask the generator to infer a coherent topic from the top sessions in a cluster. We then use this topic to scan the full memory corpus in batches and collect all sessions relevant to the topic. Given the collected evidence, the generator constructs a question asking for an aggregated summary and produces a reference answer. During validation, the judge model checks whether the instance matches the aggregation summarization task, whether the question and answer are aligned, and whether the answer is faithful to the relevant sessions.

Table 5: Human–LLM agreement on 200 pre-filter candidates. The adjudicated human labels serve as the reference. Acceptance is the positive class for precision, recall, and F1. All values except Cohen's κ are percentages.

Evaluator	Accept rate	Agreement	κ	Precision	Recall	F1
Human reference	62.0	–	–	–	–	–
MiniMax-2.5	58.5	91.5	0.823	95.7	90.3	92.9
GPT-5.5	60.0	92.0	0.832	95.0	91.9	93.4
Qwen3.5-122B-A10B	57.5	90.5	0.803	95.7	88.7	92.1

A.4 Human Validation of Memory-Gym

We assess the quality of Memory-Gym instances and the reliability of automatic validation through three complementary audits: human–LLM agreement on candidates before filtering, a human quality audit of retained instances, and an error analysis of rejected candidates.

Annotation protocol. Across all three audits, two primary annotators independently apply predefined task-specific criteria while blinded to the automatic decisions and each other's labels. A third annotator adjudicates disagreements. Inter-annotator agreement and Cohen's κ are computed from the independent labels before adjudication, whereas quality estimates and human–LLM comparisons use the adjudicated labels. The quality criteria follow the task-specific validation requirements described in Appendix A.3. All instances are assessed for evidence support, absence of answer leakage in the query, and task consistency. Single-hop and multi-hop instances are additionally assessed for answer uniqueness, while summarization instances are assessed for coverage, completeness, and faithfulness to the collected sessions. An instance passes the overall quality assessment only if it satisfies all criteria applicable to its task type.

Human–LLM agreement before filtering. We sample 200 candidates from the pre-filter pool, stratified across all nine task types and six source domains. The two primary annotators achieve 96.0% agreement and Cohen's $\kappa = 0.916$. After adjudication, 124 candidates are accepted, corresponding to a human acceptance rate of 62.0%. We evaluate the same candidates with MiniMax-2.5, GPT-5.5, and Qwen3.5-122B-A10B using identical evidence, validation criteria, prompts, and output formats across the three models. MiniMax-2.5 is the judge used in the construction pipeline; the other two models provide additional comparisons against the human reference. Precision, recall, and F1 treat acceptance as the positive class. As shown in Table 5, all three judges achieve over 90% agreement and $\kappa > 0.80$ with the adjudicated human labels. Their acceptance rates are slightly below the human reference, suggesting a moderately conservative tendency on this sample.

Human audit of retained instances. We sample 200 instances from the 16,794 retained Memory-Gym instances, including 134 single-hop or multi-hop instances and 66 summarization instances. Table 6 reports criterion-level quality and inter-annotator agreement. Overall, 190 of the 200 instances satisfy all applicable criteria, yielding a 95.0% pass rate with a 95% Wilson confidence interval of 91.0%–97.3%. Criterion-level pass rates range from 95.5% to 99.5%. These results support the quality of the retained instances while identifying a small proportion with residual quality issues.

Error analysis of rejected candidates. We sample 100 rejected candidates across task types and domains. Annotators independently assess candidate validity and assign one primary error category to invalid instances. Agreement on the binary validity decision is 98.0%, with Cohen's $\kappa = 0.789$. Among the 94 candidates judged invalid by both primary annotators before adjudication, agreement on the primary error category is 90.4%, with multiclass $\kappa = 0.875$. After adjudication, 96 candidates are confirmed as correctly rejected, while four are judged valid. Table 7 summarizes the adjudicated outcomes. Evidence-grounding failures and ambiguity are the most frequent reasons for justified rejection. The 4.0% figure refers specifically to valid instances within the sampled rejected pool, rather than to the proportion of all valid candidates rejected by the pipeline.

Table 6: Human audit of 200 retained Memory-Gym instances. Applicable denotes the number of instances assessed under each criterion. Passed counts and pass rates use adjudicated labels; agreement and Cohen's κ use the two primary annotators' labels before adjudication.

Quality criterion	Applicable	Passed	Pass rate (%)	Agreement (%)	κ
Evidence support	200	196	98.0	99.5	0.886
Answer uniqueness	134	131	97.8	99.3	0.853
Leakage-free	200	199	99.5	100.0	1.000
Task consistency	200	197	98.5	99.5	0.855
Coverage	66	64	97.0	98.5	0.792
Completeness	66	63	95.5	98.5	0.849
Faithfulness	66	65	98.5	100.0	1.000
Pass all applicable criteria	200	190	95.0	97.5	0.787

Table 7: Human error analysis of 100 rejected Memory-Gym candidates. Each invalid candidate is assigned one primary error category. Counts use adjudicated labels, and percentages are calculated over all 100 sampled rejected candidates.

Outcome / primary error type	Count	Percentage (%)
Evidence-grounding failure	33	33.0
Ambiguity/non-uniqueness	22	22.0
Query leakage or construction issue	16	16.0
Task-consistency failure	14	14.0
Coverage/completeness failure	11	11.0
Correctly rejected (subtotal)	96	96.0
Incorrectly rejected (valid candidates)	4	4.0
Total	100	100.0

Together, these audits provide human evidence for the quality of retained Memory-Gym instances and the reliability of automatic filtering on the evaluated samples.

B Experiment Details

B.1 Implementation Details

All experiments were conducted on computational nodes equipped with 8 NVIDIA H100 80GB GPUs. For SFT, we optimize the Researcher on verified trajectories with the standard next-token prediction objective. We use Qwen3.5-122B-A10B Qwen Team (2026) to generate interaction trajectories and Gemini 3 Flash Google DeepMind (2025) to filter low-quality or unsupported trajectories. A typical SFT run takes approximately 5 hours under this hardware configuration.

Unless otherwise specified, the Researcher performs at most 20 action rounds for each query. For each search action, we retrieve the top-5 candidates from BM25 and the top-5 candidates from BGE-M3, and then merge and deduplicate them. The browse model and the final answer generation model are both instantiated with the untrained Qwen3.5-4B backbone, and are not updated during SFT or RL.

For RL, we optimize the Researcher policy using Hint-guided GRPO, initialized from the SFT checkpoint and guided by the source-recall reward. To improve training efficiency, we use the Verl library Sheng et al. (2025) with vLLM-based rollouts, gradient checkpointing, and FSDP offloading. We sample $G = 8$ trajectories per query for group-relative advantage estimation and train the policy for 200 optimization steps. The policy learning rate is set to 1e-6, the KL loss coefficient β is fixed at 0.001, and the clipping ratio ϵ is set to 0.2. Hints are used only during training rollouts and are removed at test time. A typical GRPO run takes approximately 6 hours under the same hardware configuration.

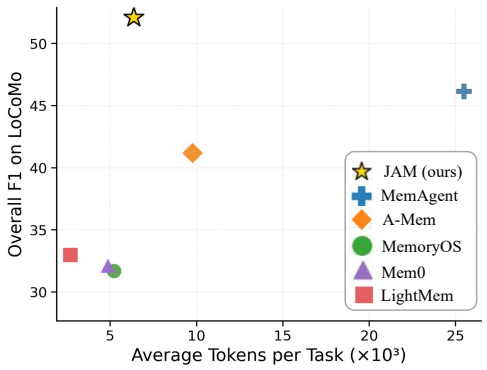


Figure 5: Efficiency comparison on LoCoMo in terms of average token consumption per task and overall F1 score.

Setting	Offline Tokens (/workspace)	Offline Time (s/workspace)	Online Tokens (/query)	Online Time (s/query)	Avg. Rounds	F1
Flat Raw Store	0	0	8174.29	17.11	3.57	49.06
JAM Workspace	36924.51	79.53	**6720.61**	**13.81**	**3.16**	**52.09**

Table 8: Effect of hierarchical workspace organization on LoCoMo. The Flat Raw Store is the *w/o Memorizer* variant in Table 4. Both settings preserve the same raw histories and use the same trained Researcher, retrievers, answer model, and action budget; the Flat Raw Store removes memos, hierarchical organization, and README navigation.

B.2 LongCodeQA: Cross-Domain Evaluation

We evaluate cross-domain transfer on 233 LongCodeQA queries, comprising 76, 92, and 65 examples in the 64K, 128K, and 256K subsets, respectively. Memory-Gym contains no code repositories or code-specific tasks, and no LongCodeQA examples are used for training or adaptation. We compare the untrained Qwen3.5-4B Researcher, the checkpoint after verified-trajectory SFT, and the full checkpoint after SFT followed by Hint-guided GRPO; hints are used only during training. Accuracy is reported for each context-length subset, while the overall score is computed over all 233 queries rather than as the unweighted mean of the three subset scores.

B.3 Efficiency Evaluation Details

We evaluate efficiency on LoCoMo under the same system configurations as the main experiments, separately accounting for one-time offline memory construction and per-query online serving.

Wall-clock latency. We report offline construction time in seconds per workspace and online serving time in seconds per query, averaged over the LoCoMo evaluation instances. For JAM, offline time covers hierarchical workspace construction by the Memorizer, while online time includes the Researcher's iterative exploration and final query processing.

Token consumption. As a hardware-agnostic measure of model-processing cost, we also report token consumption for offline construction and online serving. For the per-task comparison in Figure 5, the one-time offline cost is amortized over the queries that reuse the corresponding workspace and added to the average online token cost per query. Figure 5 shows a similar quality–cost trade-off to the wall-clock results: JAM achieves the highest LoCoMo F1 while using substantially fewer tokens per task than MemAgent, whereas the one-shot memory systems incur lower token costs but also substantially lower answer quality.

Effect of workspace organization. We further analyze the *w/o Memorizer* variant from Table 4, corresponding to a *Flat Raw Store* that preserves the same raw histories but removes hierarchical organization and navigational summaries.

Benchmark	Stopped within 5 rounds (%)	Mean	Median	P90	Budget exhausted (%)
LoCoMo	95.58	3.16	3	4	0.00
LongMemEval	82.80	3.82	3	6	0.40
NarrativeQA	84.00	3.92	3	8	0.00
HotpotQA	43.83	7.11	6	14	4.70

Table 9: Observed Researcher action rounds under a maximum budget of 20. Mean, median, and P90 are computed over all queries, with budget-exhausted queries counted as 20 rounds.

As shown in Table 8, the JAM Workspace incurs a one-time construction cost of 36,924.51 tokens and 79.53 seconds per workspace. Once constructed, however, it reduces online token consumption from 8,174.29 to 6,720.61 tokens per query, latency from 17.11 to 13.81 seconds per query, and average research rounds from 3.57 to 3.16, while improving LoCoMo F1 from 49.06 to 52.09.

This controlled comparison shows that hierarchical organization does more than improve answer quality: by providing compact navigational structure over the same underlying histories, it reduces the amount of online exploration required by the Researcher.

B.4 Researcher Runtime and Termination

We characterize the runtime behavior of the fully trained JAM Researcher across the four main benchmarks under the default maximum budget of 20 action rounds. We report the distribution of exploration length and the frequency of budget exhaustion.

Round counting and stopping criteria. An action round corresponds to one iteration of the Researcher's thinking–exploration–reflection loop and may contain multiple tool calls; the subsequent finalization step is not counted as an exploration round. The Researcher terminates when it produces a positive sufficiency assessment or reaches the 20-round budget. We report the mean, median, 90th percentile (P90), and proportion of queries stopping within five rounds. Queries that reach Round 20 without a positive sufficiency assessment are considered budget-exhausted and are counted as 20 rounds in the reported statistics.

Observed runtime behavior. Table 9 shows that at least 82.80% of queries terminate within five rounds on LoCoMo, LongMemEval, and NarrativeQA. HotpotQA requires longer exploration, with only 43.83% terminating within five rounds and a P90 of 14 rounds, compared with 4–8 on the other benchmarks. Budget exhaustion remains rare, ranging from 0% to 4.70% across benchmarks. These results show that the 20-round setting acts primarily as exploration headroom rather than a fixed inference cost.

Budget-exhaustion handling. When the exploration budget is exhausted, JAM applies the same finalization step used after sufficiency-based termination to construct a context from the evidence collected so far. The 20-round cap therefore bounds exploration cost but does not imply that an exhausted query is incorrect; likewise, a positive sufficiency assessment is an internal stopping decision rather than a guarantee of complete evidence recovery.

B.5 Inference-Time Variability

To assess the sensitivity of JAM to stochasticity during runtime exploration, we repeat inference on all four main benchmarks using three independent random seeds. The trained checkpoint, workspace, prompts, retrieval configuration, action budget, and all other evaluation settings are held fixed, so only inference-time randomness varies across runs.

Across the four benchmarks, the sample standard deviation ranges from 0.31 to 1.34 points. Variability is lowest on LongMemEval and LoCoMo and somewhat higher on NarrativeQA and HotpotQA. Overall, the results remain reasonably stable across inference seeds, with standard deviation below 1.5 points on all four benchmarks.

Benchmark	Run 1	Run 2	Run 3	Mean	Std.
LoCoMo F1	52.09	52.68	51.10	51.96	0.80
LongMemEval Acc.	65.20	64.60	65.00	64.93	0.31
NarrativeQA F1	45.72	43.37	45.21	44.77	1.24
HotpotQA F1	60.84	58.63	58.42	59.30	1.34

Table 10: Performance across three independent inference runs using the same trained JAM checkpoint. Mean and sample standard deviation are reported across inference seeds.

Method	LoCoMo	NarrativeQA	HotpotQA
Evaluated with our judge			
RAG	58.25	48.00	53.13
A-MEM	54.74	45.00	35.68
Mem0	46.23	43.00	38.02
MemoryOS	47.66	40.00	30.21
LightMem	48.57	40.00	41.15
MEM1	39.22	37.00	40.36
MemAgent	66.62	38.00	58.59
JAM	**79.16**	**68.00**	**69.79**
Reported in prior work			
Memory-R1-PPO	57.54	–	–
Memory-R1-GRPO	62.74	–	–

Table 11: LLM-judge accuracy (%) on three benchmarks. The upper block is evaluated using GPT-4o-mini. Memory-R1 results are reproduced from the original paper and are included only for reference because they are not re-evaluated under our judging pipeline. Dashes denote unreported results.

B.6 LLM-AS-A-JUDGE EVALUATION

To assess whether JAM's answer-quality gains persist beyond token-level F1, we additionally evaluate LoCoMo, NarrativeQA, and HotpotQA using an LLM judge. For each question, GPT-4o-mini with temperature 0 receives the reference answer and system prediction and returns `CORRECT` or `WRONG`; accuracy is the proportion of predictions judged `CORRECT`. We follow the Memory-R1 evaluation prompt for LoCoMo and adapt only the task description for NarrativeQA and HotpotQA. LongMemEval is not re-evaluated because its main metric is already accuracy. Memory-R1-PPO and Memory-R1-GRPO results are reproduced from the original paper and are shown separately for reference, rather than as directly comparable results under our evaluation pipeline.

As shown in Table 11, JAM achieves the highest accuracy among methods evaluated under the same judging protocol on all three benchmarks. Compared with the strongest baseline in this group, JAM improves accuracy by 12.54, 20.00, and 11.20 percentage points on LoCoMo, NarrativeQA, and HotpotQA, respectively. These results show that JAM's gains persist under a complementary answer-level evaluation, providing additional evidence that the improvements are not limited to lexical-overlap-based scoring.

B.7 MEMORIZER BACKBONE SENSITIVITY

We examine the sensitivity of JAM to the Memorizer backbone on LoCoMo by replacing the default Qwen3.5-4B Memorizer with Qwen3.5-122B-A10B and GPT-5.5. Each backbone constructs its own workspace from the same raw histories, while the trained Qwen3.5-4B Researcher, retrievers, browse and final-answer models, action budget, and other online inference settings are held fixed. The Researcher is not retrained for the alternative workspaces.

As shown in Table 12, changing the Memorizer backbone does not yield a consistent improvement in downstream performance. Qwen3.5-122B-A10B decreases F1 from 52.09 to 51.30, whereas GPT-5.5 increases it modestly to 52.62. Both alternatives incur higher workspace construction cost,

Memorizer backbone	Build time (s/workspace)	Online time (s/query)	Avg. rounds	F1
Qwen3.5-4B (default)	79.53	13.81	3.16	52.09
Qwen3.5-122B-A10B	132.43	14.36	3.25	51.30
GPT-5.5	117.65	13.19	3.03	52.62

Table 12: Effect of the Memorizer backbone on LoCoMo. Each backbone constructs its own workspace from the same raw histories, while the trained Researcher and online inference configuration are held fixed.

while online latency and average exploration rounds change only slightly. The default Qwen3.5-4B Memorizer therefore provides a favorable balance between workspace-construction cost and downstream performance in the evaluated setting.

We keep the Memorizer fixed during Researcher training because its query-agnostic workspace is reused across requests, providing a stable environment for learning query-conditioned exploration. We do not study Memorizer fine-tuning or joint Memorizer–Researcher optimization; this experiment isolates sensitivity to the Memorizer model choice.

C Case Study

To clearly illustrate the execution process of the Researcher, we present a representative NarrativeQA case in Figure 6 and Figure 7. As shown in Figure 6, the Memorizer converts the original narrative materials into a structured workspace with two major branches, `film_script/` and `source_materials/`. This structure preserves raw evidence while exposing navigable paths, allowing the Researcher to reason over the workspace at both coarse and fine granularities.

Figure 7 shows the corresponding research trajectory. After receiving the query, the Researcher first inspects the overall workspace and issues multiple complementary tool calls, including `search`, `open`, and `browse`, to explore potentially relevant regions. After observing the initial results, it identifies a more promising evidence path and performs focused browsing over the screenplay file. This coarse-to-fine trajectory allows the Researcher to progressively narrow the search space, locate the supporting evidence, and return the correct answer with source provenance. The case illustrates that JAM enables the Researcher to start from global workspace understanding, combine multiple memory-access tools for efficient exploration, and incrementally construct query-relevant evidence beyond one-shot retrieval.

D Prompt

Table 13 shows the prompt template used by the Researcher for memory exploration. The prompt specifies the Researcher's role, the available tools, the think–tool-use loop, and the required source-grounded answer format.

Case Study on NarrativeQA

Scenario: "***The Next Three Days***" (2010 film)

The story follows John Brennan, a community college instructor, whose wife Lara is wrongfully convicted of murder and sentenced to life without parole. After exhausting all legal options, John decides to break her out of jail. He researches prison escapes and meets former escapee Damon Pennington, who provides critical advice about the realities of escaping and evading police.

Question: Who advised John that evading the police would be more difficult than the initial escape?
Gold Answer: "Damon Pennington"
Predicted Answer: "Damon Pennington"

Workspace Structure:

```
workspace/
├── film_script/
│   └── partial_scripts/
│       ├── escape_execution/
│       │   ├── jail_escape_attempt_with_bump_key.md
│       │   └── scene_207_213_action_sequence.md
│       ├── escape_plan_development/
│       │   └── john_surveillance_and_break_in_sequence.md
│       └── police_investigation/
│           └── jail_scene_confrontation.md
└── source_materials/
    ├── imsdb_homepage/
    │   └── imsdb_movie_script_database_homepage.md
    └── narrativeqa_reference/
        ├── drug_dealer_home_invasion_climax.md
        ├── jail_visit_confrontation_scene.md
        ├── jail_visitation_scene_and_drug_surveillance.md
        ├── john_brennan_escape_preparation_and_manhunt.md
        ├── john_interrogation_cctv_evidence.md
        ├── john_lara_letter_exchange_and_discovery.md
        ├── john_manipulates_medical_forms_and_prepares_robbery.md
        ├── john_nicole_park_scene_dialogue.md
        ├── john_returns_home_devastated_mick_knocks.md
        ├── narrativeqa_214810_scene_sequence_memorization.md
        ├── the_last_three_months_film_script_part_2.md
        ├── the_next_three_days_film_script_part_2.md
        └── the_next_three_days_screenplay.md        ← contains answer
```

Figure 6: Case-study setup and Memorizer-constructed workspace for a NarrativeQA example.

Case Study on NarrativeQA

Research Trajectory:

System:

"You are a **Research Agent** exploring a hierarchical knowledge base to answer a question... "

Researcher:Thinking: The user is asking about a specific scene in "The Next Three Days" where someone advised John that evading the police would be more difficult than the initial escape. I need to search through the knowledge base to find this information. The knowledge base has two main directories: source_materials/ and film_script/. Let me search for relevant content, open the main reference directory to check file descriptions, and also browse a promising file about escape preparation.
{"tool": "search", "query": "John police escape difficult advice advised"}
{"tool": "search", "query": "evading police more difficult than initial escape"}
{"tool": "open", "path": "source_materials/narrativeqa_reference/"}
{"tool": "browse", "path": "source_materials/narrativeqa_reference/john_brennan_escape_preparation_and_manhunt.md", "query": "Who advised John about evading police being more difficult than the initial escape"}

System: search result (query: "John police escape difficult advice advised")...

Researcher: Thinking: The search results show several relevant files. The open result for the reference directory reveals that `the_next_three_days_screenplay.md` mentions "meeting with former escapee Damon Pennington for detailed planning information" — this seems highly relevant to the question about who advised John. However, the browse of `john_brennan_escape_preparation_and_manhunt.md` didn't contain the advice.
Let me browse the screenplay file and the second script file, and also do a more specific search.
{"tool": "browse", "path": "source_materials/narrativeqa_reference/the_next_three_days_screenplay.md", "query": "Who advised John that evading police would be more difficult than the initial escape? Find any dialogue where someone tells John about the difficulty of evading police after the initial escape."}
{"tool": "browse", "path": "source_materials/narrativeqa_reference/the_next_three_days_film_script_part_2.md", "query": "Who advised John that evading police would be more difficult than the initial escape? Find any dialogue where someone tells John about the difficulty of evading police after the initial escape."}
{"tool": "search", "query": "advised told told me difficult police manhunt after escape"}

System: browse result (path: `the_next_three_days_screenplay.md`)...

Researcher: Thinking: Based on my search results, I found the answer in the browse result from `/source_materials/narrativeqa_reference/the_next_three_days_screenplay.md`. The file clearly states that **Damon Pennington** advised John that evading police would be more difficult than the initial escape. Damon is a former escapee from jail (seven times) whom John meets in a bar in Brooklyn, NY. He is the author of the book Over The Walls. The specific dialogue shows Damon warning John that "Escaping is easy; the hardest part is staying free" and explaining how modern police protocols (post-9/11 Homeland Security plans) allow for a city lockdown in 15 minutes and a full manhunt with rolling stops on secondary roads within 35 minutes.
Answer:
{
"answer": "Damon Pennington",
"sources": ["/source_materials/narrativeqa_reference/the_next_three_days_screenplay.md"],
"notes": "This advice comes from Damon Pennington during their meeting in Brooklyn where John is planning his escape. Damon emphasizes that the difficulty lies in remaining free after the initial escape due to the speed and precision of police response."
}

Figure 7: Researcher trajectory for locating the supporting evidence in the NarrativeQA case.

Table 13: Researcher prompt for deep-research-style exploration.

Researcher Prompt Template

Your Role.
You are a **Research Agent** exploring a hierarchical knowledge base to answer a question.

Knowledge Base Structure.
A **Knowledge Base Overview** is provided at the end of this system prompt. It includes a summary of the knowledge base content and the full directory structure. Each folder has a README that summarizes the files and subfolders inside it. You can use these README files to quickly understand the workspace structure and decide which files to inspect.

Available Tools.
1. `search`. Search for files matching a query in the knowledge base. It returns relevant file names, paths, and summaries. Use this tool to find potentially relevant files. Parameter: `query`, a keyword or phrase to search for in file content.
2. `browse`. Open a file and extract information relevant to a query. An AI assistant reads the file and returns a summary of query-relevant content. Parameters: `path`, the file path to browse; `query`, a specific query asking for the exact information needed from the file.
3. `open`. Open a folder to view its README summary and directory listing. Use this tool to understand the contents and structure of a directory. Parameter: `path`, the folder path to open.

Process.
Follow the think → tool-use loop. (1) Think about what is known, what is missing, and which tool should be used. (2) Use one or more tools to gather information, with at most five tool calls per round. (3) Receive observations from the tools. (4) Think again based on the observations and decide the next step. (5) Repeat until enough information has been collected. (6) Output the final answer in <answer> tags.

Output Format During Exploration.
Each exploration round should contain one <think> block followed by one or more <tool_use> blocks. Each tool call must be wrapped in its own <tool_use> block and must contain valid JSON.

```
<think>
[Reason about what you know, what information is missing, which tool to use, and why.]
</think>
<tool_use>
{"tool": "search", "query": "your search query"}
</tool_use>
<tool_use>
{"tool": "browse", "path": "your browse path", "query": "your browse query"}
</tool_use>
```

Final Answer Format.
When enough information has been collected, output a final answer in the following format and stop the exploration loop.

```
<think>
[Summarize the collected evidence and reasoning.]
</think>
<answer>
{"answer": "Your comprehensive answer to the question", "sources":
["/path/to/source1.md", "/path/to/source2.md"], "notes": "Additional notes or
caveats"}
</answer>
```

Guidelines.
First check the Knowledge Base Overview, since it already provides the directory tree. Use multiple tools in one round when this can speed up exploration, but use at most five tool calls per round. Use multiple rounds of thinking and tool use until the evidence is sufficient. When outputting <answer>, make sure the answer is grounded in collected evidence and includes source paths. Each <tool_use> block must contain valid JSON with a "tool" field.

Begin.
Review the Knowledge Base Overview, identify the most relevant areas for the question, and start exploring.

Knowledge Base Overview: {*knowledge_base_overview*}