MEMSEEKER: QUERY-TIME MEMORY CONSTRUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory for LLM agents often relies on prebuilt semantic stores or query-time search over raw histories. Prebuilt stores incur upfront construction costs before query needs are known, while stateless search agents repeatedly spend effort locating and synthesizing evidence that earlier queries have already uncovered. To address these limitations, we define **Query-Time Memory Construction (QTMC)** as a memory construction strategy that starts with an empty store and incrementally constructs persistent memory from completed query trajectories for use by subsequent queries. We introduce **MemSeeker**, an implementation of **QTMC** that unifies efficient agentic search and persistent memory construction through structured search trajectories. Its cited working notes support efficient reasoning during search and serve as inputs to deterministic memory construction after answering. Evaluated on 2,400 questions spanning eight heterogeneous benchmarks, **MemSeeker** uses 83.5% fewer tokens while retaining comparable accuracy to the strongest baseline. Further analysis identifies two ways persistent memory supports subsequent queries: supplying evidence for direct answers and guiding the search for missing information. Code is available at `https://anonymous.4open.science/r/Memseeker-Query-Time-Memory-Construction-0D3B`.

1 INTRODUCTION

Memory is crucial for agents built on stateless LLMs to carry information across interactions and tasks (Zhang et al., 2024; Hu et al., 2026b). The information an agent needs to retrieve varies with the task: a conversational assistant may need to recover an updated user preference (Wu et al., 2025), while an agent using a past trajectory for current action may need to reconstruct how a sequence of actions led to an outcome (Zhao et al., 2026). Answering such questions requires selecting relevant records and synthesizing them into useful knowledge. When later questions depend on overlapping evidence, retaining this knowledge can spare the agent from repeating that work. Memory design must therefore account for both the cost of constructing useful knowledge and its value across subsequent queries. This raises a central question: when should an agent turn raw history into persistent memory, and how can that construction support the questions it actually encounters?

Figure 2 (left) abstracts two common memory design strategies. In **Memory as System (MaS)**, an *extract* step converts raw history into candidate memory units, and a *consolidate* step integrates them into a store before queries arrive. This construction can incur substantial upfront cost (Hu et al., 2026a; Fang et al., 2026; Pollertlam & Kornsuwannawit, 2026); queries served solely from the store cannot recover omitted details. In the stateless **Memory as Agent (MaA)** configuration, each question drives an agentic search loop: *search* locates relevant raw records, and *reason* synthesizes them into evidence for answering (Du et al., 2025; Jin et al., 2025; Salemi et al., 2026). This defers evidence selection and synthesis until the query is known, but the resulting knowledge is discarded after answering. Later questions with overlapping evidence must therefore repeat work already performed by earlier queries.

Motivation. Read side by side, the two strategies perform analogous operations: *search* selects raw records much as *extract* does, while *reason* synthesizes them into statements much as *consolidate* does. Their outputs, however, have different lifetimes: MaS retains constructed memory, whereas stateless MaA discards its intermediate knowledge after answering. Based on this asymmetry, we

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.

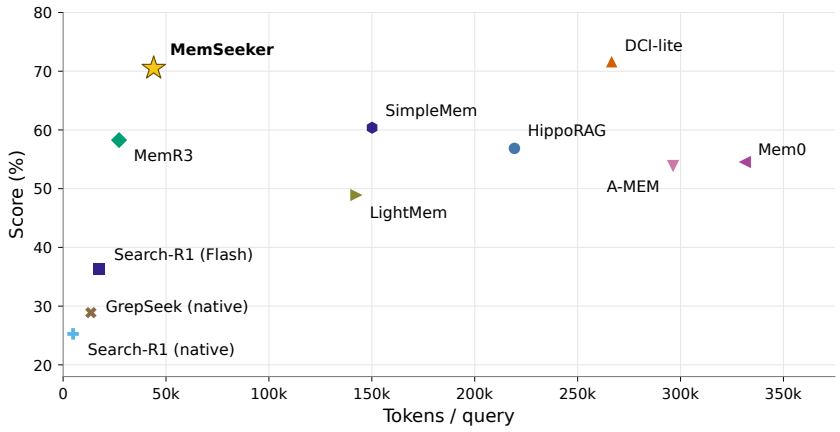

Figure 1: Accuracy versus cost on 2400 questions over eight benchmarks. Points closer to the upper-left indicate a more favorable score–cost trade-off.

propose **Query-Time Memory Construction (QTMC)**, a memory construction strategy that turns agent search trajectories into persistent memory for subsequent queries. Starting from an empty store, QTMC incrementally retains the evidence selected and synthesized during query processing for memory construction. The benefits are twofold. First, QTMC reduces token consumption by avoiding extra *extract* and *consolidate* phases while maintaining a persistent memory. Second, QTMC constructs memories that are more demand-oriented, as memory construction depends on incoming queries rather than pre-defined schemas. Realizing these benefits requires a low-cost protocol for converting agent trajectories into persistent memory: it must preserve reusable evidence while minimizing additional extraction and consolidation. This trajectory-to-store protocol is the central design problem that MemSeeker addresses. We formalize QTMC in Section 3.

MemSeeker is a memory framework built on this strategy. Its key idea is to organize reusable evidence during search, so that persistent memory can be constructed without an additional LLM pass for extraction and integration. It structures the search loop around a working note (Figure 2, right), in which each claim cites its supporting raw records and is revised in place as search proceeds. During search, the note serves as the agent's reasoning state, replacing the growing retrieval history that would otherwise be carried forward and reprocessed at each round. After answering, a deterministic pass materializes the claims and citations into a persistent **Hierarchical Evidence Tree (HET)** with no additional LLM call. The same structured evidence thus supports both efficient search and persistent memory construction. Section 4 details the framework.

Evaluation. Across 2,400 questions from eight heterogeneous memory benchmarks (Zhao et al., 2026; Tavakoli et al., 2026; Dou et al., 2026b;a; Yang et al., 2018; Maharana et al., 2024; Wu et al., 2025; Uddin et al., 2026), MemSeeker cuts token usage by 83.5% while maintaining comparable accuracy to the strongest baseline, placing it on the score–token Pareto frontier among the evaluated methods (Figure 1). To evaluate cross-query evidence reuse through HET, we construct two disjoint question sets on MuSiQue (Trivedi et al., 2022), inspired by QuALITY's use of multiple question sets over shared passages (Pang et al., 2022). Each question in the evaluation set shares supporting evidence with at least one question in the memory-construction set. Reusing the HET built while answering the memory-construction set reduces token usage by 40.2% and improves accuracy from 71.0% to 77.9% on the evaluation set. Additionally, case studies in Section 5.3 show that MemSeeker benefits from HET either by reusing previously committed claims as conclusions, or by using those earlier search records to guide subsequent search.

In summary, our contributions are threefold:

1. We propose **QTMC**, a memory construction strategy that starts from an empty store, constructs persistent memory incrementally from completed query trajectories, and serves subsequent queries from what earlier queries committed (Section 3).

2. We introduce **MemSeeker**, a memory framework that unifies efficient agentic search and persistent memory construction through structured search trajectories (Section 4).
3. We identify two ways persistent memory in QTMC benefits subsequent queries: direct evidence reuse and guidance for further search by systematically separating and studying its value while holding the agent architecture fixed (Section 5.3).

2 Related Work

Agent Memory System Agent memory systems organize history through several representations. Fact- and note-based stores retain atomic facts or linked notes (Chhikara et al., 2025; Xu et al., 2025); graph-based approaches encode temporal facts, causal trajectories, or retrieval relations (Rasmussen et al., 2025; Zhao et al., 2026; Guo et al., 2025; Gutiérrez et al., 2025). Hierarchical and multi-store designs organize memory across abstraction levels or functional stores (Kang et al., 2025; Sarthi et al., 2024; Wang & Chen, 2025), while paging mechanisms manage bounded working context and external storage (Packer et al., 2024). Complementary efficiency techniques reduce construction overhead through staged filtering, deferred consolidation, or semantic compression (Fang et al., 2026; Liu et al., 2026). For ingestion-driven stores, however, construction allocates effort before query needs are known, potentially dominating lifecycle cost when much of the history receives little subsequent demand (Dai et al., 2026; Omri et al., 2026; Pollertlam & Kornsuwannawit, 2026). Queries relying solely on the constructed store also lack access to omitted information, including facts lost during extraction (Dai et al., 2026) or causal structure absent from similarity-based representations (Zhao et al., 2026). These limitations motivate selecting and synthesizing evidence in response to actual queries.

Query-Time Search Agents Query-time search agents develop along two complementary dimensions. *Search policies* guide evidence discovery through search-engine calls or direct corpus interaction (Jin et al., 2025; Salemi et al., 2026; Li et al., 2026). *Context management* controls accumulated observations through truncation and summarization (Li et al., 2026), explicit tracking of established evidence and open gaps (Du et al., 2025), or learned evidence compression (Yu et al., 2026). Hybrid systems combine query-time reasoning with retained pages, raw files, or indexes (Yan et al., 2025; Chen et al., 2026), varying how much processing precedes a query. Two issues concern the state produced during search. Within a query, implicit state tracking can lose relevant evidence and cause repetitive search or premature stopping (Kausik et al., 2026), motivating explicit evidence and gap representations (Du et al., 2025). Across queries, stateless agents discard accumulated evidence and intermediate synthesis, requiring later questions to repeat that work. QTMC targets this cross-query limitation by constructing persistent memory from completed trajectories; MemSeeker couples this construction with the structured state maintained during search.

3 Query-Time Memory Construction

Let H be a fixed raw corpus and $(q_1, \ldots, q_Q)$ an ordered stream of Q questions, with $t \in \{1, \ldots, Q\}$ indexing the current question. Three reference configurations differ in which memory is available when answering and when that memory is constructed:

$$\begin{aligned} \text{MaS:} \quad & (\hat{y}_t, \tau_t) = A(\varnothing, M^{\mathrm{pre}}, q_t), && M^{\mathrm{pre}} = \Phi(H), \\ \text{MaA:} \quad & (\hat{y}_t, \tau_t) = A(H, \varnothing, q_t), && M_t = \varnothing, \\ \text{QTMC:} \quad & (\hat{y}_t, \tau_t) = A(H, M_{t-1}, q_t), && M_t = \mathrm{Commit}(M_{t-1}, G(\tau_t)). \end{aligned} \tag{1}$$

Here A is the query-serving procedure of the selected configuration, $\hat{y}_t$ its answer, and τ_t the completed query-processing trajectory; M_t is the persistent semantic memory after query t, and $\varnothing$ denotes its absence. MaS commits a store $M^{\mathrm{pre}} = \Phi(H)$ before the stream begins, and stateless MaA never populates M_t. QTMC starts from $M_0 = \varnothing$ and answers query t from memory that earlier queries committed; afterwards, G constructs a representation from the trajectory that query just produced and Commit incorporates it into the store. In this simplified MaS configuration, queries access the prebuilt store without reading H directly; the notation does not assume a shared serving algorithm. These are reference configurations rather than exhaustive definitions of memory systems or agents.

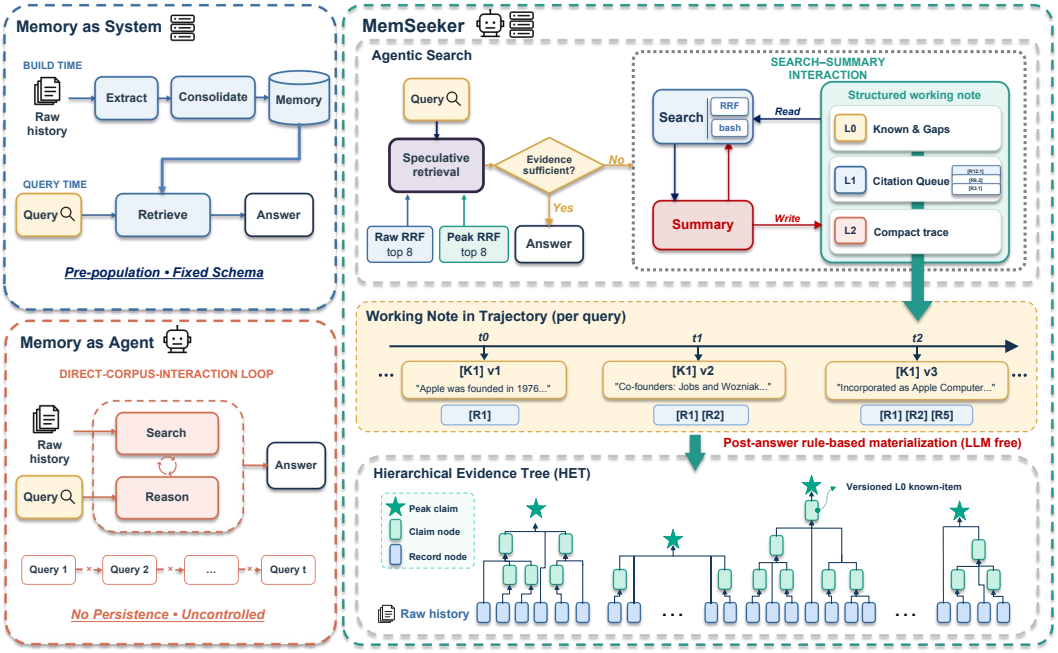

Figure 2: **Query-time work becomes persistent memory in MemSeeker.** Left: the prebuilt MaS and stateless MaA configurations. Right: MemSeeker materializes the cited working note built while answering into an HET, then retrieves its claim peaks alongside raw records for the next question. HET keeps every claim version; retrieval defaults to the final active peaks.

This is a simplified scenario for illustrating the differences among the three configurations: the incoming history is held fixed after ingestion, and queries are then processed against it. In practice, new history may arrive continuously; at any given time, the history observed so far provides a snapshot on which to interpret these construction and query-serving choices.

4 MemSeeker

MemSeeker is an implementation of QTMC, maintaining a single structure across both halves of the lifecycle. Figure 2 shows the two components this yields: an efficient direct-corpus-interaction (DCI) agent that produces a cited working note while answering (Section 4.1), and a Hierarchical Evidence Tree (HET) that stores the note's claims and their revisions for later queries (Section 4.2).

4.1 Efficient DCI Agent

Structured Query-Local State. To retrieve broadly without accumulating raw retrieval history (Section 2), we design a three-layered state representation. For agent's rth round retrieval for query q_t, it receives a context state as L_r:

$$
\begin{aligned}
&L_r = (L_r^0,\ L_r^1,\ L_r^2), \quad \text{where} \\
&L_r^0 = (\mathrm{Known}_r, \mathrm{Gap}_r), \quad \mathrm{Known}_r = \{(claim, ref)\}, \quad \mathrm{Gap}_r = \{(gap, attempt)\}, \\
&L_r^1 = FIFO\{H[ref] : ref \in \mathrm{Known}_r.ref\}, \quad |L_r^1| \le 10, \\
&L_r^2 = (e_1, \ldots, e_{r-1}), \quad e_{r'} = (tool, request, status, n, \Delta ref).
\end{aligned}
\tag{2}
$$

where H is the fixed corpus and $H[i]$ is the raw record with stable index i.

In L^0, Known_r stores $(claim, ref)$ pairs. A $claim$ is text together with its cited indices ref; each ref is a stable index that points to a chunk in raw history. Gap_r stores $(gap, attempt)$ pairs. A gap is a piece of missing information, and its corresponding $attempt$s record the agent's past failed efforts

to close the gap. L^1 is a *first-in-first-out* queue that stores the latest 10 chunks added to Known_r. L^2 uses ledger entry $e_{r'}$ to form a compacted version of the agent's past actions.

The three layers divide the context trade-off discussed in Section 2. L^0 performs the compression, and ref keeps that compression recoverable, leaving the $claim$ a source-linked interpretation rather than a verified fact. L^1 reduces the errors and misinterpretations that L^0 compression introduces by grounding new claims in $H[ref]$ (Section 5.3). L^2 keeps search history cheap enough to grow across rounds and still inform the next request. Records evicted from L^1 remain reachable through ref; all three layers reset for a new question, and HET provides cross-query persistence.

Search–Summary Loop. The *search phase* performs broad retrieval guided by the current question and structured state: $O_r = \mathrm{Execute}(\mathrm{Search}(q_t, L_r), H)$. Tool returns are only conservatively truncated for broad retrieval. The *summary phase* consolidates the retrieved evidence through $L^0_{r+1} = \mathrm{Consolidate}(q_t, L^0_r, L^1_r, O_r)$, where $\mathrm{Consolidate}$ revises cited claims and unresolved gaps, or emits an answer when the evidence suffices. After each note update, the controller refreshes the cited-record buffer and retrieval ledger and discards O_r. This permits broad exploration without accumulating raw retrieval history: the current observation and ten-record buffer bound retained raw evidence, while the compact L^2 ledger can still grow across rounds. Appendix A gives the state, tool, and retry details.

4.2 From Agent Trajectory to HET

Hierarchical Evidence Tree (HET) stores immutable raw records and claim versions as

$$\mathcal{H}_t = (H \cup V_t,\ E^{\mathrm{rev}}_t \cup E^{\mathrm{src}}_t)\,. \tag{3}$$

Here t counts completed queries. Raw corpus H is partitioned into chunks, and each chunk is a base record node. V_t is the accumulated claim-version nodes, each storing one $claim \in L^0_r$. Revision edges E^{rev}_t link a claim version to its predecessor, while E^{src}_t link claim versions to newly cited raw records.

Materialization implements G and Commit of Equation 1 by processing the committed working-note snapshots of the completed trajectory τ_t in order. Within a trajectory, a claim is matched to its preceding active version by trajectory-local identity. If both $claim$ and ref are unchanged, the existing node is reused; otherwise a new immutable version is created and HET is updated as

$$\begin{aligned} V &\leftarrow V \cup \{v\}, && \text{add a claim version,} \\ E^{\mathrm{rev}} &\leftarrow E^{\mathrm{rev}} \cup \{(v,u)\}, && \text{link its predecessor,} \\ E^{\mathrm{src}} &\leftarrow E^{\mathrm{src}} \cup \{(v, H[ref]) : ref \in v.ref \setminus u.ref\}, && \text{link new sources.} \end{aligned} \tag{4}$$

The node v stores the new $claim$ and its ref; u is its preceding version, and $u.ref$ is that version's citation set. For a claim with no predecessor, the revision-edge update is skipped and the prior ref is empty. After all snapshots are processed, these updated sets form the HET $\mathcal{H}_t$ after query t. The retrievable *peaks* $P_{\le t}$ collect the final active versions with nonempty $claim$ and ref from queries completed through t; earlier versions remain as history.

Auditable Claim Revision. Claims produced during search are provisional interpretations of source evidence. HET preserves how these interpretations change: when the agent revises an identified claim within a query trajectory, materialization creates a new version linked to its predecessor and retains that version's ref. This makes changes in $claim$ and ref inspectable while exposing only the final active version of each trajectory-local lineage for retrieval. Revision edges serve this provenance role; serving retrieves claim peaks without traversing those edges. The resulting structure supports auditing of agent-generated revisions, while semantic error detection and correction remain the responsibility of the reasoning agent.

Speculative retrieval. Before entering the agent loop, speculative retrieval for question q_t is conducted as

$$O_0 = \mathrm{Concat}(\mathrm{Retrieve}_{\mathrm{peak}}(q_t, \{claims_{<t}\}),\ \mathrm{Retrieve}_{\mathrm{raw}}(q_t, H))\,. \tag{5}$$

Both retrieval channels use top-8 by default. For approximate token-level fairness, eight claims correspond in context length to roughly two raw records (Appendix A). The agent decides whether the speculative retrieval returns sufficient evidence to answer the query. If not, the agentic loop begins, and the trajectory will update the current HET, contributing to answering the next query.

Table 1: General suite scores (%) and model tokens/query. Blue shades, darkest to lightest, mark the top three scores per column; **bold**/underline mark the two lowest costs. m denotes fine-tuned model, d denotes deepseek, CL = CLBench; CL-Life = CLBench-Life; LME = LongMemEval.

Type	Method	AMA	BEAM	CL	CL-Life	Hotpot	LoCoMo	LME	Memora	Average	Tokens/q (k)
	RRF top-10	45.00	33.49	63.27	33.44	90.00	66.67	73.33	44.95	56.27	**1.96**
	HippoRAG	40.00	32.65	67.69	45.84	83.00	63.00	72.00	50.73	56.86	219.29
	Mem0	30.33	34.26	60.88	32.74	86.00	73.67	69.00	49.44	54.54	331.22
MaS	A-MEM	53.33	25.55	63.79	38.00	69.00	67.33	63.67	50.10	53.85	296.36
	SimpleMem	53.67	36.77	65.61	40.65	83.33	74.00	81.00	48.05	60.38	150.13
	LightMem	30.33	28.82	43.16	18.85	83.00	73.33	72.00	41.77	48.91	142.51
	Search-R1^m	16.33	16.18	9.79	5.53	79.00	13.00	19.67	42.67	25.27	4.84
	Search-R1^d	47.00	24.49	32.73	13.00	82.00	24.67	22.33	44.15	36.30	17.54
MaA	GrepSeek^m	12.00	10.28	10.88	4.95	81.67	41.67	25.33	44.32	28.89	13.49
	MemR3	59.00	47.00	45.92	33.86	91.00	70.00	70.33	49.04	58.27	27.18
	DCI-lite	78.67	53.58	55.18	64.62	96.00	85.33	79.00	60.55	71.62	266.53
QTMC	MemSeeker	72.67	55.59	62.93	50.98	93.67	89.00	82.00	57.33	70.52	44.03

5 Experiments and Analysis

We study two questions through experiments: (1) How capable and efficient is MemSeeker across heterogeneous tasks? (2) How is persistent evidence reused, and how does this reuse affect MemSeeker's performance?

5.1 Experiment Setup

Benchmark Dataset Setup. Building on prior multi-task evaluations of agentic memory (Chen et al., 2026; Agent Memory Leaderboard), we assemble a general suite spanning eight benchmarks from four task types: **conversational and personalized memory** (LoCoMo-Refined (Maharana et al., 2024), LongMemEval (Wu et al., 2025), BEAM (Tavakoli et al., 2026), and Memora (Uddin et al., 2026)); **agent-trajectory recall** (AMA-Bench (Zhao et al., 2026)); **multi-document reasoning** (HotpotQA-Shared (Yang et al., 2018)); and **context learning** (CL-bench (Dou et al., 2026b) and CL-bench Life (Dou et al., 2026a)). We randomly sample 300 questions per benchmark, yielding 2,400 questions. Appendix B details dataset construction.

Baselines. We compare two families of baselines. **MaS**: Mem0 (Chhikara et al., 2025), A-MEM (Xu et al., 2025), LightMem (Fang et al., 2026), HippoRAG (Gutiérrez et al., 2025), and SimpleMem (Liu et al., 2026). **MaA**: DCI-lite (Li et al., 2026), MemR3 (Du et al., 2025), Search-R1 (Jin et al., 2025), and GrepSeek (Salemi et al., 2026). Search-R1 is evaluated in two settings: the released Qwen2.5-7B (Qwen et al., 2025) checkpoint fine-tuned with PPO (Schulman et al., 2017), and a shared-backbone LLM configuration. GrepSeek is evaluated with its released Qwen3.5-9B (Qwen Team, 2026) checkpoint fine-tuned with GRPO (Shao et al., 2024).

Evaluation Metrics. We unify evaluation using LLM-as-Judge with benchmark-specific criteria, including binary correctness and partial-credit scoring. We use judge prompts from Agent Memory Leaderboard (AML) (Agent Memory Leaderboard) for LoCoMo-Refined, LongMemEval, BEAM, and both CL benchmarks; official prompts for AMA-Bench and Memora; and a custom prompt for HotpotQA-Shared. Appendix C.3 details the scoring protocols. Tokens/q is the sum of construction, retrieval, and answer tokens divided by the number of questions.

Runtime settings. We use `deepseek-v4-flash` as the shared runtime LLM backbone, BGE-M3 for dense retrieval, BM25 for sparse retrieval, and `deepseek-v4-pro` for LLM judging. All LLM calls use temperature 0.3. In the general suite, shared-backbone MaA methods use a budget of 30 iterations; the fine-tuned Search-R1 and GrepSeek checkpoints use their default configurations. Appendix C.1 details the MaS and MaA configurations and input preprocessing.

5.2 Experiment Results

Overall performance and efficiency. Table 1 shows that MemSeeker achieves a score of 70.52, within 1.10 points of DCI-lite, while reducing tokens per query by 83.5% (44.03k versus 266.53k). It ranks first on BEAM, LoCoMo, and LongMemEval and second on four other benchmarks, placing it on the score–token Pareto frontier of the evaluated configurations. Tables 4 and 7 report the workload statistics and per-benchmark costs used in the comparisons below.

Comparison with MaS. MemSeeker improves the overall score over SimpleMem, the strongest MaS baseline, by 10.14 points with 70.7% fewer tokens. The cost advantage varies across workloads. On LongMemEval, each 104.11k-token corpus serves one query: MemSeeker scores 82.00 using 35.21k tokens per query, compared with SimpleMem's 81.00 and 512.37k. On LoCoMo, each corpus averages 20.95k tokens and serves 150 queries; all five evaluated MaS methods use fewer tokens than MemSeeker (1.32–11.44k versus 17.25k), while MemSeeker raises the score over the strongest MaS baseline from 74.00 to 89.00. This contrast is consistent with greater amortization of prebuilt memory on smaller corpora serving more queries. On the longer BEAM and Memora corpora, MemSeeker remains less costly than all five MaS methods despite multiple queries per corpus.

Comparison with MaA. MemSeeker reduces tokens relative to DCI-lite on every benchmark, with savings of 70.9%–89.1%. On LongMemEval, it raises the score from 79.00 to 82.00 with 84.5% fewer tokens. Each corpus serves a single query, so this gain arises without cross-query evidence reuse. On BEAM, where each corpus serves 20 queries, it raises the score from 53.58 to 55.59 with 89.1% fewer tokens. DCI-lite retains higher scores on AMA and CL-bench Life, whereas the less costly MemR3 trails MemSeeker by 12.25 points overall (27.18k versus 44.03k tokens per query). The following ablations separate the contributions of query-local context representation and persistent evidence reuse.

5.3 Ablation Analysis

General-suite evaluates MemSeeker's overall performance. However, the benchmarks differ in the number of queries associated with each corpus, therefore offer different opportunities for cross-query evidence reuse, and the complete-system comparison does not isolate its contribution from that of efficient agentic search. We study these components separately below: a controlled MuSiQue (Trivedi et al., 2022) setup isolates the effect of persistent memory while holding the agent architecture fixed, and General-160 isolates the effect of query-local context representation with persistence disabled.

5.3.1 Evidence Reuse

Evaluation Setup. Inspired by QuALITY's use of multiple questions over shared passages (Pang et al., 2022), we construct an evidence-reuse evaluation on MuSiQue (Trivedi et al., 2022), whose annotated supporting paragraphs enable explicit control of cross-question evidence overlap. We retain questions that require agentic search beyond an initial speculative retrieval over the raw corpus during screening. The resulting pool contains **250 native questions** over one shared corpus, partitioned into disjoint A-set (105) and B-set (145). Each B-set question is paired with an A-set anchor according to their supporting-evidence sets: `equal` ($E_A = E_B$, 45 questions), `a_larger` ($E_B \subsetneq E_A$, 47), or `b_larger` ($E_A \subsetneq E_B$, 53). The first two test reuse of a complete answer chain or a relevant subchain; the third tests whether prior evidence guides retrieval of missing information. Appendix B.3 details the filtering procedure and counts.

We first run **A-warm** sequentially from an empty HET, then copy its final HET into two independent branches. **A-seen** replays the same A-set questions, whereas **B-unseen** answers the B-set questions. Both branches start with the HET of A-warm and continue updating their own HET through incoming questions separately. As a comparison, **A-stateless** and **B-stateless** go through the same A/B-set question answering without a HET. This pairs memory-enabled and stateless answers to identical questions, separating exact-question recurrence from transfer to different questions.

Online savings, answer reuse, and knowledge accumulation. Table 2 shows three benefits of HET. First, HET already saves search steps while accumulating knowledge during the initial run, showing later questions benefit from prior memories. Compared with A-stateless, A-warm reduces

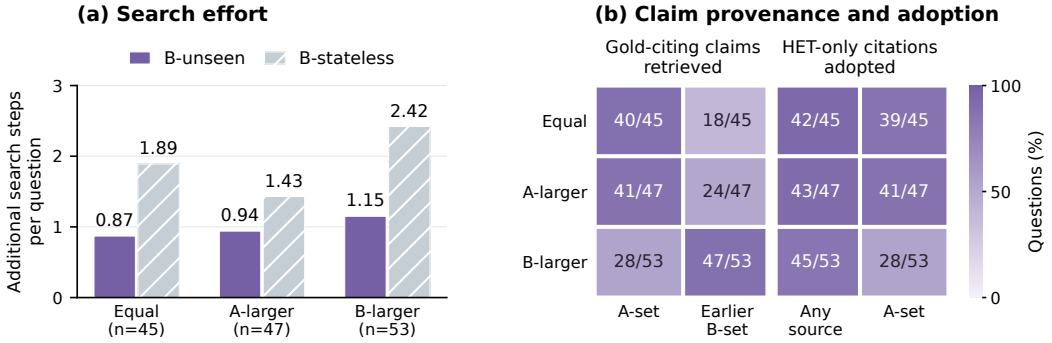

Figure 3: Search effort and claim reuse on MuSiQue B-set. (a) Mean additional search steps across evidence-overlap groups. (b) Number of questions with retrieved or adopted claims, by provenance.

Table 2: Search rate stands for the percentage of MemSeeker entering agentic search. Mean steps are conditional on entering search and exclude initial speculative retrieval. Parentheses show changes relative to A-warm within A-set and B-unseen within B-set.

Condition	Accuracy (%) ↑	Total tokens (k) ↓	Search rate (%) ↓	Mean steps ↓
A-warm	76.19	1659.47	59.05	1.74
A-seen	76.19 (0.00)	1454.92 (−12.3%)	42.86 (−16.19)	1.80 (+3.3%)
A-stateless	75.24 (−0.95)	2286.81 (+37.8%)	69.52 (+10.47)	2.12 (+21.9%)
B-unseen	77.93	2262.47	50.34	1.97
B-stateless	71.03 (−6.90)	3781.32 (+67.1%)	81.38 (+31.04)	2.37 (+20.3%)

the search rate from 69.52% to 59.05% and mean steps among questions that enter search from 2.12 to 1.74, with accuracy increasing from 75.24% to 76.19%. Second, HET enables answer reuse when questions recur. Relative to A-warm, A-seen maintains 76.19% accuracy while further reducing the search rate from 59.05% to 42.86% and total tokens by 12.3%, showing that more repeated questions can be answered directly after speculative retrieval. Third, HET accumulates knowledge that supports unseen questions. Compared with B-stateless, B-unseen increases accuracy from 71.03% to 77.93%, reduces the search rate from 81.38% to 50.34%, and lowers conditional mean steps from 2.37 to 1.97. This comparison captures knowledge accumulated from both A-set and earlier B-set questions.

Claim reuse explains lower search effort. Here we examine evidence reuse at the fine-grained evidence level. Figure 3(a) shows that HET reduces additional search steps after speculative retrieval in all three evidence relations. Panel (b) counts questions. The left two columns count questions whose retrieved claims cite gold evidence, split by A-set and earlier B-set; one question can fall in both, so the columns do not add to the group size. The right two columns count HET-only adoption, where the first summary or answer cites a span present in retrieved HET claims but absent from the same round's raw chunk retrieval. Adoption from any source occurs in 42/45 `equal`, 43/47 `a_larger`, and 45/53 `b_larger` questions, and the rightmost column restricts that count to A-set claims.

For `equal` and `a_larger`, these A-set claims can supply a complete answer chain or a relevant subchain, and 48.9% and 55.3% of questions finish after speculative retrieval, compared with 13.3% and 27.7% without HET. For `a_larger`, the saving is mainly questions that skip further search; conditional mean steps are 2.10 with HET and 1.97 without it, on different selected subsets. For `b_larger`, earlier B-set claims are the larger gold-citing source and guide search toward unresolved relations; conditional mean steps fall from 2.84 to 2.10. The saving therefore combines A-set transfer with continued B-set accumulation. Agent tokens decrease in every group (Appendix C.6). Paired trajectories in Appendix D.1 illustrate direct reuse, search guidance, and a failure where a prior claim narrows the search too far.

Table 3: General-160 architecture ablation. Cross-query persistence is disabled in all variants. Parentheses give changes relative to Full MemSeeker: percentage points for score and relative percentages for tokens.

Agent	Score (%) ↑	Tokens/query (k) ↓
Full MemSeeker	73.47	41.97
History Concat	66.44 (−7.03)	72.92 (+73.7%)
w/o L1	67.16 (−6.31)	33.08 (−21.2%)

5.3.2 Context Representation for Search Agent

Then, we study how the three-layered state representation affects agent performance, especially the role of L^1. Based on the general suite, we pick 20 questions per bench to form a general-160 subset. We exclude HET and operate MemSeeker in three context settings: MemSeeker original, without L^1 (MemR3-style), and naive history message concatenation (DCI-lite-style; Appendix C.4). Table 3 shows that the original agent scores 73.47%, against 66.44% for History Concat and 67.16% without the citation-derived L^1. Two patterns are observed: First, simply concatenating agent trajectory costs more tokens with no performance gain. Conversely, noisy context disrupts the agent's reasoning process. Structured context uses 42.4% fewer tokens than History Concat at similar retrieval rounds (1.869 versus 1.813). Second, raw records in L^1 provide an extra grounding source to avoid misunderstanding across multiple search rounds. Although removing L^1 saves 21.2% of tokens, it lowers accuracy by 6.31 points. We analyze and list some notable cases in Appendix D to show the mechanism: without the raw records, the compressed note misstates a qualifier that the source still contains, and the agent repeatedly re-reads records it has already cited while missing the detail the note dropped.

6 Conclusion

We introduced Query-Time Memory Construction (QTMC), which turns completed query trajectories into persistent memory for subsequent queries. MemSeeker implements QTMC by combining structured agentic search with deterministic memory construction. It lies on the score–token Pareto frontier of the evaluated methods and benefits from reusing stored evidence both to answer new queries directly and to guide subsequent search.

Future work will develop more effective trajectory-to-system protocols for validating and updating reusable knowledge from agent execution. A second direction is scaling QTMC to large datasets through parallel agents that read and contribute to a shared memory system. Such designs must coordinate concurrent construction, reconcile overlapping or conflicting claims, and control memory visibility so that agents can reuse one another's work while preserving provenance and consistent updates.

AI USE STATEMENT

In this work, we used generative AI tools to identify relevant literature, discuss research methodology and experiments, clean and reformat datasets, and assist with interpreting results. We developed the conceptual framework and prepared the implementation specifications. AI assistance with method implementation was limited to implementing details based on these hand-written documents, and we subsequently tested the resulting implementation against the specifications. We also used generative AI tools to suggest the manuscript structure, summarize related literature, draft portions of the manuscript, improve readability, and create or modify figures and tables. We reviewed all AI-assisted work. We did not use generative AI tools to formulate mathematical claims or to propose or refine hypotheses. Proof development, translation, and qualitative or thematic data analysis were not applicable to this work. Results for the general suite, General-160, and the MuSiQue ablation were obtained from stored experiment outputs. Language models were also used to grade answers as part of the evaluation protocol. We take full responsibility for the final manuscript, including all AI-assisted text, code, claims, and other artifacts.

ETHICS STATEMENT

Persistent memory can retain sensitive information and propagate incorrect statements. Our experiments use existing benchmark corpora; derived questions and citations preserve their relationship to those sources. Source-linked claims are not factual guarantees. Any redistribution of source text, traces, or derived material must respect upstream licenses and privacy restrictions; this manuscript does not claim that redistribution permissions or a public release have been completed.

REPRODUCIBILITY STATEMENT

The accompanying source distribution contains the MemSeeker implementation, the frozen question indexes, the MuSiQue reconstruction, and the judge prompts and scoring logic. Download the general-suite sources with `python run_benchmark.py general_suite --download --mode prepare`, and supply the official `musique_ans_v1.0_train.jsonl` for the MuSiQue ablation. Both experiments are then run with `run_benchmark.py`: `general_suite --mode run` and `het_ablation --musique-raw PATH`.

REFERENCES

Agent Memory Leaderboard. Agent Memory Leaderboard. https://agentmemoryleaderboard.ai/. Accessed: September 23, 2026.

Zhikai Chen, Jialiang Gu, Junyu Yin, Xianxuan Long, Shenglai Zeng, Xiaoze Liu, Kai Guo, Keren Zhou, and Jiliang Tang. Exploring cross-scenario generality of agentic memory systems: Diagnostics and a strong baseline, 2026. URL https://arxiv.org/abs/2606.04315.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

Zijie Dai, Shiyuan Deng, Sheng Guan, Yizhou Tian, Xin Yao, Xiao Yan, and James Cheng. Recmem: Recurrence-based memory consolidation for efficient and effective long-running llm agents, 2026. URL https://arxiv.org/abs/2605.16045.

Shihan Dou, Yujiong Shen, Chenhao Huang, Junjie Ye, Jiayi Chen, Junzhe Wang, Qianyu He, Shichun Liu, Changze Lv, Jiahang Lin, Jiazheng Zhang, Ming Zhang, Shaofan Liu, Tao Ji, Zhangyue Yin, Cheng Zhang, Huaibing Xie, Jianglu Hu, Jingcheng Deng, Lincheng Li, Minda Hu, Shaolei Wang, Syrus Zhao, Weichao Wang, Yan Lei, Yang Liu, Yanling Xiao, Yiting Liu, Zenan Xu, Zhen Guo, Ziliang Zhao, Pluto Zhou, Tao Gui, Qi Zhang, Xuanjing Huang, Yu-Gang Jiang, Di Wang, and Shunyu Yao. Cl-bench life: Can language models learn from real-life context?, 2026a. URL https://arxiv.org/abs/2604.27043.

Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, Huaibing Xie, Jianglu Hu, Shaolei Wang, Weichao Wang, Yanling Xiao, Yiting Liu, Zenan Xu, Zhen Guo, Pluto Zhou, Tao Gui, Zuxuan Wu, Xipeng Qiu, Qi Zhang, Xuanjing Huang, Yu-Gang Jiang, Di Wang, and Shunyu Yao. Cl-bench: A benchmark for context learning, 2026b. URL https://arxiv.org/abs/2602.03587.

Xingbo Du, Loka Li, Duzhen Zhang, and Le Song. Memr^3: Memory retrieval via reflective reasoning for llm agents, 2025. URL https://arxiv.org/abs/2512.20237.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. Lightmem: Lightweight and efficient memory-augmented generation, 2026. URL https://arxiv.org/abs/2510.18866.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. Lightrag: Simple and fast retrieval-augmented generation, 2025. URL https://arxiv.org/abs/2410.05779.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models, 2025. URL https://arxiv.org/abs/2405.14831.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions, 2026a. URL https://arxiv.org/abs/2507.05257.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of ai agents, 2026b. URL https://arxiv.org/abs/2512.13564.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning, 2025. URL https://arxiv.org/abs/2503.09516.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent, 2025. URL https://arxiv.org/abs/2506.06326.

Chinmaya Kausik, Adith Swaminathan, and Nathan Kallus. The context gathering decision process: A pomdp framework for agentic search, 2026. URL https://arxiv.org/abs/2605.07042.

Zhuofeng Li, Haoxiang Zhang, Cong Wei, Pan Lu, Ping Nie, Yi Lu, Yuyang Bai, Shangbin Feng, Hangxiao Zhu, Ming Zhong, Yuyu Zhang, Jianwen Xie, Yejin Choi, James Zou, Jiawei Han, Wenhu Chen, Jimmy Lin, Dongfu Jiang, and Yu Zhang. Beyond semantic similarity: Rethinking retrieval for agentic search via direct corpus interaction, 2026. URL https://arxiv.org/abs/2605.05242.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for llm agents, 2026. URL https://arxiv.org/abs/2601.02553.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents, 2024. URL https://arxiv.org/abs/2402.17753.

Yasmine Omri, Ziyu Gan, Zachary Broveak, Robin Geens, Zexue He, Alex Pentland, Marian Verhelst, Tsachy Weissman, and Thierry Tambe. Agent memory: Characterization and system implications of stateful long-horizon workloads, 2026. URL https://arxiv.org/abs/2606.06448.

OpenAI. tiktoken: A fast BPE tokenizer for use with OpenAI's models. https://github.com/openai/tiktoken. Accessed: September 23, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel R. Bowman. QuALITY: Question answering with long input texts, yes! In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5336–5358, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.391. URL https://aclanthology.org/2022.naacl-main.391/.

Natchanon Pollertlam and Witchayut Kornsuwannawit. Beyond the context window: A cost-performance analysis of fact-based memory vs. long-context llms for persistent agents, 2026. URL https://arxiv.org/abs/2603.04814.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

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

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory, 2025. URL https://arxiv.org/abs/2501.13956.

Alireza Salemi, Chang Zeng, Atharva Nijasure, Jui-Hui Chung, Razieh Rahimi, Fernando Diaz, and Hamed Zamani. Grepseek: Training search agents for direct corpus interaction, 2026. URL https://arxiv.org/abs/2605.29307.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. Raptor: Recursive abstractive processing for tree-organized retrieval, 2024. URL https://arxiv.org/abs/2401.18059.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Mohammad Tavakoli, Alireza Salemi, Carrie Ye, Mohamed Abdalla, Hamed Zamani, and J Ross Mitchell. Beyond a million tokens: Benchmarking and enhancing long-term memory in llms, 2026. URL https://arxiv.org/abs/2510.27246.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multi-hop questions via single-hop question composition, 2022. URL https://arxiv.org/abs/2108.00573.

Md Nayem Uddin, Kumar Shubham, Eduardo Blanco, Chitta Baral, and Gengyu Wang. From recall to forgetting: Benchmarking long-term memory for personalized agents, 2026. URL https://arxiv.org/abs/2604.20006.

Yu Wang and Xi Chen. Mirix: Multi-agent memory system for llm-based agents, 2025. URL https://arxiv.org/abs/2507.07957.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory, 2025. URL https://arxiv.org/abs/2410.10813.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

B. Y. Yan, Chaofan Li, Hongjin Qian, Shuqi Lu, and Zheng Liu. General agentic memory via deep research, 2025. URL https://arxiv.org/abs/2511.18423.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL https://arxiv.org/abs/1809.09600.

Jing Yu, Yibo Zhao, Jiaming Zhang, and Xiang Li. Lazymem: Retrieve broadly, construct selectively for efficient long-term agent memory, 2026. URL https://arxiv.org/abs/2607.22690.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents, 2024. URL https://arxiv.org/abs/2404.13501.

Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, Yuandong Tian, and Jishen Zhao. Ama-bench: Evaluating long-horizon memory for agentic applications, 2026. URL https://arxiv.org/abs/2602.22769.

A AGENT SPECIFICATION

A.1 STATE AND TOOL CONTRACT

L^0 is materialized as a text file with two sections, Known and Gap. Known items contain a *claim* and its *ref*, such as `(R12.1)`. Stable Known identifiers are retained across revisions. Gap entries carry stable identifiers and controller-owned active/exhausted status; nested *attempt* records describe unsuccessful searches. A summary commit replaces the full file with `write_file`. The following is a schematic schema, not a measured trajectory:

```
# Known
1. [K1] A source-supported statement. (R12.1)
# Gaps
1. [G1] active - a missing premise.
   Failed attempts:
   1. {"tool":"search","request":"focused query","reason":"returned
    records did not resolve the gap"}
```

L^1 is derived from raw records cited by Known items, uses FIFO eviction, and holds at most ten records. L^2 stores the iteration, tool name, query or script, success status, returned-item count, and *ref*s newly promoted into L^0. Each stored request is truncated to its first 100 tokens. Neither state contains model reasoning. When a memory directory is configured, the controller atomically saves L^0, L^1, and L^2 after each summary phase.

The search phase exposes `search` and `bash`; the summary phase exposes `write_file` and `answer`. Search cannot see current or earlier raw-result blocks, and summary cannot see L^2. Up to four independent tool calls share one retrieval iteration. Their combined current output is prefix-capped at 10,000 tokens and discarded after summary. The controller-owned initial search uses the original question and can skip the first search-model call. If it fails, normal search starts without consuming a retrieval iteration.

A rejected search or summary action is retried in the same phase. Rejection covers an unparseable call, a tool the phase does not expose, an invalid argument, a failed execution, a citation that does not resolve in the corpus, and an empty answer. The controller returns that error only as ephemeral feedback on the retry request. The rejected action is omitted from the retrieval budget and is not written into L^0, L^1, or L^2. A successful call that promotes no evidence into Known is not retried.

A.2 RETRIEVAL AND COMPUTATION

Raw records are partitioned into non-overlapping child chunks, with a target of 1,024 tokens, a minimum of 768, and a maximum of 1,432. A record within the maximum remains complete. On conversational benchmarks, the partition is taken inside a single dialogue. On text benchmarks, the minimum and target keep a long document from ending in a short tail. Both lexical and dense search operate on the same stable child *ref*s. Lexical and dense rankings are combined by reciprocal-rank fusion (RRF) with smoothing constant $k = 60$.

Speculative retrieval uses the original question to retrieve raw records and HET claim peaks before the first agentic search round, with top-8 as the default for each channel. We interpret retrieval size at the token level: eight compact claims occupy approximately the same context length as two raw records. This provides an approximate token-level-fair comparison of evidence volume, rather than equating the number of claims with the number of raw records; actual lengths vary by retrieved item. Peak text and its citation snapshot form one memory document. No reranking across the two channels is performed. Query embeddings are computed per request; corpus embeddings and lexical indexes are persisted. Index identity includes source, tokenizer, chunk configuration, and embedding model.

The computation tool runs in a networkless sandbox with the corpus mounted read-only at `/memory` and writable scratch space. It supports matching, filtering, enumeration, sorting, deduplication, aggregation, and corpus-grounded calculations. Its output shares the current-result cap. Retaining exact records in L^1 can help check numbers and qualifiers lost in notes, but the ten-record FIFO policy can also evict relevant evidence.

A.3 HARDWARE

The fine-tuned Search-R1 and GrepSeek checkpoints were served locally on four RTX 3090 GPUs, with sampling temperature 0.3. BGE-M3 embeddings were served in FP16 on one RTX 3090.

A.4 HET IMPLEMENTATION AND PROVENANCE DETAILS

Section 4.2 defines the construction rule and peak view. Here we specify the remaining implementation and auditing details.

Identifier matching and citation validation. The materializer resolves each cited ref against the current corpus before storing it as support. Stable item identifiers match a $claim$ to its preceding version. Version numbers start at one for a new lineage and increase for subsequent revisions. The implementation maintains separate previous- and current-snapshot item maps, replacing the former after each snapshot.

Store isolation and resume. Stores are isolated by episode and workload. A completed query is committed before the next query on the same corpus is served. Resuming a workload must preserve both the ordered query stream and its existing causal prefix. If a store is constructed offline from all saved trajectories, retrieval must still filter each query's candidates by corpus and strictly earlier trajectory order. All nine HET workloads passed causal audits, with no reported source-isolation or future-visibility violation. These audits check access boundaries, not claim correctness or beneficial use.

B BENCHMARK CONSTRUCTION DETAILS

B.1 BENCHMARK SOURCES AND TRANSFORMATIONS

The general suite uses locally normalized samples from AMA-Bench (Zhao et al., 2026), BEAM (Tavakoli et al., 2026), CL-bench (Dou et al., 2026b), CL-bench Life (Dou et al., 2026a), HotpotQA-Shared (Yang et al., 2018), LoCoMo-Refined (Maharana et al., 2024), LongMemEval (Wu et al., 2025), and the Genies Memora benchmark (Uddin et al., 2026). The CL-family tasks retain their rubric-based scoring distinction. The model settings used in these local comparisons differ from those of the official leaderboard. (Agent Memory Leaderboard). The general-suite selection uses seed 42 and fixes 300 questions per benchmark for all systems. For non-CL sources, the sampler prioritizes larger shared episodes, randomizes ties and within-category question order, and cycles through categories; the CL family samples by cycling through categories after shuffling within each category. HotpotQA-Shared independently shuffles eligible native questions, selects 300, and pools their complete context paragraphs, retaining distractors and deduplicating identical paragraphs.

B.2 GENERAL-SUITE WORKLOAD STATISTICS

Table 4 summarizes the corpora associated with the 300 selected queries per benchmark. A corpus is the full raw history available to one episode, and query counts refer to the selected evaluation workload. Corpus length is measured with `cl100k_base` over the stored record text and separators, excluding query text, answers, and agent prompts. Lengths are averaged equally across corpora; queries per corpus equals the number of selected queries divided by the number of corpora.

The suite contains 953 corpora. LoCoMo's two corpora serve 121 and 179 queries; within every other benchmark, each corpus serves the same number of selected queries. CL, CL-Life, and LongMemEval have one query per corpus and therefore provide no cross-query reuse opportunity under corpus-isolated memory. The remaining benchmarks support multiple queries per corpus. Shared corpora provide opportunities for reuse, but the amount of overlapping evidence depends on the questions. Corpus length, query count, and task content vary together across benchmarks, so these statistics contextualize the system comparisons rather than isolating their individual effects.

Table 4: General-suite workload statistics. Each benchmark contributes 300 queries. Length is measured in thousands of tokens; averages weight each corpus equally.

Benchmark	Corpora	Avg. tokens/corpus (k)	Avg. queries/corpus
AMA	25	53.16	12
BEAM	15	627.56	20
CL	300	16.12	1
CL-Life	300	18.75	1
Hotpot	1	398.11	300
LoCoMo	2	20.95	150
LME	300	104.11	1
Memora	10	775.35	30

B.3 MuSiQue A-set and B-set Construction

QuALITY motivates our use of multiple questions over shared passages (Pang et al., 2022), but its QA pairs do not establish necessary overlap in the evidence needed to answer those questions. We therefore use MuSiQue (Trivedi et al., 2022), whose question-level supporting-paragraph annotations allow us to construct pairs with equal or nested evidence sets. MuSiQue composes connected single-hop questions into two- to four-hop questions (Trivedi et al., 2022). We use native MuSiQue-Ans v1.0 training questions, preserving their question text, answers, and reasoning decompositions. Canonical paragraph identifiers allow supporting-evidence sets to be compared across questions. A frozen selection assigns disjoint questions to A-set and B-set and gives each B-set question a designated A-set anchor with equal or strictly nested support. The corpus pools and deduplicates the selected contexts into 3,971 paragraphs (528,772 tokens under `cl100k_base`, including paragraph identifiers and separators), retaining every selected question's gold support. Corpus retrieval is shared rather than restricted to each question's original context.

Filtering for unresolved information needs. The starting pool contains 150 A-set and 234 B-set questions. A preliminary screen retrieves ten raw records using BM25/BGE-M3 reciprocal-rank fusion and records the first summary action. We exclude the 85 questions that immediately emit an answer (45 A-set and 40 B-set), together with 64 B-set questions whose designated A-set anchor was excluded; 15 B-set questions belong to both sets. The union removes 134 questions, leaving 105 A-set and 145 B-set. This selects questions that requested further search in the screening run; it neither guarantees that later stateless runs will search nor tests the correctness of the screen's direct answers. The resulting ablation targets reuse when initial raw retrieval did not immediately resolve the information need.

Table 5: Statistics of the retained MuSiQue ablation. Hop counts follow the native decompositions. B-set relation counts are 45 equal, 47 A-larger, and 53 B-larger.

Split	2-hop	3-hop	4-hop	Total
A-set	23	42	40	105
B-set	32	59	54	145
Total	55	101	94	250

The relation is defined relative to the designated anchor, not the union of all A-set evidence. In particular, three retained B-larger questions have all their gold evidence covered by the union of A-set supports. Set containment also need not correspond to a literal prefix or exactly one missing hop. Gold support identifies possible reuse; it does not imply that the agent successfully encoded that support in HET. The input questions contain no answers, decompositions, anchor assignments, or relation labels.

C Configuration and Cost Accounting

C.1 Baseline Configuration

We evaluate the five MaS baselines and five MaA configurations in Table 1 on the same 300 questions per benchmark. The settings below describe the evaluated configurations.

Shared settings and input units. All LLM calls use temperature 0.3, including memory construction, retrieval planning, reflection, answer generation, and judging. Shared-backbone configurations use `deepseek-v4-flash`; the native Search-R1 and GrepSeek configurations use their released fine-tuned checkpoints. Dense retrieval uses BGE-M3. Each memory store and raw corpus is isolated by episode. For conversational benchmarks, input segmentation and truncation operate at message boundaries, with an individual message as the basic unit. We retain complete messages when enforcing an input budget. The 6,000-character chunk setting for document inputs therefore does not define the truncation unit for conversational histories. SimpleMem's conversational windows are likewise counted in messages.

MaS construction and retrieval. MaS baselines construct their memory stores before answering the query stream. Table 6 summarizes their construction and retrieval settings. Retrieved memories are passed to the shared answer model. For Mem0, LightMem, and A-MEM, the answer context is capped at 20 retrieved items and 5,000 tokens. Document inputs use 6,000-character chunks where indicated; conversational inputs follow the message-based policy above.

Table 6: MaS baseline configurations. All LLM temperatures are 0.3. Character-based input chunk sizes apply to document inputs; conversational histories use message boundaries.

System	Construction	Retrieval
Mem0	Fact extraction; document chunks of 6,000 characters, batched up to 24 items or 6,000 characters.	Score threshold 0.1; up to 20 items within 5,000 tokens.
LightMem	LLMLingua-2 pre-compression, topic segmentation, memory extraction, and offline updates; 6,000-character document chunks.	Embedding search; up to 20 items within 5,000 tokens.
SimpleMem	Sequential extraction over windows of 40 messages with an overlap of 2; thinking disabled.	Hybrid search with semantic, keyword, and structured retrieval depths of 25, 5, and 5; planning and up to 2 reflection rounds.
A-MEM	Note analysis and evolution every 100 notes over 5 neighbors; 6,000-character document chunks; persistent Chroma storage.	Search depth 20; up to 20 items within 5,000 tokens.
HippoRAG	OpenIE over each episode with a 2,048-token generation cap; synonym edges and a graph for personalized PageRank; 6,000-character document chunks.	Linking depth 5, retrieval depth 200, and 5 passages passed to the answer model.

MaA query-time execution. DCI-lite uses a bash/read ReAct loop over the raw corpus, with a maximum of 30 iterations. MemR3 uses a retrieve–reflect–answer loop, also capped at 30 iterations. Search-R1 with the shared DeepSeek backbone retains a 30-step harness and retrieves the top three passages per search.

The native Search-R1 checkpoint uses four steps, a 500-token output cap, a 4,096-token rolling context budget, retrieval depth three, and a 2,000-character observation cap. Native GrepSeek uses four steps, a 1,024-token output cap, 1,000-token observations, and a 16k context budget. Its long-question adapter retains the first 4,000 and last 16,000 characters of the question. These configurations follow their model cards. This question-text limit is separate from conversational-history truncation, which follows message boundaries. The native checkpoints use smaller iteration budgets because of their context-length limits and inference speed, and because neither provides efficient default compression of accumulated context.

C.2 PER-BENCHMARK TOKEN COSTS

Table 7 reports tokens per query for each benchmark. For a system evaluated on benchmark b, we divide the sum of construction, retrieval, and answer input and output tokens by its 300 selected queries. This includes construction in the workload cost rather than reporting query-serving cost alone. Judging is excluded. Corpus lengths in Table 4 use a common tokenizer to describe workload size, whereas execution costs use the token counters recorded for the evaluated system. HET materialization requires no additional LLM call. Since all benchmarks contain 300 queries, the overall tokens per query is the unweighted mean of the eight per-benchmark costs, computed before rounding.

Table 7: General-suite tokens per query (k = 1,000), including construction, retrieval, and answering. Abbreviations follow Table 1.

Method	AMA	BEAM	CL	CL-Life	Hotpot	LoCoMo	LME	Memora
HippoRAG	19.41	82.91	241.47	318.91	16.49	8.94	942.87	123.33
Mem0	40.19	269.15	211.65	265.62	14.57	4.31	1563.80	280.49
A-MEM	66.54	295.47	145.80	224.68	27.51	2.72	1308.17	299.99
SimpleMem	39.78	155.22	129.48	157.35	17.56	11.44	512.37	177.89
LightMem	27.85	172.36	33.82	76.12	6.85	1.32	601.43	220.29
Search-R1 (native)	3.65	4.50	6.26	6.70	2.26	5.01	5.03	5.33
Search-R1 (Flash)	24.18	2.58	48.94	38.69	4.87	14.41	4.65	2.03
GrepSeek (native)	10.69	23.59	16.57	14.30	5.69	11.82	14.36	10.95
MemR3	26.31	17.76	33.20	60.34	6.20	13.97	18.57	41.12
DCI-lite	191.28	481.52	188.83	346.41	32.36	59.26	227.23	605.39
MemSeeker (HET)	33.26	52.44	49.71	83.47	8.54	17.25	35.21	72.35

The cost breakdown distinguishes two sources of workload variation. LongMemEval provides no query sharing to amortize construction: the five MaS methods use 512.37k–1563.80k tokens per query, compared with 35.21k for MemSeeker. LoCoMo combines smaller corpora with many queries per corpus, and all five MaS methods use fewer tokens per query than MemSeeker. BEAM and Memora contain much longer corpora, and their multiple queries do not eliminate the observed MaS cost gap. These comparisons describe the evaluated workloads; Section 5.3 uses controlled ablations to examine query-local context and cross-query reuse separately.

C.3 JUDGE PROMPT PROVENANCE AND SCORING

Table 8 records the source of each general-suite judge. Five benchmarks use the public Agent Memory Leaderboard (AML) evaluators (Agent Memory Leaderboard): LoCoMo-Refined and LongMemEval share a binary inclusion and non-contradiction template; BEAM scores each rubric nugget at 0, 0.5, or 1 and averages within a question; the CL family preserves per-requirement decisions and an all-or-nothing task score, while the main comparison uses requirement-level rubric-micro scores. AMA-Bench keeps its official yes/no judge, and Memora keeps the official FAMA evaluation-question checks, averaging their yes/no results. HotpotQA-Shared uses a semantic-answer template written for this work, because the official HotpotQA metric is normalized exact match and F1 rather than an LLM judge; those token-overlap metrics remain diagnostic. Appendix E.3 reproduces the HotpotQA-Shared template. Dispatch and aggregation are in `bench/scoring.py`. This is a local evaluation protocol, not an official AML leaderboard submission.

C.4 HISTORY CONCATENATION ON GENERAL-160

History Concat sets `context_mode` to `append_only_history` in the format of:

```
user: # Controller-owned speculative retrieval
      <raw records; not a tool call>
user: search
      <original question, phase, round>
assistant: search tool call
```

Table 8: Judge-prompt provenance for the eight general-suite benchmarks.

Benchmark	Judge prompt source
LoCoMo-Refined	Agent Memory Leaderboard
LongMemEval	Agent Memory Leaderboard
BEAM	Agent Memory Leaderboard
CL-bench	Agent Memory Leaderboard
CL-bench Life	Agent Memory Leaderboard
AMA-Bench	Native benchmark
Memora	Native benchmark
HotpotQA-Shared	This work (Appendix E.3)

```
tool: <raw payload; kept in full across rounds>
assistant: bash tool call
tool: <raw payload; kept in full across rounds>
...
assistant: answer
tool: <tool result>
---
next round: resubmit this full list, then append the new turn
omitted: failed attempts that were not submitted
kept: messages from a successful retry
```

C.5 MuSiQue Runtime and Memory Visibility

All conditions use DeepSeek-v4-Flash with a ten-round budget and a 10,000-token combined tool-result cap. HET-enabled speculative retrieval presents eight claim peaks followed by eight raw records; stateless speculative retrieval presents ten raw records. The shared corpus and retrieval index remain fixed across conditions. The corpus contains 3,971 paragraphs and 528,772 tokens under `cl100k_base` (OpenAI), including paragraph identifiers and separators. A-warm processes all 105 A-set questions serially. A-seen and B-unseen start from independent copies of its completed HET and continue writing within their own branch, with no cross-branch access. Stateless runs expose no persistent HET.

C.6 MuSiQue Results and Attribution

Accuracy is measured with the public AML binary answer-correctness prompt used for LoCoMo-Refined and LongMemEval (Agent Memory Leaderboard), evaluated by `deepseek-v4-pro` at temperature 0.3. The judge receives the question, the MuSiQue gold answer, and the prediction, and returns `CORRECT` or `WRONG`. The same prompt and judge settings are used across all conditions. Table 9 reports paired B-set results. A direct answer finishes immediately after speculative retrieval, with no additional search round. Additional rounds count distinct non-speculative retrieval iterations and are averaged over every question in the group. Agent tokens are each trajectory's total input-plus-output tokens, excluding judging.

Table 9: Paired B-set results. Each paired cell shows B-unseen / B-stateless. Tokens are rounded thousands per question.

Relation (n)	Accuracy (%)	Direct (%)	Rounds	Tokens (k)
Equal (45)	80.00 / 66.67	48.89 / 13.33	0.87 / 1.89	14.0 / 25.7
A-larger (47)	76.60 / 72.34	55.32 / 27.66	0.94 / 1.43	14.9 / 21.0
B-larger (53)	77.36 / 73.58	45.28 / 15.09	1.15 / 2.42	17.6 / 30.9

Figure 3(b) in the main text reports claim provenance and HET-only citation adoption for these same groups. Provenance counts gold-citing retrieved claims by source; adoption tracks use in the first summary or answer, excluding citations also available in the simultaneous raw retrieval.

C.7 TOKEN ACCOUNTING

Reported agent cost sums input and output tokens across query-serving calls. HET materialization adds no LLM extraction call. This metric covers query-serving LLM API input and output tokens; embedding, indexing, storage, screening, and judging costs are excluded. We omit wall-clock measurements because API and network variability during the experiments prevents a stable runtime comparison. B-set costs quantify subsequent-query serving after A-warm. They exclude the preceding 105-query warm-up and do not measure total lifecycle savings including warm-up.

D CASE STUDIES

The boxes use an L^2-style display: `iter` is the recorded retrieval iteration, `tool` names the call, and `query` contains the search query or bash script. Results, $claim$ and L^0 excerpts, reasoning, and final-answer calls are supplementary trajectory information; they are not additional fields persisted in L^2. Lines beginning with `//` are editorial annotations. Search requests and results are capped at 480 characters, $claim$/L^0 and reasoning excerpts at 360, and answer excerpts at 480; `[... truncated]` marks a cut and `[... excerpt]` marks an omitted prefix. Only whitespace is normalized. The L^1-ablation cases show selected calls in chronological order and task-relevant L^0 excerpts from the expanded case report. Initial speculative retrieval uses the original benchmark question. Iteration IDs and later calls follow the report's trajectory-step tables; final outcomes are summaries rather than verbatim answer quotations. These paired single-run examples illustrate representative behavior rather than establish causality. These display limits do not change the agent's runtime L^2 policy.

Benchmark: BEAM
Query: How did you explain the connection between my heart's pumping efficiency and the symptoms I might experience during physical activities, and what ongoing steps did you recommend to manage these effects?
Experiment: General-160: Full MemSeeker vs. w/o L^1

Agent: MemSeeker default

```
// Speculative retrieval: original
// question.

iter: 1 | tool: search
query: How did you explain the
  connection between my heart's
    pumping
  efficiency and the symptoms I might
  experience during physical
  activities, and what ongoing steps
  did you recommend to manage these
  effects?
result: R2.1, R2.29, R7.8, R7.40,
  R4.19, R9.42

// R2.1 and R7.8 enter L0; raw text
// stays in L1.

iter: 2 | tool: search
query: reduced pumping efficiency
  ejection fraction symptoms during
  physical activity shortness of
    breath
  fatigue

// Search finds R2.7: fatigue,
// shortness of breath, and gradual
// exercise. Reported answer covers
// symptoms and management advice.
```

Agent: MemSeeker w/o L^1

```
// Speculative retrieval: original
// question.

iter: 1 | tool: search
query: How did you explain the
  connection between my heart's
    pumping
  efficiency and the symptoms I might
  experience during physical
  activities, and what ongoing steps
  did you recommend to manage these
  effects?
result: R2.1, R2.29, R7.8, R7.40,
  R4.19, R9.42

// R2.1 and R7.8 enter L0; no L1 cache
    .
// L0 excerpt:

l0: stable EF at 58% means the heart
  pumps a consistent volume of blood
  with each beat, slightly below the
  typical 55-70% range

// 58% is inside 55-70%; the note
// merges source qualifications.

iter: 2 | tool: bash
query: grep -nE '^\[R2\.1\]|^\[R7
    \.8\]'
  corpus.txt

// Rereads records already cited in
// Known.

iter: 2 | tool: bash
query: grep -niE 'ejection
  fraction|pumping|efficien|exercise|
    physical
  activit|exertion|symptom' corpus.txt

// Broad output is truncated before
    the
// missing symptom evidence. Reported
// answer omits symptoms and
    management
// details.
```

Benchmark: Memora
Query: Can you recommend a book by authors I like?
Experiment: General-160: Full MemSeeker vs. w/o L^1

Agent: MemSeeker default

```
// Speculative retrieval: original
// question.

iter: 1 | tool: search
query: Can you recommend a book by
  authors I like?
result: R1152.1, R1842.1, R29.1,
  R224.1, R1870.1, R271.1, R577.1,
  R1805.1

// Six references enter L0; raw text
// stays in L1. Selected calls in time
// order:

iter: 2 | tool: bash
query: grep -in -E "junger|perfect
  storm|death in
  belmont|freedom|sebastian|tribe|war"
  corpus.txt | grep -i -E
  "book|read|author|recommend|title"

iter: 3 | tool: bash
query: grep -in -E
  "recommend|suggest|should read|you
  might like|you'd like|enjoyed
  reading|read .*book" corpus.txt |
  head -50

iter: 4 | tool: bash
query: grep -n "R1842.1" corpus.txt

// Reported answer: recommends The
// Perfect Storm or War. Total: 4
// rounds, 1 search, 6 bash calls.
// Selected calls shown.
```

Agent: MemSeeker w/o L^1

```
// Speculative retrieval: original
// question.

iter: 1 | tool: search
query: Can you recommend a book by
  authors I like?
result: R1152.1, R1842.1, R29.1,
  R224.1, R1870.1, R271.1, R577.1,
  R1805.1

// Six references enter L0; no L1
// cache. Selected calls in time order
    :

iter: 6 | tool: bash
query: grep -n "Junger" corpus.txt |
  head -50
result: R1476.1, R1546.1, R1842.1

iter: 6 | tool: bash
query: grep -inE "perfect storm|A
    Death
  in Belmont|\bWar\b|Freedom"
  corpus.txt | head -50

// Summary 6 records the absence of
// other titles. L0 excerpts present
    by
// round 7:

l0: Sebastian Junger is the clearest
  currently preferred author. Only
  three corpus records mention Junger
  by name. No Junger book title other
  than "Tribe" appears in those
  records.

iter: 7 | tool: bash
query: grep -n "Junger" corpus.txt
result: R1476.1, R1546.1, R1842.1

iter: 9 | tool: bash
query: grep -n "Junger" corpus.txt
result: R1476.1, R1546.1, R1842.1

// Rereads the same author records,
// then declines to recommend a book.
// Total: 9 rounds, 10 searches, 10
// bash calls. Selected calls shown.
```

D.1 MuSiQue: Direct Reuse and Search Guidance

The following cases summarize paired B-unseen and B-stateless trajectories. Quoted *claim* and query strings are verbatim excerpts from the saved analysis; surrounding descriptions summarize the traces. A-set-source and earlier-B-set-source labels refer to the recorded *claim* provenance. They identify where the retrieved information came from, rather than inferring reuse from the working note alone.

Direct reuse of a subchain (A-larger). Question `2hop__470630_791590` asks, "What country does Rule No. 1's director hold a citizenship?" Two claims from the same A-warm trajectory supply the director and nationality. B-unseen answers Singapore after speculative retrieval, with zero additional search rounds and 3,637 tokens. B-stateless uses three additional rounds and 26,837

tokens, then declines to infer citizenship from "Singaporean director." The recorded judge marks B-unseen correct and B-stateless incorrect.

B-unseen: A-set-sourced claims

```
Rule No. 1 (Rule #1) is a 2008
    Hong Kong horror film directed
    by Kelvin Tong (R1975.1).

Kelvin Tong is a Singaporean
    director; It's a Great, Great
    World is a Singapore film
    directed by him, indicating
    Singapore as his birthplace
    (R65.1).
```

Paired outcome (summary)

```
B-unseen
Answer: Singapore (correct)
Additional search rounds: 0
Agent tokens: 3,637

B-stateless
Declines to infer citizenship
Judge: incorrect
Additional search rounds: 3
Agent tokens: 26,837
```

The retained nationality claim suffices for the judged answer; its additional birthplace inference is not needed and illustrates why persisted claims still require source scrutiny.

Whole-chain reuse with confirmation (equal). For `3hop2__8509_159915_19700`, the question links the people who triggered the Slavic migration to a football tournament's host country and asks for their population there. A-set-sourced claims identify Germanic tribes, list candidate tournament hosts including Brazil, and give five million people of German ancestry in Brazil. B-unseen confirms the referent in one additional round and answers *5 million*, using 18,058 tokens. B-stateless spends five additional rounds and 70,261 tokens but remains uncertain which tournament is intended. This trace shows reuse of previously assembled facts and disambiguation, with a final confirmation search.

Search guidance (B-larger). For `4hop2__53235_66475_57802_311656`, retrieved claims identify Boris Yeltsin as Russia's leader after the Soviet collapse. The A-set-sourced claim cites `R2830.2`, and an earlier B-set claim reinforces the identification with `R487.1`; neither citation is in the initial raw retrieval packet. The first working note then makes Yeltsin's place of death the remaining gap. B-unseen searches for that relation and returns Moscow (31,484 tokens). B-stateless selects Lenin instead, searches for Lenin's place of death, and ultimately declines to answer (49,181 tokens). This case illustrates how prior knowledge reduces the candidate entities explored; both A-set and earlier B-set contribute to that knowledge.

B-unseen: *claim* and next query

```
After the dissolution of the
    Soviet Union, Russian
    President Boris Yeltsin took
    over from Gorbachev (R2830.2).

query: Boris Yeltsin death place
    where died
```

B-stateless: working-note excerpt

```
Europe's largest country by square
    miles is Russia; after the
    Russian Empire's collapse the
    Soviet Union arose, whose
    leader was Vladimir Lenin
    (R789.1, R1003.1).

query: Vladimir Lenin place of
    death died where
```

Failure: wrong geographic scope. For `4hop3__656765_419859_14750_86234` (B-larger), earlier B-set claims associate Philipsburg with Saint Martin. The question asks when the French arrived in the relevant region. B-unseen focuses on Saint Martin and its French/Dutch partition, answering 1648 from the Treaty of Concordia; the reference instead concerns French arrival in the broader Caribbean in 1625. B-stateless answers 1625 correctly. The retrieved location is relevant, but the subsequent search fixes its granularity too narrowly. Prior knowledge can therefore guide retrieval toward an incorrect interpretation as well as reduce unnecessary search.

E FULL PROMPT TEMPLATES

The following blocks reproduce the MemSeeker agent templates and the only scoring judge written for this work. Braced fields denote runtime inputs, not omitted instructions. Agent templates use the general-suite budget of 30 retrieval iterations, ten-record L^1 cache, four parallel retrieval calls, and 10,000-token current-result cap. Search and summarize rounds each include their full system and user messages, including their shared policy. Tool schemas are supplied separately through the tool interface. Table 8 lists the source of each benchmark's judge; Appendix E.3 is the project-defined semantic-answer judge.

E.1 MEMSEEKER SEARCH ROUND

System message.

```
# Role
You are a memory retrieval agent. Answer the question from an unchanged
    indexed
corpus. Retrieve evidence, maintain compact task memory, connect evidence
     across
records, and finish with a concise supported answer.

# Immutable corpus format
`corpus.txt` is the sole agent-visible corpus used by all retrieval
    operators.
Every physical line has exactly the form `[R<parent>.<chunk>] <single-
    line text>`.
The citation is literal and stable within this index; original newlines
    inside a
chunk have been flattened to spaces. Search the labeled text directly and
     preserve
the `(Rn.n)` citation when promoting evidence into L0. Do not spend
    retrieval steps
rediscovering the corpus layout. Do not use `wc`, `head`, or citation-
    label
enumeration merely to count records/chunks or map the reference space;
    those
operations do not provide question evidence.

# Prior claim-peak evidence
The controller may add a `Hierarchical Evidence Tree claim peaks` block
    to the
first retrieval result. Each claim peak is a complete claim retained from
     a
causally prior query and includes its current supporting `(Rn.n)`
    references.
Treat it as derived, provenance-grounded evidence rather than a new raw
    source:
preserve its raw-record citations when promoting it into L0, and retrieve
     the raw
records when the claim is ambiguous, stale, conflicting, or needs
    verification.

# Memory definition
The working context has three memory layers.

## L0: durable semantic memory workspace
It is a structured Markdown file with known evidence and gap-bound failed
     attempts:

'''text
# Known
1. short fact or conclusion (Rn.n)
```

```
# Gaps
1. [G1] active | short missing information need
   Failed attempts:
   1. {"tool":"search","request":"focused query","reason":"why it failed
    "}
```

```
Known and gap items use contiguous `1.`, `2.`, `3.` numbering, restarting
     in each
section. Gap IDs such as `G1` are stable identities and every gap has
    controller-owned
status and zero or more nested failed attempts.

- `# Known`: concise facts, conclusions, intermediate deductions, and
    constraints
  supported by retrieved evidence. Corpus-derived items cite their
    supporting
  child records as `(Rn.n)`.
- `# Gaps`: specific missing information. `active` gaps remain valid
    retrieval aims;
  `exhausted` gaps are durable negative memory and cannot be targeted
    again.
- Nested failed attempts are added by the controller only after a
    successful tool
  execution returns no evidence promoted into known. Tool errors are
    never attempts.
- A gap becomes exhausted after three distinct failed attempts spanning
    both search
  and bash, or after an unproductive same-tool request overlaps a prior
    failed
  request for that gap by at least 80%.

L0 is model-edited only in summary phase; gap status and failed-attempt
    children are
controller-owned.

## L1: citation-derived raw evidence
L1 contains the raw corpus records cited by the current `known` items. It
     is an
automatic FIFO with a maximum of 10 records, ordered by when each
    citation
was newly introduced into `known`. The controller derives L1 from L0; the
     model does
not edit L1 directly.

## L2: compact retrieval history
L2 contains only committed `search` and `bash` history. It contains no
    reasoning and
no raw tool-return text. Each tool call is one item in this form:

`[iter n] [tool_name] gap_id=<Gk> request=<query-or-script> status=<
    success|failed> returned=<item_count> promoted=<returned_item_ids|
    none>`

For `search`, `request` is the exact query. For `bash`, it is the exact
    script. A
request is shown in full up to 100 tokens; a longer request keeps
its first 100 tokens and carries an explicit truncation marker.
`returned` is the number of result items. `promoted` identifies which
    returned items
were incorporated into L0 by the following summary phase. The controller
    appends L2
after summary phase and persists the three memory layers.

# Search-summary loop
```

```
Execution alternates between two phases for at most 30 iterations.

## Search phase
The model performs one retrieval step using search or bash tool.

## Summary phase
The model receives search phase tool result. The model either calls `answer` when the
answer is supported, or replaces `l0.txt` with `write_file` to commit
the tool return's useful information before the next search phase.

# Problem strategy
Match the current gap to one question type, then follow that type's search recipe.
## One-hop
Definition: the answer is one fact about a named entity, event, or time
(e.g. "what is A", "when A happens").
How to search: bash-filter or grep that entity or time directly. Stop once
the matching record is found; do not expand into related checks.

## Multi-hop
Definition: the question is a chain A -> B -> C. A is known and C is asked,
so B is the missing bridge.
How to search: retrieve only the next missing bridge. Keep evidence that
advances the chain; drop side facts that do not produce the next hop.

## Enumeration
Definition: the question asks for every member, interval, or item inside a
defined scope. (e.g. "need to find A, B ,C, ..., and these items are independent of each other.")
How to search: pin the scope first, then list or filter members with bash.
Track covered vs missing items. Stop if two consecutive searches return no new items.

## Inference
Definition: the answer is a conclusion deduced from premises, not a sentence
that already states it. (e.g. "base on A, what can be concluded")
How to search: retrieve only a missing premise that could change the
conclusion. Do not search for the deduction itself for checking.

# Current controller state
phase: search
retrieval_iteration: {iteration}/30
## Search phase state and action rules
Select one or more concrete `active` gaps. Every search or bash call must set
`gap_id` to the stable ID of the gap it aims to resolve. Exclude exhausted gaps and
requests already represented by that gap's failed attempts or L2 history.

- Produce one retrieval step containing one to 4 parallel tool calls.
- Parallel calls pursue distinct information needs or materially different evidence
  sources. Calls may share a gap_id only when their evidence paths differ.
- Use `search` for semantic retrieval when the corpus wording or location is unknown.
- Use `bash` for exact or regex matching, structured filtering, enumeration,
  aggregation, and corpus-grounded calculation.
```

```
- `bash` already starts in `/memory`. Access the corpus directly as `
    corpus.txt`
  (or `/memory/corpus.txt`); never run `cd /memory`.

The exposed native tool schemas are the authoritative action set for this
     phase.
```

User message.

```
# Question
{question}

# L0 workspace file: l0.txt
{l0}

# L1: citation-derived raw evidence (FIFO, max 10)
{l1}

# L2: compact retrieval history
{l2}
```

E.2 MEMSEEKER SUMMARIZE ROUND

System message.

```
# Role
You are a memory retrieval agent. Answer the question from an unchanged
    indexed
corpus. Retrieve evidence, maintain compact task memory, connect evidence
     across
records, and finish with a concise supported answer.

# Immutable corpus format
`corpus.txt` is the sole agent-visible corpus used by all retrieval
    operators.
Every physical line has exactly the form `[R<parent>.<chunk>] <single-
    line text>`.
The citation is literal and stable within this index; original newlines
    inside a
chunk have been flattened to spaces. Search the labeled text directly and
     preserve
the `(Rn.n)` citation when promoting evidence into L0. Do not spend
    retrieval steps
rediscovering the corpus layout. Do not use `wc`, `head`, or citation-
    label
enumeration merely to count records/chunks or map the reference space;
    those
operations do not provide question evidence.

# Prior claim-peak evidence
The controller may add a `Hierarchical Evidence Tree claim peaks` block
    to the
first retrieval result. Each claim peak is a complete claim retained from
     a
causally prior query and includes its current supporting `(Rn.n)`
    references.
Treat it as derived, provenance-grounded evidence rather than a new raw
    source:
preserve its raw-record citations when promoting it into L0, and retrieve
     the raw
records when the claim is ambiguous, stale, conflicting, or needs
    verification.
```

```
# Memory definition
The working context has three memory layers.

## L0: durable semantic memory workspace
It is a structured Markdown file with known evidence and gap-bound failed
    attempts:

'''text
# Known
1. short fact or conclusion (Rn.n)

# Gaps
1. [G1] active | short missing information need
   Failed attempts:
   1. {"tool":"search","request":"focused query","reason":"why it failed
    "}
'''

Known and gap items use contiguous `1.`, `2.`, `3.` numbering, restarting
    in each
section. Gap IDs such as `G1` are stable identities and every gap has
    controller-owned
status and zero or more nested failed attempts.

- `# Known`: concise facts, conclusions, intermediate deductions, and
    constraints
  supported by retrieved evidence. Corpus-derived items cite their
    supporting
  child records as `(Rn.n)`.
- `# Gaps`: specific missing information. `active` gaps remain valid
    retrieval aims;
  `exhausted` gaps are durable negative memory and cannot be targeted
    again.
- Nested failed attempts are added by the controller only after a
    successful tool
  execution returns no evidence promoted into known. Tool errors are
    never attempts.
- A gap becomes exhausted after three distinct failed attempts spanning
    both search
  and bash, or after an unproductive same-tool request overlaps a prior
    failed
  request for that gap by at least 80%.

L0 is model-edited only in summary phase; gap status and failed-attempt
    children are
controller-owned.

## L1: citation-derived raw evidence
L1 contains the raw corpus records cited by the current `known` items. It
    is an
automatic FIFO with a maximum of 10 records, ordered by when each
    citation
was newly introduced into `known`. The controller derives L1 from L0; the
    model does
not edit L1 directly.

## L2: compact retrieval history
L2 contains only committed `search` and `bash` history. It contains no
    reasoning and
no raw tool-return text. Each tool call is one item in this form:

`[iter n] [tool_name] gap_id=<Gk> request=<query-or-script> status=<
    success|failed> returned=<item_count> promoted=<returned_item_ids|
    none>`
```

```
For `search`, `request` is the exact query. For `bash`, it is the exact
    script. A
request is shown in full up to 100 tokens; a longer request keeps
its first 100 tokens and carries an explicit truncation marker.
`returned` is the number of result items. `promoted` identifies which
    returned items
were incorporated into L0 by the following summary phase. The controller
    appends L2
after summary phase and persists the three memory layers.

# Search-summary loop
Execution alternates between two phases for at most 30 iterations.

## Search phase
The model performs one retrieval step using search or bash tool.

## Summary phase
The model receives search phase tool result. The model either calls `
    answer` when the
answer is supported, or replaces `10.txt` with `write_file` to commit
the tool return's useful information before the next search phase.

# Problem strategy
Match the current gap to one question type, then follow that type's
    search recipe.
## One-hop
Definition: the answer is one fact about a named entity, event, or time
(e.g. "what is A", "when A happens").
How to search: bash-filter or grep that entity or time directly. Stop
    once
the matching record is found; do not expand into related checks.

## Multi-hop
Definition: the question is a chain A -> B -> C. A is known and C is
    asked,
so B is the missing bridge.
How to search: retrieve only the next missing bridge. Keep evidence that
advances the chain; drop side facts that do not produce the next hop.

## Enumeration
Definition: the question asks for every member, interval, or item inside
    a
defined scope. (e.g. "need to find A, B ,C, ..., and these items are
    independent of each other.")
How to search: pin the scope first, then list or filter members with bash
    .
Track covered vs missing items. Stop if two consecutive searches return
    no new items.

## Inference
Definition: the answer is a conclusion deduced from premises, not a
    sentence
that already states it. (e.g. "base on A, what can be concluded")
How to search: retrieve only a missing premise that could change the
conclusion. Do not search for the deduction itself for checking.

# Current controller state
phase: summary
retrieval_iteration: {iteration}/30
## Summary phase state and action rules
Summarize the current retrieval result into L0 memory.

- Call `answer` when current known context is sufficient to answer the
    user's question.
```

```
- Otherwise edit `l0.txt` and commit every decision-relevant change
    exposed by the
  current result:
  - add or revise `known` with concise supported information and `(Rn.n)`
     citations;
  - remove resolved active gaps, refine their text without changing
    gap_id, or add
    new active gaps with new `G<number>` IDs.
- Preserve existing gap status and nested failed-attempt lines. They are
  controller-owned; do not invent, edit, or delete them. New gaps have no
     attempts.
- Every file-edit call must change `l0.txt`. When no L0 change is needed
    and the
  answer is supported, call `answer`.
- Each `known` item must have at least one `(Rn.n)` citation.
- Multiple citations may be grouped as `(R1.1, R2.1, R3.2)`.
- Cite explicit records only; citation ranges such as `(R1.1-R9.1)` are
    invalid.
- `known` contains only question-relevant facts or conclusions supported
    by record
  contents. Never store corpus/chunk counts, unexamined-reference
    inventories,
  search diagnostics, or tool-call bookkeeping as `known` items.
- Keep each item short and concise.

`l0.txt` has exactly this Markdown format, and every individual item is
    at most
500 characters:

# Known
1. short fact or conclusion with (Rn.n)

# Gaps
1. [G1] active | short missing information need
   Failed attempts:
   1. {"tool":"search","request":"focused query","reason":"why it failed
    "}
2. [G2] exhausted | another missing premise

Use `write_file` to replace the complete file with `path="l0.txt"` and `
    content`.

The resulting file must contain `# Known` then `# Gaps`. Known and top-
    level gap items
use contiguous numbering restarting from `1.` in each section. Every
    known item
includes at least one `(Rn.n)` citation. Every gap line uses
`N. [Gk] active|exhausted | text`; nested controller-owned attempts are
    numbered JSON
objects under an explicit `Failed attempts:` label. The committed L0 must
     remain
concise, self-consistent, and sufficient for
planning the next search without raw result text. L1 and L2 are
    controller-derived
and are not file-edited.

The exposed native tool schemas are the authoritative action set for this
     phase.
```

User message.

```
# Question
{question}
```

```
# L0 workspace file: l0.txt
{l0}

# L1: citation-derived raw evidence (FIFO, max 10)
{l1}

# Current retrieval result (ephemeral)
retrieval_iteration: {iteration}
{search_result}
```

E.3 HotpotQA-Shared Judge

Judge message.

```
You are grading a concise multi-hop question-answering response.

Inputs:
- Question: {question}
- Reference answer: {gold_answer}
- Predicted answer: {generated_answer}

Judge whether the predicted answer is semantically equivalent to the
    reference
answer in the context of the question.

Rules:
- Accept aliases, abbreviations, paraphrases, and harmless explanatory
    text.
- For yes/no questions, the polarity must match exactly.
- Named entities, numbers, dates, and comparison direction must be
    factually
  consistent with the reference.
- Extra text is acceptable only when it does not contradict or replace
    the
  reference answer.
- Do not require the prediction to reproduce supporting facts or
    reasoning when
  the requested answer is concise.
- Mark WRONG if the prediction is missing, ambiguous, answers a different
  relation, or contradicts the reference.

Return JSON only:
{"label": "CORRECT"} or {"label": "WRONG"}
```