Context as an Environment: Programmatic Context Management for Long-Horizon Agents

Yin Lin[†]
yin.lin@alibaba-inc.com
Alibaba Group

Elaine Ang[†§]
ra3448@columbia.edu
Columbia University

Erkang Zhu
erkang.zhu@alibaba-inc.com
Alibaba Group

Bolin Ding
bolin.ding@alibaba-inc.com
Alibaba Group

Jingren Zhou
jingren.zhou@alibaba-inc.com
Alibaba Group

ABSTRACT

LLM agents increasingly take on long-running tasks whose history grows far beyond a single model context window. Existing approaches compress earlier interactions or extract selected information into fixed memory representations, committing to what to preserve before future needs are known. We present *Scroll*, a context manager that treats each agent session as an executable *Session Environment*. The environment is backed by an append-only Event Log and a sandboxed, persistent Python kernel. The kernel maintains a typed namespace across model calls, allowing tool outputs, retrieved history, and derived state to be bound to variables rather than serialized into the prompt at each call. Model-written code searches, materializes, and transforms session state through `exec`; only explicitly printed projections enter the model's working view for the next call. Context management thus becomes a programming task that inherits the improving coding abilities of LLMs, while the Event Log preserves lossless historical ground truth. As the working view approaches its budget, stale spans are evicted but remain recoverable: an eviction index keeps compact landmarks tied to exact Event Log addresses, so that the agent navigates directly to evicted regions instead of searching the full log. With Qwen3.8-Max as the backbone, Scroll achieves **94.8%** on LongMemEval$_S$; **73.1%** on BEAM$_{10M}$, surpassing the best published memory system by 5.1 points; and **86.7%** on LOCA$_{256K}$, exceeding the best published long-horizon agent by 37.4 points.

1 Introduction

LLM agents are increasingly used for long-running tasks such as repository-level software engineering [Jimenez et al., 2024, Yang et al., 2024] and deep research over the open web [Zheng et al., 2025, Wei et al., 2025]. Unlike single-turn generation, these tasks unfold over extended trajectories of model calls, tool executions, observations, failures, and revisions. As trajectories grow, a central challenge for the *agent harness* is context management: session history accumulates continuously, while each model invocation operates over a bounded context window. Moreover, the effective context a model can reliably exploit is far smaller than its nominal window, as long-input retrieval and reasoning degrade with input length [Modarressi et al., 2025, Zeng et al., 2026].

[†]Equal contribution.
[§]Work done during an internship at Alibaba Group.

Current systems largely address this problem through *context compression* or *external memory*. Compression is the dominant approach in practice: existing methods truncate stale spans, discard tool outputs, or replace earlier trajectory segments with summaries [Kang et al., 2025, Ye et al., 2025], and production agents such as Claude Code, Codex CLI, and Cursor reportedly employ similar compaction mechanisms as the context window approaches its limit. External memory systems extract selected facts or episodes into a separate store and later retrieve them through semantic or structured interfaces [Packer et al., 2023, Wu et al., 2025, Cao et al., 2026]. Both are fundamentally lossy in the same way: the agent sees history only through the summary or the memory store, so any detail they fail to keep is out of reach—even if the raw log still exists on disk. Long-horizon tasks, however, may require exact historical evidence or nontrivial computation over past events, such as comparing tool outputs produced far apart in the trajectory. Neither the relevant information nor the required operation is known in advance, so no summary produced at observation time can be guaranteed to preserve what is later needed.

We present *Scroll*, which keeps the agent's history outside the model context [Zhang et al., 2025] and represents it as an executable *Session Environment*. An append-only Event Log preserves the interaction trajectory with stable addresses and provenance, while a sandboxed, persistent Python kernel survives across model calls and maintains a typed namespace of resident variables. Any of this state can therefore be materialized as Python objects and reused across reasoning steps without being serialized into the prompt.

This turns context management into *writing programs*—something current models are already highly proficient at. The model issues `exec` actions to search and expand the Event Log, access permitted resources, invoke tools [Anthropic, 2025b], and compute over resident variables. Retrieved records, tool outputs, and intermediate computations remain in the kernel unless explicitly emitted through `print`, which the harness inserts as an observation into the next model context. Thus, `exec` determines how the environment is accessed and transformed, while `print` determines which projection enters the model's *working view* over the Session Environment.

As the working view approaches its budget, the harness evicts stale spans. Unlike compaction, eviction changes only the view, never the underlying record: evicted events remain verbatim in the Event Log under their stable addresses, where the model's programs can search for and materialize them on demand. Scroll additionally keeps an *eviction index* in the view: a compact map of what has left it. Where search recovers only what the agent thinks to ask for, the index keeps the agent aware of history it can no longer see; each entry anchors the exact addresses of the evicted events, from which the originals are materialized on demand.

Our contributions are threefold:

- We formulate long-horizon context management as choosing, at each step, a working view over a persistent Session Environment. Existing approaches fix this choice before future needs are known; Scroll defers it to query time as a program the model writes.
- We implement an executable context substrate combining an append-only Event Log, durable storage, and a sandboxed persistent Python kernel. The model operates on the environment through `exec`; within it, only explicit `print` output crosses into the model-visible context.
- We introduce an algorithm that keeps the working view within budget without losing history: evicted spans remain intact in the Event Log, indexed by compact address-anchored entries the agent can navigate directly.

2 Scroll Context Manager

We introduce Scroll, a context manager for long-horizon LLM agents. The key insight is that an agent's accumulated history should not be serialized into the model's prompt but should instead be treated as *an environment that the model programmatically interacts with*. The prompt then carries only a working view,

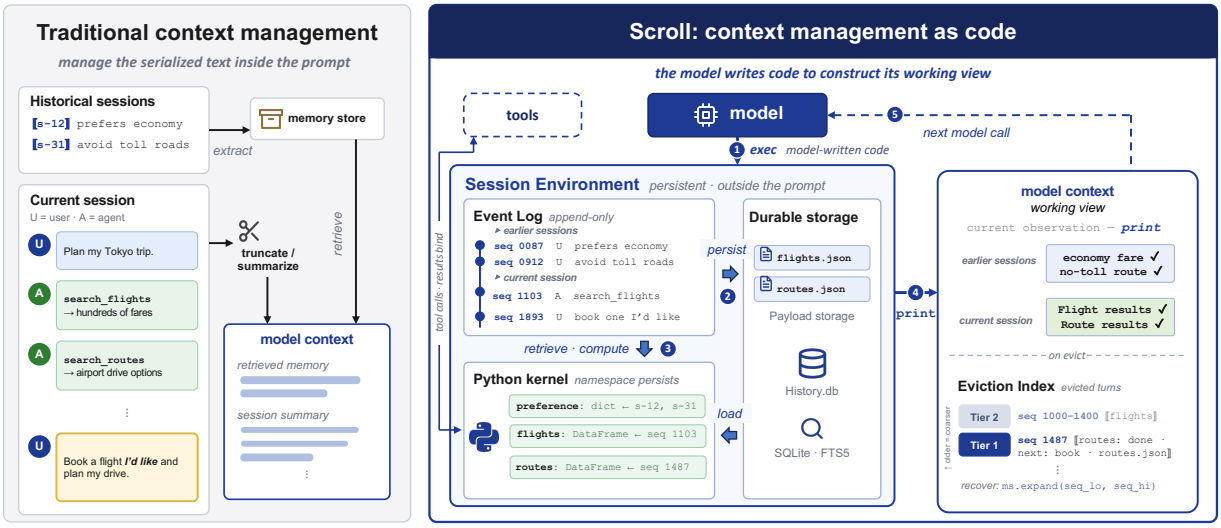

Figure 1: Overview of Scroll. Scroll keeps the full session in a persistent, executable Session Environment; model-written code retrieves and computes over it via `exec`, and `print` selects the working view exposed to the next model call.

while the session lives outside the context window without loss. We first formalize the problem this design addresses (§2.1), then describe the Session Environment (§2.2), the programmatic interface through which the model constructs its own context (§2.3), and the eviction mechanism that keeps the working view bounded while preserving recoverability (§2.4).

2.1 Problem Formulation

Session state and working view. An agent session produces a growing sequence of *events* $e_1, e_2, \ldots$ (user messages, model responses, tool calls, tool results), each with an associated *payload* (its raw content, e.g., a full tool output). We write the session state after t agent steps as

$$S_t = (L_t,\ P_t,\ V_t), \tag{1}$$

where L_t is the event sequence with per-event metadata, P_t the payloads referenced by L_t, and V_t auxiliary derived state (in Scroll, a variable namespace; in other systems, a memory store or summary buffer). Each model call, however, consumes a *working view* c_t with $|c_t| \le C$ tokens, where C is the model's nominal context window. The *context-management problem* is to choose, at every step, the next view: the map $S_t \mapsto c_{t+1}$.

When the selection is made. Existing approaches fix the map $S_t \mapsto c_{t+1}$ before future needs are known: compression applies a lossy operator ϕ as the trajectory grows, $c_{t+1} = \phi(c_t, e_t)$, deciding which information survives when each segment is compacted; external memory applies an extraction operator ψ at ingestion, $V_t = \psi(V_{t-1}, e_t)$, fixing what is stored and how it can later be retrieved. Either way, the reduced representation replaces the history it summarizes, so anything it omits is unrecoverable.

Scroll instead defers selection to query time. The full state S_t persists losslessly outside the context, and the map $S_t \mapsto c_{t+1}$ is a *program* π_t the model writes at step t: the program executes on S_t, updates V_t, and emits a bounded observation for the next call. The model decides what to recall, compute, and expose; the harness makes those decisions safe via durable storage, stable addressing, and sandboxed execution.

Operation	Interface	Role
LOCATE	`ms.search(query, k, ...)`	Find candidate Event Log records via BM25-ranked full-text search and structured scope, kind, and time filters; each hit carries its stable `seq` address.
MATERIALIZE	`ms.expand(seq)` `ms.expand(seq_lo, seq_hi)`	Recover exact turns or sequence spans, and load externalized payload contents behind lazy handles.
COMPUTE	Ordinary Python and permitted database, filesystem, and tool interfaces	Filter, join, aggregate, resolve updates, inspect artifacts, or construct derived state.
EXPOSE	`print(value)`	Return the selected projection as a bounded observation; everything else stays in the kernel.

Table 1: The model-facing interface factorizes context construction into location, materialization, computation, and exposure.

Because the policy is expressed as code, it inherits the full generality of programs and improves with the backbone's coding ability, at no change to the harness.

2.2 Persistent Session Environment

Scroll realizes S_t as a *Session Environment* (Figure 1, right), with three components corresponding to L_t, P_t, and V_t.

Append-only Event Log (L_t). The Event Log is the durable, ground-truth record of an agent's sessions: a single append-only log that spans session boundaries. Every interaction appends a typed event carrying the metadata future queries need—role, session and agent identifiers, timestamps, and tool state—and receives its immutable, monotonically increasing `seq`. Our implementation stores events in SQLite. Search defaults to BM25 rather than embeddings: it is deterministic and requires no index-time model calls.

Durable storage (P_t). A payload is the raw content an interaction produced, such as a full tool result or a generated artifact. The log records that the interaction occurred but need not store every byte of it in the event row: small payloads remain inline in SQLite, while large ones are moved into JSON or artifact storage on the filesystem, with the row retaining a bounded preview and a recovery pointer. Externalized payloads are accessed through lazy handles (`ToolResultRef`, `ArtifactRef`).

Persistent runtime and resident namespace (V_t). A sandboxed Python kernel persists across model calls throughout the session; its namespace holds *environment objects*: resident Python values and lazy handles, each carrying type, size, and provenance metadata identifying the events it derives from. Tool invocations issued through the programmatic tool interface [Anthropic, 2025b] return Python objects that later programs can operate on. The harness prepends to every call a *namespace digest*: a short listing of each resident variable's name, type, and shape, with small scalar values shown inline. Model-authored code runs in a fail-closed sandbox: the Event Log is read-only from the kernel, and database, filesystem, network, and tool access are limited to capabilities the harness explicitly declares.

2.3 Programmatic Context Construction

Scroll uses a CodeAct-style interface [Wang et al., 2024] for both task execution and context construction. A controlled capability object, `ms`, forms the model-facing *memory surface* over durable history, abstracting the physical backend behind four operations (Table 1).

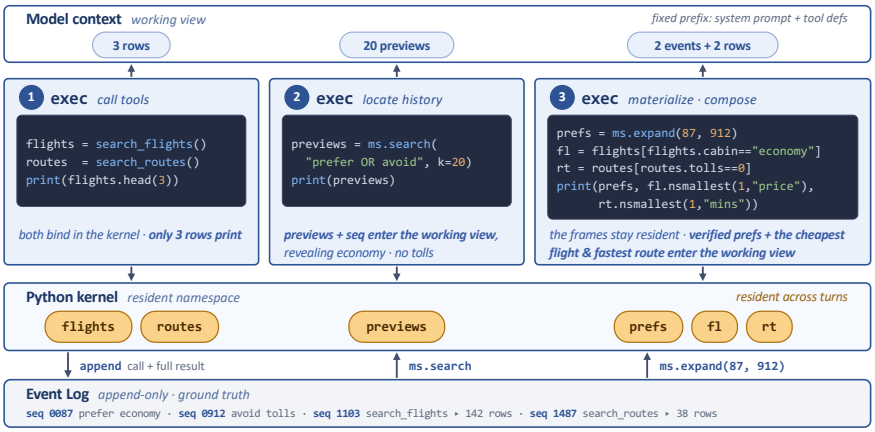

Figure 2: Programmatic context construction. Three `exec` turns compute over resident state in the kernel; only `print` output crosses into the next model context/working view.

Algorithm 1 Recoverable context eviction

Input: working view c, Event Log $\mathcal{L}$, eviction index $\mathcal{I}$, budget ρC, tier width k
Output: bounded c and updated $\mathcal{I}$; removed spans stay recoverable from $\mathcal{L}$

if $|c| > \rho C$ **then**
  $\mathcal{L} \leftarrow \textsc{Persist}(c, \mathcal{L})$ ▷ live turns become durable
  $R \leftarrow \textsc{Protected}(c)$ ▷ active turn, recent tail, newest tool results
  $c \leftarrow R \cup \textsc{FoldPayloads}(c \setminus R)$ ▷ payloads → `seq` pointers
  $E \leftarrow \textsc{SelectSpan}(c \setminus R,\ |c| - \rho C)$ ▷ oldest completed span above budget
  $c,\ \mathcal{I} \leftarrow \textsc{EvictToIndex}(c,\ E,\ \mathcal{H}[E])$ ▷ E leaves the view; its headlines enter $\mathcal{I}$, shown in place
  $\mathcal{I} \leftarrow \textsc{RollUp}(\mathcal{I},\ k)$
end if
return $(c, \mathcal{I})$

Figure 2 traces the trip-planning task of Figure 1 through three `exec` cells. Cell 1 binds full tool results to the resident variables `flights` and `routes` in the Python kernel, printing only a few rows. Cell 2 searches the Event Log for stated preferences; the matching previews and `seq` addresses enter the working view, revealing the user's preference for economy cabins and toll-free routes. Cell 3 expands the two events for the verbatim record, filters and ranks the resident variables accordingly, and prints the two preference turns alongside the cheapest economy flight and the fastest toll-free route. The bulk tool results never enter the working view; every call is appended to the Event Log with its full result, addressable by `seq`.

2.4 Eviction and Off-Context Navigation

The working view must stay bounded as the session grows. Scroll bounds it with an eviction procedure (Algorithm 1) that triggers whenever the working view exceeds a budget ρC. The procedure first persists any live turns to the Event Log and protects the active turn, the recent tail, and the newest tool results. The remainder is evicted in increasing order of recovery cost: completed tool payloads are folded first, since a single `seq` pointer suffices to recover them; whole spans are removed only if the view remains over budget. What leaves the view is not lost: it stays verbatim in the Event Log, and the procedure's one invariant is that everything it removes stays addressable.

Headlines as navigation anchors. Lexical search recovers an evicted span only when the agent recalls its wording; Scroll therefore also maintains *landmarks* for position-based navigation. As part of each response, the model writes a short *headline*—task, verified state, next action, and a status—which Scroll binds at append time to the `seq` assigned by the Event Log, yielding a map $\mathcal{H}$ from address to headline. When a span is evicted, its headlines enter a tiered index (Figure 1). A flat index would grow linearly with the session, so Scroll rolls it up: each tier holds at most k blocks; when a tier fills, the newest block retains full detail while the $k-1$ older ones collapse to one line each and merge into the next tier. After n evictions, the index occupies $O(k \log_k n)$ blocks, providing fine anchors for recent history, coarse ranges for distant history, each backed by a `seq` span.

3 Experimental Setup

3.1 Benchmarks

We evaluate Scroll in two long-horizon settings: (1) retrieving and reasoning over interaction histories that exceed the live context, and (2) reasoning and acting in an agentic environment whose state grows over time.

Long-term memory retrieval and reasoning. `LongMemEval` [Wu et al., 2025] poses questions over a history of prior user–assistant conversations. Answering may require locating evidence scattered across sessions, resolving temporal dependencies and knowledge updates, and reasoning over the retrieved evidence. The benchmark provides three settings with increasing amounts of distractor history: `Oracle`, where the history contains only the evidence sessions; and `S` and `M`, where each question is paired with roughly 50 and 500 sessions ($\sim$115K and $\sim$1.5M tokens) of history, respectively.

`BEAM` [Tavakoli et al., 2025] extends this evaluation to substantially longer coherent histories. Its questions may require collecting non-adjacent evidence, tracking changes over time, deduplicating repeated information, or aggregating facts distributed throughout the history. The benchmark spans four history scales (128K, 500K, 1M, and 10M tokens); at the largest scale, BEAM_{10M}, histories cannot be consumed directly within current model context windows.

Long-context reasoning and acting. `LOCA` [Zeng et al., 2026] evaluates agents that must reason, invoke tools, and modify an environment as the available environment state accumulates. LOCA measures whether an agent can continue to reason and act throughout a growing tool-use trajectory. The benchmark scales the *environment description length* (the token count of the full environment state as seen through tool outputs) across seven regimes from 8K to 256K tokens; we evaluate on the two largest regimes (128K and 256K).

3.2 Agent Configuration

Our main experiments use Qwen3.8-Max as the agent backbone. All context management methods are implemented and evaluated on top of QwenPaw [Agentscope Team, 2026], an agent operating system providing tool invocation and execution infrastructure, and orchestrated with Harbor [Harbor Framework Team, 2026] in the benchmark-provided environments. Our implementation to reproduce all reported results is available at `https://github.com/niceIrene/QwenPaw/tree/scroll-research`.

Scroll exposes its functionality to the agent through a set of tools, of which the following two implement the `exec` action of Section 2.

- `repl_exec` executes a model-generated Python cell in the persistent kernel. Environment tools are exposed as Python functions forwarded to the underlying services, enabling programmatic tool calling;

all intermediate computation stays in the kernel, and only explicit, budgeted `print` output enters the model's working window.

- `recall_history_python` executes a cell with the memory surface `ms` bound: `ms.search` locates evicted records and `ms.expand` materializes them as Python objects, which the model filters, combines, or aggregates in the kernel before printing a distilled result.

We use a single system prompt and one set of context-management rules across all benchmarks, with no few-shot demonstrations. Each memory benchmark contributes only a short rubric specifying its data layout, memory-surface usage, evidence-selection conventions, and answer format (see detailed prompts in Appendix C). For LOCA, we use the benchmark's official task instructions unmodified, adding only environment metadata (available APIs and workspace paths).

To test generality across backbones, we additionally evaluate Qwen3.7-Max, Deepseek-v4-pro, GLM-5.2, Kimi-K2.7, and Qwen3.6-35B-A3B (an open-weight model with a smaller active-parameter footprint), changing only the foundation model.

3.3 Evaluation Protocol

For the two memory benchmarks, we ingest each conversation history into Scroll session by session, in chronological order. At each session boundary, the raw context is cleared, and only Scroll's internal state (the eviction index and the Event Log) is carried forward to subsequent sessions. For LOCA, each task starts from the benchmark-provided initial environment state. The agent explores and acts on the environment directly, with Scroll managing its context as the trajectory grows.

LongMemEval and BEAM are scored with their benchmark-provided LLM-as-a-judge prompts, using Qwen3.6-flash at temperature 0 as the judge; we report accuracy for LongMemEval and the judge score for BEAM. LOCA is scored with its native rule-based verifier, which checks the final environment state, and we report accuracy. Unless otherwise noted, each task is evaluated once in the benchmark-provided container with a random seed; we also record model-facing input and output tokens and the number of interaction turns for each task.

4 Results

4.1 Comparison with Existing Systems

Retrieval accuracy comparison with long-term memory systems (Table 2). Agents do not natively retain information across sessions, so answering questions over prior interactions requires an external memory system. We compare Scroll against dedicated long-term memory systems. For each system, we report the best publicly available result under its own preferred configuration (backbone model, retrieval budget, and judge) as of August 15, 2026.[1] Appendix A provides per-category breakdowns of Scroll on the `S` and `M` splits of LongMemEval and on BEAM_{10M}.

As shown in Table 2, Scroll is competitive with the strongest reported systems on LongMemEval_S and beats the best-performing system by 5.1 points on BEAM_{10M}. Existing memory systems follow a three-stage paradigm: at ingestion, an LLM processes the history into a derived store through fact extraction, summarization, or knowledge-graph construction; at query time, a retrieval pipeline selects candidate memories from that store; and a reader model then reasons over the returned snippets to produce the answer. Scroll instead ingests the raw history as-is, composes retrieval code per question, and needs no separate reader: the agent that writes and executes the queries also produces the final answer directly.

[1] We do not reproduce the baselines ourselves, as independent reproductions in this area have repeatedly led to disagreement over evaluation setup [Zep, 2025, Mem0, 2025].

Table 2: Comparison with existing long-term memory systems. For each baseline, we report the best publicly available result under that system's own setup as of August 15, 2026; "–" denotes no publicly reported result. These are reference points from the literature rather than a controlled comparison: reader models differ across rows and can substantially affect scores—EmergenceMem (GPT-4o), Zep (GPT-5.4), Mastra OM (GPT-5 mini), Mem0 (GPT-5), Hindsight (Gemini 3 Pro), Exabase M-1 (Gemini 3 Flash), RAG and LIGHT (Llama-4-Maverick); Honcho uses a multi-model pipeline and Cognee does not report its reader.

Method	LongMemEval$_S$	BEAM$_{10M}$
RAG [Tavakoli et al., 2025]	–	24.9
LIGHT [Tavakoli et al., 2025]	–	26.6
Zep [Zep, 2026]	90.2	–
Mem0 [Chhikara et al., 2025, Mem0, 2026]	94.4	48.6
Hindsight [Bartholomew, 2026]	94.6	64.1
EmergenceMem [Emergence AI, 2026]	86.0	–
Honcho [Plastic Labs, 2026]	90.4	40.6
Mastra OM [Barnes, 2026]	94.9	–
Cognee [Marković, 2026]	–	67.0
Exabase M-1 [Exabase, 2026]	**96.4**	68.0
Scroll (ours)	94.8	**73.1**

Table 3: LOCA accuracy (%) of different context-management strategies at the two largest environment description lengths. All agent loops use Qwen3.8-Max as the backbone; Δ denotes the absolute drop from 128K to 256K.

Agent loop	128K	256K	Δ
Summarization Agent	86.7	65.3	-21.4
Retrieval Agent	88.0	66.7	-21.3
CodeAct Agent	**89.3**	85.3	-4.0
Scroll (ours)	**89.3**	**86.7**	**-2.6**

Comparison of context-management strategies on LOCA (Table 3). On LOCA, we compare four agents that share the same backbone (Qwen3.8-Max) and toolset, and differ only in how they manage a growing context: (i) a *summarization agent*, a ReAct agent [Yao et al., 2023] that periodically compacts its interaction history into a summary; (ii) a *retrieval agent*, a ReAct agent whose overflowing history is evicted and made accessible through a recall tool; (iii) a *CodeAct agent* [Wang et al., 2024] that interacts with the environment through programmatic tool calling; and (iv) Scroll. A comparison against the best published numbers from the LOCA paper [Zeng et al., 2026] and its leaderboard is in Appendix B.

Table 3 reports accuracy at the two largest environment description lengths. The CodeAct agent and the agent with Scroll, which bind intermediate results to environment objects instead of carrying raw text in context, achieve the best performance and the smallest decrease as context grows.

4.2 Can Different Backbone Models Use Scroll Effectively?

Scroll provides an environment for managing context but does not dictate its use: what state to keep where, and what code to write, are left to the model. A natural question is *whether the ability to use Scroll effectively is specific to one backbone or shared across models of varying capability*. We rerun both regimes across six backbones with the harness, tools, prompts, and context-management rules held fixed (Table 4).

Every backbone can use Scroll, but stronger models benefit more. On LongMemEval$_S$, where the model

Table 4: Scroll across backbones. Only the foundation model changes; harness, tools, prompts, and context-management rules are identical.

Backbone	LongMemEval$_S$	BEAM$_{10M}$	LOCA 128K	LOCA 256K
Qwen3.8-Max	94.8	73.1	89.3	86.7
Qwen3.7-Max	92.8	66.6	78.7	60.0
Deepseek-v4-pro	93.2	70.2	69.3	58.7
GLM-5.2	93.6	70.7	66.7	62.7
Kimi-K2.7	92.0	67.2	30.7	32.0
Qwen3.6-35B-A3B	88.8	58.1	37.3	22.7

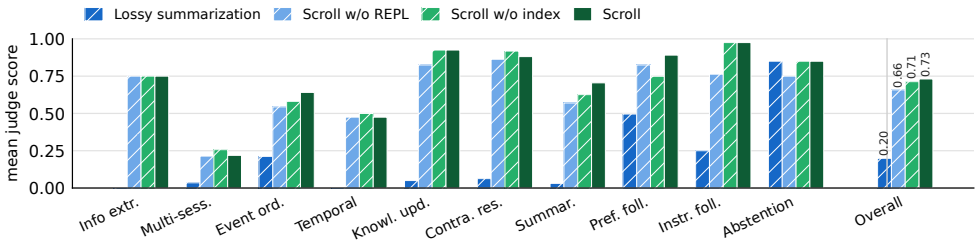

Figure 3: Ablating Scroll's components on BEAM$_{10M}$ (judge scores; hatched bars are Scroll variants; Qwen3.8-Max, thinking on). Lossy summarization: summaries replace the originals at ingestion. Scroll w/o REPL: `ms` exposed as ordinary tool calls, with no persistent kernel. Scroll w/o index: the eviction index is removed, leaving keyword search only. Scroll: the full system.

queries the history database with short programs, all backbones benefit similarly—even the 35B model reaches 88.8, within six points of the best (94.8). On BEAM$_{10M}$ the gap stays within 15 points. On LOCA, however, tasks demand longer trajectories and more complex program synthesis, so the spread widens to 64 points at 256K (86.7 vs. 22.7). Failures are not protocol-level: all backbones adhere to the CodeAct interface, but weaker models commit more execution errors or terminate prematurely on aggregation-heavy tasks. Scroll's ceiling on such tasks thus rises with multi-step query planning and the ability to decide when evidence suffices, suggesting room for post-training on frontier model traces.

4.3 Ablation Study

We ablate the core components of Scroll on BEAM$_{10M}$. Figure 3 reports judge scores per category and overall. First, to assess the utility of the Event Log, we compare Scroll against a lossy variant whose history is summarized at ingestion, with the originals discarded. Second, to evaluate the programmatic interface, we compare against *Scroll w/o REPL*, which exposes `search`, `expand`, and `sql_query` as ordinary tool calls, with no persistent kernel. Third, to assess index-guided navigation, we remove the eviction index, leaving the agent to locate history through keyword search alone rather than index ranges.

Discarding the original records is the most damaging ablation: the lossy variant falls to 19.9 overall, with near-zero scores wherever the answer must preserve exact values from the history, such as information extraction, temporal reasoning, and knowledge update. Scroll w/o REPL underperforms full Scroll by 7.3 points, since serialized tool results cannot be filtered, joined, or aggregated in the kernel; the difference is concentrated in abilities that require composing evidence from many records, such as knowledge update (92.5 vs. 82.5) and instruction following (97.5 vs. 76.3), while single-lookup abilities are unaffected. Removing the eviction index costs 1.8 points overall, but the effect concentrates where evidence is scattered across the history and must otherwise be collected by keyword search: preference following (89.1 vs. 74.9),

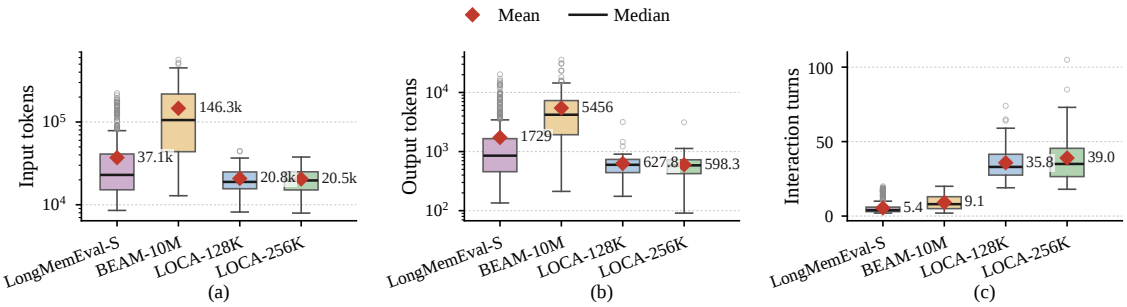

Figure 4: Per-task cost of Scroll (backbone: Qwen3.8-Max): (a) input tokens, (b) output tokens, (c) agent turns. Boxes span the IQR with the median marked; red diamonds are means. Log scale in (a) and (b).

summarization (70.5 vs. 62.6), and event ordering (64.1 vs. 58.1).

4.4 Cost and Efficiency

Scroll exposes only a small fraction of the corpus to the model. Ingestion involves no additional LLM calls, and at query time records are filtered inside the Python kernel, so only printed output enters the context. Figure 4 shows the per-task distribution of input tokens, output tokens, and agent turns: median input on BEAM_{10M} is 105K tokens, about 1% of the corpus, and output is an order of magnitude smaller than input across all three benchmarks. Note that for LongMemEval_S and BEAM_{10M} we measure retrieval alone, whereas for LOCA we measure full task completion, hence its longer trajectories. We report token counts rather than latency or dollar cost, as both depend on serving configuration.

5 Related Work

Context compression and external memory. Most context-management systems either compress the active trajectory or store selected information externally. Compression methods summarize, clear, or fold earlier interactions into shorter representations [Kang et al., 2025, Ye et al., 2025, Zhou et al., 2026, Kontonis et al., 2026]. External-memory systems instead extract facts, episodes, or notes into a separate store and retrieve them when relevant [Packer et al., 2023, Tan et al., 2026, Letta, 2026]. Both approaches reduce the history the model sees by deciding, before future needs are known, which information survives, in what form, and through which interface it can later be reached. Scroll instead retains the original interaction events and referenced payloads; summaries and indexes provide compact working views without becoming the sole representation of historical evidence.

Code as the agent–environment interface. CodeAct introduced executable Python as a general action interface for LLM agents [Wang et al., 2024], while programmatic tool-calling and code-execution systems allow tool results to remain in sandbox variables and enter context only through selected projections [Anthropic, 2025b,a]. Related work has also explored programmatic access to externalized long input prompts [Zhang et al., 2025] and structured working state [Li, 2026, VISTA, 2026]. Scroll applies this principle to the continuously evolving state of an agent session. Its persistent Python kernel retains typed variables across model calls: `exec` retrieves and transforms session state, while only explicit `print` outputs cross the observation boundary.

Lossless session history and navigation. Prior systems have explored verbatim recall storage, event-sourced interaction logs, lossless pointers, and provenance-linked memory [Packer et al., 2023, Nakajima,

2026, Ehrlich and Blackman, 2026, Zhang et al., 2026]. Scroll combines a queryable append-only Event Log with external payload references and an executable resident namespace. Its within-session eviction index is a navigation layer over this retained state: recent history is represented by fine-grained, sequence-addressed headlines, while older history is represented by coarser ranges. Once a relevant region is located, the original events and payloads are recovered programmatically. In the terminology of CoALA [Sumers et al., 2024] and context-engineering surveys [Mei et al., 2025], Scroll connects executable working state with verbatim episodic history through a persistent Session Environment.

6 Conclusion

In this report, we present Scroll, a context manager that makes context management an explicit model policy over a persistent Session Environment: the model uses `exec` to retrieve and compute over externalized state, and uses `print` to decide what enters the next context, while the harness provides deterministic storage, execution, and recovery. This policy can in turn be distilled from frontier models into smaller ones. Successful trajectories supervise two decisions: *context retrieval* (when and how to write retrieval code over the agent history) and *context injection* (which computed results should be printed back into the working window). We plan to use frontier-model traces for supervised fine-tuning or policy distillation, keeping the underlying context mechanisms fixed.

References

Agentscope Team. QwenPaw. https://qwenpaw.agentscope.io/, 2026.

Anthropic. Code execution with MCP: Building more efficient agents. https://www.anthropic.com/engineering/code-execution-with-mcp, 2025a.

Anthropic. Programmatic tool calling. https://platform.claude.com/docs/en/agents-and-tools/tool-use/programmatic-tool-calling, 2025b.

Tyler Barnes. Observational memory: 95% on LongMemEval. https://mastra.ai/research/observational-memory, February 2026.

Ben Bartholomew. Hindsight is #1 on BEAM — the benchmark that tests memory at 10M tokens. https://hindsight.vectorize.io/blog/2026/04/02/beam-sota, 2026.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 16803–16822, 2026.

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

Clint Ehrlich and Theodore Blackman. LCM: Lossless context management. *Voltropy PBC technical report*, 2026.

Emergence AI. SOTA on LongMemEval with RAG. https://www.emergence.ai/blog/sota-on-longmemeval-with-rag, 2026.

Exabase. Exabase reports state-of-the-art results on BEAM memory benchmark. https://www.hpcwire.com/aiwire/2026/07/28/

exabase-reports-state-of-the-art-results-on-beam-memory-benchmark/, July 2026.

Harbor Framework Team. Harbor: A framework for building and running agent evaluations at scale. https://github.com/laude-institute/harbor, 2026.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. Acon: Optimizing context compression for long-horizon llm agents. *arXiv preprint arXiv:2510.00615*, 2025.

Vasilis Kontonis, Yuchen Zeng, Shivam Garg, Lingjiao Chen, Hao Tang, Ziyan Wang, Ahmed Awadallah, Eric Horvitz, John Langford, and Dimitris Papailiopoulos. Memento: Teaching llms to manage their own context. *arXiv preprint arXiv:2604.09852*, 2026.

Letta. Context repositories: Version-controlled memory for agents. Letta Blog, 2026. https://www.letta.com/blog/context-repositories/.

Bojie Li. User as code: Executable memory for personalized agents. *arXiv preprint arXiv:2606.16707*, 2026.

Vasilije Marković. cognee on BEAM: SOTA results without a benchmark-specific memory system. https://www.cognee.ai/blog/deep-dives/benchmarking-cognee-on-beam, 2026.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. A survey of context engineering for large language models, 2025. URL https://arxiv.org/abs/2507.13334.

Mem0. Revisiting Zep's 84% LoCoMo claim: Corrected evaluation & 58.44% accuracy. https://github.com/getzep/zep-papers/issues/5, 2025.

Mem0. Memory evaluation. https://docs.mem0.ai/core-concepts/memory-evaluation, 2026.

MiniMax. MiniMax M3: Frontier coding, 1M context, native multimodality — all in one model. https://www.minimax.io/blog/minimax-m3, 2026.

Ali Modarressi, Hanieh Deilamsalehy, Franck Dernoncourt, Trung Bui, Ryan A. Rossi, Seunghyun Yoon, and Hinrich Schütze. NoLiMa: Long-context evaluation beyond literal matching. In *International Conference on Machine Learning (ICML)*, 2025.

Yohei Nakajima. The log is the agent: Event-sourced reactive graphs for auditable, forkable agentic systems. *arXiv preprint arXiv:2605.21997*, 2026.

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

Plastic Labs. Honcho: Memory infrastructure for stateful agents. https://github.com/plastic-labs/honcho, 2026.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research (TMLR)*, 2024.

Juntao Tan, Liangwei Yang, Wenting Zhao, Jielin Qiu, Ming Zhu, Rithesh Murthy, Silvio Savarese, Huan Wang, Shelby Heinecke, and Caiming Xiong. A lightweight, domain-adaptive memory system for LLM agents. In *International Conference on Learning Representations (ICLR)*, 2026.

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. *arXiv preprint arXiv:2510.27246*, 2025.

VISTA. LLM agents are latent context managers: Typed working memory and state proprioception. *arXiv preprint arXiv:2606.30005*, 2026.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents, 2024. URL https://arxiv.org/abs/2402.01030.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents, 2025. URL https://arxiv.org/abs/2504.12516.

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

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, et al. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*, 2025.

Weihao Zeng, Yuzhen Huang, and Junxian He. Loca-bench: Benchmarking language agents under controllable and extreme context growth. *arXiv preprint arXiv:2602.07962*, 2026.

Zep. Lies, damn lies, and statistics: Is mem0 really SOTA in agent memory? https://blog.getzep.com/lies-damn-lies-statistics-is-mem0-really-sota-in-agent-memory/, 2025.

Zep. Research: Zep benchmark results. https://www.getzep.com/research/, 2026.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025.

Yuhan Zhang, Zhiyuan Guo, Ziheng Zeng, Wei Wang, Wentao Wu, and Lijie Xu. Mandol: An agglomerative agent memory system for long-term conversations. *arXiv preprint arXiv:2606.29778*, 2026.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. DeepResearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*, 2025.

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

A Detailed Breakdowns of Benchmark Results

A.1 Per-Question-Type Results on LongMemEval

Table 5 reports Scroll's per-question-type accuracy on the `S` and `M` splits of LongMemEval under the protocol of Section 3.3. The degradation from `S` to `M` is concentrated in question types that require aggregating evidence across many sessions: multi-session accuracy drops from 88.0% to 81.2% and single-session (preference) from 100.0% to 83.3%. With more irrelevant sessions, the agent-written code reliably locates a single supporting session but often misses part of the evidence.

Table 5: Per-question-type accuracy (%) of Scroll on LongMemEval$_S$ and LongMemEval$_M$ (backbone: Qwen3.8-Max). Question types follow the benchmark taxonomy [Wu et al., 2025].

Question type	LongMemEval$_S$	LongMemEval$_M$
Single-session (user)	98.6	100.0
Single-session (assistant)	100.0	100.0
Single-session (preference)	100.0	83.3
Multi-session	88.0	81.2
Temporal reasoning	94.0	93.6
Knowledge update	98.7	87.2
Overall	94.8	89.6

A.2 Per-Category Results on BEAM

Table 6 breaks BEAM$_{10M}$ down by memory ability. We compare against Mem0, Hindsight, and Exabase M-1, the baselines for which per-category BEAM$_{10M}$ results are publicly available as of August 15, 2026 [Mem0, 2026, Bartholomew, 2026, Exabase, 2026]; Cognee and Honcho report only overall scores. Mem0, as a representative of the ingestion-heavy, fixed-pipeline paradigm, makes the contrast with Scroll's query-time approach most visible at the category level.

Table 6: Per-category judge scores on BEAM$_{10M}$. Categories follow the benchmark's ten memory abilities [Tavakoli et al., 2025]. Scroll uses Qwen3.8-Max with thinking on; baseline breakdowns are taken from their published evaluations and use different configurations, so cross-system comparison is indicative rather than controlled.

Category	Mem0	Hindsight	Exabase M-1	Scroll (ours)
Information extraction	56.2	51.2	66.3	**75.0**
Multi-session reasoning	**26.1**	16.6	9.6	21.9
Event ordering	20.2	61.6	**67.5**	64.1
Temporal reasoning	16.3	41.3	**58.8**	47.5
Knowledge update	75.0	65.0	45.0	**92.5**
Contradiction resolution	32.5	56.9	58.8	**88.1**
Summarization	46.9	78.2	**91.9**	70.5
Preference following	90.4	**97.5**	95.0	89.1
Instruction following	82.5	92.5	**97.5**	**97.5**
Abstention	40.0	80.0	**90.0**	85.0
Overall	48.6	64.1	68.0	**73.1**

Scroll leads by the widest margins where the answer hinges on a few exact records that must be located

in the raw history and then ordered or reconciled: knowledge update (92.5 vs. 45.0–75.0), contradiction resolution (88.1 vs. 32.5–58.8), and information extraction (75.0 vs. 51.2–66.3). These categories punish write-time compression: resolving an update or a contradiction needs both sides of the value timeline, in order, with provenance, whereas retrieval over an ingested store typically surfaces the current state of a fact without its ordered history. The degree varies by system (Mem0's add-only extraction preserves old facts and holds up on knowledge update at 75.0), but none matches recovering the evidence by address from the verbatim Event Log.

Conversely, Scroll underperforms the strongest baselines where the graded artifact is itself a condensed view over many records: summarization (70.5 vs. 91.9 for Exabase M-1), preference following (89.1 vs. 97.5 for Hindsight), and temporal reasoning (47.5 vs. 58.8 for Exabase M-1). Ingestion-heavy pipelines build digests and preference profiles at write time, so the condensed view already exists when the question arrives; Scroll must reconstruct it from raw events per query, and its residual failures there are errors of query formulation rather than retrieval (Appendix D).

Multi-session reasoning remains the weakest category for every system (9.6–26.1); Scroll's misses stem from over-precise filters that undercount the evidence set rather than from unreachable records. This profile is consistent across our repeated runs: contradiction resolution and knowledge update are among Scroll's strongest categories in every run, multi-session reasoning and summarization among its weakest.

B Additional LOCA Results

Comparison with published results. Table 7 places Scroll alongside the best publicly reported results from the LOCA paper [Zeng et al., 2026] and its leaderboard. These systems use different backbone models, so the comparison is system-level rather than controlled.

Table 7: System-level comparison on LOCA. Results for prior systems are taken from the LOCA paper [Zeng et al., 2026] or its public leaderboard and use different backbone models; "–" denotes no publicly reported result.

System	128K	256K
GPT-5.2-Medium + ReAct [Yao et al., 2023]	38.7	21.3
GPT-5.2-Medium + PTC [Anthropic, 2025b]	49.3	–
Claude-4.5-Opus + ReAct [Zeng et al., 2026]	34.0	14.7
MiniMax M3 + ReAct [MiniMax, 2026]	–	49.3
Scroll (Qwen3.8-Max)	**89.3**	**86.7**

C Full Prompts and Rubrics

Beyond the shared system prompt and context-management rules of Section 3.2, each memory benchmark contributes one rubric, reproduced below. For LOCA we use the benchmark's official task instructions unmodified, adding only environment metadata (available APIs and workspace paths).

BEAM rubric

```
The user's benchmark conversation history is stored in your durable history as rows
with kind='beam_chat_turn'.  Rows are chronologically ordered by seq; each session
has a distinct session_id; rows carry an ISO created_at.
```

```
Recall it with the recall_history_python tool:  pass a Python cell that uses the
pre-bound ms surface.  Search with concise keyword or synonym queries, e.g.
ms.search(QUERY, all_agents=True, kind='beam_chat_turn', k=10).  Uppercase OR passes
through as a boolean operator; otherwise terms are AND-combined.  Start with k=5 or
k=10 and increase it only when the evidence is insufficient.  If a question
combines multiple named systems, efforts, or entities, search each one separately
instead of putting every name into one AND query.  Extract one directly relevant
value for each part, preserve its unit, and only then calculate.  Each hit already
includes the full turn text together with its seq, session_id, role, and metadata;
do not call ms.expand merely to pair a user message with its assistant reply.  If
you must reread a returned turn, call ms.expand only with that hit's exact seq
neighbours (lo, hi).  For a question about one source date or an inclusive date
range, filter on the created_at column (ISO-8601 text, lexically sortable) with
ms.sql_query, e.g.  "SELECT seq, session_id, role, created_at, content FROM
hist.conversation_history WHERE kind='beam_chat_turn' AND substr(created_at,1,10)
BETWEEN '2024-07-01' AND '2024-07-31' ORDER BY seq LIMIT 50".  For elapsed calendar
days between two dates, use ms.days_between(d1, d2).

Treat role='user' rows as evidence of the user's facts, actions, and preferences;
an assistant suggestion is not evidence that the user adopted it.  Preserve exact
numbers, units, and version labels from the most directly relevant user evidence;
do not replace them with illustrative values.  Do not add repeated historical
mentions as separate quantities unless the question explicitly asks for that.  When
a fact changed, use the latest user evidence only after confirming that the rows
describe the same project and the same fact, not merely similarly named work.  Base
your answer only on recalled conversation evidence.  Do not use information from
other history kinds.  Follow any output-count or formatting constraint in the
question exactly.  If the requested fact is absent, say so clearly.

Grounding rule (strict):  every concrete claim --- each fact, number, name, date,
quantity, or event --- must come verbatim in meaning from a turn you retrieved.
Never invent specifics to make an answer sound complete.

Decide between answering and saying "not enough information" by what you actually
retrieved, not by how hard you searched:  if a turn directly states the asked-for
fact, give it; if none does, say there is not enough information in the
conversation.  A turn on a merely related topic is not the fact, and "not enough
information" is itself a correct, expected answer.

Always finish with submit_answer and a non-empty, natural-language answer.  Once
more searching stops improving your answer, commit it rather than continuing until
you run out.  Never end without one.
```

LongMemEval rubric

```
The user's benchmark conversation history is stored in your durable history as rows
with kind='context_msg' (user turns) or kind='model_turn' (assistant turns).  Every
turn's content opens with a [Session N | YYYY-MM-DD] role:  tag.  Rows are
chronologically ordered by seq; each session has a distinct session_id; rows carry
an ISO created_at.

Recall it with the recall_history_python tool:  pass a Python cell that uses the
pre-bound ms surface.  Search with concise keyword or synonym queries, e.g.
ms.search(QUERY, all_agents=True, kind='context_msg', k=10).  Uppercase OR passes
through as a boolean operator; otherwise terms are AND-combined.  Start with k=5 or
k=10 and increase it only when the evidence is insufficient.  If a question
combines multiple named systems, efforts, or entities, search each one separately
instead of putting every name into one AND query.  Extract one directly relevant
value for each part, preserve its unit, and only then calculate.  Each hit already
includes the full turn text together with its seq, session_id, role, and metadata;
```

```
do not call ms.expand merely to pair a user message with its assistant reply.  If
you must reread a returned turn, call ms.expand only with that hit's exact seq
neighbours (lo, hi).  For a question about one source date or an inclusive date
range, filter on the created_at column (ISO-8601 text, lexically sortable) with
ms.sql_query, e.g.  "SELECT seq, session_id, role, created_at, content FROM
hist.conversation_history WHERE kind='context_msg' AND substr(created_at,1,10)
BETWEEN '2023-04-01' AND '2023-04-30' ORDER BY seq LIMIT 50".  For elapsed calendar
days between two dates, use ms.days_between(d1, d2).

Treat role='user' rows as evidence of the user's facts, actions, and preferences;
an assistant suggestion is not evidence that the user adopted it.  Preserve exact
numbers, units, and version labels from the most directly relevant user evidence;
do not replace them with illustrative values.  Do not add repeated historical
mentions as separate quantities unless the question explicitly asks for that.  When
a fact changed, use the latest user evidence only after confirming that the rows
describe the same project and the same fact, not merely similarly named work.  Base
your answer only on recalled conversation evidence.  Do not use information from
other history kinds.  Follow any output-count or formatting constraint in the
question exactly.  If the requested fact is absent, say so clearly.

GROUNDING (strict):  every concrete claim -- each fact, number, name, date,
quantity, or event -- must come verbatim in meaning from a turn you retrieved.
Never invent specifics to make an answer sound complete.

Decide between answering and abstaining by what you actually retrieved, not by how
hard you searched:  if a turn directly states the asked-for fact, give it; if none
does, abstain with the exact phrase "I don't have that information from our
conversations." A turn on a merely related topic is not the fact, and abstaining is
itself a correct, expected answer when the fact is truly absent.

Always finish with submit_answer and a non-empty, natural-language answer.  Once
more searching stops improving your answer, commit it rather than continuing until
you run out.  Never end without one.
```

D Example Trajectories

This appendix reproduces four trajectories from the BEAM$_{10M}$ run reported in Table 6 (Scroll with Qwen3.8-Max, extended reasoning enabled): two successes from the categories where Scroll scores highest (knowledge update, 92.5; contradiction resolution, 88.1) and two failures from the categories where it trails the best published systems (preference following, 89.1 vs. 97.5 for Hindsight; summarization, 70.5 vs. 91.9 for Exabase M-1).

Each trajectory is shown in its logged JSON format (`task_id`, `metrics`, `steps`), abridged as follows. Model-authored code is moved from each step's `"source"` field into the referenced code block and is otherwise unedited, including the model's own comments. `"reasoning"` (the model's thinking before the action) and `"observation"` strings are excerpted, with elisions marked `[...]`; newlines inside strings are rendered literally , and non-ASCII punctuation is normalized. Gray `//` lines are our annotations, not part of the log; the steps our trajectory analysis identifies as decisive for the probe's outcome are flagged in bold color (**`// >>`** steps that drive the success, **`// >>`** steps where the failure is decided). `seq` denotes an event's immutable Event Log address; `[Session n | date]` prefixes are part of the stored events. Judge scores are those of the benchmark-provided LLM judge.

D.1 Success (knowledge update): resolving a superseded value by Event Log order

Probe (task `10M-1`, category *knowledge update*; judge score 1.0 on all four rubric criteria):
"How many tasks have I logged in Jira for the sprint on 2024-11-05, and what is my sprint completion target percentage?"

The queried values are updated *within* the relevant session (Session 42; 14→17 tasks, 85%→88%), amid decoy sprints in Sessions 4, 22, and 47. The trajectory resolves the update by Event Log order: one search, one SQL materialization, then in-kernel filters over the cached variable.


```
{"task_id": "beam/10M-1",
 "metrics": {"tokens_in": 51224, "tokens_out": 1471, "wall_time_s": 36.62, "
    step_count": 5},
 "steps": [
  // >> Success:  One coarse search pins the right session (42) among the
  // >> decoy sprints (Sessions 4, 22, 47), and both competing values already surface
  // >> in the hits
  {"step_index": 0,
   "reasoning": "I need to find the mentions of Jira tasks for the sprint on
    2024-11-05, and the sprint completion target percentage. Let me search for these
    .",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 1}},
   "observation": "
      stdout:
      8349 42 [Session 42 | 2024-11-05] user: I'm trying to update the task count in
    Jira to reflect the new total of 17 tasks, and I want to make sure I'm doing it
    correctly to meet our sprint completion target of
      [...]
      8123 42 [Session 42 | 2024-11-05] user: I'm trying to plan my sprint and I've
    added 14 tasks for dynamic resizing, aiming for 85% sprint completion. I'm using
    Jira 9
      [...]
      8350 42 [Session 42 | 2024-11-05] assistant: Great! Let's update the task count
     in Jira to reflect the new total of 17 tasks and ensure you're on track to meet
    your sprint completion target of 88%.
      [...]
      565 4 [Session 4 | 2024-07-05] user: I've added 10 feasibility tasks to my Jira
     board, targeting 85% completion within t
      [...]
      4283 22 [Session 22 | 2024-08-05] user: I'm using Jira 9.5.0 to manage my
    project, and I've added 18 tasks for ingestion strateg
      [...]"},

  // The queried values changed within Session 42 -- materialize the whole session in
  // address order, bound to the kernel variable rows
  {"step_index": 1,
   "reasoning": "The search results show Session 42 (2024-11-05) which is the
    relevant session. [...] the value CHANGED over time. Let me look at the full
    session 42 to find the most recent values mentioned. [...]",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 2}},
   "observation": "
      stdout:
      seq=8047 role=user: [Session 42 | 2024-11-05] user: I'm working on enhancing
    the RAG system's context window management, specifically focusing on dynamic
    resizing based on query complexity, an
```

```
      [...]
      [output too long: 96806 chars printed, over the 32000-char limit -- the rest is
     hidden to protect your context window. Your variables persist, so re-run printing
     LESS: a count or list of seqs, snippet=True for a bounded triage view, or
    aggregate in a variable and print only the result -- not whole rows.]"},

  // The 32K observation cap truncated the print, but rows persists --
  // re-filter in the kernel with no Event Log re-access
  {"step_index": 2,
   "reasoning": "The result was too long. Let me print less - just the user turns and
     key snippets about task counts and sprint completion percentages from Session
    42.",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 3}},
   "observation": "
      [...]
      seq=8047: [Session 42 | 2024-11-05] user: I'm working on enhancing the RAG
    system's context window management, specifically focus
      [...]"},

  // >> Success:  establishes intra-session order by seq -- 8123 (14 tasks,
  // >> 85%) precedes 8349 (17 tasks, 88%) -- exactly the evidence needed to ensure
  // >> latest information holds
  {"step_index": 3,
   "reasoning": "I found key evidence: Session 42 on 2024-11-05. [...] I need to
    check if there are any later mentions in session 42 that update these values
    further. [...]",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 4}},
   "observation": "
      [...]
      seq=8123: [Session 42 | 2024-11-05] user: I'm trying to plan my sprint and I've
     added 14 tasks for dynamic resizing, aiming for 85% sprint completion. I'm using
    Jira 9.6.0 to track my progress, but I'm not sure how to prioritize my tasks and
    allocate my time e
      [...]
      seq=8349: [Session 42 | 2024-11-05] user: I'm trying to update the task count
    in Jira to reflect the new total of 17 tasks, and I want to make sure I'm doing it
     correctly to meet our sprint completion target of 88%
      [...]"},

  // later address supersedes earlier: seq 8349 (17 tasks, 88%) follows seq 8123 (14
  // tasks, 85%); Session 47's sprint is ruled out by date.
  {"step_index": 4,
   "reasoning": "[...] 1. seq=8123: User says "I've added 14 tasks for dynamic
    resizing, aiming for 85% sprint completion" [...] The question asks specifically
    about "the sprint on 2024-11-05". The most recent mention within that sprint is
    seq=8349 with 17 tasks and 88% completion target. [...]",
   "action": {"tool": "submit_answer",
              "args": {"answer": "
      For the sprint on 2024-11-05, you logged **17 tasks** in Jira (updated from an
    earlier count of 14), and your sprint completion target percentage is **88%** (
    updated from the initial 85%)."}}}
]}
```

Code Block 1: Knowledge update: full-text search; each hit returns its Event Log address (`seq`), session, and snippet.

```
hits = ms.search("sprint 2024-11-05", scope='task', k=20)
for h in hits:
    print(h["seq"], h["step_index"], h["snippet"][:200])
```

Code Block 2: SQL materialization of Session 42 in address order, bound to the kernel variable `rows`.

```
rows = ms.sql_query(
    "SELECT seq, role, content FROM hist.conversation_history WHERE kind='
   conversation' AND step_index=42 ORDER BY msg_index"
)
for r in rows:
    # Print just the first 300 chars to get an overview
    print(f"seq={r['seq']} role={r['role']}: {r['content'][:300]}")
    print("---")
```

Code Block 3: In-kernel filter of the cached `rows` (user turns only); no Event Log access.

```
# Print only user turns from session 42, just first 200 chars each
for r in rows:
    if r['role'] == 'user':
        print(f"seq={r['seq']}: {r['content'][:200]}")
        print("---")
```

Code Block 4: Second in-kernel filter (user turns, `seq`$\geq$8100), exposing the value progression.

```
# Look at user turns from seq 8100 onwards in session 42 to see progression
for r in rows:
    if r['role'] == 'user' and r['seq'] >= 8100:
        # Print seq and a snippet around "task" or "sprint"
        content = r['content']
        print(f"seq={r['seq']}: {content[:250]}")
        print("---")
```

D.2 Success (contradiction resolution): retrieving both sides of a conflict and ordering them

Probe (task `10M-10`, category *contradiction resolution; judge score 1.0 on all four rubric criteria*):
"Has Pamela ever helped coordinate with vendors or saved setup time during my events?"

The rubric requires stating that the record is contradictory, citing the user's correction, citing the earlier claim, and resolving which is correct. The trajectory operationalizes this as a retrieval program: one query per side of the conflict, provenance checks on both, and an address-bounded search that proves no later reversal.

```
{"task_id": "beam/10M-10",
 "metrics": {"tokens_in": 104160, "tokens_out": 5758, "wall_time_s": 132.78, "
    step_count": 10},
 "steps": [
  // search the affirmative side of the conflict
  {"step_index": 0,
   "reasoning": "[...] I need to find all mentions of Pamela and vendor coordination/
    setup time, identify the conflict, and determine which statement is the latest or
    a correction. [...]",
   "action": {"tool": "recall_history_python",
```

```
            "args": {"source": Code Block 5}},
 "observation": "
    stdout:
    16900 73  ... Certainly! It sounds like Pamela took initiative to gather quick
  ratings from three vendors, which helped streamline the feedback process and saved
    [...]
    8434 36  ... 1. **Morning Setup**:
       - **Pamela**: Supervise the arrival and setup of vendors and decorators.

    [...]
    7254 31  ... Great to hear that Pamela is here to help! Given her experience
  and maturity, she can be a valuable asset in managing t
    [...]"},

// >> Success:  the model authors an opposing, negation-term query for the
// >> 'against' side of the conflict; the user's correction surfaces at seq 8867
{"step_index": 1,
 "reasoning": "The search results are showing that Pamela coordinated with vendors.
   Next I need to find the contradictory information. [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 6}},
 "observation": "
    [...]
    8867 37 [Session 37 | 2023-07-15] user: How can I make sure Pamela is aware she
   didn't actually brief any entertainers or save me coordination time, as that
    [...]"},

// materialize the correction exactly
{"step_index": 2,
 "reasoning": "The search result at seq 8867 (Session 37) looks like it has key
  information [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 7}},
 "observation": "
    stdout:
    8867 user
    [Session 37 | 2023-07-15] user: How can I make sure Pamela is aware she didn't
  actually brief any entertainers or save me coordination time, as that was a
  misunderstanding?
    ---"},

// session-scoped SQL around the correction; the original delegation (seq 8675)
// sits in the same session
{"step_index": 3,
 "reasoning": "[...] Now I need to check what was said earlier about Pamela
  coordinating with vendors or saving setup time. [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 8}},
 "observation": "
    [...]
    seq 8675 [user]: [Session 37 | 2023-07-15] user: What's the most efficient way
  to brief the 5 entertainers, as Pamela had taken on this task earlier?
    [...]"},

// expand the later, seemingly re-affirming mentions from Session 73
{"step_index": 4,
 "reasoning": "[...] Let me look at the seq 16900 and 16902 hits from Session 73,
  which seem to discuss Pamela's vendor ratings and saved time. [...]",
 "action": {"tool": "recall_history_python",
```

```
            "args": {"source": Code Block 9}},
 "observation": "
    [...]
    seq 16902 [assistant]:
    [Session 73 | 2023-08-22] assistant: You're welcome! I'm glad that clarifying
  Pamela's actions helped you understand how her quick calls saved you time and
  provided immediate insights.
    [...]
    2. **Time Saved**: By handling these calls herself, Pamela saved you
  approximately 15 minutes of follow-up time.
    [...]"},

// provenance check: those are assistant turns -- verify what the user actually
// said nearby; the user's own later confirmation surfaces
{"step_index": 5,
 "reasoning": "[...] this is assistant-generated text, not the user's own statement
  . I need to check whether the user in session 73 actually said something that led
  to this assistant response, or if this wa [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 10}},
 "observation": "
    [...]
    seq 16899 [user]: [Session 73 | 2023-08-22] user: Can you help me understand
  why Pamela called 3 vendors for quick ratings, and how it saved me 15 minutes of
  follow-up?
    [...]
    seq 16901 [user]: [Session 73 | 2023-08-22] user: Yeah, that makes sense.
  Pamela's quick calls definitely saved me time. It's good to get those immediate
  insights without having to follow up myself. Thanks for clarifying!
    [...]"},

// the reasoning separates the two claims: 'coordination time' during setup
// (corrected) vs. post-event 'follow-up time' (confirmed)
{"step_index": 6,
 "reasoning": "[...] The correction in Session 37 is about Pamela NOT briefing
  entertainers or saving coordination time (a misunderstanding). [...] In Session
  73, the user confirms Pamela called 3 vendors for quick ratings and saved 15
  minutes of follow-up time. [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 11}},
 "observation": "
    [...]
    seq 8868 [assistant]: [Session 37 | 2023-07-15] assistant: To clarify the
  misunderstanding with Pamela and ensure she is aware that she did not actually
  brief any entertainers or save coordination time, it's important to communicate
  this information
    [...]"},

// retrieve the claim side verbatim (Sessions 31 and 36)
{"step_index": 7,
 "reasoning": "[...] **Earlier claims (pre-correction):** Multiple sessions (31,
  36) had assistant suggestions about Pamela supervising vendor arrival/setup,
  coordinating with vendors, etc. [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 12}},
 "observation": "
    [...]
    seq 7496 [assistant]: [Session 31 | 2023-07-10] assistant: Delegating tasks
  like decor oversight to someone like Pamela can have a significant positive impact
```

```
    on the overall timeli
      [...]
      seq 8434 [assistant]: [Session 36 | 2023-07-15] assistant: That sounds like an
   excellent plan! By leveraging Pamela's skills and experience, you can ensure that
   the guest check-in
      [...]"},

  // >> Success:  the disconfirming query -- an address-bounded search proving
  // >> nothing after Session 73 overturns the correction.
  // >> Across this run, successful trajectories issue such a query before
  // >> submitting; but failed ones never do
  {"step_index": 8,
   "reasoning": "[...] The later statement (Session 73) would normally win. But I
    need to check if there's a later correction after Session 73 as well. [...]",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 13}},
   "observation": "
      stdout:
      18077 77 [Session 77 | 2023-08-24] user: Yeah, it definitely caused some
    frustration. Those 5 guests w
      [...]"},

  // each rubric criterion maps to evidence retrieved explicitly
  {"step_index": 9,
   "reasoning": "[...] So the resolution is nuanced: The user corrected that Pamela
    did NOT brief entertainers or save coordination time during setup (that was a
    misunderstanding), but later confirmed Pamela DID help by calling vendors for
    quick ratings, saving follow-up time [...]",
   "action": {"tool": "submit_answer",
              "args": {"answer": "
      The conversation contains conflicting information about this, and here's how it
     resolves:

      **The initial claim:** In earlier sessions (around Sessions 31-36), plans were
    made for Pamela to coordinate with vendors, supervise vendor arrivals/setup, and
    brief entertainers -- with the expectation that this would save you coordination
    time during the event.

      **Your correction (Session 37, July 15, 2023):** You explicitly clarified that
    this was a misunderstanding -- Pamela did NOT actually brief any entertainers or
    save you coordination time during setup. You asked how to make sure Pamela was
    aware of this correction.

      **A later, narrower confirmation (Session 73, August 22, 2023):** You confirmed
     that Pamela did call 3 vendors for quick ratings after the event, which saved you
     about 15 minutes of follow-up time. You said: "Pamela's quick calls definitely
    saved me time."

      **Resolution:** The broad claim that Pamela coordinated vendors and saved setup
     time during your events was corrected by you as a misunderstanding (Session 37).
    However, Pamela did help in a more limited, post-event way -- gathering quick
    vendor ratings that saved you ~15 minutes of follow-up (Session 73). So she didn't
     save setup time or coordinate vendors during the event itself, but she did
    contribute afterward with vendor feedback calls."}}}
]}
```

Code Block 5: Contradiction resolution: full-text search on the claim side.

```
hits = ms.search("Pamela vendor", scope='task', k=20)
for h in hits:
    print(h["seq"], h["step_index"], h["snippet"][:150])
```

Code Block 6: Negation-term search for the correction side.

```
hits2 = ms.search("Pamela didn't OR wasn't OR never OR not", scope='task', k
    =20)
for h in hits2:
    print(h["seq"], h["step_index"], h["snippet"][:150])
```

Code Block 7: Exact materialization of the correction (`ms.expand`).

```
rows = ms.expand([8867])
for r in rows:
    print(r["seq"], r["role"])
    print(r["content"][:2000])
    print("---")
```

Code Block 8: Session-scoped SQL over Session 37.

```
# Look at the assistant response after seq 8867 and surrounding context in
    session 37
rows = ms.sql_query("SELECT seq, role, content FROM hist.conversation_history
     WHERE kind='conversation' AND step_index=37 ORDER BY seq LIMIT 30")
for r in rows:
    print(f"seq {r['seq']} [{r['role']}]: {r['content'][:200]}")
    print("---")
```

Code Block 9: Expansion of the post-correction mentions (`seq` 16900, 16902).

```
# Check later references (after the correction at seq 8867) about Pamela and
    vendor coordination/setup time
rows = ms.expand([16900, 16902])
for r in rows:
    print(f"seq {r['seq']} [{r['role']}]:")
    print(r["content"][:600])
    print("---")
```

Code Block 10: Provenance check—the user turns around `seq` 16900.

```
# Check what the user said in session 73 around seq 16900
rows = ms.sql_query("SELECT seq, role, content FROM hist.conversation_history
     WHERE kind='conversation' AND step_index=73 AND seq BETWEEN 16890 AND
    16910 ORDER BY seq")
for r in rows:
    print(f"seq {r['seq']} [{r['role']}]: {r['content'][:300]}")
    print("---")
```

Code Block 11: The assistant's uptake of the correction (`seq` 8868).

```
# Check the assistant response after the user's correction at seq 8867
rows = ms.sql_query("SELECT seq, role, content FROM hist.conversation_history
     WHERE kind='conversation' AND step_index=37 AND seq BETWEEN 8867 AND
    8870 ORDER BY seq")
```

```
for r in rows:
    print(f"seq {r['seq']} [{r['role']}]: {r['content'][:500]}")
    print("---")
```

Code Block 12: The original claim side (Sessions 31 and 36).

```
# Let me also check the earlier sessions (31, 36) where Pamela was assigned
   vendor coordination roles, to get the full picture of the "claim" side
rows = ms.expand([7496, 8434])
for r in rows:
    print(f"seq {r['seq']} [{r['role']}]: {r['content'][:400]}")
    print("---")
```

Code Block 13: `seq_range`-bounded search over the remainder of the history, proving no later reversal.

```
# Check if there are any later corrections about Pamela after session 73
hits3 = ms.search("Pamela misunderstanding OR correction OR didn't OR wasn't"
   , scope='task', k=15, seq_range=(16900, 23716))
for h in hits3:
    print(h["seq"], h["step_index"], h["snippet"][:200])
```

D.3 Failure (preference following): competent retrieval on the wrong axis

Probe (task `10M-8`, category *preference following*; judge score 0.0 on all three rubric criteria):
"I'm planning the route for our upcoming trip. How would you suggest I map it out?"

All three criteria concern one stated preference: avoiding tolls. The retrieval patterns that succeed elsewhere all execute correctly here—the failure is upstream, in query formulation: the model frames the probe as *which tool* the user prefers and never queries the *route-constraint* axis. The graded preference is abundant in the Event Log, in the user's own words (seq 805: "which route would you say is better for someone who wants to avoid tolls as much as possible?"; seq 807; Session 3, seq 617–631, where the hybrid route is adopted because it saves $15 in tolls).

```
{"task_id": "beam/10M-8",
 "metrics": {"tokens_in": 82830, "tokens_out": 2471, "wall_time_s": 69.19, "
    step_count": 7},
 "steps": [
  // >> Failure:  the outcome is decided here, before any retrieval runs --
  // >> the probe is framed as 'which mapping tool', all six queries name tools, and
  // >> none of the trajectory's 14 queries contains toll, fee, avoid, or cost,
  // >> which is the axis all three rubric criteria grade
  {"step_index": 0,
   "reasoning": "[...] I need to find a preference they established earlier -- a
    chosen tool, version, style, or constraint. [...]",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 14}},
   "observation": "
      stdout:
      10016 43  ... Drag and drop the route lines to adjust the path according to the
     detour.
      [...]
      8148 34  ... Compare the routes to see which one avoids tolls.
      [...]
```

```
      662 3  ... Sync with Google Maps
      While you cannot directly sync Google Maps with GasBuddy, you can use both apps
     together
      [...]"},

  // >> Failure:  the toll preference brushes past in the previous step's hits
  // >> (seq 8148, 'avoids tolls') and goes unpursued; the model chases the Maps.me
  // >> tool candidate instead
  {"step_index": 1,
   "reasoning": "[...] Let me also check seq 16614 which mentions Maps.me as a "
    Current Tool". [...]",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 15}},
   "observation": "
      [...]
      16614 [Session 75 | 2023-04-20] assistant: Improving your detour accuracy and
    saving time with navigation tools like Maps.me is a great achievement. Here are
    some other n
      [...]"},

  // sound provenance discipline: the Maps.me hit is discarded as assistant-authored;
  // five preference-phrase queries follow -- still tool-framed
  {"step_index": 2,
   "reasoning": "[...] that's an assistant response. I need to find where the user
    explicitly states their preference for a mapping tool. [...]",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 16}},
   "observation": "
      [...]
      2349 10  ... That sounds like a solid plan! I'll make sure to use Google Maps
    and Roadtrippers to map out the hybrid route and share it with Chris. I'll also
    download off
      [...]
      17175 77 [Session 77 | 2023-04-22] user: That sounds great! I'll definitely use
     mapping tools like Google Maps and Waze to plan my routes and find good stopping
    points.
      [...]"},

  // recency check over the final sessions:
  // no later preference supersedes the finding
  {"step_index": 3,
   "reasoning": "[...] Let me look more closely at the most recent user turns about
    mapping, especially in later sessions (82-100), [...]",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 17}},
   "observation": "
      (no output)"},

  // user-authored evidence for the tool preference accumulates (Sessions 1, 10, 13)
  {"step_index": 4,
   "reasoning": "[...] the user has consistently used **Roadtrippers** and **Google
    Maps** together. [...]",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 18}},
   "observation": "
      [...]
      2335 10  ... That sounds perfect! I'll use Google Maps and Roadtrippers to map
    out the hybrid route and share it with Chris. We'll definitely need to download
    offline maps for a
```

```
      [...]
      25 [Session 1 | 2022-05-01] user: How does the Roadtrippers app work, and can
    it really help me visualize a route that covers at least 5 national parks?
      [...]"},

  // scope the upcoming trip (Session 100):
  // a 1,200-mile northern route, four landmarks
  {"step_index": 5,
   "reasoning": "[...] check if there's anything in the most recent sessions (S100,
    seq 22221-22560) about a specific route they're planning for the "upcoming trip
    [...]",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 19}},
   "observation": "
      [...]
      22221 user [Session 100 | 2023-05-31] user: What's the best way to plan a
    1,200-mile northern route within 9 months, aiming for 4 key landmarks, like I've
    targeted?
      [...]"},

  // The answer is grounded and well-structured, but without toll information.
  // Judge: 0.0 on all three criteria.
  {"step_index": 6,
   "reasoning": "[...] The most recent and repeated preference is **Google Maps and
    Roadtrippers** used together. The user consistently says they'll use both to map
    out routes. [...]",
   "action": {"tool": "submit_answer",
              "args": {"answer": "
      Based on our past conversations, you've consistently used **Google Maps and
    Roadtrippers** together to map out your routes -- you mentioned this combination
    multiple times, including when planning the hybrid route with Chris and when
    scouting scenic areas. I'd suggest sticking with that approach for your upcoming
    1,200-mile northern route to your 4 key landmarks:
      [...]
      1. **Roadtrippers first** -- Plot your 4 landmarks as waypoints and use it to
    discover scenic stops, detours, and points of interest along the northern route.
    It's great for visualizing the overall tri
      [...]
      This two-tool combo has worked well for you across multiple trips, so it should
     serve you well for this one too."}}}
 ]}
```

Code Block 14: Preference following: multi-query sweep, de-duplicated by address; all six queries name tools.

```
# Search for mapping tool preferences in the conversation
hits = []
for q in ["map route", "Google Maps", "Roadtrippers", "mapping tool", "app
    route", "map out"]:
    hits += ms.search(q, scope='task', k=10)

# Print unique hits with snippets for triage
seen = set()
for h in hits:
    if h["seq"] not in seen:
        seen.add(h["seq"])
        print(h["seq"], h["step_index"], h.get("snippet", "")[:150])
```

```
        print("---")
```

Code Block 15: Expansion of the Maps.me candidate (`seq` 16614).

```
# Let me look at the user's own turns about mapping tools, especially from
    later sessions
# First check seq 16614 which mentions Maps.me as "Current Tool"
rows = ms.expand([16614])
for r in rows:
    print(r["seq"], r["content"][:500])
    print("---")
```

Code Block 16: Preference-phrase sweep plus a Roadtrippers query—still tool-framed.

```
# Search for user turns specifically about their preferred mapping tool
hits2 = []
for q in ["I'll use", "I prefer", "my favorite", "I've been using", "I use"]:
    hits2 += ms.search(f"{q} map", scope='task', k=10)

# Also search for Roadtrippers specifically in user turns
hits3 = ms.search("Roadtrippers route plan", scope='task', k=15)

seen2 = set()
for h in hits2 + hits3:
    if h["seq"] not in seen2:
        seen2.add(h["seq"])
        print(h["seq"], h["step_index"], h.get("snippet", "")[:160])
        print("---")
```

Code Block 17: `seq_range`-bounded recency check over the final sessions.

```
# Look at user turns in later sessions about mapping tools
hits4 = ms.search("map route plan", seq_range=(18323, 22716), scope='task', k
    =20)
for h in hits4:
    if h.get("role") == "user":
        print(h["seq"], h["step_index"], h.get("snippet", "")[:180])
        print("---")
```

Code Block 18: User-turn search plus session-scoped SQL over Session 1.

```
# Check user turns specifically mentioning their chosen mapping approach
# Focus on user turns that state what they'll use or prefer
hits5 = ms.search("Google Maps Roadtrippers map out", scope='task', k=15)
for h in hits5:
    if h.get("role") == "user":
        print(h["seq"], h["step_index"], h.get("snippet", "")[:200])
        print("---")

# Also check the earliest session where Roadtrippers was introduced
print("\n=== Session 1 user turns about Roadtrippers ===")
rows_s1 = ms.sql_query("SELECT seq, role, content FROM hist.
    conversation_history WHERE kind='conversation' AND step_index=1 AND role
    ='user' AND content LIKE '%Roadtrippers%' ORDER BY msg_index LIMIT 5")
```

```
for r in rows_s1:
    print(r["seq"], r["content"][:300])
    print("---")
```

Code Block 19: Reading the opening of Session 100 to scope the upcoming trip.

```
# Check the most recent session (S100) for context about the upcoming trip
rows_s100 = ms.sql_query("SELECT seq, role, content FROM hist.
   conversation_history WHERE kind='conversation' AND step_index=100 ORDER
   BY msg_index LIMIT 10")
for r in rows_s100:
    print(r["seq"], r["role"], r["content"][:250])
    print("---")
```

D.4 Failure (summarization): positional sampling misses mid-session evidence

Probe (task `10M-8`, category *summarization*; judge score 0.42—two of six rubric criteria fully satisfied, one partially):

"Can you summarize how my travel decisions and habits evolved from April 8, 2023 to April 25, 2023 and how they influenced my overall experience and personal growth?"

Coverage at session granularity is complete: cells 0–6 page and merge all 4,448 events in the date range (Sessions 61–80, verified gap-free). The score is then decided by sampling choices within that coverage: the trajectory reads long mixed-topic sessions *positionally* (head and tail) rather than *thematically*, and the graded mid-session facts—the 3-hour driving cap with its fatigue reduction (Session 77, seq 17171–17207) and the sleep and hydration goals (Session 66, seq 14809–14817)—never enter an observation.

```
{"task_id": "beam/10M-8",
 "metrics": {"tokens_in": 197973, "tokens_out": 8846, "wall_time_s": 221.2, "
    step_count": 14},
 "steps": [
  // date-range SQL over event metadata; both the 1000-row cap and the 32K
     observation
  // cap bite -- the model pivots to paging
  {"step_index": 0,
   "reasoning": "[...] First, I need to find the conversations within this date range
    . I'll try using SQL to filter by date.",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 20}},
   "observation": "
      stdout:
      1000
      13611 61 2023-04-08 user [Session 61 | 2023-04-08] user: What's the best way to
     spend our last day in Los Angeles bef
      [...]
      [note] a sql_query hit the 1000-row cap -- matching rows beyond the first 1000
    were NOT returned; narrow with WHERE or page with LIMIT/OFFS
      [...]"},

  // aggregate per session in the kernel: page 1 reaches only 2023-04-13
  {"step_index": 1,
   "reasoning": "The result got cut off at 1000 rows. Need to paginate. [...]",
   "action": {"tool": "recall_history_python",
              "args": {"source": Code Block 21}},
```

```
 "observation": "
    stdout:
    1000 13611 14610 2023-04-13
    61 ['2023-04-08'] 188
    62 ['2023-04-09'] 222
    63 ['2023-04-10'] 196
    64 ['2023-04-11'] 190
    65 ['2023-04-12'] 194
    66 ['2023-04-13'] 10"},

// page the remainder; steps 3-5 rerun this query changing only the OFFSET
{"step_index": 2,
 "reasoning": "[...] I need to fetch the remaining period (April 13-25). [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 22}},
 "observation": "
    stdout:
    1000
    66 ['2023-04-13'] 268
    67 ['2023-04-14'] 200
    68 ['2023-04-15'] 198
    69 ['2023-04-15'] 196
    70 ['2023-04-15'] 138
    15600 2023-04-15"},

{"step_index": 3,
 "reasoning": "I need to continue paging through the data and keep collecting the
  remaining pages until I reach April 25th.",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 22 (OFFSET 1000)}},
 "observation": "
    stdout:
    1000 16600 2023-04-20
    [...]"},

{"step_index": 4,
 "reasoning": "On track -- continuing the paging to cover April 20-25.",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 22 (OFFSET 2000)}},
 "observation": "
    stdout:
    1000 17600 2023-04-24
    [...]"},

{"step_index": 5,
 "reasoning": "[...] Let me fetch the last page (the remainder from April 24th
  onwards) to complete the coverage, [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 22 (OFFSET 3000)}},
 "observation": "
    stdout:
    458
    79 ['2023-04-24'] 178
    80 ['2023-04-25'] 280"},

// merge and de-duplicate by seq in the kernel: 4,448 events, Sessions 61-80,
  verified
// gap-free -- session-level coverage is complete. Note Session 77's size: 470
  events
```

```
{"step_index": 6,
 "reasoning": "[...] Next I need to merge everything into one complete dataset (
  deduplication), [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 23}},
 "observation": "
    [...]
    4448 13611 2023-04-08 18058 2023-04-25
    [...]
    61 ['2023-04-08'] 188 13611 13798
    [...]
    77 ['2023-04-22'] 470 16909 17378
    [...]
    80 ['2023-04-25'] 280 17779 18058"},

// >> Failure:  the first of the two sampling choices that decide the score
// >> -- positional reading (first 3 + last 2 user turns per session);
// >> head-and-tail reads of 200-470-row mixed-topic sessions systematically
// >> skip the middle, where habit threads sit
{"step_index": 7,
 "reasoning": "[...] that's about 2200 user turns, way too much. Better approach:
  for each session, print the first few and last few user turns.",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 24}},
 "observation": "
    [...]
    === Session 61 (2023-04-08) -- 94 user turns ===
    [...]
    13611: [S61 | 2023-04-08] U: What's the best way to spend our last day in Los
   Angeles before heading home, considering we're starting the final stretch of our
   2,500-mi
    [...]"},

// >> Failure:  the deep read stays head-only (first 6 user turns of 7 sessions);
// >> the graded driving-limit arc sits  100 user turns into Session 77
// >> (seq 17171-17207) and is never reached -- even though this read surfaces 'I'm
// >> trying to manage my travel fatigue' (Session 63) as a lead
{"step_index": 8,
 "reasoning": "[...] To grasp the main components, let me try sampling a few turns
  from each session in more detail. [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 25}},
 "observation": "
    [...]
    ########## SESSION 61 (2023-04-08) ##########
    [...]
    --- seq 13611 ---
    [Session 61 | 2023-04-08] user: What's the best way to spend our last day in
   Los Angeles before heading home, considering we're starting the final stretch of
   ou
    [...]"},

// >> Failure:  the second sampling choice -- the trajectory's only thematic
// >> search covers growth vocabulary exclusively; the probe asks about
// >> 'decisions AND habits', but no query ever names driving hours, fatigue,
// >> sleep, or hydration (the graded habit dimensions)
{"step_index": 9,
 "reasoning": "[...] Let me search for growth-related turns and key decisions
  within a few sessions. [...]",
```

```
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 26}},
 "observation": "
    [...]
    15489  ... I feel like I learned a lot about myself and our relationship during
   that hike.
    [...]"},

// verbatim grounding of the growth quantifications (40% patience, 35% flexibility,
// 82% self-rating) -- these earn the fully satisfied criteria
{"step_index": 10,
 "reasoning": "[...] Let me expand a few of the key seqs to get verbatim details.
  [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 27}},
 "observation": "
    [...]
    === seq 15489 | role=user ===
    [Session 70 | 2023-04-15] user: Yeah, the Grand Canyon hike was definitely a
  standout moment. It pushed us both physically and mentally, and seeing the canyon
  was breathtaking. It made me realize how much I can accomplish with determination
  and teamwork. Plus, it gave me a new perspective on life and made me appreciate
  nature more. I feel like I learned a lot about myself and our relationship during
  that hike.


    [...]"},

// budget details land ($70 Pittsburgh delay, $250 food overspend); the
// sleep/hydration cluster (seq 14809-14817, mid-Session 66) is never touched
{"step_index": 11,
 "reasoning": "[...] let me gather a few more specific points: budget details ($250
   food overspend, Pittsburgh delay $70), habit changes (session 72, screen time),
  [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 28}},
 "observation": "
    [...]
    HIT 17368  ... In this case, the total overspend on food is $250, with $60
  coming from a single diner visit in LA. Here's a structured approach to
    [...]"},

// final spot checks; the model declares coverage sufficient
{"step_index": 12,
 "reasoning": "[...] Actually, I have enough material. [...]",
 "action": {"tool": "recall_history_python",
            "args": {"source": Code Block 29}},
 "observation": "
    [...]
    [Session 79 | 2023-04-24] assistant: Limiting trips to 2 stops max played a
  significant role in my overall personal growth, particularly in handling
  unexpected detours and fee
    [...]"},

// judge: 0.42 -- two of six criteria fully satisfied, one partial. Everything the
// rubric graded was inside the materialized sessions; the misses (3-hour driving
// cap, Session 77; 8-hour sleep and hydration goals, Session 66) fell outside the
// sampled positions
{"step_index": 13,
```

```
   "reasoning": "[...] Let me do a final coverage check against what the question is
    ask [...]",
   "action": {"tool": "submit_answer",
              "args": {"answer": "
      **The arc, April 8-25, 2023 (Sessions 61-80):** Your travel decisions evolved
    through three clear phases: (1) **wrap-up logistics** in Los Angeles (Apr 8-10),
    (2) **deliberate rest and the journey home** (Apr 13-16), and (3) **post-trip
    processing, sharing, and lesson-building** back in New Jeffreytow
      [...]
      You **quantified your growth**: 40% more patience (later framed as a 40%
    increase in confidence and adaptability), 35% more flexibility after the 5
    unplanned detours, and an overall trip-growth self-rating of 82%.

      [...]
      By April 25 you had distilled **4 key trip lessons** -- "Flexibility is Key"
    and "Embrace Spontaneity" among them -- and begun journaling to sustain the gains.
     Your unplanned stops reshaped your future travel philosophy: looser itineraries
    with buffer time, openness to local recommendations, and fewer, deeper stops
    rather than rigid day-by-day plans."}}}
 ]}
```

Code Block 20: Summarization: date-range SQL over event metadata (2023-04-08 to 2023-04-25).

```
rows = ms.sql_query(
    "SELECT seq, step_index, json_extract(metadata,'$.date') AS date, role,
    headline, substr(content, 1, 200) AS preview "
    "FROM hist.conversation_history WHERE kind='conversation' "
    "AND json_extract(metadata,'$.date') BETWEEN '2023-04-08' AND
    '2023-04-25' "
    "ORDER BY msg_index")
print(len(rows))
for r in rows:
    print(r['seq'], r['step_index'], r['date'], r['role'], (r['preview'] or '
    ')[:120].replace('\n',' '))
```

Code Block 21: Per-session aggregation of page 1 in the kernel.

```
rows_all = list(rows)
print(len(rows_all), rows_all[0]['seq'], rows_all[-1]['seq'], rows_all[-1]['
    date'])
# Aggregate per session: date range
from collections import defaultdict
sess = defaultdict(lambda: {'dates': set(), 'roles': 0})
for r in rows_all:
    sess[r['step_index']]['dates'].add(r['date'])
    sess[r['step_index']]['roles'] += 1
for s in sorted(sess):
    print(s, sorted(sess[s]['dates']), sess[s]['roles'])
```

Code Block 22: Paging query; steps 3–5 rerun it changing only the `OFFSET`.

```
rows2 = ms.sql_query(
    "SELECT seq, step_index, json_extract(metadata,'$.date') AS date, role,
    substr(content,1,180) AS preview "
    "FROM hist.conversation_history WHERE kind='conversation' "
    "AND json_extract(metadata,'$.date') BETWEEN '2023-04-13' AND
```

```
    '2023-04-25' "
    "ORDER BY msg_index LIMIT 1000 OFFSET 0")
print(len(rows2))
rows2 = list(rows2)
from collections import defaultdict
sess2 = defaultdict(lambda: {'dates': set(), 'n': 0})
for r in rows2:
    sess2[r['step_index']]['dates'].add(r['date'])
    sess2[r['step_index']]['n'] += 1
for s in sorted(sess2):
    print(s, sorted(sess2[s]['dates']), sess2[s]['n'])
print(rows2[-1]['seq'], rows2[-1]['date'])
```

Code Block 23: Kernel-side merge of all five pages, de-duplicated by `seq`.

```
# Merge all pages by seq (dedupe)
all_turns = {}
for coll in [rows_all, rows2, rows3, rows4, rows5]:
    for r in coll:
        all_turns[r['seq']] = r
merged = [all_turns[k] for k in sorted(all_turns)]
print(len(merged), merged[0]['seq'], merged[0]['date'], merged[-1]['seq'],
    merged[-1]['date'])
from collections import defaultdict
sess_all = defaultdict(lambda: {'dates': set(), 'n': 0, 'first_seq': None, '
    last_seq': None})
for r in merged:
    d = sess_all[r['step_index']]
    d['dates'].add(r['date'])
    d['n'] += 1
    if d['first_seq'] is None:
        d['first_seq'] = r['seq']
    d['last_seq'] = r['seq']
for s in sorted(sess_all):
    print(s, sorted(sess_all[s]['dates']), sess_all[s]['n'], sess_all[s]['
    first_seq'], sess_all[s]['last_seq'])
```

Code Block 24: Positional sampling—first 3 and last 2 user turns per session.

```
from collections import defaultdict
sess_turns = defaultdict(list)
for r in merged:
    sess_turns[r['step_index']].append(r)

# Print first 3 and last 2 user turns per session to get arc
for s in sorted(sess_turns):
    users = [r for r in sess_turns[s] if r['role'] == 'user']
    print(f"=== Session {s} ({users[0]['date']}) -- {len(users)} user turns
    ===")
    for r in users[:3] + users[-2:]:
        txt = r['preview'].replace('[Session ', '[S').replace('] user: ', ']
    U: ')
        print(f"  {r['seq']}: {txt[:160]}")
    print()
```

Code Block 25: Head-only deep read (first 6 user turns) of sessions 61, 63, 66, 70, 76, 77, 80.

```
# Pull full user-turn content for decision-rich sessions and print to read
import json
def full_content(seq):
    rr = ms.sql_query("SELECT content FROM hist.conversation_history WHERE
   seq=?", (seq,))
    return rr[0]['content'] if rr else None

# Sample: first ~6 user turns of sessions 61,63,66,70,76,77,80
for s in [61,63,66,70,76,77,80]:
    ut = [r for r in sess_turns[s] if r['role']=='user']
    print(f"\n########## SESSION {s} ({ut[0]['date']}) ##########")
    for r in ut[:6]:
        c = full_content(r['seq'])
        print(f"\n--- seq {r['seq']} ---")
        print(c[:600])
```

Code Block 26: The trajectory's only thematic search—growth vocabulary, bounded to the date range.

```
hits = []
for q in ["personal growth", "lessons learned", "personal growth trip", "
   growth", "changed me", "learned about myself"]:
    hits += ms.search(q, seq_range=(13611, 18058), scope='task', k=6)
seen = {}
for h in hits:
    if h['seq'] not in seen:
        seen[h['seq']] = h
print(len(seen))
for seq, h in sorted(seen.items()):
    print(seq, h.get('snippet','')[:160].replace('\n',' '))
```

Code Block 27: Verbatim expansion of eight growth-evidence addresses.

```
rows = ms.expand([15138, 15489, 16441, 16678, 16824, 17216, 17712, 17932])
for r in rows:
    print(f"=== seq {r['seq']} | role={r['role']} ===")
    print(r['content'][:900])
    print()
```

Code Block 28: Targeted expansions plus two budget searches.

```
rows = ms.expand([16009, 16018])
for r in rows:
    print(f"=== seq {r['seq']} | role={r['role']} ===")
    print(r['content'][:800])
    print()
# also search budget / Pittsburgh delay / food overspend
hits2 = ms.search("Pittsburgh delay", seq_range=(13611,18058), scope='task',
   k=5)
for h in hits2:
    print("HIT", h['seq'], h.get('snippet','')[:200].replace('\n',' '))
hits3 = ms.search("food overspend", seq_range=(13611,18058), scope='task', k
   =5)
for h in hits3:
```

```
    print("HIT", h['seq'], h.get('snippet','')[:200].replace('\n',' '))
```

Code Block 29: Final point reads (`seq` 16685, 17712).

```
# Quick final check: confirm the "2 stops max" and "future travel" decision
    details from seq 17712 and 16678 context
r1 = ms.sql_query("SELECT content FROM hist.conversation_history WHERE seq
    =16685")
print(r1[0]['content'][:500])
r2 = ms.sql_query("SELECT content FROM hist.conversation_history WHERE seq
    =17712")
print(r2[0]['content'][:300])
```