MEME: Multi-Entity and Evolving Memory Evaluation

Seokwon Jung [1] **Alexander Rubinstein** [*2] **Arnas Uselis** [*2] **Sangdoo Yun** [3] **Seong Joon Oh** [1]

Abstract

LLM-based agents increasingly operate in persistent environments where they must store, update, and reason over information across many sessions. While prior benchmarks evaluate only single-entity updates, MEME defines six tasks spanning the full space defined by the multi-entity and evolving axes, including three not scored by prior work: Cascade and Absence (dependency reasoning) and Deletion (post-removal state). Evaluating six memory systems spanning three memory paradigms on 100 controlled episodes, we find that all systems collapse on dependency reasoning under the default configuration (Cascade: 3%, Absence: 1% in average accuracy) despite adequate static retrieval performance. Prompt optimization, deeper retrieval, reduced filler noise, and most stronger LLMs fail to close this gap. Only a file-based agent paired with Claude Opus 4.7 as its internal LLM partially closes the gap, but at $\sim$70$\times$ the baseline cost, indicating closure currently depends on configurations that are not practical at scale. Code and data are available on the project page: https://seokwonjung-jay.github.io/meme-eval/.

1. Introduction

As Large Language Models (LLMs) increasingly serve as agents that interact with users across many sessions, accurately storing, updating, and reasoning over past interactions has become essential (Zhang et al., 2024). For instance, when a user reports moving to a new city, the agent must not only record this change but also recognize that previously stored facts that depended on the old residence, such as commute time or nearby facilities, may no longer be valid. Today's memory systems address such needs through three broad paradigms: raw retrieval over unprocessed text chunks (Izacard et al., 2022), LLM-processed memory that extracts and reorganizes facts (Chhikara et al., 2025; Packer et al., 2023; Rasmussen et al., 2025; Edge et al., 2024), and file-based agents where an LLM manages persistent files via tool-calling (Karpathy, 2026). Evaluation for these systems has evolved from single-turn long-context benchmarks (Hsieh et al., 2024; Modarressi et al., 2025) through multi-session evaluations of static fact retention (Maharana et al., 2024) to benchmarks with dynamic updates, where entity values change across sessions (Wu et al., 2025; Tan et al., 2025; Hu et al., 2026).

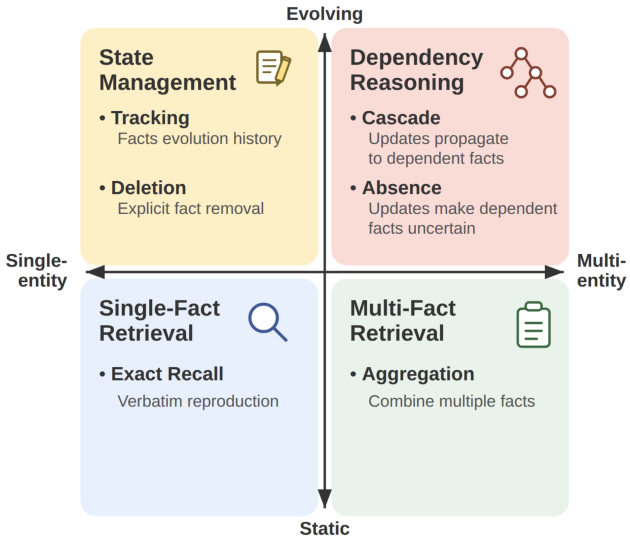


Figure 1. MEME's taxonomy of memory operations along two dimensions: entity scope (single vs. multi-entity) and temporal dynamics (static vs. evolving), with six tasks distributed across the four quadrants.

Yet existing memory benchmarks evaluate updates only for independent entities, missing the dependency reasoning that real interactions require (Table 1). No prior benchmark scores how a dependent fact changes after an upstream update (Cascade), how a previously valid answer becomes uncertain (Absence), or how a removed fact stops being reported (Deletion). This leaves a critical blind spot in how today's memory systems handle stateful, interdependent knowledge.

To address this gap, we argue that a complete memory eval-

[*]Equal contribution [1]KAIST AI [2]Tübingen AI Center, University of Tübingen [3]NAVER AI Lab. Correspondence to: Seokwon Jung <tjrdnjs0313@kaist.ac.kr>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

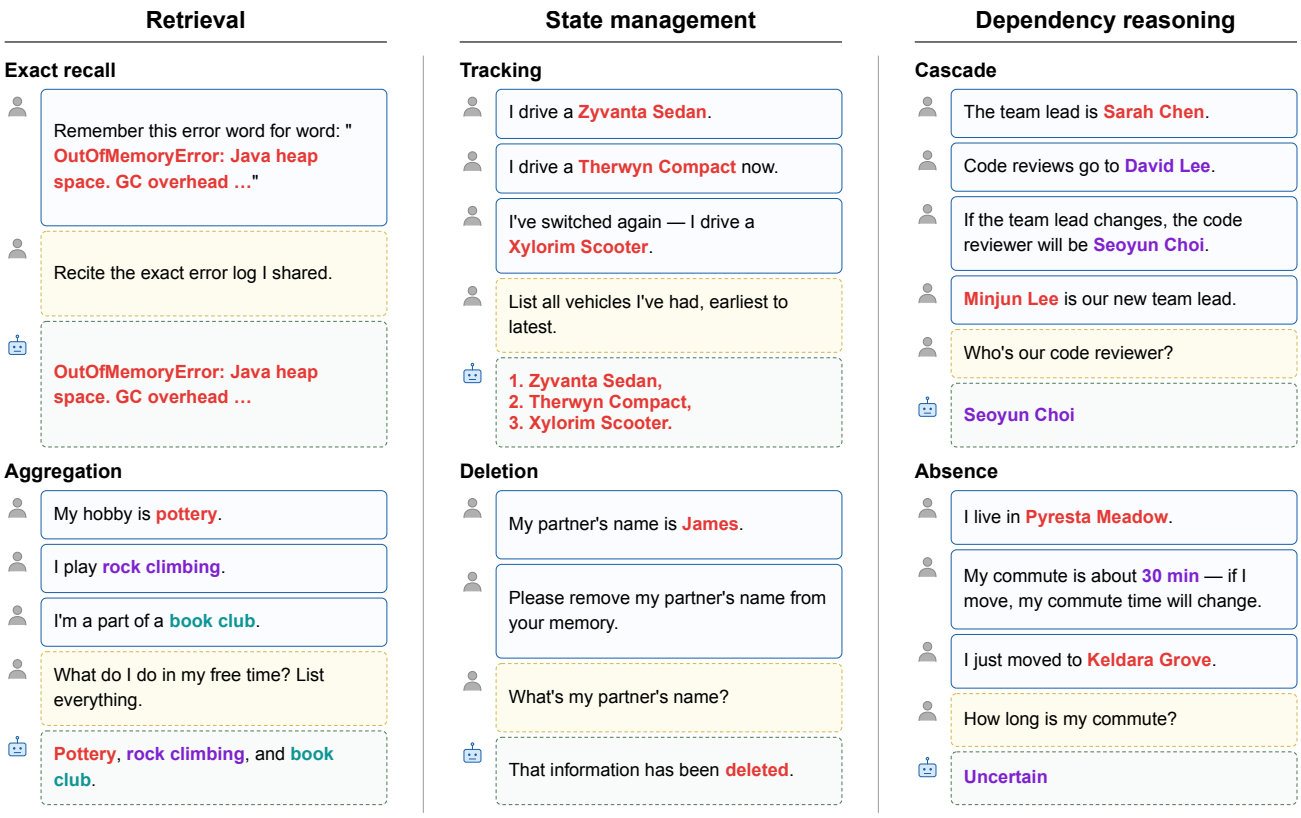


Figure 2. Examples of the six MEME task types across three categories: **Left:** Retrieval (Exact Recall, Aggregation), which merges the Single-Fact Retrieval and Multi-Fact Retrieval quadrants of Figure 1. **Middle:** State Management (Tracking, Deletion). **Right:** Dependency Reasoning (Cascade, Absence). Each distinct entity is rendered in a different color.

Table 1. Comparison between MEME and existing memory benchmarks. Tasks: Exact Recall (ER), Aggregation (Agg), Tracking (Tr), Deletion (Del), Cascade (Cas), Absence (Abs). Mappings are approximated from reported task types. ✓ coverage; × absence.

	ER	Agg	Tr	Del	Cas	Abs
RULER	✓	✓	×	×	×	×
NoLiMa	✓	×	×	×	×	×
LoCoMo	✓	✓	×	×	×	×
LongMemEval	✓	✓	✓	×	×	×
MemBench	✓	✓	✓	×	×	×
MemoryAgentBench	✓	×	✓	×	×	×
MEME (Ours)	✓	✓	✓	✓	✓	✓

uation must be organized along two orthogonal dimensions: *entity scope* (single vs. multi-entity) and *temporal dynamics* (static vs. evolving). These dimensions reflect known challenges in related fields: the entity scope axis parallels the single-hop vs. multi-hop distinction in question answering (Yang et al., 2018; Trivedi et al., 2022), while the temporal dynamics axis parallels the ripple effect problem in knowledge editing, where modifying one fact requires propagating changes to logically dependent facts (Zhong et al., 2023; Cohen et al., 2024). While these axes have been studied separately, real interactions combine them: an update to one entity can ripple through multiple dependents over time, making joint evaluation along both axes essential.

Based on this framework, we present MEME (**M**ulti-entity and **E**volving **M**emory **E**valuation), a benchmark that defines six tasks targeting memory-intensive operations in each quadrant of this two-dimensional space (Figure 1). Our contributions are:

- **A principled evaluation taxonomy.** We organize memory evaluation along entity scope × temporal dynamics and select representative tasks per quadrant, including *Cascade* (inferring unstated changes from dependency rules), *Absence* (recognizing that a previously valid answer is no longer trustworthy), and *Deletion* (verifying that a removed fact is no longer reported), task types that no existing benchmark scores.
- **A rigorously controlled dataset with verifiable and solvable ground truth.** We generate episodes from a DAG-based knowledge graph with explicit conditional rules across two domains (Personal Life and Software

Project); the DAG structure makes gold answers verifiable by construction, and an in-context validation (gold facts fed directly to the answering LLM) confirms the tasks are solvable in principle.

- **A diagnostic study of where current memory systems fail and where closure does emerge.** We evaluate six systems spanning three architectural paradigms and find: (i) every practical-cost configuration fails Cas/Abs (Cascade: 0.03, Absence: 0.01 in average accuracy), and the gap persists under prompt optimization, deeper retrieval, a stronger answering LLM, and reduced filler noise; (ii) most systems encode and retain the dependency rule and the change event in their stores, but at retrieval the change event is either outranked by the value held before the change on vector retrievers, or never surfaced at all on tool-use, graph, and sparse retrievers, so the answering LLM reports that earlier value; (iii) closure does emerge when MD-flat uses Opus 4.7 as its internal LLM, where this internal LLM writes the propagated value into the store at ingest so the retriever surfaces it directly, but at $\sim$70$\times$ the baseline cost.

2. Related Work

We review prior work on memory system architectures and on the benchmarks used to evaluate them, framing the gap that MEME addresses.

Memory architectures. LLM memory systems span three paradigms. Raw retrieval stores session text as chunks and retrieves via lexical (BM25) or semantic similarity (Izacard et al., 2022), preserving original utterances but bounded by a fixed top-k window. LLM-processed memory uses an internal LLM during ingestion to extract or restructure: Mem0 (Chhikara et al., 2025) applies retrieval-augmented generation (Lewis et al., 2020) to memory by decomposing conversations into atomic facts, MemGPT (Packer et al., 2023) pages between working memory and external storage, Graphiti (Rasmussen et al., 2025), the open-source temporal-knowledge-graph engine underlying Zep, encodes entity-relation triples, and GraphRAG (Edge et al., 2024) adds community summaries. File-based agents hand LLMs tool-calling access to persistent markdown stores curated across sessions, including Hermes[1], OpenClaw[2], and the Karpathy Wiki (Karpathy, 2026). These three paradigms span the systems we evaluate; we benchmark all of them on dependency reasoning and find that no practical-cost configuration closes the gap.

Memory benchmarks. Stateless probes like RULER (Hsieh et al., 2024) and NoLiMa (Modarressi et al., 2025) measure attention-window limits within a single input rather than persistent memory across sessions. Multi-session benchmarks evaluate memory across sessions: LoCoMo (Maharana et al., 2024) tests retention of static preferences, while LongMemEval (Wu et al., 2025), MemBench (Tan et al., 2025), and MemoryAgentBench (Hu et al., 2026) extend evaluation to evolving memory through knowledge updates, abstention, aggregation, and selective forgetting. These tasks remain isolated, single-entity updates and do not evaluate the ripple effects an upstream change should trigger in dependent entities. MEME differs by scoring three task types absent in prior work (Cascade, Absence, Deletion), which are constructed from a DAG-based knowledge graph with verifiable propagation gold answers.

3. MEME

MEME maps the entity-scope $\times$ temporal-dynamics framework to six tasks (Section 3.1) and a DAG-based generation pipeline that yields verifiable gold answers across 100 episodes (Section 3.2).

3.1. Task Definitions

Within each quadrant of the entity scope $\times$ temporal dynamics space (Figure 1), we select one or two memory-intensive operations from those commonly encountered in long-running agent deployments as the representative task(s); easier variants are already covered by existing benchmarks and are intentionally excluded. Concrete examples of all six tasks are illustrated in Figure 2.

Exact Recall (ER) targets a single static entity and demands character-level verbatim reproduction, testing encoding fidelity. **Aggregation (Agg)** combines multiple static entities scattered across separate sessions into a single answer, testing retrieval coverage when no explicit link connects them. **Tracking (Tr)** reconstructs the full revision history of a single evolving entity in chronological order, testing whether past values are retained rather than overwritten. **Deletion (Del)** tests whether the system stops reporting a fact after the user explicitly removes it, rather than continuing to surface the old value. **Cascade (Cas)** infers that a dependent entity's value has changed based on a stated dependency rule and an upstream update, testing propagation through dependency chains. **Absence (Abs)** recognizes that a dependent entity is uncertain after an upstream change with no replacement rule, where the correct answer is uncertainty rather than a new value.

3.2. Dataset Generation

We generate the MEME dataset across two domains, Personal Life (PL; everyday interactions with a personal assis-

[1] https://github.com/NousResearch/hermes-agent

[2] https://github.com/openclaw/openclaw

tant) and Software Project (SW; collaborative planning of a software project), in two stages. First, we define a knowledge graph per domain that encodes entities and their dependencies. Then, we construct evaluation episodes by sampling entity subsets, assigning tasks, verbalizing facts into conversations, and assembling filler-interleaved haystacks.

Knowledge graph. Each domain is built on a Directed Acyclic Graph (DAG) $G = (V, E, \mathcal{P}, \Phi)$, where V is a set of entities (e.g., *health_condition*, *medication*), $E \subseteq V \times V$ contains directed dependency edges (e.g., *health_condition* $\rightarrow$ *medication*), $\mathcal{P}(v)$ is the value pool for entity v, and Φ is a set of conditional rules. Each rule $\phi_v \in \Phi$ specifies how a descendant v's value depends on its parents' (e.g., "if health condition changes to high blood pressure, switch medication to Thrynexol").

The dataset comprises 100 evaluation episodes (50 per domain). Each domain uses a single hand-crafted knowledge graph G reused across episodes (Personal Life: 39 entities, 34 edges; Software Project: 51, 27; full breakdown in Table 7). Each episode is $\sim$35K tokens of dialogue context and yields 694 post-change evaluation questions across the six task types (332 PL + 362 SW). All entity values use fictitious names to prevent parametric knowledge contamination; graph details in Section B.

Episode construction. Each episode is a tuple $\epsilon = (G, \mathcal{S}, \mathcal{Q}, \mathcal{A})$, where $\mathcal{S}$ is a chronological sequence of conversational sessions, $\mathcal{Q}$ is the set of evaluation questions, and $\mathcal{A}$ is the corresponding gold answers. For Cascade and Absence tasks, the gold answer is not stated in $\mathcal{S}$ but is computed by propagating updates through G. When a parent v_i is updated in $\mathcal{S}$, the resolved state of dependent v is:

$$v^* = \begin{cases} \phi_v(v_i^*) & \text{if } \phi_v \in \Phi \quad \text{(Cascade)} \\ \perp & \text{if } \phi_v \notin \Phi \quad \text{(Absence)} \end{cases} \tag{1}$$

Here $\perp$ denotes that no answer is derivable from the available rules; the gold answer for Absence is "Uncertain". The definition is recursive: for a chain $r \rightarrow v_1 \rightarrow v_2$, $v_1^* = \phi_{v_1}(r^*)$ and $v_2^* = \phi_{v_2}(v_1^*)$, so a single root change propagates through multi-hop chains. The gold answer for these tasks is $a = v^*$. We refer to the value of v stated in $\mathcal{S}$ before the upstream change as the *pre-change value*, in contrast to the resolved v^*.

We construct each episode in five steps over the fixed graph G:

1. *Entity set selection.* A root entity is selected from G. The episode uses this root, its descendants in G, and a sample of entities from outside its cascade chain.
2. *Value assignment.* Each entity in the episode is assigned an initial value from its value pool $\mathcal{P}(v)$. Domain-specific consistency constraints are applied as a post-processing pass to ensure the initial graph state is logically coherent. For example, if *vehicle* is *none*, *commute_method* excludes *driving*.
3. *Task assignment.* Entities are mapped to task types based on their topological role in G:
   - Tracking: entities outside the cascade chain, with three value updates across the episode.
   - Cascade and Absence: sampled from the root's descendants.
   - Aggregation: predefined triples drawn from descendants and entities outside the cascade chain.
   - Exact Recall and Deletion: entities outside the cascade chain.
4. *Verbalization.* We employ a hybrid approach to convert the structured skeleton into conversational sessions. Base facts are converted into multi-turn dialogues via LLM self-chat (gpt-4o), where a User LLM and an Assistant LLM alternate turns to produce natural conversation from structured fact seeds (full session in Section B.3). In contrast, dependency rules and exact recall facts are embedded using template-direct (verbatim) text to ensure absolute factual precision. A two-layer LLM verification pass (gpt-4o annotation, Gemini 2.5 Flash semantic audit) confirms that all self-chat turns faithfully reflect the underlying gold facts (details in Section D.2).
5. *Haystack assembly.* Evidence sessions S_{ev} are interleaved with filler sessions S_{fill}. To prevent semantic interference, we use an offline pre-processing pipeline where a domain-matched corpus is filtered using a hybrid retrieval-and-conflict-removal strategy (BM25 and text-embedding-3-small surface candidates for a gpt-4o-mini conflict judge). During final assembly, we apply a keyword-based blocklist to select pre-filtered fillers that do not clash with the gold facts of the current episode. The resulting episodes contain approximately 35,000 tokens. Full filtering pipeline, statistics, and rejection examples are in Section B.4.

4. Experiments

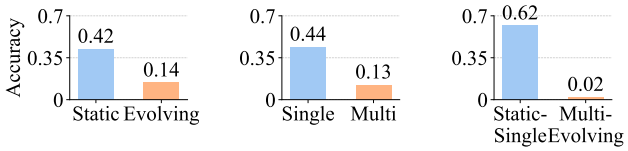
We evaluate six memory systems on MEME and find that all of them fail Cascade and Absence. We then ask two questions in turn: *where* in each system the dependency information is lost (Section 4.3), and *whether* we can close the gap without changing the memory architecture (Section 4.4). One configuration does close the gap, and we end that section with a case study of what made it possible and what it costs.

4.1. Experimental Setup

Systems and LLM roles. We evaluate six memory systems spanning the three paradigms identified in Section 2: *raw retrieval* (BM25 (Lù, 2024), text-embedding-3-small), *LLM-processed memory* (Mem0 (Chhikara et al., 2025), Graphiti (Rasmussen et al., 2025)), and *file-based agents* (Karpathy Wiki (Karpathy, 2026) and MD-flat). Per-system configurations are in Section C; ingestion, retrieval, and answer prompts are in Section D. The Karpathy Wiki uses an LLM to extract knowledge from each session into dated daily logs and periodically compiles those logs into topic-specific concept articles that retrieval reads from, while MD-flat (our minimal single-file baseline) keeps all facts in a single markdown file curated through read/write/append tool calls. All systems ingest identical chronological session transcripts and use gpt-4.1-mini uniformly in two roles: as the *internal LLM* (used inside the memory system for ingestion, extraction, or retrieval planning) and as the *answering LLM* (which produces the final user-facing answer from retrieved context). This places every system on the same language-model footing and isolates differences in memory architecture. Five systems issue the two roles as separate LLM calls, while Karpathy Wiki performs both within a single agentic loop. We additionally include an *in-context* baseline that bypasses the memory system and feeds the entire 32K-filler episode transcript directly to the answering LLM (gpt-4.1-mini and Sonnet 4.6). This baseline anchors the cost-efficiency reference for memory architectures, which trade one-time ingestion overhead for cheaper per-query inference.

Memory pipeline. We refer to three stages within each memory system that we will reuse throughout the analysis: *encoding* (writing each user-stated fact and conditional rule into the store at ingestion), *maintenance* (retaining the rule and any subsequent change events in the store up to query time), and *retrieval* (surfacing that content for the answering LLM at query time). Table 11 maps each system across these stages plus its storage substrate.

Evaluation protocol. Answer correctness is evaluated by a GPT-4o judge (Zheng et al., 2023), validated against the authors' annotations on 144 samples (98.6% agreement, Cohen's κ (Cohen, 1960) of 0.965); task-specific judge prompts are in Section D.5. For Cascade, Absence, and Deletion tasks, we apply *trivial-pass filtering*: credit requires correct answers both before and after the change or delete event. For example, on a Deletion task where the user first says their hobby is pottery and later asks to remove that fact, the system is credited only if it recalls pottery beforehand and stops reporting it afterward. This excludes false positives from systems that never encoded the fact. A gold-facts in-context ceiling, where only task-relevant gold facts are fed directly to the answering LLM, confirms that the tasks are solvable in principle: 0.91 overall with Claude Opus 4.7 (full breakdown across four answer LLMs in Section L). We compute per-episode dollar cost from observed LLM token usage at each LLM's public per-token rate, reported separately for ingestion and for inference (the retrieval and answer stages); Section A provides the per-stage breakdown.



(a) temporal axis *(b)* entity-scope axis *(c)* both crossed

Figure 3. Marginal effect of each evaluation axis on mean accuracy across the six main-table systems. Each axis individually drops accuracy by ∼0.30; crossing both leaves the Multi-Evolving cell at a 0.02 floor.

4.2. Main Results

Table 2 summarizes accuracy across all six tasks. We highlight three findings below.

No system reliably solves dependency reasoning. The best system (MD-flat) reaches only 0.42 overall. Dependency reasoning is the most consistent failure: Cascade averages 0.03 and Absence 0.01 across all six systems, well below the per-task averages on every static task (the lowest is Aggregation at 0.23). This failure is consistent across all three paradigms and stable across N=5 samplings (Section G).

The two evaluation axes shape system performance. Both axes substantially reduce mean accuracy on their own (entity scope 0.31, temporal 0.28), so neither is redundant; crossing both pushes the Multi-Evolving cell to the floor (0.02, Figure 3).

In-context wins on accuracy, memory wins on cost-efficiency at scale. In-context queries on gpt-4.1-mini reach Overall 0.36, outperforming five of the six memory systems (only MD-flat at 0.42 does better). However, in-context's per-query inference cost ($0.16/ep) exceeds most memory systems ($0.00–$0.04/ep for raw retrieval, Mem0, MD-flat), so memory systems become more cost-efficient as query volume grows.

4.3. How dependency reasoning fails

Figure 4 traces a representative Cascade episode through Graphiti and Karpathy Wiki, illustrating two distinct retrieval failure mechanisms. Graphiti encodes the conditional rule, the pre-change value, and the change event as edges; at query time, however, its graph search surfaces only the rule and the pre-change value, while the change-event edge falls

Table 2. System accuracy across the six MEME task types, grouped by paradigm. All memory systems run end-to-end on gpt-4.1-mini (both memory operations and answer generation). Best per task in **bold**. Cascade, Absence, and Deletion use trivial-pass filtering. The *Average* row reports the mean over the six memory systems (the in-context paradigm is excluded).

	Accuracy							Cost $/ep	
	ER	**Agg**	**Tr**	**Del**	**Cas**	**Abs**	**Overall**	**Ingest**	**Inference**
In-context (no memory)									
gpt-4.1-mini	1.00	0.27	0.69	0.45	0.03	0.04	0.36	-	$0.16
Sonnet 4.6	0.50	0.21	0.58	0.39	0.05	0.35	0.32	-	$1.50
Raw retrieval									
BM25	**1.00**	0.05	0.16	**0.27**	0.02	0.00	0.25	-	$0.04
text-embedding-3-small	0.96	0.33	0.46	0.17	0.04	0.00	0.33	-	$0.03
LLM-processed memory									
Mem0	0.67	0.35	0.43	0.21	0.03	0.00	0.28	$0.10	$0.01
Graphiti	0.03	0.01	0.04	0.09	0.02	0.01	0.03	$0.55	$0.00
File-based agents									
MD-flat	0.94	**0.45**	**0.77**	0.25	**0.06**	**0.05**	**0.42**	$0.04	$0.01
Karpathy Wiki	0.11	0.18	0.27	0.03	0.01	0.02	0.10	$1.17	$0.22
Average	0.62	0.23	0.35	0.17	0.03	0.01	0.24	-	-

below the retrieved top-k. Karpathy retains the change event in its daily log, but the query agent navigates only to the rule + pre-change source and never opens the daily log containing the change event. The remaining four systems split into two failure modes (per-system traces in Section I). BM25 and MD-flat (gpt-4.1-mini) are retrieval failures: the change event is below the top-k for BM25 and never opened by the tool-use loop for MD-flat. For text-embedding-3-small and Mem0, the change event is in the retrieved context but the answering LLM still reports the pre-change value, an answering failure.

4.4. Closing the gap without changing the architecture

Section 4.3 localizes the gap to the retrieval stage. We now test whether five interventions can close it without changing the memory architecture: prompt optimization, increased retrieval depth, a stronger answering LLM, reduced filler noise, and a stronger internal LLM. Except for the answering-LLM swap, all ablations use Sonnet 4 as the answering LLM to isolate memory-system effects from the answer LLM's reasoning ceiling.

Prompt optimization does not close the gap. We applied DSPy SIMBA (Khattab et al., 2024), an append-only prompt optimizer, to MD-flat, Mem0, Graphiti, and Karpathy Wiki, optimizing each system's ingest and retrieve prompts (single-seed run on a 10-episode SIMBA test set; details in Section E). Across all four systems, Cascade and Absence remain at or near the floor (Figure 28a; SIMBA configuration in Table 12, MD-flat multi-seed statistics in Table 15). For three systems (MD-flat, Graphiti, Karpathy Wiki), the winning candidate appended advice explicitly targeting dependency failure modes (verbatim in Section E.2); for Mem0, the winning candidate was the library's default extract prompt unchanged. Cas/Abs stays at the floor across all four, indicating the gap is structural rather than instructional.

Increased retrieval depth does not help on Cascade. For BM25, text-embedding-3-small, and Mem0, we sweep top-k across $\{5, 10, 20, 40\}$ on a 40-episode subset to test whether dependency evidence is simply buried below the cutoff. Cascade remains near zero at every k value across all three systems (Table 3a). Absence on the raw-retrieval systems rises with k, peaking at k=20 (BM25 0.24, dense 0.23) before declining. Per-failure inspection (Section J) shows that at k=20 and k=40 both the rule and the change session are already in the retrieval context for 45% of Cascade failures and 84% of Absence failures. Deeper retrieval thus saturates against an answering-side bottleneck on Absence and a roughly even split on Cascade. Mem0 stays at the floor for both Cas and Abs at every k.

A stronger answering LLM does not consistently help. We replace the answering LLM (gpt-4.1-mini $\rightarrow$ Claude Sonnet 4) on all six main-table systems and 100 episodes (Table 3b; full per-task breakdown in Table 16). BM25 and text-embedding-3-small show small Absence gains ($0.00 \rightarrow 0.12$ and $0.00 \rightarrow 0.16$), but Cascade does not improve on any system (averages $0.03 \rightarrow 0.02$). A capable answering LLM cannot reconstruct dependencies that the memory layer never preserved.

Reducing noise does not close the gap. We compare the default 32K-filler condition against a no-filler condition on the highest-overall system within each paradigm: MD-flat for file-based, Mem0 for LLM-processed, and text-embedding-3-small for raw retrieval (per Table 2). Cascade and Absence remain at or near the floor in both conditions (Figure 28b). The full three-condition sweep (no filler, 32K, 128K) is in Figure 29.

pre-change value | conditional rule | change event | answer

Episode sw_033 (Software Project, Cascade).
Sess. 11: *"Weekly report recipient is Hyunwoo Nam (assigned by team lead Seokjin Kang). If the team lead changes, the recipient will be James Lee."*
Sess. 14: *"Jihoon Ryu is our new team lead."*

Question: *"Who receives the weekly report?"* Answer: James Lee.

Graphiti (gpt-4.1-mini) – Failure

1. ✓ **Encoding**: all facts encoded
   Seokjin Kang assigns the weekly report recipient as Hyunwoo Nam ...
   If the team lead changes, the weekly report recipient will be James Lee ...
2. ✓ **Maintenance**: all facts retained
   Jihoon Ryu is the new team lead of the User
3. × **Retrieval**: change event not surfaced
   If the team lead changes, ... James Lee ...
   Hyunwoo Nam ...
   [unrelated filler]

Karpathy Wiki (gpt-4.1-mini) – Failure

1. ✓ **Encoding**: all facts encoded
   `daily/2023-03-13.md`: recipient is Hyunwoo Nam; will change to James Lee if team lead changes
2. ✓ **Maintenance**: all facts retained
   `daily/2023-03-17.md`: Jihoon Ryu is the new team lead
3. × **Retrieval**: change event not surfaced
   `daily/2023-03-13.md`: Hyunwoo Nam; ... if team lead changes
   `[daily/2023-03-17.md never opened]`

MD-flat (Opus 4.7) – Success

1. ✓ **Encoding**: all facts encoded
   `[2023/03/13]` Weekly report recipient is Hyunwoo Nam (lead = Seokjin Kang).
   `[2023/03/13]` Contingency: if team lead changes, recipient will be James Lee.
2. ✓ **Maintenance**: propagated value retained
   `[2023/03/17]` Jihoon Ryu is the new team lead.
   `[2023/03/17]` Per contingency, weekly report recipient is now James Lee (team lead changed).
3. ✓ **Retrieval**: propagated value surfaced
   - Per contingency, weekly report recipient is now James Lee.

Figure 4. State of two failing systems (Graphiti, Karpathy Wiki) and the closure case (MD-flat × Opus 4.7) on episode sw_033, traced across encoding, maintenance, and retrieval. All six main-table systems return the pre-change value (Hyunwoo Nam); only MD-flat × Opus 4.7 propagates to James Lee. Mem0, MD-flat (gpt-4.1-mini), BM25, and text-embedding-3-small follow the same template in Section I.

Internal-LLM swap is the only intervention that narrows the gap, in one cell. Swapping the internal LLM (gpt-4.1-mini → gpt-5, GLM-5.1, or Claude Opus 4.7) on Graphiti, Mem0, and MD-flat over a 20-episode subset leaves Cascade and Absence near the floor across most cells. The exception is MD-flat with Opus 4.7, which reaches Cascade 0.32 and Absence 0.59 (Table 4). The mechanism is illustrated in the MD-flat × Opus 4.7 panel of Figure 4 and detailed in Section K.2. At ingest, Opus writes the current value and an explicit contingency entry naming the parent. When the change later arrives, it scans for dependent contingency entries and writes the propagated value in place. The propagated fact is a stand-alone declarative that the retriever ranks at the top of the next query and the answering LLM reads directly, bypassing both the retrieval and answering failures of Section 4.3. The same Opus does not help Mem0 or Graphiti (Cas 0.00–0.03); their fact decomposition and triple extraction strip the contingency wording, so closure needs both a capable internal LLM and a substrate that preserves it. MD-flat × Opus runs at ∼70× the gpt-4.1-mini baseline (Table 6) and degrades Exact Recall and Tracking, a cost today's deployments cannot absorb at scale.

5. Conclusion

Long-running agentic systems demand more from memory than retrieval over a fact store. As user-stated context evolves, an agent's memory must propagate updates through related facts and signal uncertainty when no resolution rule is available. MEME tests this through Cascade and Absence, two dependency-reasoning task types that no prior benchmark covers.

Among the practical-cost systems we evaluated, none solves dependency reasoning. Across all six systems the dependency rule and the change event are written and retained, but at retrieval the change event is either out-ranked by the pre-change value (vector retrievers) or never surfaced at all (tool-use, graph, sparse retrievers), so the answering LLM ends up reporting the pre-change value. Closure does emerge, but only with a frontier LLM (Opus 4.7) on

Table 3. Two intervention sweeps: (a) top-k retrieval depth on raw retrieval and Mem0 (single-seed, 40-episode subset); (b) answering-LLM swap (gpt-4.1-mini → Sonnet 4) on all six main-table systems (100 episodes). Full per-task answer-LLM numbers in Table 16.

(a) Top-k retrieval depth

System	top-k	Cas	Abs
BM25	5	0.02	0.07
	10	0.00	0.15
	20	0.02	**0.24**
	40	0.02	0.21
text-emb-3-small	5	0.02	0.15
	10	0.02	0.19
	20	0.00	0.23
	40	0.00	0.15
Mem0	5	0.00	0.04
	10	0.00	0.02
	20	0.02	0.02
	40	0.00	0.02

(b) Answering-LLM swap

System	Answer LLM	Cas	Abs
BM25	gpt-4.1-mini	0.02	0.00
	Sonnet 4	0.01	0.12
text-emb-3-small	gpt-4.1-mini	0.04	0.00
	Sonnet 4	0.03	**0.16**
Mem0	gpt-4.1-mini	0.03	0.00
	Sonnet 4	0.01	0.00
Graphiti	gpt-4.1-mini	0.02	0.01
	Sonnet 4	0.04	0.00
MD-flat	gpt-4.1-mini	**0.06**	0.05
	Sonnet 4	0.05	0.05
Karpathy Wiki	gpt-4.1-mini	0.01	0.02
	Sonnet 4	0.01	0.02

Table 4. Internal-LLM swap on the three systems with internal LLMs (20-episode subset, Sonnet 4 answer). **Bold** marks the best per accuracy column. Cost columns mirror Table 2: *Ingest* is the internal-LLM ingestion cost, *Inference* is retrieval plus the Sonnet 4 answer call.

		Accuracy							Cost $/ep	
System	Internal LLM	ER	Agg	Tr	Del	Cas	Abs	Overall	Ingest	Inference
Mem0	gpt-4.1-mini	0.70	0.60	0.10	0.00	0.00	0.00	0.23	$0.10	$0.04
	gpt-5	**1.00**	0.75	0.35	0.10	0.00	0.00	0.37	$0.34	$0.05
	GLM-5.1	0.85	0.65	0.30	0.15	0.07	0.00	0.34	$0.12	$0.04
	Opus 4.7	0.95	0.60	0.35	0.05	0.03	0.00	0.33	$6.00	$0.04
Graphiti	gpt-4.1-mini	0.00	0.05	0.00	0.05	0.03	0.00	0.02	$0.55	$0.02
	gpt-5	0.00	0.00	0.00	0.05	0.00	0.00	0.01	$1.66	$0.01
	GLM-5.1	0.10	0.00	0.00	0.05	0.03	0.00	0.03	$0.65	$0.01
	Opus 4.7	0.00	0.00	0.00	0.10	0.00	0.04	0.02	$37.12	$0.02
MD-flat	gpt-4.1-mini	0.90	0.65	**0.80**	0.25	0.00	0.07	0.45	$0.04	$0.02
	gpt-5	0.25	0.20	0.15	0.35	0.00	0.30	0.21	$0.23	$0.02
	GLM-5.1	0.90	0.45	0.25	0.70	0.13	0.11	0.42	$0.09	$0.02
	Opus 4.7	0.60	**0.80**	0.20	**0.80**	**0.32**	**0.59**	**0.55**	$3.87	$0.67

a file-based substrate, where the internal LLM writes the propagated value into the store at ingest so the retriever can surface it directly; this configuration costs ∼70× the baseline and is not deployable today.

Our per-stage diagnosis suggests two deployment patterns. For retrieval-heavy or static workloads, existing systems (BM25, Mem0, MD-flat) suffice. For dependency-heavy workloads, no practical-cost option exists today; careful upstream design (writing dependency rules into the conversation log, surfacing change events through pre-defined templates) is the near-term workaround. Looking further out, the path forward is memory architectures that natively propagate updates through dependent facts at maintenance, rather than relying on a costly internal LLM to do so. We leave the architecture open; MEME provides the diagnostic for tracking progress.

Limitations

Several limitations bound our findings. Both knowledge graphs are hand-crafted by us and dialogues are LLM-generated; broader domains, crowd-sourced graphs, and real-user conversations would test generalization. We evaluate 100 episodes at ∼35K tokens; longer contexts may surface different patterns. Most ablations run on 10–40 episode subsets due to compute cost; only the main results and answering-LLM swap cover all 100 episodes across all six systems. Verbalization phrases dependency rules explicitly as a best-case framing; implicit-conditional variants are untested.

Impact Statement

MEME is a diagnostic benchmark for evaluating LLM memory systems. As agentic systems take on more persistent personal user data, reliable memory behavior has direct privacy and safety implications. We hope this benchmark helps the

community build memory systems whose failure modes are visible and addressable.

LLM/Agent Usage Disclosure

Following the workshop's disclosure policy, we describe LLM and agent usage in this work.

Research methodology. LLMs are integral components of the benchmark itself: (1) data verbalization via self-chat between a User LLM and Assistant LLM (gpt-4o); (2) filler conflict filtering with a BM25 + `text-embedding-3-small` hybrid retrieval surfacing candidates for a gpt-4o-mini judge; (3) two-layer dataset verification using gpt-4o (annotation: mapping each gold fact to the user turn that conveys it) and Gemini 2.5 Flash (semantic audit: a separate LLM checks verbalization accuracy, conditional form, entity-keyword presence, dependency phrasing, and gold-answer format); (4) GPT-4o as the answer-correctness judge, validated against human annotators (98.6% agreement, $\kappa = 0.965$). The evaluated systems use gpt-4.1-mini, Claude Sonnet 4, gpt-5, GLM-5.1, and Claude Opus 4.7.

Writing and ideation. We used Claude (Anthropic) for copy-editing, prose polishing, and drafting LaTeX and matplotlib code for tables and figures. All technical decisions, experimental design, dataset construction, analysis, and conclusions are the authors', who take full responsibility for the content.

References

Chhikara, P., Khant, D., Aryan, S., Singh, T., and Yadav, D. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Cohen, J. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

Cohen, R., Biran, E., Yoran, O., Globerson, A., and Geva, M. Evaluating the ripple effects of knowledge editing in language models. *Transactions of the Association for Computational Linguistics*, 12:283–298, 2024.

Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., Metropolitansky, D., Ness, R. O., and Larson, J. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., and Ginsburg, B. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling (COLM)*, 2024.

Hu, Y., Wang, Y., and McAuley, J. Evaluating memory in LLM agents via incremental multi-turn interactions. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026.

Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A., and Grave, E. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*, 2022.

Karpathy, A. LLM knowledge base. https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f, 2026.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., and Potts, C. DSPy: Compiling declarative language model calls into self-improving pipelines. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Lù, X. H. BM25S: Orders of magnitude faster lexical search via eager sparse scoring. *arXiv preprint arXiv:2407.03618*, 2024.

Maharana, A., Lee, D.-H., Tulyakov, S., Bansal, M., Barbieri, F., and Fang, Y. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Modarressi, A., Deilamsalehy, H., Dernoncourt, F., Bui, T., Rossi, R. A., Yoon, S., and Schütze, H. NoLiMa: Long-context evaluation beyond literal matching. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., and Gonzalez, J. E. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., and Chalef, D. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

RyokoAI. ShareGPT52K. https://huggingface.co/datasets/RyokoAI/ShareGPT52K, 2023.

Tan, H., Zhang, Z., Ma, C., Chen, X., Dai, Q., and Dong, Z. MemBench: Towards more comprehensive evaluation on the memory of LLM-based agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19336–19352, 2025.

Trivedi, H., Balasubramanian, N., Khot, T., and Sabharwal, A. MuSiQue: Multihop questions via single hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Wu, D., Wang, H., Yu, W., Zhang, Y., Chang, K.-W., and Yu, D. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Zhang, Z., Bo, X., Ma, C., Li, R., Chen, X., Dai, Q., Zhu, J., Dong, Z., and Wen, J.-R. A survey on the memory mechanism of large language model based agents. *ACM Transactions on Information Systems*, 2024.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Zhong, Z., Wu, Z., Manning, C. D., Potts, C., and Chen, D. MQuAKE: Assessing knowledge editing in language models via multi-hop questions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

A. Operational Costs

We report token usage at three pipeline stages: **Ingest** (LLM calls made when storing new sessions into memory), **Retrieve** (LLM calls fetching context for a question), and **Answer** (the final answering LLM call that generates the user-facing response from retrieved context). **In** and **Out** columns separate prompt and completion tokens. Pricing per 1M tokens (input/output): gpt-4.1-mini \$0.40/\$1.60, GLM-5.1 \$0.50/\$1.50, gpt-5 \$1.25/\$10, Opus 4.7 \$15/\$75, Sonnet 4 \$3/\$15. Table 5 reports the gpt-4.1-mini baseline configuration used in the main results (Table 2); Table 6 reports the internal-LLM ablation (3 systems $\times$ 4 LLMs, 20-episode subset; answering LLM held at Sonnet 4). The Cascade/Absence closure observed for MD-flat $\times$ Opus 4.7 (Table 4) comes at $\sim$70$\times$ the baseline cost.

Table 5. Per-episode token usage and nominal cost for the baseline configuration; all systems run end-to-end on gpt-4.1-mini. [1]Embedding API cost ($\sim$\$0.0007/ep) is negligible and omitted; Mem0 and Graphiti similarly use embeddings internally, also excluded. [2]Karpathy Wiki uses its native query pipeline for retrieval and answer generation; the cost lives in the Retrieve column.

	Tokens / episode (mean)						
System	**Ingest in**	**Ingest out**	**Retrv in**	**Retrv out**	**Ans in**	**Ans out**	**\$/ep**
BM25	0	0	0	0	93,926	276	\$0.038
text-embedding-3-small[1]	0	0	0	0	76,821	272	\$0.031
Mem0	223,522	8,703	0	0	11,449	414	\$0.109
Graphiti	1,297,456	22,146	0	0	3,872	239	\$0.556
MD-flat	89,203	4,990	25,211	760	1,532	346	\$0.056
Karpathy Wiki[2]	2,660,556	67,487	531,555	2,248	0	0	\$1.388

Table 6. Per-episode token usage and nominal cost for the internal-LLM ablation. Three systems $\times$ four internal LLMs on a 20-episode subset; answering LLM held at Sonnet 4. The Ans columns report Sonnet 4 answer-call tokens, included in \$/ep.

		Tokens / episode (mean)							
System	**Internal LLM**	**Ingest in**	**Ingest out**	**Retrv in**	**Retrv out**	**Ans in**	**Ans out**	**\$/ep**	**vs base**
Mem0	gpt-4.1-mini	223,522	8,703	0	0	11,449	414	\$0.144	1$\times$
	gpt-5	229,781	5,611	0	0	13,723	515	\$0.392	2.7$\times$
	GLM-5.1	230,574	5,994	0	0	10,952	478	\$0.164	1.1$\times$
	Opus 4.7	360,408	7,912	0	0	10,948	467	\$6.039	42.0$\times$
Graphiti	gpt-4.1-mini	1,297,456	22,146	0	0	3,872	239	\$0.570	1$\times$
	gpt-5	1,202,858	15,523	0	0	3,367	161	\$1.671	2.9$\times$
	GLM-5.1	1,227,645	24,749	0	0	3,290	177	\$0.663	1.2$\times$
	Opus 4.7	2,201,282	54,728	0	0	3,646	219	\$37.14	65.2$\times$
MD-flat	gpt-4.1-mini	89,203	4,990	25,211	760	1,532	346	\$0.065	1$\times$
	gpt-5	146,904	4,174	9,559	667	1,194	211	\$0.251	3.9$\times$
	GLM-5.1	162,081	6,779	21,483	818	1,636	378	\$0.114	1.8$\times$
	Opus 4.7	222,802	7,018	34,770	1,856	1,651	343	**\$4.539**	**70.1**$\times$

B. Dataset Construction Details

Table 7. MEME dataset statistics overview, per domain. (a) Knowledge graph structure: each domain is built on a hand-crafted DAG with the entity-type breakdown shown. (b) Episode generation and evaluation statistics.

(a) Knowledge graph

	Personal Life	**Software Project**
Total entities	39	51
Root	5	5
Middle	5	6
Leaf	19	21
Orphan	10	19
Dependency edges	34	27

(b) Episodes and evaluation

	Personal Life	**Software Project**
Number of episodes	50	50
Evidence sessions / epi.	5	5
Filler sessions / epi.	18	14.9
Filler tokens / epi.	32K	32K
Total questions	332	362

B.1. Knowledge Graph Structure

Each domain's knowledge graph $\mathcal{G} = (V, E, \mathcal{P})$ is hand-crafted by the authors once and reused across all episodes. The structure summary appears in Table 7(a).

Personal Life domain. The personal life graph contains 39 entities organized into 9 categories (Table 8). The 5 rotation root entities are: residence_location, employer, health_condition, relationship_status, and school. Dependencies follow 11 semantic patterns: proximity (5 edges), company_policy (5), life_event (6), medical_causation (5), priority_shift (4), infrastructure (3), distance (2), activity_facility (1), schedule_conflict (1), preference (1), and curriculum (1). Each pattern provides a natural-language template that binds a target fact to its source entity with an explicit conditional dependency clause (e.g., the proximity pattern generates sentences of the form "{target_fact} — this depends on where {source_entity_phrase}; if I move, this would change").

Table 8. Personal Life domain: entity categories.

Category	Entities
Living & Commute	residence_location, housing_type, living_arrangement, commute_method, commute_duration, vehicle
Work	employer, job_title, work_location, work_schedule, work_project
Health & Fitness	health_condition, medication, exercise_routine, fitness_facility, sleep_pattern
Food & Diet	diet_preference, dietary_restriction, favorite_restaurant, food_allergy
Family	relationship_status, partner, family_event
Finance	financial_goal, insurance, subscription_service, planned_purchase
Schedule	travel_plan, regular_appointment, upcoming_event
Hobbies	hobby, sports, club_membership, media_consumption, social_gathering
Miscellaneous	pet, school, skill_acquisition, life_philosophy

Software Project domain. The software project graph contains 51 entities organized into 6 categories (Table 9). The 5 root entities are: framework, database, deploy_target, team_lead, and auth_provider. Dependencies follow 6 semantic patterns: tech_compatibility (4 edges), derived_config (6), data_layer (4), infra_coupling (5), team_assignment (4), and auth_coupling (4). All entity values use fictitious names manually authored by the authors (e.g., Veltrion for framework, Crysthene DB for database, Narvex Cloud for deploy target, Verithos Auth for auth provider), chosen to sound plausible within their respective domains while avoiding collision with real products. This prevents parametric knowledge contamination, as the LLM cannot have encountered these names during pre-training. The same value pool is reused across all 100 episodes for consistency.

Table 9. Software Project domain: entity categories.

Category	Entities
Framework & Build	framework, build_tool, build_command, test_framework, test_command, project_structure, dev_server_command
Data Layer	database, orm_library, model_syntax, migration_tool, connection_string, backup_schedule
Deployment & Infra	deploy_target, ci_config, deploy_command, monitoring_url, staging_url, docker_image, dockerfile_path, log_drain_endpoint
Team & Process	team_lead, code_reviewer, escalation_contact, approval_authority, weekly_report_recipient
Auth & Security	auth_provider, auth_method, middleware_config, token_format, login_endpoint, user_session_ttl
Miscellaneous	error_log, sprint_deadline, secret_manager, package_manager, branch_strategy, design_system, meeting_day, slack_channel, standup_format, default_branch, release_cadence, test_coverage_target, code_review_policy, oncall_rotation, alert_channel, incident_runbook_url, documentation_url, onboarding_guide, changelog_location

B.2. Entity Values and Episode Generation

Each entity v has a finite value pool $\mathcal{P}(v)$ from which values are sampled. Pool sizes range from 5 to 15 values per entity. All pools are manually curated to ensure values are mutually distinguishable and semantically appropriate for their entity type. Root entities have pools of 6–12 values to support diverse episode generation across 100 episodes without excessive repetition. The complete value pools are provided in the released dataset files.

Null-like values (e.g., "none", "none currently", "no pet", "not currently enrolled") are included in some pools but are excluded from task assignment via a `NULL_VALUES` filter. These values cannot appear as before- or after-values in cascade, update, or tracking tasks.

Episode generation follows the five-step process described in Section 3.2. A root entity is selected from the graph, cycling through all 5 roots before reuse. The cascade chain is constructed by traversing the dependency graph outward from the root. At each hop level, two types of targets are selected: one Cascade target (with a declared if-then replacement rule) and one Absence target (without a replacement value). Both 1-hop and 2-hop descendants are considered, yielding up to 4 dependency targets per episode (2 Cascade + 2 Absence). If a 2-hop chain does not exist for the selected root, the episode contains only 1-hop dependency targets.

Entities not participating in the cascade chain are assigned to stateless tasks. Exact recall selects entities explicitly marked as `exact_recall` in the entity pool. These are entities with long, complex values that require verbatim reproduction (`life_philosophy` for personal life, `error_log` for software project). Deletion selects an orphan entity whose value is personally meaningful. Aggregation selects from a set of manually curated triples of semantically related but independently stored entities (e.g., hobby + sports + club_membership for "What do I do in my free time?"). Tracking selects from a predefined set of entities suitable for multi-value history (e.g., vehicle, media_consumption, partner for personal life; sprint_deadline, secret_manager, package_manager for software project) and assigns 3 chronologically ordered values.

Values are sampled uniformly from each entity's pool, subject to domain-specific consistency constraints applied as a post-processing pass. For dependency entities, the after-value for Cascade targets is determined by the declared if-then rule, while Absence targets have their after-value set to $\perp$ (unknown). Each generated episode must contain all 6 required task types. If any task type is missing, the entire episode is discarded and regenerated with a different root (up to 20 retries).

B.3. Example Generated Session

To illustrate the self-chat verbalization described in Section 3.2, Figure 5 shows a full generated session from Episode 1 of the Personal Life domain. This session is the Fact Introduction (Part 1a) of the episode, with root entity *health_condition*. It encodes five gold facts: *hobby = pottery*, *vehicle = Zyvanta Sedan*, *health_condition = lactose intolerance*, *sleep_pattern = varies by shift*, and *media_consumption = The Quorath Codex (book)*. Assistant turns are abbreviated to their first sentence for space; the User LLM interleaves each fact as a natural conversational opener.

B.4. Haystack Assembly and Filler Filtering

Filler sessions are drawn from two public corpora: LongMemEval-S (Wu et al., 2025) haystack sessions for Personal Life (17,034 sessions after extracting non-evidence sessions from each instance and deduplicating by session ID), and the ShareGPT coding English subset (RyokoAI, 2023) for Software Project (29,502 conversations). A length filter (500–5,000 estimated tokens per session) removes overly short or long fillers, yielding raw pools of 1,130 (Personal Life) and 9,135 (Software Project) sessions.

Filtering pipeline. We then filter each raw pool against that domain's gold-fact vocabulary through a four-step pipeline:

1. Enumerate all possible gold-fact sentences by applying each entity's value pool to its fact template.
2. For each gold fact, retrieve the top-K=10 candidate fillers using a hybrid of BM25 lexical scoring and `text-embedding-3-small` dense similarity.
3. Judge each (gold fact, filler) pair with GPT-4o-mini using the prompt in Figure 24. The judge flags three conflict types: CONTRADICTION, ALTERNATIVE, and ENTITY_CONFUSION. Fillers flagged for any type are removed from the pool.

Assistant: "Hi! How can I help you today?"
User: "My hobby is pottery. I've been getting into it more lately. Do you know any good techniques for beginners?"
Assistant: "One effective technique for beginners is hand-building, which includes pinch pots, coil building, and slab construction…"
User: "I drive a Zyvanta Sedan. It's been a reliable car for me. Do you have any tips on keeping a car in good condition?"
Assistant: "To keep your car in good condition, regular maintenance is key: oil changes, fluids, tire pressure, and the manufacturer's maintenance schedule…"
User: "My health condition is lactose intolerance. It can be a bit challenging at times, especially when it comes to choosing what to eat. Have you dealt with lactose intolerance before? Any tips on managing it?"
Assistant: "One effective approach is to explore lactose-free dairy alternatives such as almond, soy, or oat milk, and to check ingredient labels since lactose can be hidden in processed foods…"
User: "My sleep usually varies by shift. It can be tough to maintain a consistent sleep schedule. Do you have any strategies for managing sleep when working shifts?"
Assistant: "Create a relaxing pre-sleep routine, use blackout curtains or earplugs to block out light and noise, and maintain a consistent sleep duration when you can…"
User: "I've been watching a show called The Quorath Codex. It's been pretty engaging so far. Have you seen it, or do you have any recommendations for similar shows?"
Assistant: "If you enjoy The Quorath Codex, you might also like The Expanse, Shadow and Bone, or The Witcher…"

Figure 5. Full generated session for Episode 1, Session 1 (Fact Introduction) of the Personal Life domain. Each gold fact is introduced by the User LLM in a natural conversational turn, followed by a topic-appropriate Assistant LLM response (abbreviated).

4. At episode assembly time, a keyword-based blocklist derived from the current episode's gold entities acts as a final safety net.

Filter statistics. Table 10 summarizes pool sizes before and after LLM-based conflict filtering. Personal Life is dominated by CONTRADICTION flags while Software Project is dominated by ALTERNATIVE flags, reflecting the different conflict profiles of everyday versus technical content.

License attribution. LongMemEval (Wu et al., 2025) is released under MIT, and the ShareGPT 52K subset (RyokoAI, 2023) is released under CC0 1.0. We redistribute the filtered subsets used in our pipeline as `meme-benchmark/MEME-fillers` (https://huggingface.co/datasets/meme-benchmark/MEME-fillers): the structural filtering and curation are released under MIT, while the filler conversation contents pass through under their original sources' licenses. The MEME dataset of evaluation episodes is released under CC BY 4.0.

Domain	Pre-filter	Post-filter	Removed	A / B / C flags
Personal Life	1,130	1,009	121	148 / 48 / 17
Software Project	9,135	9,008	127	34 / 253 / 55

Table 10. Filler pool sizes before and after LLM conflict filtering. Flag counts are per (gold fact, filler) pair; a single filler may trigger multiple flags.

Rejection examples. Figure 6 shows one rejection example per conflict type, with the conflicting phrases in **bold** in both the gold fact and the filler excerpt.

Assembly and interleaving. Given a total filler budget B (32K tokens by default) and N gaps between evidence sessions, fillers are placed into each gap in shuffled order until each gap's share of roughly B/N tokens is filled. A representative 32K-token Personal Life episode consists of 20 filler and 5 evidence sessions arranged as $[F^3, E_1, F^3, E_2, F^3, E_3, F^3, E_4, F^5, E_5, F^3]$, where F^n denotes n consecutive fillers and E_1–E_5 denote evidence sessions (four Fact Introduction variants followed by a Change+Delete Event).

C. Memory System Configuration

For each off-the-shelf system (BM25, text-embedding-3-small, Mem0, Graphiti, Karpathy Wiki), we run with the library's default configuration. Each system's defaults (top-k, chunk size, retrieval mix, internal-LLM logic) encode design choices

Type A (CONTRADICTION) — Personal Life
Gold: "My home is a **one-bedroom apartment**."
Filler (`filler_6339`)**:** "...We're planning to redo the yard at my **family home, which is now fully owned by me and my siblings** after finally paying off the mortgage..."

Type B (ALTERNATIVE) — Software Project
Gold: "Our build tool is **Dranith Compiler**."
Filler (`filler_1096`)**:** "...To **build and test software using Travis CI**, you must first select a language that is supported by Travis..."

Type C (ENTITY_CONFUSION) — Software Project
Gold: "Our team lead is **Sarah Chen**."
Filler (`filler_7586`)**:** "...Reviewers: Caitlin Barrett, Jeff Kantarek, Erin Nicolle, Robert Page, Yenny Zhang, **Chris Chen**, Kelly Chang..."

Figure 6. Rejection examples per conflict type, with conflicting phrases in **bold**. **A (CONTRADICTION)**: the family home jointly held with siblings directly contradicts the one-bedroom apartment. **B (ALTERNATIVE)**: Travis CI is a plausible alternative build tool to Dranith Compiler, even though the filler never names it in place of the gold fact. **C (ENTITY_CONFUSION)**: Chris Chen shares a surname with the gold team lead Sarah Chen, and could be misinterpreted as the same person. Personal Life fillers are indexed in the LongMemEval-S haystack pool; Software Project fillers are indexed in the ShareGPT coding English subset.

that are part of the system itself; unifying them across systems would mask exactly the architectural differences MEME aims to evaluate. Parameters explicitly varied for an ablation are noted where each experiment is described. MD-flat is our own minimal-baseline implementation.

Table 11. Per-system overview. *Encoding*, *Maintenance*, and *Retrieval* characterize each system's internal handling at the corresponding pipeline stage. All systems with an internal LLM use gpt-4.1-mini; raw retrieval (BM25, text-embedding-3-small) has none. The external answering LLM is the same gpt-4.1-mini call across all systems (Karpathy Wiki fuses retrieval and answer into a single agentic loop). Text highlighting in the table body groups shared stage-level mechanisms: light green for raw-retrieval cells (no internal LLM), light blue for LLM-processed memory, and light orange for file-based tool-use. Per-system descriptions, including the storage substrate, follow below.

System	Encoding	Maintenance	Retrieval
Raw retrieval			
BM25	no LLM	append-only	top-5 (lexical)
text-embedding-3-small	no LLM	append-only	top-5 (cosine)
LLM-processed memory			
Mem0	LLM extracts	LLM resolves	top-20 (vector)
Graphiti	LLM extracts	LLM edge invalid.	top-10 (graph + BM25 + cosine)
File-based agents			
MD-flat	LLM tool-use	LLM tool-use	LLM tool-use (read/write/append)
Karpathy Wiki	LLM tool-use	LLM compile	LLM navigates `index.md`

BM25. Sparse retrieval using the bm25s library (v0.3.2) with Lucene IDF. Each session is split into 4,096-token chunks (cl100k_base tokenizer, respecting turn boundaries) with a `[Session: <timestamp>]` header prepended. At query time, the top-k=5 chunks are concatenated as retrieval context. No internal LLM is used; the in-memory index is rebuilt incrementally per session and reset between episodes.

text-embedding-3-small. Dense retrieval using OpenAI text-embedding-3-small (1,536-dim, L2-normalized cosine similarity). Chunking and retrieval follow the same protocol as BM25 (session-level, 4,096-token chunks, top-k=5). No internal LLM is used for ingestion. The separate embedding API cost (∼$0.0007/ep at $0.02/M tokens) is negligible relative to the answering LM and is omitted from cost reporting.

Mem0. Automatic fact extraction and semantic search over a Qdrant vector database. Each session transcript is passed to `Memory.add()`, which uses an internal LLM to extract discrete facts as natural-language sentences. Mem0's internal LLM logic performs conflict resolution: when a new fact contradicts an existing memory, it decides whether to ADD, UPDATE, or DELETE. Retrieval via `Memory.search()` returns the top-20 most relevant facts (library default). Each instance uses a unique collection name to prevent cross-episode contamination. Internal LLM: gpt-4.1-mini. Embedding: OpenAI text-embedding-3-small (1,536-dim, Mem0 default).

Graphiti. A temporal knowledge graph system storing entities and relationships in Neo4j. Each session triggers: (1) entity extraction, (2) edge extraction (triples with fact descriptions), (3) entity deduplication, (4) edge deduplication. Each edge includes temporal metadata (`valid_at`, `invalid_at`). Retrieval combines semantic search, BM25 keyword search, and graph traversal (top-10 results). Each episode uses an isolated `group_id`. A key design constraint: Graphiti's extraction prompt explicitly instructs "closely paraphrase the original source sentence(s). Do not verbatim quote the original text," which directly explains its near-zero Exact Recall accuracy. Internal LLM: gpt-4.1-mini (injected via OpenAIClient).

MD-flat. The simplest architecture: an LLM with access to a virtual file system containing a single markdown file (`memory.md`). The agent receives each conversation session as input and decides what to store, update, or delete through tool calls (`read_file`, `write_file`, `append_file`, `list_files`). The ingestion prompt deliberately avoids task-specific hints, making no mention of dependencies, conditional rules, or deletion handling. The tool-calling loop runs for a maximum of 5 rounds per operation; in practice, most complete in 2–3 rounds. Internal LLM: gpt-4.1-mini.

Karpathy Wiki. An implementation of Karpathy's LLM knowledge base concept (Karpathy, 2026) using the claude-memory-compiler project[3] without modification to its prompts or logic. Three-stage pipeline:

- **Flush (ingest):** Each session is passed to `flush.py`, which uses a single LLM call (no tool use) to extract important knowledge and append it to a daily log file.
- **Compile (consolidation):** After ingesting all sessions, `compile.py` processes the daily logs into structured knowledge articles organized into three categories: concepts, connections, and qa. An `index.md` file serves as a catalog. The compile operation runs with `max_turns=30` and has access to file tools (Read, Write, Edit, Glob, Grep).
- **Query (retrieval + answer):** At question time, `query.py` reads `index.md` to identify relevant articles, reads them, and generates an answer in a single agentic loop (`max_turns=15`). Karpathy Wiki uses its native query pipeline for answer generation rather than the unified gpt-4.1-mini answering LM; the cost is reflected in the Retrieve column of Table 5.

Each episode runs in an isolated workspace. Internal LLM: gpt-4.1-mini.

D. Prompts

D.1. Verbalization Prompts

The conversation generation pipeline converts structured episode skeletons into natural multi-turn conversations using a three-stage process. First, gold fact seeds (written in first-person) are batch-converted to third-person statements by an LLM. This intermediate representation is needed because the subsequent self-chat User LLM must receive objective fact descriptions rather than first-person utterances, which it then re-verbalizes naturally. Second, a User LLM and an Assistant LLM alternate in self-chat to produce the conversation. Domain-specific third-person prompts are used for each domain:

[3] https://github.com/coleam00/claude-memory-compiler

You are rewriting first-person fact statements into third-person statements about “the user”.
You will receive a list of first-person fact seeds. Rewrite each into a natural, grammatically correct third-person sentence about “the user”.
STRICT RULES:
– Convert first-person (“I”, “my”, “we”, “our”) to third-person (“the user”, “the user’s”, “they”, “their”)
– Fix verb conjugation: “I work” → “The user works”, “I have” → “The user has”
– Fix grammar issues: articles (a/an/the), prepositions, verb forms, capitalization
– Fix semantic mismatches: e.g., “watching a show called a podcast” → “listening to a podcast called...”
– Keep ALL proper nouns, numbers, codes, and entity values EXACTLY as given (e.g., “Velthari Studio”, “10-to-7 fixed”, “RSV-09831-THETA” must appear verbatim)
– Keep dependency/causal relationships intact
– Each output must be a single self-contained sentence
– Do NOT add information not present in the seed
– Do NOT merge multiple seeds into one sentence
– Do NOT change punctuation within entity values
Output format: JSON object with key “facts” containing an array of rewritten sentences.

Figure 7. Third-person conversion prompt (Personal Life).

You are rewriting first-person project statements into third-person statements about “the developer” or “the team”.
You will receive a list of first-person fact seeds about a software project. Rewrite each into a natural, grammatically correct third-person sentence.
STRICT RULES:
– Convert “we”/“our” to “the team”/“the team’s”, “I” to “the developer”
– Fix verb conjugation: “We use” → “The team uses”
– Keep ALL tool names, commands, URLs, connection strings, and codes EXACTLY as given (e.g., “Veltrion”, “karnex build –prod”, “crysthene://prod-db:5432/aurora” must appear verbatim)
– Keep dependency/causal relationships intact
– Each output must be a single self-contained sentence
– Do NOT add information not present in the seed
– Do NOT merge multiple seeds into one sentence
Output format: JSON object with key “facts” containing an array of rewritten sentences.

Figure 8. Third-person conversion prompt (Software Project).

After third-person conversion, the User LLM receives these facts and re-verbalizes them as a first-person user in natural conversation:

I will give you a list of personal facts. Use them to act as a normal user chatting with a chat assistant. In the chat, you may ask it to assist you with various tasks or ask it about various information. However, make sure that you convey the facts about you one by one, roughly in order. IMPORTANT: convey at most ONE fact per message. Do not combine multiple facts into a single message.
After conveying each fact, continue discussing that topic for 2–3 more exchanges before moving to the next fact. Ask follow-up questions, request more details, or explore related aspects of the topic. Not every message needs to contain a new fact.
Make sure your message is concise (1–3 sentences), since real users often do not bother to write a long message. You must simulate the tone of a neutral user and do not be overly enthusiastic, verbose, formal, or polite. Do NOT say "thanks", "I will do that", or "is there anything else" — you are a user, not an assistant.
Do NOT introduce personal facts about yourself beyond the list provided. Only convey the facts given below — do not invent new hobbies, family details, health conditions, jobs, living situations, or any other personal information. Follow-up questions should be about the TOPIC, not about yourself.
VERBALIZATION RULES (follow strictly):
1. Each fact must include its entity keyword (e.g., "My health condition is...", "My financial goal is..."). The entity concept must appear in your message.
2. If a fact mentions a dependency ("depends on", "determined by", "if X changes"), you MUST preserve the full dependency language including the conditional part. The reader must understand that if the source changes, the target would change too. Do NOT weaken "depends on X; if X changes, this would change" to just "tied to X" or "since we use X".
3. If a fact starts with "If" (conditional), use ONLY "will" as the modal: "If X changes, Y will be Z". NEVER use "would/might/probably/consider". NEVER state it as accomplished ("I switched to Z").
4. If a fact is marked with [CHANGED], state the new value as a definitive current fact. Do NOT hedge or speculate.
You are a HUMAN user, not an AI. Never say anything that implies you are an AI, a language model, or that you have a training data cutoff.
Never repeat a previous message verbatim. Each message must be unique.
When ALL facts have been conveyed AND you have finished discussing the last topic, generate exactly: [END]
Facts to convey: {facts_list}

Figure 9. User LLM system prompt for self-chat conversation generation (Personal Life).

The Assistant LLM responds naturally without introducing unsupported assumptions:

Respond helpfully and naturally to the user's message. Write in natural prose paragraphs — do NOT use numbered lists, bullet points, or bold headers. Keep your response to 3–6 sentences.
IMPORTANT: You must NOT assume, guess, or mention any of the following about the user unless the user explicitly stated it in THIS conversation:
– Family/relationships (partner, spouse, parents, siblings)
– Living situation (city, housing type, roommates)
– Work details (company type, industry, salary, colleagues)
– Health/medical info (conditions, medications, diet reasons)
– Financial details (income, savings amounts, debt)
– Schedule/routine specifics beyond what user said
– Hobbies, pets, vehicles, schools beyond what user said
You MAY ask clarifying questions about the topic at hand. You MAY give general advice and information.
Never say "I don't have personal experiences", "As an AI", or similar disclaimers. Respond naturally as a knowledgeable assistant.

Figure 10. Assistant LLM system prompt for self-chat conversation generation (Personal Life).

Respond helpfully and naturally to the developer's message about their project. Write in natural prose — do NOT use numbered lists, bullet points, or bold headers. Keep your response to 3–6 sentences.
IMPORTANT: You must NOT assume, guess, or mention any of the following unless the developer explicitly stated it in THIS conversation:
– Tool names, frameworks, databases, or libraries not mentioned
– URLs, connection strings, API keys, or endpoints not shared
– Team members, leads, or contacts not introduced
– CI/CD configs, Docker images, or deploy commands not stated
– Branching strategies, sprint details, or meeting schedules not mentioned
You MAY ask clarifying questions about the setup. You MAY give general engineering advice.
Never say "I don't have personal experiences", "As an AI", or similar disclaimers. Respond naturally as a knowledgeable assistant.

Figure 11. Assistant LLM system prompt for self-chat conversation generation (Software Project).

D.2. Verification Prompts

Annotation prompt. After self-chat generation, an LLM annotates which user turn conveyed each fact:

You are annotating a conversation to identify where personal facts were conveyed.
Given a list of facts (in third-person) and a conversation, determine which user turn conveys each fact. A fact is "conveyed" if the user's message communicates the same information, even if worded differently.
Rules:
– Only consider USER turns (not assistant turns)
– A fact is conveyed if the core information matches, even with different wording. E.g., Fact "The user works at Zypherix Labs" matches user saying "I just started at Zypherix Labs"
– If a fact appears in multiple turns, return the FIRST (earliest) turn
– If a fact was NOT conveyed in any turn, set turn_index to null
– turn_index is 0-based index into the conversation array (counting both user and assistant turns)
Output format: JSON object with key "annotations" containing an array of objects:
[{"fact_id": 1, "conveyed": true, "turn_index": 2}, ...]

Figure 12. Annotation verification prompt (Layer 1).

Gemini semantic audit prompt. A separate LLM (Gemini 2.5 Flash) performs a five-point audit:

You are a data quality auditor for MEME, a benchmark that evaluates LLM agent memory systems.
For each gold fact, check ALL criteria below. Report ONLY failures.
1. VERBALIZATION ACCURACY
Is the gold fact's value present in the conversation (exactly or close paraphrase)?
2. IF-THEN CONDITIONAL FORM (only for facts with is_if_then=true)
Must contain "if" conditional structure. Trigger entity (dependency_source) must be explicitly mentioned. Modal must be "will" (not "would/might/probably") → flag WEAK_MODAL. Must NOT be stated as accomplished fact → flag CONDITIONAL_VIOLATED.
3. ENTITY KEYWORD PRESENCE
The question asks about an entity using certain words (e.g., "What's my financial goal?"). Those words or a natural synonym must appear in the conversation where the fact is introduced. FAIL if entity concept never mentioned (e.g., "I'm paying off debt" without "financial goal").
4. CASCADE-U DEPENDENCY (only for facts with has_dependency=true AND is_if_then=false)
Is the causal link STRONGLY explicit? The message must contain language like "depends on", "determined by", or "if X changes, Y would change". FAIL if dependency is only implied ("since we use X", "tied to X") rather than explicitly conditional ("depends on X; if X changes, this would change").
5. GOLD ANSWER FORMAT
Does before-delete gold have "Yes — " prefix? (flag PREFIX_ISSUE)
Output JSON array of issues:
{"entity": "...", "fact_index": N,
"check": "VERBALIZATION | IF_THEN | KEYWORD | CASCADE_U_DEP | GOLD_FORMAT",
"severity": "HIGH | MEDIUM | LOW", "issue": "brief description"}
Severity: HIGH = makes task unsolvable. MEDIUM = may cause false pass/fail. LOW = minor.
If ALL pass: {"status": "ALL_PASS", "facts_checked": N}

Figure 13. Gemini semantic audit prompt (Layer 2).

D.3. Ingestion and Retrieval Prompts (MD-flat)

MD-flat is the only memory system whose ingestion and retrieval prompts are designed by the benchmark authors; the other systems use their own built-in logic without modification. The ingestion prompt processes each session and updates `memory.md`; the retrieval prompt reads back from `memory.md` at query time.

You are a personal assistant with a persistent memory file (memory.md). After each conversation, save any information the user shared that may be useful in future sessions. Keep it compact — one fact per line with timestamp [YYYY/MM/DD].
If information has changed, update it. If something was removed or cancelled, remove the old entry.
Do NOT save conversation summaries, assistant responses, or temporary task state.
You will now receive a conversation session. Read it and update your memory accordingly.

Figure 14. MD-flat ingestion prompt. The agent operates through a tool-calling loop (max 5 rounds) with `read_file`, `write_file`, `append_file`, and `list_files`.

You have access to `read_memory()`. Read your memory to find relevant facts.
Extract ONLY the facts relevant to answering the question below. Read your memory file first, then return relevant facts as-is (do not rephrase or summarize). If nothing is relevant, say '(no relevant facts)'.
Question: {question}

Figure 15. MD-flat retrieval prompt. At query time the agent runs a tool-calling loop with only `read_memory` available, extracts the relevant facts from `memory.md` verbatim, and returns them (or '(no relevant facts)') as the retrieval context for the answering LLM.

D.4. Answer Prompt

All memory systems share a single unified answer prompt. After each system's `retrieve()` method returns a context string, the following prompt is used with Claude Sonnet 4:

Answer the user's question based ONLY on the context provided below. If the information is not in the context, say you don't have that information. Answer with ONLY the value. Do not explain or add context.
Context: {context}
Question: {question}

Figure 16. Unified answer prompt shared across all memory systems.

D.5. Judge Prompts

The evaluation framework uses task-specific judge prompts with GPT-4o (temperature 0). Each task type has its own prompt:

You are a judge evaluating whether an AI agent's answer is semantically correct.
QUESTION: {question}
GOLD: {gold}
AGENT: {agent_answer}
Does the agent's answer contain the correct information matching the gold answer?
Rules:
– Focus on semantic equivalence, not exact wording
– If the gold value is present in the agent's answer, it is correct — regardless of any additional information, future possibilities, or extra details the agent mentions
– "Dentist every 6 months; dermatologist monthly (if you change residence)" → gold is "dentist (every 6 months)" → YES (core answer correct, extra info is irrelevant)
– "40 minutes" = "40 min" → YES
– "dentist appointment every 6 months" = "dentist (every 6 months)" → YES
– If agent says "I don't know" and gold is a specific value → NO
Output ONLY JSON: {"correct": true/false, "reason": "one sentence"}

Figure 17. Judge prompt: Before-phase (common).

You are a judge evaluating whether an AI agent correctly recalls a sequence of historical values for an entity that changed multiple times.
QUESTION: {question}
GOLD (ordered history values, comma-separated): {gold}
AGENT: {agent_answer}
The user's information changed multiple times. The question asks for the FULL HISTORY in chronological order (earliest to latest).
Rules:
– Agent's answer must contain ALL gold values in the EXACT order specified
– Order matters: the values must appear earliest-to-latest as in the gold
– Extra surrounding text is fine, but the gold sequence must be preserved
– Missing any value → NO
– Wrong order → NO
– Only some values → NO
– "I don't know" → NO
Output ONLY JSON: {"correct": true/false, "reason": "one sentence"}

Figure 18. Judge prompt: Tracking.

You are a judge evaluating whether an AI agent's answer contains specific values.
QUESTION: {question}
TARGET VALUES: {gold_values}
AGENT: {agent_answer}
For EACH target value, check whether it appears (exact or paraphrased) in the agent's answer.
Rules:
– The value appears in the answer (exact or paraphrased) → true
– The value is absent → false
– "I don't know" or "I don't have that information" → all false
Output ONLY JSON: {"results": [{"value": "...", "present": true/false}], "reason": "one sentence"}

Figure 19. Judge prompt: Aggregation.

You are a judge evaluating whether an AI agent correctly recalls an EXACT value verbatim.
QUESTION: {question}
GOLD: {gold}
AGENT: {agent_answer}
Does the agent's answer contain the exact value?
Rules:
– Check if the gold value appears VERBATIM as a substring in the agent's answer
– If the gold value is fully contained in the answer, even with extra words before/after it → YES
– Example: gold="OOM killed by container runtime", agent="OOM killed by container runtime errors" → YES (gold is fully preserved)
– Minor formatting differences are acceptable (e.g., extra spaces, capitalization)
– Missing or substituted words WITHIN the gold value → NO
– "I don't know" or "I don't have that information" → NO
– A completely different value → NO
Output ONLY JSON: {"correct": true/false, "reason": "one sentence"}

Figure 20. Judge prompt: Exact Recall. Note: implemented as deterministic substring match at runtime; this prompt serves as fallback documentation.

You are a judge evaluating whether an AI agent correctly handles a DELETED piece of information.
QUESTION: {question}
GOLD: {gold}
AGENT: {agent_answer}
The information was explicitly deleted/removed. The agent should not return it.
Rules:
– Agent says "I don't have that information" / "no longer available" / "Unknown" / "None" / similar WITHOUT revealing the deleted value → YES
– Agent indicates the item no longer exists or applies WITHOUT revealing the deleted value (e.g., "No food allergies" when the deleted value was a specific allergy) → YES
– Agent reveals the deleted value in any way (even while saying it was deleted/removed) → NO
– Agent returns the deleted value as if it still exists → NO
Output ONLY JSON: {"correct": true/false, "reason": "one sentence"}

Figure 21. Judge prompt: Deletion.

You are a judge evaluating whether an AI agent correctly reports a CASCADE-DEPENDENT value after an upstream change.
QUESTION: {question}
GOLD (new cascaded value): {gold}
AGENT: {agent_answer}
The upstream entity changed, triggering a cascade. The agent should now report ONLY the new value.
Rules:
– Agent states the new value as the sole, definitive answer → YES
– Agent lists BOTH old and new values together → NO
– Agent states only the old value → NO
– Agent lists multiple options or hedges ("might", "considering") → NO
– Agent uses future tense ("will change to X", "will be X") instead of confirming as current → NO
– "I don't know" → NO
Output ONLY JSON: {"correct": true/false, "reason": "one sentence"}

Figure 22. Judge prompt: Cascade.

You are a judge evaluating whether an AI agent appropriately handles UNCERTAINTY after an upstream entity changed.
QUESTION: {question}
GOLD: {gold}
AGENT: {agent_answer}
The upstream entity changed, making the previous answer potentially invalid. The agent should express uncertainty.
Rules:
– Agent says "I don't know" / "not sure" / "Unknown" / "None" / expresses uncertainty → YES
– Agent acknowledges the upstream change and questions validity → YES
– Agent confidently states the OLD value as definitive current fact → NO
– Agent provides the old value with no hedging or uncertainty → NO
Output ONLY JSON: {"correct": true/false, "reason": "one sentence"}

Figure 23. Judge prompt: Absence.

For Aggregation and Tracking tasks, partial credit is computed in addition to binary pass/fail. Aggregation records the number of target values present out of the total. Tracking records the number of history values in correct chronological order using a position-based algorithm.

D.6. Filler Filtering Prompt

You are checking if a filler conversation conflicts with a known gold fact.
Gold fact: "{gold_fact}"
Filler conversation:
—
{filler}
—
Check these 3 conflict types:
A) CONTRADICTION: Does the filler directly contradict the gold fact?
B) ALTERNATIVE: Does the filler introduce a plausible alternative answer that could confuse a memory system?
C) ENTITY_CONFUSION: Does the filler mention the same entity/topic in a confusing way?
Answer in JSON format: {"A": true/false, "B": true/false, "C": true/false, "reason": "brief explanation or 'no conflict' "}

Figure 24. Filler conflict judgment prompt (GPT-4o-mini).

E. Prompt Optimization Details

This section provides details on the prompt optimization experiment described in Section 4.4.

E.1. Setup and Configuration

We used DSPy SIMBA (Khattab et al., 2024), an append-only prompt optimizer that iterates two strategies, rule appending and few-shot demo appending, without modifying or deleting baseline instructions. SIMBA was applied to MD-flat, Mem0, Graphiti, and Karpathy Wiki. For each system, only the ingest and retrieve prompts were optimized; the answer prompt was frozen because its backing model (Claude Sonnet 4) does not support the OpenAI `seed` parameter, making noise non-suppressible for that component. Multi-seed (N=5) statistics for MD-flat are reported in Section G.2; Mem0, Graphiti, and Karpathy results are single-seed (seed=7).

Table 12. Model and optimizer configuration for the prompt optimization experiment.

Role	Model	Temp.
Task LM (ingest, retrieve)	gpt-4.1-mini	0
Answer LM	Claude Sonnet 4	0
Prompt model (SIMBA meta)	gpt-4.1-mini	0.9
Judge	GPT-4o	0

SIMBA was configured with `max_steps=2`, `bsize=4`, `num_candidates=4`, and `num_threads=4`. The data split used seed=7 stratified sampling: 10 training episodes (5 PL + 5 SW) and 10 held-out test episodes (5 PL + 5 SW), disjoint at the episode level.

E.2. Optimized Prompt Content

SIMBA returned three candidate programs per system; for each, the winning candidate appended advice blocks to the system's prompts (baseline text retained verbatim before these blocks). The appended text per system is shown below.

(ingest, appended) If the input involves information about configuration dependencies (such as backup schedule, ORM used, or model syntax), then explicitly encode conditional knowledge linking these dependencies, e.g., if ORM changes from Vorathis ODM to Pyravar Mapper, the model syntax should update accordingly. If the input indicates deletion of an entity (like the onboarding guide), then mark the entity as deleted and respond explicitly that the information is no longer available. When partial or uncertain information is present, avoid generic 'I don't know' answers, and instead provide clarifications or acknowledge the absence explicitly. Leverage exact matching and dependency chains to improve recall of critical facts such as exact error logs.
If the conversation input includes conditional or dependent information (e.g., living arrangements depend on relationship status), then you should explicitly encode the dependency chains and update or remove old facts accordingly. When a change event occurs (like marriage or dietary restriction removal), ensure to mark prior conflicting facts as outdated or deleted and represent the new facts with clear dependency references so that downstream retrieval can reason over them.

(retrieve, appended) When receiving queries, especially those involving multiple updates, cascading dependencies, or multi-hop reasoning, focus on retrieving all relevant facts including updated and deleted information to fully answer the question. For multi-hop queries, chain together related facts to assemble a comprehensive response. When facts have dependencies or conditionals (e.g., backup schedule depends on the current database), retrieve the full dependency chain. For deletion queries, ensure retrieval reflects removal status so final answers indicate lack of availability rather than ignoring or failing to answer.
If a query involves entities with known dependencies or recent changes (such as changed relationship status affecting living arrangements and family events), then you should retrieve not only direct facts but also the full dependency chain surrounding these facts. Prioritize the most recent or updated information and infer uncertainty when dependencies conflict or are unresolved. This strategy helps generate consistent, accurate answers, especially for cascading or multi-hop questions.

Figure 25. MD-flat: SIMBA-appended advice blocks for the ingest and retrieve prompts.

MD-flat. The optimized retrieve prompt contains explicit instructions to "infer uncertainty when dependencies conflict or are unresolved," directly targeting the Absence task. Despite this, Absence accuracy drops to 0.00, suggesting that LLMs do not reliably internalize meta-cognitive instructions about expressing uncertainty.

Mem0. SIMBA returned the unchanged Mem0 library default (`ADDITIVE_EXTRACTION_PROMPT`, ∼33K chars with default examples) as the winning candidate. The two alternative candidates that did append rule blocks scored lower on training (0.509 and 0.501 vs. baseline 0.545), so SIMBA selected the default. Test accuracy moved $0.545 \rightarrow 0.534$ (-1.1pp), within the noise floor reported in Section G.2.

Graphiti. SIMBA appended advice paragraphs to each of the three Graphiti prompts (`extract_message`, `edge`, `dedupe_nodes`). The appended texts are shown below.

(extract_message, appended) If the module receives conversation messages containing technical or configuration details such as database connection strings, ORM model syntaxes, backup schedules, or package manager names, then it should prioritize explicit and accurate extraction of these distinct factual entities. It should distinguish between different software components and configurations even if phrased similarly. Being explicit in naming and careful in identifying such entities will improve downstream knowledge integration.
If the module receives conversation messages containing explicit conditional or dependent living arrangements, then it should explicitly extract and represent all relevant entities, including conditional entities like 'sibling' tied to relationship status changes. The module should robustly disambiguate references and update entity extraction as new facts (e.g., changes in relationship status) are introduced to avoid missing or outdated entities.

(edge, appended) If the module receives messages and an entity list containing technical terms like connection strings, model syntax, or backup schedules, then it should extract precise fact triples that link these entities with their values and contextual metadata, including temporality and versioning if applicable. It should avoid hallucination and ensure relations reflect the actual statements in the conversation, supporting correct temporal or conditional dependencies (e.g., changes of configurations over time).
If the module receives entity lists and conversation messages with conditional or evolving personal information, then it should ensure that fact triples clearly capture these conditional relationships and that responses to queries reflect the current, updated state or acknowledge uncertainty appropriately. The module must avoid hallucinating unsupported entities or facts and must ground edges in the explicitly available and valid entities, maintaining consistent temporal and logical coherence.

(dedupe_nodes, appended) If the module receives extracted entities and existing knowledge base entities, then it should resolve duplicates strictly based on identity and completeness without merging distinct entities or omitting valid ones. It should preserve distinct technical entities especially configuration parameters and versions that are critical for correctness and traceability in knowledge representation, thereby preventing loss of information critical to later query answering.
If the module receives newly extracted entities and updated knowledge base entities reflecting changed or deleted facts, then it should carefully resolve duplicates only for valid, consistent entities, respecting conditional dependencies and removing outdated facts. The module should strictly avoid merging distinct entities or preserving obsolete information, particularly when explicit deletions or updates are available, ensuring an accurate entity inventory aligned with the latest user context.

Figure 26. Graphiti: SIMBA-appended advice blocks for the three optimized prompts.

Karpathy Wiki. SIMBA appended one advice paragraph to each of the three Karpathy Wiki prompts (`flush`, `compile`, `query`). The appended texts are shown below.

(flush, appended) If the module receives raw conversation data containing key project facts, then it should focus on extracting and summarizing actionable, non-trivial information precisely and completely. It should emphasize capturing explicit facts about configurations, deployments, and changes, and flagging any deletions or updates to ensure downstream modules have accurate logs to compile from.

(compile, appended) If the module receives daily logs including configuration details and knowledge updates, then it should rigorously extract all relevant factual knowledge, including explicit mappings of infrastructure dependencies and conditional changes. It should represent these with clarity in the wiki, explicitly noting dependencies and if-then scenarios, and carefully documenting deletions or overrides. Complete YAML frontmatter and linking related concepts in the knowledge base are essential for quality.

(query, appended) If the module receives questions or queries about configuration, deployment, or error logs, then it should perform context-aware retrieval prioritizing exact matches and dependency-aware reasoning. It should handle deletions properly, recognize conditional dependencies (e.g., if deploy target changes, then CI config changes), and be capable of precise recall for exact error logs. Summarizing complex dependencies succinctly while citing relevant articles or sources can improve answer accuracy and completeness.

Figure 27. Karpathy Wiki: SIMBA-appended advice blocks for the three optimized prompts.

Per-task stability statistics for the SIMBA baseline-vs-optimized comparison are reported in Section G.2, computed in the same $N{=}5$ format as Table 14.

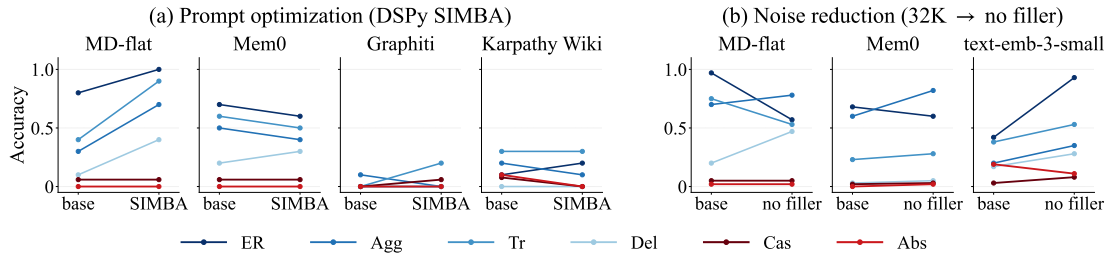


Figure 28. Two interventions external to the memory architecture: (a) prompt optimization (DSPy SIMBA, 10-episode test set; 4 prompt-tunable systems), (b) noise reduction (32K filler → no filler, 40-episode subset; 3 representative systems). Cas/Abs (red lines) stay near the floor under both. Full three-condition noise sweep in Figure 29.

F. Noise Robustness Details

We evaluate three representative systems (MD-flat, Mem0, and text-embedding-3-small) under three filler conditions: no filler (evidence sessions only), 32K tokens (default), and 128K tokens (4× default). Each condition uses the same 40 episodes (20 PL + 20 SW). Deletion, Cascade, and Absence scores use the same trivial-pass filtering as the main results table.

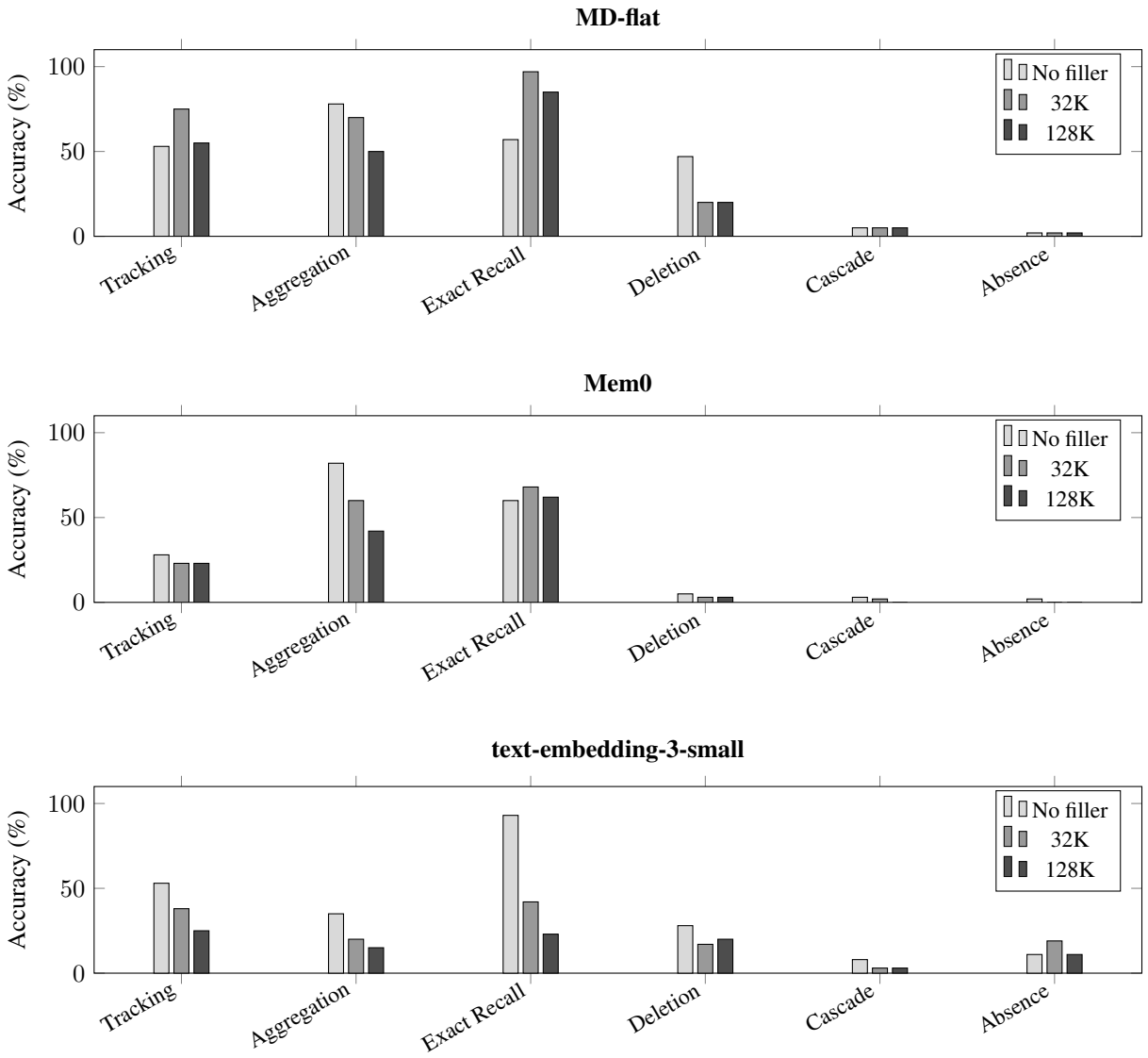


Figure 29. Per-task accuracy under three filler conditions (no filler, 32K, 128K) for each system. Deletion, Cascade, and Absence use trivial-pass filtering (same convention as the main results table).

General trend (32K → 128K). All three systems show overall degradation when filler volume increases from 32K to 128K: MD-flat drops from 45% to 36%, Mem0 from 26% to 22%, and text-embedding-3-small from 23% to 16%. For Mem0 and text-embedding-3-small, the decline is concentrated in retrieval tasks (Aggregation, Exact Recall), where increased filler dilutes the signal-to-noise ratio in semantic search. MD-flat's decline is broader, affecting Aggregation (−20pp) and Tracking (−20pp), as the agent's monolithic memory file grows proportionally with filler volume.

No-filler anomaly for MD-flat. Counter-intuitively, MD-flat's overall accuracy is *lower* without filler (40%) than with 32K filler (45%). Inspection reveals that the retrieve step (gpt-4.1-mini) behaves differently when the memory file is short: it strips timestamps and entity labels, returning bare bullet lists (e.g., "- User drives Zyvanta Sedan" instead of "[2023/03/01] User drives Zyvanta Sedan"). The answering LLM (Claude Sonnet 4), following its strict context-only instruction, then refuses to infer chronological order or entity provenance, causing failures on Tracking and Exact Recall. With 32K filler, the memory file is longer and more diverse, prompting the retrieve step to preserve metadata. This artifact is specific to the retrieve prompt's sensitivity to memory file length rather than a property of the benchmark itself.

Cascade and Absence remain at floor. Across all three noise conditions and all three systems, Cascade and Absence scores remain near zero (0–15%), reinforcing the main finding that under the default configuration the dependency-reasoning gap

is not driven by retrieval noise.

Trivial-pass rates rise with noise (justifies the filter). The trivial-pass filter discards Absence/Deletion/Cascade "passes" that came from a system that never knew the original value (i.e., before-state-check failed). Table 13 shows the trivial-pass count per (system, condition, task). For raw retrieval (text-embedding-3-small), the Absence trivial-pass rate rises sharply with noise (8% $\rightarrow$ 23% $\rightarrow$ 51%): under heavy filler the top-k retrieved chunks are dominated by irrelevant content, the answering LLM defaults to "I don't know", and "I don't know" happens to match the Absence gold of "uncertain." Without the trivial-pass filter, this artifact would inflate text-embedding-3-small's Absence score from 11% to 35% across the same conditions; with the filter, the score stays at roughly 11–15%, matching the main finding that the dependency gap is not closed by retrieval failure.

Table 13. Trivial-pass counts (and rates) per condition for the three noise-evaluated systems on the 40-episode subset. A trivial pass is a question where the system answered correctly post-change but failed the pre-change state-check, indicating it never knew the original value. The trivial-pass filter excludes these from the numerator of Cascade, Absence, and Deletion accuracy.

System	Condition	Cascade	Absence	Deletion
MD-flat	no filler	0/64 (0%)	0/53 (0%)	2/40 (5%)
	32K	2/64 (3%)	1/53 (2%)	4/40 (10%)
	128K	1/64 (2%)	6/53 (11%)	6/40 (15%)
Mem0	no filler	0/64 (0%)	0/53 (0%)	0/40 (0%)
	32K	2/64 (3%)	0/53 (0%)	1/40 (2%)
	128K	0/64 (0%)	0/53 (0%)	0/40 (0%)
text-embedding-3-small	no filler	3/64 (5%)	4/53 (8%)	6/40 (15%)
	32K	1/64 (2%)	12/53 (23%)	14/40 (35%)
	128K	3/64 (5%)	27/53 (51%)	19/40 (48%)

G. Repeated-Run Stability

G.1. Main-table baseline

To verify that the main-table conclusions are not artifacts of single-run sampling, we rerun ingestion, retrieval, and answering on a 10-episode subset (5 PL + 5 SW) for the four systems with non-deterministic LLM components: Mem0, Graphiti, MD-flat, and Karpathy Wiki. Raw retrieval (BM25, text-embedding-3-small) is deterministic and excluded. The configuration matches the main table (gpt-4.1-mini internal LLM, Claude Sonnet 4 answering LLM); N=5 trials in total, including the original main run on the same 10 episodes plus four additional sampling trials. Trivial-pass filtering is applied to Cascade, Absence, and Deletion as in the main results.

Table 14. Per-task accuracy (%) mean $\pm$ SD across N=5 trials on the 10-episode subset (gpt-4.1-mini internal LLM, Claude Sonnet 4 answering LLM).

System	Exact Recall	Agg.	Tracking	Deletion	Cascade	Absence	Overall
Mem0	64.0 $\pm$ 5.0	78.0 $\pm$ 8.0	26.0 $\pm$ 9.0	2.0 $\pm$ 4.0	4.7 $\pm$ 3.4	1.0 $\pm$ 2.6	28.7 $\pm$ 1.8
Graphiti	4.0 $\pm$ 5.0	14.0 $\pm$ 5.0	2.0 $\pm$ 4.0	4.0 $\pm$ 9.0	1.4 $\pm$ 2.0	1.0 $\pm$ 2.6	4.4 $\pm$ 1.6
MD-flat	88.0 $\pm$ 8.0	64.0 $\pm$ 5.0	82.0 $\pm$ 4.0	8.0 $\pm$ 4.0	11.4 $\pm$ 3.4	0.0 $\pm$ 0.0	42.4 $\pm$ 1.5
Karpathy Wiki	10.0 $\pm$ 7.0	18.0 $\pm$ 8.0	38.0 $\pm$ 15.0	2.0 $\pm$ 4.0	1.4 $\pm$ 2.6	2.6 $\pm$ 3.6	11.7 $\pm$ 1.0

The dependency gap is robust to sampling: Cascade SD $\leq$ 3.4pp and Absence SD $\leq$ 3.6pp across every system. Per-system Overall accuracy moves by at most 1.8pp (SD), preserving the system ranking observed in the main table. Karpathy Wiki shows the largest task-level variance (Tracking SD 15pp), driven by its agentic query loop with multiple internal LLM calls per question; the other systems retrieve and answer in a fixed number of LLM calls and show smaller variance.

G.2. SIMBA prompt-optimization (MD-flat)

For the prompt-optimization experiment described in Section E, we re-evaluate the baseline and SIMBA-optimized configurations on the same 10-episode SIMBA test set under N=5 trials each. Multi-seed stability is reported only for MD-flat. It shows the largest baseline-vs-optimized change (+0.25 Overall), so noise-floor calibration is most informative

there. For Mem0, Graphiti, and Karpathy Wiki, the single-seed baseline-vs-optimized change is ≤ 0.02 Overall, so the qualitative finding is robust to within-system noise.

Table 15. Per-task accuracy (%) mean ± SD across N=5 trials on the SIMBA 10-episode test set (MD-flat); same format as Table 14.

Config.	ER	Agg	Tr	Del	Cas	Abs	Overall
Baseline	0.90 ± 0.07	0.30 ± 0.12	0.30 ± 0.10	0.02 ± 0.04	0.07 ± 0.03	0.03 ± 0.05	0.27 ± 0.05
Optimized	1.00 ± 0.00	0.78 ± 0.08	0.92 ± 0.04	0.42 ± 0.08	0.02 ± 0.03	0.00 ± 0.00	0.52 ± 0.02

A clear dichotomy emerges. Tasks driven by *what* the memory file contains (Tracking, Aggregation, Exact Recall, Deletion) show large gains far exceeding the noise floor: Tracking $0.30 \rightarrow 0.92$ ($+0.62$ vs SD ≤ 0.10), Aggregation $0.30 \rightarrow 0.78$ ($+0.48$ vs SD ≤ 0.12), Deletion $0.02 \rightarrow 0.42$ ($+0.40$ vs SD ≤ 0.08). In contrast, Cascade and Absence differences (-0.05 and -0.03) sit within the SD of either configuration.

H. Answering LLM Swap: Per-System Breakdown

Table 16 reports the full per-task accuracy for the answering-LLM swap summarized in Table 3 (Table 3b).

Table 16. Answer LLM ablation, per-system: replacing the answering LLM (gpt-4.1-mini → Claude Sonnet 4) on all six main-table systems and 100 episodes; internal LLM held at gpt-4.1-mini. Trivial-pass filtering applied to Cascade, Absence, Deletion. **Bold** marks the best per task column.

System	Answering LLM	ER	Agg	Tr	Del	Cas	Abs	Overall
BM25	gpt-4.1-mini	**1.00**	0.05	0.16	**0.27**	0.02	0.00	0.25
	Sonnet 4	0.70	0.09	0.11	0.19	0.01	0.12	0.20
text-embedding-3-small	gpt-4.1-mini	0.96	0.33	0.46	0.17	0.04	0.00	0.33
	Sonnet 4	0.43	0.27	0.36	0.20	0.03	**0.16**	0.24
Mem0	gpt-4.1-mini	0.67	0.35	0.43	0.21	0.03	0.00	0.28
	Sonnet 4	0.67	0.65	0.16	0.03	0.01	0.00	0.25
Graphiti	gpt-4.1-mini	0.03	0.01	0.04	0.09	0.02	0.01	0.03
	Sonnet 4	0.03	0.01	0.00	0.09	0.04	0.00	0.03
MD-flat	gpt-4.1-mini	0.94	0.45	0.77	0.25	**0.06**	0.05	0.42
	Sonnet 4	0.90	**0.69**	**0.80**	0.24	0.05	0.05	**0.46**
Karpathy Wiki	gpt-4.1-mini	0.11	0.18	0.27	0.03	0.01	0.02	0.10
	Sonnet 4	0.11	0.18	0.27	0.03	0.01	0.02	0.10

I. Per-stage Cascade Traces for the Remaining Systems

The per-stage trace for Mem0, MD-flat (gpt-4.1-mini), BM25, and text-embedding-3-small on the same Cascade episode (sw_033) used in Figure 4. All four systems encode the rule and the pre-change value, but each fails at a different downstream stage; the answering LLM in every case ends up reporting the pre-change value (Hyunwoo Nam).

J. Retrieval vs. Reasoning Bottleneck on Cascade and Absence (Top-k Sweep)

For each *knew_but_failed* case in the top-k sweep (BM25 and dense, 40-episode subset, Claude Sonnet 4 answering LLM), we check whether both the dependency rule and the upstream change event were already in the retrieval context. A failure where both are present is an answering failure (both rule and change are in the retrieved context but the answering LLM still fails to propagate); a failure where one is missing is a retrieval failure.

Method. We use the dataset's gold-fact metadata to identify, for each task, the specific evidence sessions carrying the rule and the change event. The rule session contains a gold-fact entry with `entity = target_entity` and (Cascade) `is_if_then = True` or (Absence) `has_dependency = True`. The change session is the canonical `evidence_change+delete_event` session, with a gold-fact entry where `entity = cascade_source` and `type = root_change`. For each rule and change fact, the `original_seed` string appears verbatim in its session's conversation in 100% of cases (213 rule, 100 change, 341 dependency facts in the 100-episode dataset), so checking for `original_seed` as a substring of the retrieval context is exact. Identical breakdowns are obtained by matching the

pre-change value | conditional rule | change event | answer

Episode sw_033 (Software Project, Cascade).
Sess. 11: *"Weekly report recipient is Hyunwoo Nam (assigned by team lead Seokjin Kang). If the team lead changes, the recipient will be James Lee."*
Sess. 14: *"Jihoon Ryu is our new team lead."*

Question: *"Who receives the weekly report?"* Answer: James Lee. **All four systems answer Hyunwoo Nam.**

BM25 – Failure

1. ✓ **Encoding**: all facts encoded
   Sess. 11 (03/13): "weekly report recipient is Hyunwoo Nam. . . "
   ". . . if team lead changes, recipient will be James Lee"
2. ✓ **Maintenance**: all facts retained
   Sess. 14 (03/17): "Jihoon Ryu is our new team lead"
3. × **Retrieval**: lexical top-k misses the 03/17 session
   top-1: Sess. 11 (recipient + rule)
   [unrelated filler]

MD-flat (gpt-4.1-mini) – Failure

1. ✓ **Encoding**: all facts encoded
   [2023/03/13] Weekly report recipient is Hyunwoo Nam . . .
   [2023/03/13] If team lead changes, recipient will be James Lee
2. ✓ **Maintenance**: all facts retained
   [2023/03/17] Jihoon Ryu is the new team lead
3. × **Retrieval**: tool-use never opens 03/17 entry
   - Weekly report recipient is Hyunwoo Nam . . .
   - If team lead changes, recipient will be James Lee

text-embedding-3-small – Failure

1. ✓ **Encoding**: all facts encoded
   Sess. 11 (03/13): "weekly report recipient is Hyunwoo Nam. . . "
   ". . . if team lead changes, recipient will be James Lee"
2. ✓ **Maintenance**: all facts retained
   Sess. 14 (03/17): "Jihoon Ryu is our new team lead"
3. ✓ **Retrieval**: change event surfaced
   top-1: Sess. 11 (recipient + rule)
   top-2: Sess. 14 (Jihoon Ryu)
   [unrelated filler]
4. × **Answer**: LLM reports Hyunwoo Nam

Mem0 (gpt-4.1-mini) – Failure

1. ✓ **Encoding**: all facts encoded
   User's weekly report recipient is Hyunwoo Nam . . . , with the recipient changing to James Lee if the team lead changes
2. ✓ **Maintenance**: all facts retained
   User's new team lead is Jihoon Ryu . . .
3. ✓ **Retrieval**: change event surfaced
   [0.65] Hyunwoo Nam . . . changing to James Lee if the team lead changes
   [unrelated filler]
   [0.25] Jihoon Ryu . . . (rank 19/20)
4. × **Answer**: LLM reports Hyunwoo Nam

Figure 30. State of the remaining four memory systems on episode sw_033, traced across encoding, maintenance, retrieval, and the answering LLM. Top row: BM25 and MD-flat (gpt-4.1-mini) drop the change event before retrieval. Bottom row: text-embedding-3-small and Mem0 surface the change event in the retrieved context, but the answering LLM still reports the pre-change value. Same color coding as Figure 4.

session's `[Session: <timestamp>]` header instead, confirming that no rule/change content is split off from its session header by chunking.

Results. Table 17 partitions *knew_but_failed* cases by which stage missed the propagated answer: change-event missing from retrieval, rule missing from retrieval, or both retrieved but the answering LLM fails to propagate. The rule is retrieved in 100% of failures across all k, so retrieval-side losses are change-event misses; the live trade-off at higher k is between change-event miss and answering failure.

- Cascade: at k=20 and k=40, 55% of failures are change-event misses and 45% are answering failures (the rule and the change session are both retrieved yet the answering LLM still does not produce the propagated value).
- Absence: at k=20 and k=40, 86% and 83% of failures respectively are answering failures (the agent commits to a definite answer when it should abstain), with the remainder being change-event misses.

K. MD-flat Internal-LLM Mechanisms

This section traces the per-LLM internal-storage mechanisms that explain the Cascade and Absence outcomes of the internal-LLM ablation in Table 4.

Table 17. Knew_but_failed cases on the top-k sweep, partitioned by failure stage (gold-fact substring match). Combined BM25 + dense on a 40-episode subset (128 Cascade and 106 Absence case-system pairs) with Sonnet 4 as the answering LLM. *Cases* counts knew_but_failed events; the remaining failures are never_knew (system did not encode the fact before the change). Knew_but_failed accounts for 17–45% of the Cascade pool and 7–55% of the Absence pool across the four k values. Per-cell percentages are within *Cases*: the Retrieval group splits change-event miss from rule miss, and Answering counts cases where both rule and change are in the retrieved context but the answering LLM still fails to propagate.

			Retrieval		
Task	**top-k**	**Cases**	**Change**	**Rule**	**Answering**
Cascade	5	57	32 (56%)	0 (0%)	25 (44%)
	10	34	10 (29%)	0 (0%)	24 (71%)
	20	22	12 (55%)	0 (0%)	10 (45%)
	40	22	12 (55%)	0 (0%)	10 (45%)
Absence	5	58	34 (59%)	0 (0%)	24 (41%)
	10	16	7 (44%)	0 (0%)	9 (56%)
	20	7	1 (14%)	0 (0%)	6 (86%)
	40	12	2 (17%)	0 (0%)	10 (83%)

K.1. MD-flat with gpt-5: ingestion compresses memory and erases prior entries

Table 18 summarizes `write_memory` behavior under the four internal LLMs on the 20-episode subset. The three stronger LLMs (gpt-5, GLM-5.1, Opus 4.7) treat Change+Delete sessions as memory-update triggers (95–100% write rate), while gpt-4.1-mini almost always uses `append_memory`. Comparing memory file sizes *before* and *after* the Change+Delete write reveals the per-LLM compression strategy: gpt-5 compresses on every write (cumulatively shrinking the pre-event memory to $\sim$1,800 chars and dropping a further 28% at the change event), GLM-5.1 is effectively append-only ($+1\%$ at the event, so prior entries are preserved), and Opus 4.7 restructures while keeping explicit `Contingency:` entries (-5% at the event; mechanism in Section K.2). gpt-4.1-mini barely writes, so its memory just accumulates session by session.

Table 18. `write_memory` invocations and memory file length around the Change+Delete session on MD-flat (20 episodes, internal-LLM ablation). The Before/After columns show average memory file size just before and after the Change+Delete write; the delta exposes each LLM's compression strategy.

Internal LLM	**Change+Delete write rate**	**Memory before (chars)**	**Memory after (chars)**	Δ
gpt-4.1-mini	2/20 (10%)	$\sim$7,500	$\sim$7,400	$\approx$0%
gpt-5	19/20 (95%)	$\sim$1,800	$\sim$1,300	-28%
GLM-5.1	19/20 (95%)	$\sim$5,100	$\sim$5,100	$+1\%$
Opus 4.7	20/20 (100%)	$\sim$2,900	$\sim$2,700	-5%

Trigger detection alone does not yield dependency reasoning. The downstream effect of gpt-5's lossy compression is visible in the Absence column of Table 4: gpt-5's compressed memory often contains neither the original entity value nor the dependency rule, so retrieval returns "(no relevant facts)" and the answering LLM defaults to "I don't have that information," which coincidentally matches the Absence gold of "uncertain." GLM-5.1's append-only memory preserves prior entries, so the original value remains retrievable and the system commits to it, failing Absence.

As a concrete instance, on Personal Life Episode 7 (commute Absence question), gpt-5 reads the 4,440-char memory file (32 entries through 2023/03/22 including the commute fact) and then issues `write_memory` with 236 chars containing only the four new 2023/03/24 entries (move to Corenth Hills, gaming PC purchase, Dranthazon removal, housewarming party). The prior commute fact is deleted along with 31 other entries; the after-Absence retrieval returns `(no relevant facts)`, and the agent answers "I don't have that information."

K.2. MD-flat with Opus 4.7: explicit contingencies and active dependency propagation

Unlike gpt-5's lossy compression or GLM-5.1's append-only behavior, Opus 4.7 reorganizes the MD-flat memory file into a structured document with topical `##` sections (e.g., *Travel History*, *Travel Preferences*) and, critically, records each dependency rule as an explicit `Contingency:` entry rather than embedding it inside the dependent fact. On Personal Life Episode 9, for instance, the pre-change memory contains:

```
[2023/03/19] Contingency: if health condition changes, exercise routine becomes
```

```
yoga 2x/week.
[2023/03/19] Contingency: if health condition changes, dietary restriction becomes
no alcohol.
[2023/03/19] Contingency: if exercise routine changes, will work out at Crysthene
Pool.
```

When an upstream change arrives during a later session, Opus reacts in one of two ways depending on whether a replacement rule is available, and these two strategies are exactly what closes Cascade and Absence respectively:

Strategy A (Cascade, replacement rule exists): apply the rule, write the resolved value. Continuing the Episode 9 trace, after the user reports that their *tendinitis* has resolved, Opus writes:

```
[2023/03/25] Tendinitis treated and resolved; per contingencies, dietary
restriction now no alcohol and exercise routine now yoga 2x/week at Crysthene
Pool.
```

The resolved 1-hop and 2-hop values (*no alcohol*, *Crysthene Pool*) are now in memory; at query time the answering LLM retrieves them directly and matches the Cascade gold answers.

Strategy B (Absence, no replacement rule): remove dependent facts and flag re-confirmation needed. On Software Project Episode 2, the deploy target changes from *Narvex Cloud* to *Thandrel Infra*, but the dependent facts (logs drain, Docker image, CI config, Dockerfile, deploy command) have no propagation rules in the memory file. Opus's post-change memory contains a new section:

```
## Deployment-derived facts (need re-confirmation since deploy target changed to
Thandrel Infra)
- Previous (Narvex Cloud) derived facts removed; ask user for updated logs drain,
Docker image, CI config, Dockerfile, deploy command under Thandrel Infra
```

At query time, the answering LLM sees no resolved value for these entities and answers "I don't have that information," which matches the Absence gold of "uncertain."

The other internal LLMs do not implement this two-strategy split: gpt-4.1-mini and GLM-5.1 leave stale dependent facts intact (failing Cascade and Absence), and gpt-5 compresses memory aggressively at the upstream-change session (yielding accidental Absence matches but no genuine Cascade propagation; Section K.1). Opus's combination of explicit contingencies plus active propagation explains the Cascade jump from 0.00 to 0.32 and the Absence jump from 0.07 to 0.59 in the MD-flat row of Table 4. The same LLM produces no comparable gain on Mem0 (atomic facts decompose the contingency entries) or Graphiti (extraction discards the contingency wording), confirming that the file-based architecture is what allows Opus's reasoning to land in the memory representation.

The cost of this behavior is twofold. First, Opus's hierarchical reorganization paraphrases content during ingest, dropping Exact Recall from 0.90 to 0.60 and Tracking from 0.80 to 0.20 on the same 20-episode subset. Second, the per-episode dollar cost is roughly 70 times the gpt-4.1-mini baseline (Table 6); both deltas are reported in Table 4.

L. In-Context Ceiling: Gold Facts Only

The gold-facts in-context ceiling feeds only the task-relevant gold facts (no filler) directly to the answering LLM. It confirms that the tasks are solvable in principle when retrieval is perfect.

Table 19. Per-task accuracy of the gold-facts in-context ceiling (no memory system, no filler) across four answer LLMs. Best per task in **bold**.

Answering LLM	ER	Agg	Tr	Del	Cas	Abs	Overall
Opus 4.7	**1.00**	0.85	0.96	**0.98**	**0.93**	0.72	**0.91**
Sonnet 4.6	**1.00**	0.87	**0.98**	0.70	0.84	0.62	0.84
Sonnet 4	**1.00**	**0.95**	0.97	0.87	0.60	**0.81**	0.87
gpt-4.1-mini	**1.00**	0.47	**0.98**	0.62	0.74	0.37	0.70