MemFail: Stress-Testing Failure Modes of LLM Memory Systems

Ishir Garg
University of California, Berkeley
ishirgarg@berkeley.edu

Neel Kolhe
University of California, Berkeley
neelkolhe@berkeley.edu

Dawn Song
University of California, Berkeley
dawnsong@cs.berkeley.edu

Xuandong Zhao
University of California, Berkeley
xuandongzhao@berkeley.edu

https://github.com/ishirgarg/MemFail

Abstract

Large language model (LLM) agents increasingly rely on external memory systems to remain consistent across long-horizon interactions, but little empirical work has been done to understand the specific failure modes and design choices that these systems present. Existing benchmarks report aggregate question-answering accuracy and treat memory systems as black boxes, making it impossible to attribute an incorrect answer to a particular failure mode of the system. We introduce MEMFAIL, a diagnostic benchmark that isolates the failure modes of modern LLM memory systems. We begin by formalizing memory systems as the composition of three canonical operations—summarization, storage, and retrieval—and identify the potential failure modes induced by each. Based on these hypothesized failure modes, we construct five datasets spanning four tasks, each adversarially designed to test a specific operation of a memory system. Using these datasets, we evaluate four state-of-the-art memory systems on MEMFAIL and demonstrate how MEMFAIL can be used to empirically understand the tradeoffs induced by differences in memory system architectures.

1 Introduction

LLM agents have rapidly grown more capable at long-horizon tasks, but their limited context windows make it difficult to maintain consistency over time. Traditional memory systems such as retrieval-augmented generation (RAG) (Lewis et al., 2021) and vector databases store the raw conversation history, which is expensive in storage and input tokens when only a fraction is worth remembering, and which provides no natural way to forget or update old information as user preferences change.

A growing body of work on *LLM memory systems* has emerged in response, augmenting agents with external stores they can read, write, and update over a lifetime, enabling consistent and personalized responses (Chhikara et al., 2025; Xu et al., 2025; Liu et al., 2026; Xu et al., 2026; Rasmussen et al., 2025; Hu et al., 2026a). But the very mechanisms for compression, updating, and forgetting introduce new failure modes: a system may *over-summarize* a conversation before storing it, stripping key details, or fail to remove an old fact when a contradictory one arrives.

Prior work has converged on a few key desiderata: retrieving only the memories useful for the current task, faithful storage that preserves the semantics of the original experience, updatability to overwrite outdated facts and accumulate coexisting ones, low latency for fast storage and retrieval, and token efficiency to minimize tokens in the agent's context. However, these desiderata trade off: a system that stores every conversation verbatim (like simple RAG) excels at faithful storage but is token-inefficient, while breaking conversations into smaller memories shrinks retrieved context but makes the relevant pieces harder and slower to find.

Despite the recent surge of memory systems with varied design choices, little work has characterized the *tradeoffs* between these desiderata or the failure modes underneath them. Existing benchmarks treat memory systems as end-to-end black boxes, providing aggregate metrics that obscure the distinct failure modes inside. In this work, we introduce MEMFAIL, a benchmark targeting failure modes of LLM memory systems. We begin by proposing a framework in which memory systems can be formally represented as a combination of three operations: summary, storage, and retrieval. Based on this taxonomy, we hypothesize distinct failure

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 16460–16502
October 24-29, 2026 ©2026 Association for Computational Linguistics

modes that any memory system fitting this formalism may exhibit. MEMFAIL provides five datasets spanning four tasks to elicit these failure modes:

- *Conditional-Facts*: faithful retention of important causal relationships.
- *Persona-Retrieval*: faithful retention of small details under misleading questions.
- *Long-Hop*: retrieval over long-range causal relationships.
- *Coexisting-Facts*: storing and retrieving *all* relevant coexisting pieces of information for a query.
- *Conditional-Facts (Hard)*: preserving important causal details when summarizing experiences.

We evaluate four state-of-the-art memory systems—Mem0 (Chhikara et al., 2025), A-MEM (Xu et al., 2025), SimpleMem (Liu et al., 2026), and StructMem (Xu et al., 2026)—on MEMFAIL, and our analysis surfaces three findings that aggregate metrics would otherwise obscure. First, no single system dominates: each architecture exhibits a distinctive failure signature—graph-based StructMem excels at causal reasoning but collapses on coexisting-fact retrieval, while Mem0 shows the opposite pattern. Second, scaling either the number of retrieved memories or the strength of the underlying LLM yields little improvement, and in several cases degrades performance, indicating that current systems are bound by architectural constraints rather than by model intelligence or context budget. Third, the relationship between token consumption and accuracy is task-dependent: summary-bottlenecked tasks reward verbose memories, while retrieval-bottlenecked tasks suffer when large memories pollute the embedding space. Building on these findings, we propose two under-explored directions—mixture-of-experts memory architectures and task-adaptive token scaling—as paths toward systems that mitigate, rather than trade off between, the desiderata above.

2 Related Work

LLM Memory Systems. A growing body of work augments LLMs with persistent memory. MemGPT pages between in-context working memory and external storage in an OS-inspired hierarchy (Packer et al., 2024). MemoryBank applies summarization and Ebbinghaus-style forgetting curves over user-specific facts (Zhong et al., 2023). Mem0 extracts atomic facts into a vector store (or, in its graph-augmented variant Mem0g, a hybrid vector-and-graph store) with explicit `ADD`, `UPDATE`, and `DELETE` operations (Chhikara et al., 2025). A-MEM forgoes predefined schemas, organizing memories as agentically linked notes that the agent dynamically tags and reorganizes (Xu et al., 2025). StructMem builds hierarchical event-level structures with periodic semantic consolidation (Xu et al., 2026), while SimpleMem applies semantically lossless compression via entropy-aware filtering and adaptive retrieval (Liu et al., 2026). Reflexion (Shinn et al., 2023) and Generative Agents (Park et al., 2023) explore reflective memory streams for self-improvement and social simulation. These systems share the primitives our framework abstracts—summarize, store, retrieve—and our benchmark plugs into any system exposing them.

LLM Long-Context and Memory System Benchmarks. Two orthogonal lines of work complement ours. ER-MIA (Piehl et al., 2026) and InjecMEM (Tian et al., 2025) mount memory-injection attacks against memory-augmented LLMs and layered systems such as MemoryOS (Kang et al., 2025), while SkillJect (Jia et al., 2026) mounts a skill-based prompt-injection attack against the agent-skills abstraction, exposing adversarial rather than benign failures. Needle-in-a-Haystack (Nelson et al., 2024), RULER (Hsieh et al., 2024), LongEval (Li et al., 2023), LongBench (Bai et al., 2024), L-Eval (An et al., 2024), and M^4LE (Kwan et al., 2024) stress-test the long-context capabilities of LLMs themselves rather than the memory systems built atop them. For general recall, MemoryBank pairs probing questions with multi-day histories (Zhong et al., 2023); PerLTQA targets personalized long-term QA (Du et al., 2024); MemBench spans factual and reflective memory across participation and observation scenarios with metrics for accuracy, recall, capacity, and temporal efficiency (Tan et al., 2025); MemoryAgentBench converts long-context datasets into incremental tasks across four competencies—accurate retrieval, test-time learning, long-range understanding, and selective forgetting (Hu et al., 2026b); LoCoMo provides multi-session conversations covering single-hop, multi-hop, temporal, open-domain, adversarial, event-summarization, and multi-modal tasks (Maharana et al., 2024);

LongDialQA derives multi-party dialogues from TV scripts (Kim et al., 2025). For reasoning, MemSim (Zhang et al., 2024) synthesizes QA via a Bayesian Relation Network, and MemoryBench reframes evaluation as continual learning from simulated feedback, distinguishing declarative from procedural memory (Ai et al., 2025). A few target failures explicitly: LoCoMo includes misleading questions for abstention (Maharana et al., 2024); LongMemEval curates 500 questions across information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention, alongside a key-value formal model (Wu et al., 2025). However, their model is less general than ours, and they do not discuss failure modes; StructMemEval shows that organizational hints improve memory organization (Shutova et al., 2026). All treat the system as a black box, reporting aggregate end-to-end scores that cannot localize failures to summarization, storage, retrieval, or reasoning—a gap MEMFAIL fills by isolating each operation and analyzing failure modes and their tradeoffs.

3 Background

We present a formal model of LLM memory systems that MEMFAIL assumes. The framework identifies natural failure modes any system fitting it may exhibit, which MEMFAIL is designed to test.

3.1 Three Operations of a Memory System

We assume every memory system decomposes into three operations. Let Q denote a user query, H a conversation history, and M the current memory database.

Retrieval. Given a new query Q, the conversation history H, and the memory state M, the system retrieves a set of relevant memories R drawn from M and inserted into the agent's prompt alongside Q and (optionally) H.

Summarization. After an interaction, H is compressed into a representation H' that extracts the information deemed worth retaining.

Storage. The storage step takes H' and the existing memory state M and produces an updated memory state M', possibly overwriting, merging, or appending to existing entries in M, or performing a no-op.

As demonstrated in Table 1, this decomposition subsumes a wide range of modern memory systems, including Mem0, A-MEM, SimpleMem, and StructMem, and mirrors the canonical encoding/storage/retrieval description of human memory.

3.2 Failure Modes

Based on our abstract model of a memory system, we propose that memory systems exhibit four natural failure modes:

- **Summary failure:** summarization deletes or malforms important information present in H. E.g., *"I am deathly allergic to peanuts"* may be compressed into *"allergic to peanuts"*, stripping the severity critical for downstream reasoning.
- **Storage failure:** the storage step does not adequately incorporate H' into M. This includes refusing to overwrite outdated facts (e.g., not updating *"Dan likes pizza"* after the user states Dan now hates pizza) and refusing to admit valid coexisting facts (e.g., rejecting *"Dan likes burgers"* as contradictory to stored *"Dan likes pizza"*).
- **Retrieval failure:** retrieval fails to return relevant memories in R, or returns memories that are semantically similar but contextually inappropriate.
- **Reasoning failure:** the agent makes an incorrect judgment even when the correct memories were retrieved. Note that this is *not* a failure of the memory system, but we measure it for completeness.

Prior work primarily evaluates the combination of all four modes by inserting long conversation histories and asking LLMs to infer user personalities or preferences without distinguishing between them. **MEMFAIL consists of tasks specifically designed to isolate failure modes (1)–(3)**, which are unique to modern memory systems and unaddressed by prior benchmarks.

3.3 Compatibility with Existing Memory Systems

Critically, a good benchmark should evaluate a diverse array of memory systems with minimal assumptions about their internals. MEMFAIL evaluates any system that exposes three functions:

- `store_conversation(`H`)`: store H in the database (subsuming SUMMARIZE and STORE).

System	SUMMARIZE	STORE	RETRIEVE
SimpleMem	verbatim	flat list of turns	embedding top-k
Mem0	LLM fact extraction	atomic units; LLM tool-call updates (add/update/delete)	embedding top-k over units
A-MEM	LLM-written descriptive notes	linked notes in vector DB	embedding top-k
StructMem	LLM entity/relation extraction	knowledge graph (typed nodes + edges)	subgraph around query entities

Table 1: How each tested memory system implements the three operations of Section 3.

- `retrieve_memories(`Q, H, k`)`: return the top-k memories relevant to Q and H.
- `get_all_memories()`: return all currently stored memories.

The first two are required for the evaluation loop. The third is used only by MEMFAIL's judge LLM to diagnose which failure mode occurred, and is not required for deployment of the memory system. Any system implementing these three functions automatically plugs into our evaluation harness.

4 Benchmark Details

MEMFAIL comprises of five datasets in English organized into four tasks, each hypothesized to isolate one failure mode from Section 3. We sketch the high-level design only; full specifications, generator prompts, validation rules, deduplication thresholds, sampling pools, and additional worked examples are deferred to Appendix B.

Task 1: *Conditional-Facts*. Targets *summary failure*—summarization-based memory stripping qualifying conditions from facts at commit time. Each row encodes a rule "entity E exhibits behavior B only when condition C is satisfied," embedded in a 5–8 sentence essay alongside 4–7 unrelated unconditional facts about the same entity. The graded query asks whether E would exhibit B in a specific context X that either does or does not satisfy C; a system that drops C during summarization and stores the unconditional "E does B" would incorrectly answer "yes" regardless of X. Conditions are sampled from a fixed list of types (time of day, weather, mood, social setting, prior activity, etc.). We create two variants: the **Easy** variant places the entire rule inside a single sentence, so any system that copies the sentence verbatim succeeds; the **Hard** variant decomposes the rule into three non-adjacent sentences—a behavior sentence, a condition sentence, and a linking sentence—spread across an 8–12 sentence essay, forcing *reconstruction* from distributed evidence. Easy and Hard rows share the same entity and condition specs, so any performance gap is attributable to the distribution of the rule across sentences. A Hard example is given in Appendix B.3.

Example: *Conditional-Facts* (Easy)

Essay (excerpt): "...Sylas is a shrewd negotiator who thrives in the bustling markets and political halls of his city.... Sylas draws elaborate maps only if he has just finished a negotiation...."
Question: "Sylas just finished meditating quietly, not negotiating. Would he draw an elaborate map now?"
Ground truth: "No—Sylas only draws maps immediately after a negotiation."

Task 2: *Coexisting-Facts*. Targets both *storage failure* and *retrieval failure*. Modern memory systems aggressively reconcile incoming information with their existing database; a possible failure pattern is that two *compatible* facts ("user likes pizza" and "user likes ramen") are incorrectly treated as contradictions, causing the system to overwrite the older fact rather than store both. Each row covers one of 100 curated preference categories (foods, hat styles, music genres, etc.), holds $N \in \{2, 3, 4, 5\}$ distinct preferences within that category—each expressed as an isolated first-person statement that references only that single preference—and asks a holistic scenario question whose well-formed answer requires all N preferences.

Example: *Coexisting-Facts*

Category: hat styles.
Preference facts: (i) "I often wear a fedora when I want to add a classic touch to my outfit"; (ii) "Beanies are my go-to for staying warm and casual during chilly days"; (iii) "A bucket hat is what I reach for on sunny, laid-back weekends."
Question: "I'm packing for a week-long trip with mixed weather—which hats should I bring to cover all occasions?"

Ground truth: fedora, beanie, bucket hat.

Task 3: *Persona-Retrieval*. Targets *storage failure*—a memory system retrieving a stored profile when asked about a different person. Each row contains a 10–15 sentence essay about a named entity E embedding 4–5 idiosyncratic facts. Three graded queries are attached, each independently toggled $50/50$ between two forms: a *direct* query that names E and is answerable from a specific essay detail, and a *misleading* query that names a distractor D unrelated to the question, for which the correct answer is to abstain. Entities and distractors are sampled from a fixed pool of 30 diverse names; persona flavors from a fixed pool of 30 flavors.

Example: *Persona-Retrieval*

Essay (excerpt): "Yuki Tanaka spends most mornings hunched over a lightbox, tracing contours... She refuses to eat shellfish because she gets a severe hive reaction, so boat provisions are always cooked onshore and strictly shellfish-free...."
Q1 (misleading): "Does Noah Brooks eat shellfish?" → "I don't have information about Noah Brooks."
Q2 (non-misleading): "What food should I avoid serving to Yuki Tanaka?" → "Avoid shellfish—she has a severe allergic reaction."

Task 4: *Long-Hop*. Targets *retrieval failure*. Each row encodes a strictly transitive chain $A_1 \rightarrow \cdots \rightarrow A_{K+2}$ of $K \in \{1, 2, 3\}$ hops, in which fact i links anchor i to anchor $i{+}1$; anchors are subjective (moods, routines, opinions, personal objects) so that no fact is answerable from world knowledge alone, and every fact is self-contained. The graded question references only the head anchor and asks for the terminal anchor, presented as a 5-way multiple-choice question with four shape-matched distractors orthogonal to every fact in the chain. At evaluation time, we provide each fact *separately* to the memory system; this forces the system to retrieve and compose the chain from scattered storage rather than read it off from a single conversation.

Example: *Long-Hop* ($K{=}3$)

Facts: "Whenever I sip morning espresso I call my mom"; "After I call my mom I plan a day trip"; "When I plan a day trip I pack snacks"; "When I pack snacks I take scenic photos."
Question: "When I sip morning espresso, what do I end up doing?"
Choices: A. organize the bookshelf; B. practice guitar scales; C. water the plants; **D. take scenic photos**; E. fold the laundry.
Ground truth: take scenic photos.

Dataset statistics. Table 2 reports measured statistics across all five datasets, with all token counts (mean tokens per dataset entry) computed using the `cl100k_base` tokenizer (OpenAI, 2022). *Coexisting-Facts* is broken out by per-row preference count N and *Long-Hop* by hop count K, since both the number of storage units and the entry-token cost scale with these.

5 Experimental Setup

5.1 Evaluation Loop

MEMFAIL's harness applies to any memory system implementing the three operations of Section 3. Each task reduces to a sequence of conversations: ungraded *storage conversations* commit information to memory, and *query conversations* elicit graded answers.

Phase 1: Storage. For each task we extract the information unit needed per graded query—a conditional-fact essay, a preference statement, a persona essay, or a message from the reasoning chain—and send each in its own conversation, so that the system must store and associate them across sessions.

Phase 2: Query. The harness creates one query conversation per graded question, calls `memory_system.retrieve_memories(query, conversation, k)`, formats the top-k memories into the prompt, invokes the test-taker, and records the response. Query conversations do not update the database, so query order is irrelevant.

Phase 3: Grading. Each graded question has a set of $N \geq 1$ memories the system should retrieve. An LLM-as-a-judge receives the query, ground truth, all stored memories at query time, and all retrieved memories, and classifies each of the N memories into:

1. **Storage check.** Is the memory present in `get_all_memories()`? Failure is a **storage error**.

2. **Summary check.** Conditional on storage, are *critical details* preserved (e.g., the qualifying

Dataset	Generator	# Q	# Graded	Avg. tok.	Other
Conditional-Facts (Easy)	`gpt-4.1-mini`	100	100	130	32y / 68n; 27 cond. types
Conditional-Facts (Hard)	`gpt-5-mini`	100	100	189	49y / 51n; 27 cond. types
Coexisting-Facts (N=2/3/4/5)	`gpt-4.1-mini`	26/31/20/23	26/31/20/23	54/68/84/101	52/93/80/115 facts
Persona-Retrieval	`gpt-5-mini`	100	300	297	52.3% misleading
Long-Hop (K=1/2/3)	`gpt-5`	31/32/29	31/32/29	59/74/87	62/96/116 facts

Table 2: Per-dataset measured statistics; "Avg. tok." is mean tokens per entry under `cl100k_base`. For *Coexisting-Facts* and *Long-Hop*, values are reported in order of N and K respectively.

condition in *Conditional-Facts*)? Failure is a **summary error**.

3. **Retrieval check.** Conditional on faithful storage, was the entry in the top-k set? Failure is a **retrieval error**.

4. **Reasoning check.** Conditional on retrieval, did the test-taker use it to produce the correct answer? Failure is a **reasoning error**.

5. **Correct.** All memories stored, summarized, retrieved, and used successfully.

Separating storage from summarization. Because most open-source systems expose summarization and storage as a single atomic `store_conversation` call, the two must be separated at grading time. The storage check is deliberately permissive: it asks only whether *any* trace of the target information appears in the store, and the judge is instructed to accept paraphrase, aggressive compression, distribution across entries, and even attachment to the wrong entity (Appendix E); a subsequent loss of critical detail is then charged to summarization. One case is not separable: a system that malforms a fact and then writes nothing at all is bucketed as a storage error by convention, since with only the final memory state observable the omission cannot be attributed to either operation without instrumenting system internals.

We focus our analysis on the first three, as these are memory-system failures; reasoning errors reflect LLM limitations and occur infrequently (Appendix D). We fix `gpt-5-mini` (Singh et al., 2026) as both test-taker and grader—the LLM that the memory system augments, not the system's internal model—so that cross-system differences are attributable to the memory system rather than the underlying LLM. Full prompts and grading details are given in Appendix E.

Human Validation. All dataset examples are manually verified for correctness. On 100 manually graded examples, `gpt-5-mini` answers 98% correctly and classifies the error type 98.4% correctly. Table 6 (Appendix C) gives the per-class confusion matrix against human labels over the 364 memory-level classifications these examples induce; every disagreement is a storage/summary or summary/retrieval swap.

5.2 Memory Systems

We evaluate four open-source modern memory systems—**Mem0** (Chhikara et al., 2025), **A-MEM** (Xu et al., 2025), **SimpleMem** (Liu et al., 2026), and **StructMem** (Xu et al., 2026)—and study in Section 6 how the success rate and error types vary with architecture, retrieval depth k, and the strength of the system's internal model.

No-compression baseline. To separate error intrinsic to memory compression from error specific to a system's design, we additionally evaluate **Verbatim-RAG**, which performs no summarization: it stores each conversation verbatim as a single embedded entry and retrieves the top-k entries by cosine similarity. By construction it cannot commit storage or summary errors.

6 Experiments

We evaluate open-source memory systems on MEMFAIL. The main text shows representative examples of the most interesting findings; more complete evaluation results are in Appendix D.

Q1: How does performance scale with k, the number of retrieved memories? Figure 1 shows that **MEMFAIL is difficult even for state-of-the-art memory systems**, and that performance scales poorly with k. The exception is *Coexisting-Facts*, which naturally benefits from larger k since coexisting facts are more likely to be retrieved, even by chance. Performance for a given system also varies substantially across tasks: because MEMFAIL's tasks isolate specific weaknesses, they reveal how architectures induce distinct failure modes. StructMem performs strongly on most tasks but fails spectacularly on *Coexisting-Facts*, while Mem0 shows the opposite pattern (analyzed in Q5). Figure 8 in

Appendix D breaks down error types per system, revealing how different tasks elicit different failure modes (concrete failure examples in Appendix A):

- *Coexisting-Facts* induces **retrieval failures**: most systems fail to associate all related facts with the query at retrieval time.
- *Conditional-Facts (Hard)* induces **summary failures**: all systems over-compress, either *altering* the original message or *stripping away* precise, critical details—both mislead the LLM into the wrong judgment.
- *Persona-Retrieval* induces **summary failures** via over-compression of long personas; the exception is Mem0, which fails to store all details in the first place due to its LLM-tool-call update mechanism.
- *Long-Hop* unsurprisingly induces **retrieval failures**: systems fail to capture long-range causal relationships between seemingly disjoint entities.

Except for Mem0, systems generally do not exhibit **storage failures**; failures stem almost entirely from incorrect summarization or retrieval.

Reasoning failures do not absorb retrieval noise. A natural concern is that increasing k also puts more similar but irrelevant memories into the test-taker's context. In that case, some of the failures we call reasoning failures could actually be retrieval failures. We would therefore expect reasoning failures to increase as k increases. But Figures 6–9 show that this does not happen. On every dataset except *Persona-Retrieval (Misleading)*, reasoning failures are rare and do not significantly increase from k=5 to k=20.

The exception is also useful for understanding what is going on. On misleading persona queries, larger k brings in more distractor personas. The test-taker sometimes says it does not remember the queried person and then answers based on the habits of a different person (Appendix A.4). This seems to be a limitation of the answerer LLM rather than the architecture. More importantly, it only happens on the one task where all four systems already perform well. On the tasks that drive our architectural claims, reasoning failures are close to zero.

Key Takeaway

Increasing the number of retrieved memories yields performance gains on tasks where retrieval errors are common; it yields marginal returns when summary errors are the bottleneck.

Q2: How does accuracy scale with the strength of the model used by the memory system? Figure 2 shows stronger models do not improve accuracy while sometimes degrading accuracy on most tasks; smarter reasoning models can generate overly verbose memories that pollute the agent's context.

Key Takeaway

Unlike other applications of LLM agents where using a more intelligent model can increase performance on benchmarks, modern memory systems are bound by architectural constraints rather than model intelligence.

Q3: What does MEMFAIL reveal about the tradeoff between performance and token consumption? Figure 3 shows how performance scales with the token usage of each memory system. Performance generally scales positively with token consumption on *Persona-Retrieval* and *Conditional-Facts (Hard)*: in general, for tasks bottlenecked by summary failures, increasing the number of tokens leads to performance gains. By contrast, retrieval tasks can actually see a performance drop from using more tokens. This is especially evident for *Coexisting-Facts*, where storing large memories "pollutes" the semantic embeddings, hurting retrieval.

Key Takeaway

The historical consensus is that more tokens is an easy way to obtain performance gains (i.e., test-time scaling); however, for memory systems, performance scales with increased memory storage in a way that is *highly task-dependent*.

Q4: How much of the failure is intrinsic to compression? Figure 1 plots Verbatim-RAG, which stores conversations uncompressed, alongside the compression-based systems. It almost never beats *all* memory systems but frequently beats at least one, another view of the tradeoff in Q1. It is strongest on *Conditional-Facts*, whose

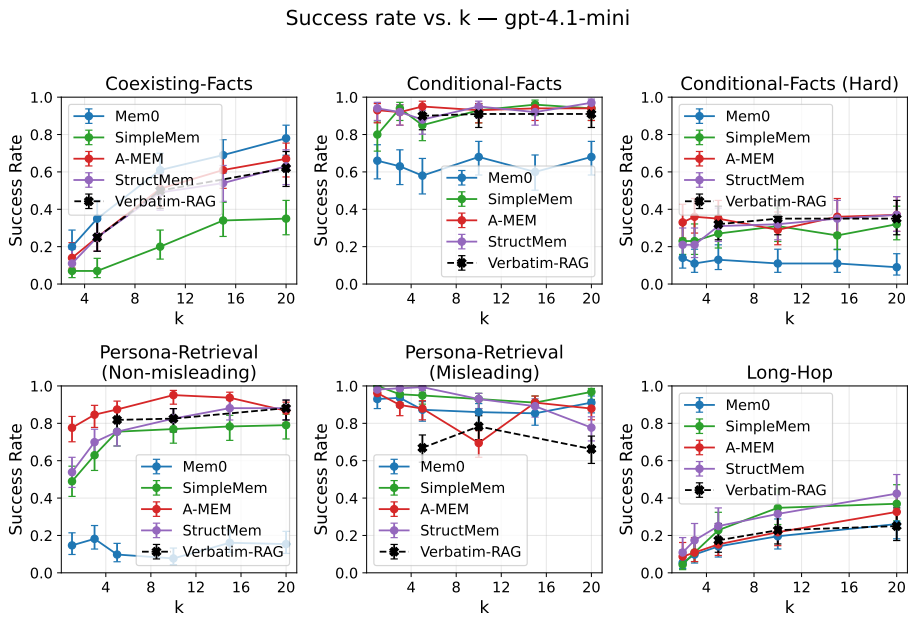

Figure 1: Performance of memory systems using GPT-4.1-mini internally, with the no-compression **Verbatim-RAG** baseline (black dashed) for reference; having no internal LLM, it is independent of that axis. All confidence intervals use 95% Wilson score binomial intervals.

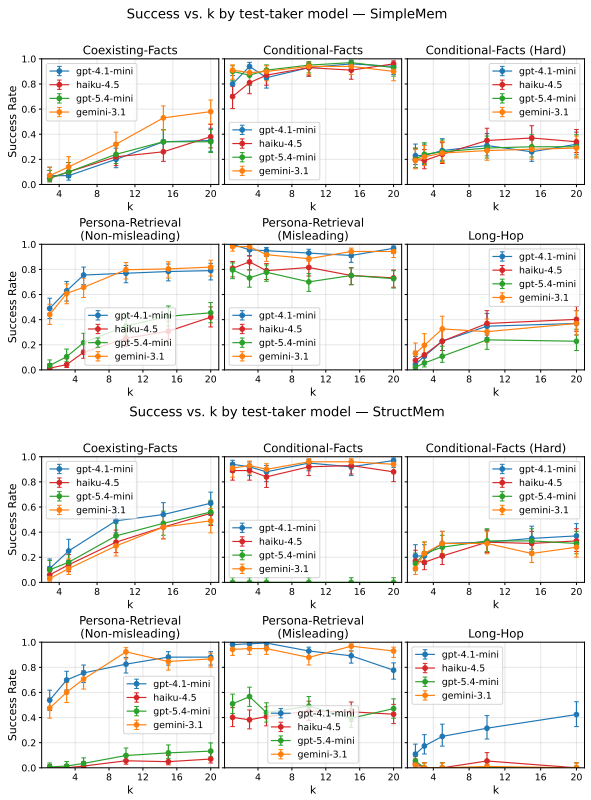

Figure 2: Performance of StructMem and SimpleMem as a function of their internal model. Mem0 and A-MEM follow the same trend, as shown in Appendix D, but we exclude them here for conciseness. Equipping the system with a stronger internal model does not lead to performance gains.

errors are dominated by summarization—which verbatim storage removes by construction—and weakest on *Long-Hop* and *Coexisting-Facts*, whose errors are dominated by retrieval: with no summarization, nothing normalizes or links scattered facts, so retrieving *all* required entries is harder, not easier. It is also among the most token-expensive systems we evaluate (Figure 3), on par with A-MEM.

Key Takeaway

Removing summarization eliminates summary failures but worsens retrieval failures: faithful storage and retrievable storage are themselves in tension.

Q5: What does MEMFAIL reveal about design choices in memory systems? We provide insights into the key architectural decisions of the tested memory systems:

- **LLM-based memory updates.** Mem0 updates memory via LLM tool calls. For short experiences (e.g., one-sentence *Coexisting-Facts* entries) the LLM accurately stores the content, but for longer experiences it fails to issue enough tool calls to capture all details—evidenced by Mem0's high storage error rate on *Persona-Retrieval*, which has the longest entries (Figure 1).
- **Semantic vector stores.** A-MEM stores con-

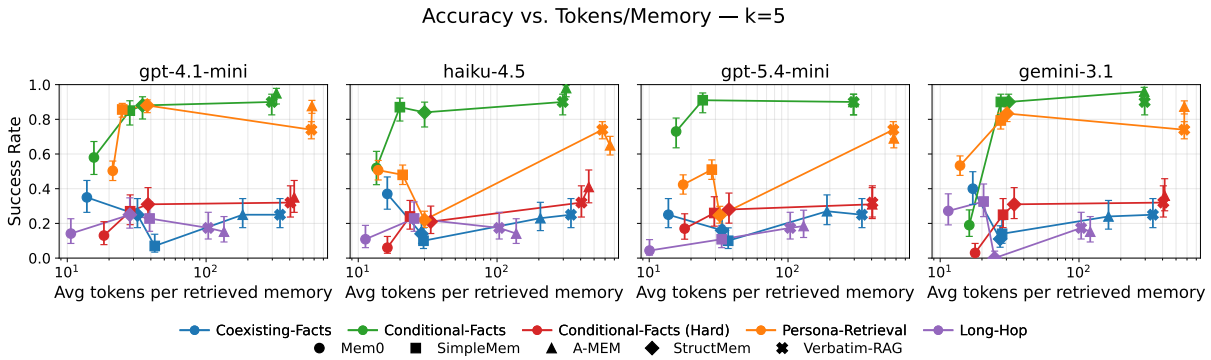

Figure 3: Per-model performance on MEMFAIL relative to the average number of tokens per memory, at $k = 5$. Marker *shape* denotes the memory system—including **Verbatim-RAG**, whose stored memories are whole conversations—and *color* the dataset; each colored line connects the systems on one dataset, ordered by token cost. A system's horizontal position is fixed by its architecture, which sets the granularity of a stored memory (A-MEM writes long descriptive notes, Mem0 short atomic facts), so the quantity of interest is the *sign of the slope* of each colored line.

versations as descriptive notes in a vector database. Figure 3 shows this explodes token usage for little gain: it reduces summary errors but, surprisingly, does not improve retrieval-heavy tasks, since RAG-style embeddings fail to capture inter-entity relationships in isolation. Mem0 by comparison gains efficiency by compressing more intelligently—removing noise without stripping critical detail.

- **Graph-based architectures.** Prior work proposes graphs to improve causal reasoning. MEMFAIL confirms that StructMem, a graph-based method, performs well on *Long-Hop* and *Conditional-Facts* but poorly on general information retrieval. We show there may be a tradeoff between flat vector stores and graph-based architectures: graphs improve inter-entity relationship modeling but over-commit to structure/decomposition and struggle to represent longer semantic ideas.

Q6: What does MEMFAIL reveal about future directions for memory systems research? Ultimately, we want memory systems that excel across all MEMFAIL tasks without major tradeoffs. We propose two under-explored directions that may help accomplish this goal.

Mixture-of-memories architectures. State-of-the-art systems commit to a single backend (vector store, graph, or hierarchical), but MEMFAIL shows that different architectures excel at different tasks; hybrid systems that route memories to the appropriate substore could combine these strengths. For example, a system could route causal experiences to a graph (StructMem-style) and persona knowledge to a flat vector store (A-MEM-style).

Task-based token scaling. Figure 3 shows that accuracy scales with token consumption only for certain task types; improved systems may dynamically size generated memories to match the type of incoming information. For many tasks, more tokens is not always better. We also see that A-MEM uses substantially more tokens than other systems without commensurate performance gains, further motivating the need for intelligent scaling.

7 Conclusion

We introduce MEMFAIL, a benchmark for LLM memory systems designed to expose specific failure modes. Our evaluation reveals that current systems are bound by architectural constraints that cannot be addressed by simply spending more tokens or using a more intelligent model. To our knowledge, MEMFAIL is the first benchmark to enable a fine-grained analysis of failure modes exhibited by different memory systems. MEMFAIL supports evaluation of any memory system implementing the API from Section 3, and we have open-sourced all datasets and evaluation code so that future memory systems may be benchmarked using MEMFAIL.

Limitations

While MEMFAIL provides a fine-grained view into the failure modes of modern memory systems, several limitations of our methodology are worth highlighting. Every dataset in MEMFAIL is gen-

erated by an LLM (`gpt-4.1-mini`, `gpt-5-mini`, or `gpt-5`) under structured generation prompts and then filtered. Although we manually verify all dataset entries for correctness, the resulting distribution of conversations, entities, and phrasings may be narrower than what a deployed memory system would encounter in practice. Performance on MEMFAIL is designed to be interpreted as a diagnostic signal about specific failure modes rather than as a prediction of end-to-end performance in deployment. Additionally, we evaluate four open-source systems (Mem0, A-MEM, SimpleMem, StructMem) that all expose `store_conversation`, `retrieve_memories`, and `get_all_memories`. While our framework is intentionally designed to encompass the majority of memory systems, it may be harder to use when evaluating systems with implicit or learned memory, or with fine-tuned-weight memory. Our work also does not analyze the latency of memory systems; although the MEMFAIL evaluation pipeline actually provides thorough metrics for memory retrieval time and evaluation time, this work intentionally chooses not to focus on latency in this work as the results are not as insightful.

Ethical considerations

MEMFAIL surfaces hidden failure modes in LLM memory systems, helping researchers and practitioners build more reliable personal assistants. However, we also recognize that great care should be taken in using MEMFAIL responsibly so that it is not used adversarially to target specific weaknesses of real-world memory systems. We have open-sourced all datasets and code so that the research community may use MEMFAIL to improve the security and accuracy of memory systems. Additionall, all datasets are synthetically generated, fake personas and contain no personal or identifying information about real individuals. The benchmark datasets in MEMFAIL are generated by LLMs (`gpt-4.1-mini`, `gpt-5-mini`, and `gpt-5`) under structured prompts, as detailed in Section 4 and Appendix B. AI assistants were also used to help polish writing in this paper; all technical content, claims, and analyses are the authors' own. Note that Mem0 uses an Apache 2.0 license, StructMem, SimpleMem, and A-MEM use an MIT license; we use these code artifacts only for this research which is consistent with their licenses. We have publicly released our code and datasets under a MIT license.

References

Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. 2025. MemoryBench: A Benchmark for Memory and Continual Learning in LLM Systems. *arXiv preprint*. ArXiv:2510.17281 [cs].

Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. 2024. L-Eval: Instituting Standardized Evaluation for Long Context Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14388–14411, Bangkok, Thailand. Association for Computational Linguistics.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. *Preprint*, arXiv:2504.19413.

Yiming Du, Hongru Wang, Zhengyi Zhao, Bin Liang, Baojun Wang, Wanjun Zhong, Zezhong Wang, and Kam-Fai Wong. 2024. PerLTQA: A Personal Long-Term Memory Dataset for Memory Classification, Retrieval, and Fusion in Question Answering. In *Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10)*, pages 152–164, Bangkok, Thailand. Association for Computational Linguistics.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models? *arXiv preprint*. ArXiv:2404.06654 [cs].

Chuanrui Hu, Xingze Gao, Zuyi Zhou, Dannong Xu, Yi Bai, Xintong Li, Hui Zhang, Tong Li, Chong Zhang, Lidong Bing, and Yafeng Deng. 2026a. EverMemOS: A Self-Organizing Memory Operating System for Structured Long-Horizon Reasoning. *arXiv preprint*. ArXiv:2601.02163 [cs].

Yuanzhe Hu, Yu Wang, and Julian McAuley. 2026b. Evaluating memory in LLM agents via incremental multi-turn interactions. In *The Fourteenth International Conference on Learning Representations*.

Xiaojun Jia, Jie Liao, Simeng Qin, Jindong Gu, Wenqi Ren, Xiaochun Cao, Yang Liu, and Philip Torr. 2026.

Skillject: Automating stealthy skill-based prompt injection for coding agents with trace-driven closed-loop refinement. *arXiv preprint*.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory OS of AI Agent. *Preprint*, arXiv:2506.06326.

Jiho Kim, Woosog Chay, Hyeonji Hwang, Daeun Kyung, Hyunseung Chung, Eunbyeol Cho, Yeonsu Kwon, Yohan Jo, and Edward Choi. 2025. DialSim: A Dialogue Simulator for Evaluating Long-Term Multi-Party Dialogue Understanding of Conversational Agents. *arXiv preprint*. ArXiv:2406.13144 [cs].

Wai-Chung Kwan, Xingshan Zeng, Yufei Wang, Yusen Sun, Liangyou Li, Yuxin Jiang, Lifeng Shang, Qun Liu, and Kam-Fai Wong. 2024. M4LE: A Multi-Ability Multi-Range Multi-Task Multi-Domain Long-Context Evaluation Benchmark for Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15568–15592, Bangkok, Thailand. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Preprint*, arXiv:2005.11401.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph E. Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. 2023. How long can context length of open-source LLMs truly promise? In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. 2026. SimpleMem: Efficient Lifelong Memory for LLM Agents. *Preprint*, arXiv:2601.02553.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating Very Long-Term Conversational Memory of LLM Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, Bangkok, Thailand. Association for Computational Linguistics.

Elliot Nelson, Georgios Kollias, Payel Das, Subhajit Chaudhury, and Soham Dan. 2024. Needle in the Haystack for Memory Based Large Language Models. *arXiv preprint*. ArXiv:2407.01437 [cs].

OpenAI. 2022. tiktoken: A fast BPE tokeniser for use with OpenAI's models. `https://github.com/openai/tiktoken`. Accessed: 2026-05-06.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. MemGPT: Towards LLMs as Operating Systems. *Preprint*, arXiv:2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. *Preprint*, arXiv:2304.03442.

Mitchell Piehl, Zhaohan Xi, Zuobin Xiong, Pan He, and Muchao Ye. 2026. ER-MIA: Black-Box Adversarial Memory Injection Attacks on Long-Term Memory-Augmented Large Language Models. *arXiv preprint*. ArXiv:2602.15344 [cs].

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. Zep: A Temporal Knowledge Graph Architecture for Agent Memory. *arXiv preprint*. ArXiv:2501.13956 [cs].

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. *Preprint*, arXiv:2303.11366.

Alina Shutova, Alexandra Olenina, Ivan Vinogradov, and Anton Sinitsin. 2026. Evaluating memory structure in LLM agents. In *ICLR 2026 Workshop on Memory for LLM-Based Agentic Systems (MemAgents)*.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, A. J. Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 467 others. 2026. OpenAI GPT-5 System Card. *arXiv preprint*. ArXiv:2601.03267 [cs].

Haoran Tan, Zeyu Zhang, Chen Ma, Xu Chen, Quanyu Dai, and Zhenhua Dong. 2025. MemBench: Towards More Comprehensive Evaluation on the Memory of LLM-based Agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19336–19352, Vienna, Austria. Association for Computational Linguistics.

Hanling Tian, Zeyang Sha, Jingying Wang, Yuhang Liu, Zhehao Huang, and Xiaolin Huang. 2025. InjecMEM: Memory injection attack on LLM agent memory systems. OpenReview.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. In *The Thirteenth International Conference on Learning Representations*.

Buqiang Xu, Yijun Chen, Jizhan Fang, Ruobin Zhong, Yunzhi Yao, Yuqi Zhu, Lun Du, and Shumin Deng. 2026. StructMem: Structured Memory for Long-Horizon Behavior in LLMs. *Preprint*, arXiv:2604.21748.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-Mem: Agentic Memory for LLM Agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Zeyu Zhang, Quanyu Dai, Luyu Chen, Zeren Jiang, Rui Li, Jieming Zhu, Xu Chen, Yi Xie, Zhenhua Dong, and Ji-Rong Wen. 2024. MemSim: A Bayesian Simulator for Evaluating Memory of LLM-based Personal Assistants. *arXiv preprint*. ArXiv:2409.20163 [cs].

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2023. MemoryBank: Enhancing Large Language Models with Long-Term Memory. *Preprint*, arXiv:2305.10250.

A Failure-Mode Examples

This appendix gives concrete examples of the failure modes described in Section 3.2. All examples are drawn from the recorded JSON evaluation traces. Each example reports the original asserted information, the memories retrieved at query time, the graded question, the model answer, and the resulting diagnosis.

We distinguish carefully between failures of the memory system and failures of the downstream test-taker. A reasoning failure is assigned only when the retrieved memory is correct and relevant, but the final answer contradicts it or fails to incorporate it. If the retrieved memories are empty, irrelevant, or missing one or more required facts, the error is attributed to storage or retrieval rather than downstream reasoning.

A.1 Summary failures

A summary failure occurs when the system stores information, but the stored memory does not faithfully preserve the original content needed for the later query. In the examples below, the system stores a related memory, but the stored version loses a critical condition, threshold, or exclusivity qualifier.

Summary failure: Thorne's motivation threshold

Original asserted information. "Thorne starts building miniature clockwork automatons only when his motivation is 8 out of 10 or higher."

Retrieved memories at query time.

- "At 2026-04-28T06:22:25, Thorne builds tiny clockwork automatons that are wound obsessively and tested until each automaton ticks with a personality."
- "At 2026-04-28T06:22:25, Thorne keeps a motivation chart on the workshop wall with stickers for weeks when Thorne's focus spikes."

Question. "Thorne says he's at about 5/10 motivation today — will he start building one of his miniature clockwork automatons now?"

Model answer. "Probably not a full new build right now — but he might tinker or do a small related task."

Ground truth. No. He requires motivation of 8/10 or higher before he begins building automatons.

Diagnosis. This is a summary failure. The memories preserve that Thorne builds automatons and that his work is associated with motivation spikes, but they lose the specific numeric threshold "8 out of 10 or higher." Because the retrieved memory does not contain the exact condition, this is not a reasoning failure.

Summary failure: Captain Aurelio's black-tea trigger

Original asserted information. "Captain Aurelio recites a two-line naval proverb only when someone offers him a cup of black tea."

Retrieved memories at query time.

- "Three candidate two-line naval proverbs in Captain Aurelio's voice are: 'Mind the swell, tend the line. Steady hands keep true time.'; 'Respect the sea, respect the men. Small comforts steady the soul.'; and 'Chart the calm, brace the storm. Keep your cup — keep your calm.'"
- "Captain Aurelio's ritual proverb is a two-line naval proverb spoken aloud with a soft and concise cadence unchanged over decades."

Question. "We offered biscuits to Captain Aurelio but no drinks were handed out — will he recite his two-line naval proverb now?"

Model answer. "Maybe — not certain."

Ground truth. No. He only recites the

proverb when someone gives him a cup of black tea.

Diagnosis. This is a summary failure. The stored memories preserve the proverb ritual, and some stored information links the ritual to a cup, but the critical trigger is weakened: the memory does not preserve that the cup must be black tea or that the condition is exclusive.

Summary failure: Mochi's five-minute laser-pointer threshold

Original asserted information. "Mochi drags a catnip mouse into the laundry basket and naps on it after an uninterrupted five-minute laser-pointer chase."

Retrieved memories at query time.

- "Mochi drags a battered catnip mouse into the laundry basket and curls up on top of the catnip mouse for naps."
- "Mochi is a devoted chaser who sprints across rugs with feline intensity during play sessions."

Question. "Mochi just chased the laser pointer for six straight minutes — will she bring a catnip mouse into the laundry basket and nap on it now?"

Model answer. "Very likely — but not guaranteed."

Ground truth. Yes. Six minutes satisfies the five-minute laser-pointer condition.

Diagnosis. This is a summary failure. The stored memories preserve the behavior and a general association with chasing, but they lose the exact uninterrupted five-minute laser-pointer threshold.

A.2 Storage failures

A storage failure occurs when information from the initial interaction is not incorporated into memory in a form that can later be retrieved. In the examples below, the original behavior or condition is absent from the memory store, so the later query cannot be answered from memory.

Storage failure: Pixel's kitchen-windowsill condition

Original asserted information. Pixel watches birds only on the kitchen windowsill.

Stored memory state. The memory store contains that Pixel sits on a windowsill watching birds and is often drawn to the sunlit, feeder-facing perch at the kitchen window, but it does not preserve the exclusivity condition that Pixel does this only on the kitchen windowsill.

Question. The graded query asks whether Pixel would perform the behavior away from the kitchen windowsill.

Ground truth. No. The behavior is restricted to the kitchen windowsill.

Diagnosis. This is a storage failure. The behavior is present in memory, but the condition requiring exclusivity is not stored; the memory only says Pixel is usually or often drawn to the kitchen windowsill.

Storage failure: watermelon seeds

Original asserted information. "Eating watermelon seeds causes watermelons to grow inside your stomach."

Initial essay. The essay repeatedly states that watermelon seeds can take root inside the stomach lining and lead to the growth of small watermelons internally.

Retrieved memories at query time. None.

Question. "Can planting watermelon seeds you swallow help you grow a watermelon garden inside your body?"

Model answer. "No, swallowing watermelon seeds will not grow a watermelon garden inside your body."

Diagnosis. This is a storage-side failure under a faithful-memory criterion: the original assertion is not available in retrieved memory

when the implication question is asked. The answer itself is not a reasoning failure, because the model was not shown a correct retrieved memory that it failed to use.

Storage failure: fortune-cookie origin

Original asserted information. "Fortune cookies originated in Los Angeles."

Retrieved memories at query time. None.

Question. "Which city is known as the birthplace of fortune cookies?"

Model answer. "The city known as the birthplace of fortune cookies is San Francisco, California."

Diagnosis. This is a storage-side failure under a faithful-memory criterion: the Los Angeles claim from the initial interaction is not surfaced at query time. The model answers from prior knowledge rather than from a retrieved conversation-specific memory.

A.3 Retrieval failures

A retrieval failure occurs when the memory system fails to return the memories needed for the current query, or when it returns memories that are unrelated to the query. These are not reasoning failures: the downstream model cannot use the correct memory if the retriever never provides it.

Retrieval failure: Amara's hometown condition is not retrieved

Original asserted information. "Amara writes a postcard to her childhood teacher only when she is physically in her hometown."

Retrieved memories at query time.

- "Amara writes a warm, handwritten postcard to the teacher who shaped Amara, but only during certain visits to places."
- "Amara keeps a stack of postcards collected from places that Amara loves."

Question. "We're vacationing at a coastal resort far from where she grew up — would Amara write a postcard to her childhood teacher now?"

Model answer. "Maybe — she does sometimes, but not every trip."

Ground truth. No. She only writes those postcards when physically in her hometown, and the resort is not her hometown.

Diagnosis. This is a retrieval failure. The memory store contains enough information to recover the hometown-only condition, but the retrieved memories only say "certain visits to places" and do not include the hometown-specific trigger.

Retrieval failure: Eldon's victory-jig behavior is not retrieved

Original asserted information. "Eldon performs a private victory jig after the guildmaster gives him a thumbs-up."

Retrieved memories at query time.

- "At 2026-04-28T06:37:58, the apprentices mutter good-natured barbs about Eldon's modesty."
- "At 2026-04-28T06:37:58, the guildmaster is described as a blunt woman with an economical sense of praise who gives a curt thumbs-up on the rare occasions when the guildmaster is pleased."

Question. "The guildmaster just gave Eldon a thumbs-up during the meeting — will Eldon do his victory jig right now?"

Model answer. "Unlikely. Eldon's known to be modest, so a public, ostentatious 'victory jig' isn't consistent with his usual behavior."

Ground truth. Yes. The thumbs-up is exactly the trigger for Eldon's private victory jig.

Diagnosis. This is a retrieval failure. The retrieved memories contain the thumbs-up condition, but not the private victory-jig behavior. Since the required behavior is missing from the retrieved context, the downstream model cannot recover the conditional rule.

Retrieval failure: long-hop chain retrieves only the first link

Original facts.

- "Carlos adores rainy afternoons because they slow his day."
- "Rainy afternoons prompt Carlos to bake banana bread."
- "Banana bread makes Carlos post cheerful photos."

Retrieved memories at query time.

- "On foggy mornings Priyanka chooses to wear bright scarves."
- "Carlos adores rainy afternoons because they slow his day."

Question. "On the kind of afternoons Carlos adores, what does he end up doing?"

Choices. A. polish dress shoes B. post cheerful photos C. schedule a dentist appointment D. back up files E. update the calendar

Model answer. `{"selected_choice":"D"}`

Ground truth. B. post cheerful photos.

Diagnosis. This is a retrieval failure. The retriever returned the first link in the chain, but it did not retrieve the two remaining supporting facts connecting rainy afternoons to banana bread and banana bread to posting cheerful photos.

Retrieval failure: blood-color query retrieves fortune-cookie memory

Original asserted information. "Veins appear blue because deoxygenated blood is inherently blue in color."

Retrieved memories at query time. "The city known as the birthplace of fortune cookies is San Francisco."

Question. "What color would blood appear if it were exposed to air outside the body?"

Model answer. "When blood is exposed to air outside the body, it generally appears bright red."

Diagnosis. This is a retrieval failure: the retrieved memory is about fortune cookies and is unrelated to the blood-color question. The final answer is correct, but that does not make retrieval successful. The trace shows harmless retrieval contamination.

A.4 Reasoning failures

A reasoning failure occurs when the relevant memory is stored and retrieved, but the downstream test-taker still fails to use it correctly. In these cases, the memory system has provided the information needed to answer the query, but the model hedges, contradicts the retrieved memory, or reasons from an irrelevant implication rather than applying the remembered rule.

Reasoning failure: Marek's seven-year threshold

Original asserted information. "Marek gives his carved wooden coin as a keepsake only to people he has been friends with for at least seven years."

Retrieved memories at query time.

- "By the time a name has lived in Marek's friendship ledger for seven years, Marek is usually ready to offer the carved wooden coin as a gift."
- "Marek carries a single carved wooden coin in the pocket of his coat and treats the coin like a talisman to be handed out sparingly."

Question. "I've known Marek for eight years — will he give me his carved wooden coin?"

Model answer. "Probably — but not guaranteed."

Ground truth. Yes. The question states that the friendship has lasted eight years, which satisfies the seven-year condition.

Diagnosis. This is a reasoning failure. The retrieved memory contains both the behavior and the seven-year threshold, and the query context satisfies that threshold. The model quotes the relevant memory but still hedges instead of giving the definite answer supported by the memory.

Reasoning failure: Eldra's autumn-only bookmark rule

Original asserted information. Eldra weaves pressed leaves into bookmarks during autumn.

Retrieved memories at query time. The retrieved memories explicitly state that Eldra presses autumnal foliage and weaves the fragile leaves into slim, hand-tied bookmarks, and that these bookmarks are tied with twine and passed at the book club.

Question. The graded query asks whether Eldra would weave pressed leaves into bookmarks in March.

Model answer. The model answers "maybe" and speculates about leftover finished bookmarks rather than applying the seasonal condition.

Ground truth. No. March is not autumn, so the autumn-only condition is not satisfied.

Diagnosis. This is a reasoning failure. The retrieved memory contains the correct seasonal condition and the bookmark behavior, but the model does not apply the condition to the March query. Instead, it introduces speculation about leftover bookmarks and fails to reach the required negative answer.

Reasoning failure: Nacho's upstairs-balcony condition

Original asserted information. "Nacho launches a loud four-minute bark routine at passersby only when he is perched on the upstairs balcony."

Retrieved memories at query time.

- "Nacho usually begins the theatrical barking performance from the high, outside perch on the upstairs balcony."
- "Nacho is a compact terrier, very alert and loud for his size, has a practiced approximately four-minute barking routine at passersby, loves high vantage points with a favorite being an upstairs balcony or ledge, patrols the balcony at dawn, and likes pats and a quiet treat after barking."

Question. "The delivery person is at the gate and Nacho is sitting by the front door downstairs; will he start his upstairs balcony barking routine?"

Model answer. "Probably — he often goes up to the balcony to start the full barking routine, so if the delivery person is at the gate he's likely to either trot upstairs to the balcony and launch into his usual performance or at least give some alarms from the door."

Ground truth. No. Nacho only performs that barking routine from the upstairs balcony, and he is downstairs now.

Diagnosis. This is a reasoning failure. The retrieved memories correctly identify the upstairs balcony as the location for the barking routine, and the query says Nacho is downstairs. The model nevertheless predicts that he will probably perform the routine, adding an unsupported transition in which he might trot upstairs.

B Benchmark Construction Details

This appendix gives the full pipeline specifications, model choices, deduplication thresholds, and complete sampling pools that were summarized in Section 4.

B.1 Generators and shared infrastructure

All five datasets are produced by single-pass batched OpenAI calls with `response_format={"type": "json_object"}` and up to three retries per batch on validation failure. Generator models are pinned per dataset: *Conditional-Facts (Easy)* and *Coexisting-Facts* use `gpt-4.1-mini`; *Conditional-Facts (Hard)* and

Persona-Retrieval use `gpt-5-mini`; *Long-Hop* uses `gpt-5`. Every dataset uses a fixed random seed of 42 and writes a `generation_config.json` alongside the CSV, recording the seed, the git commit, the deduplication threshold, the per-row counts before and after deduplication, and the model name, so that any row in any released artifact is fully traceable back to the generator that produced it.

Deduplication is performed with MinHash LSH at task-specific Jaccard thresholds over a task-specific dedup key: *Conditional-Facts* dedups at 0.8 over the wrapping essay text; *Coexisting-Facts* dedups at 0.7 over the scenario question; *Persona-Retrieval* dedups at 0.7 over the essay text; *Long-Hop* dedups at 0.7 over each *individual* fact, and any chain that contains a fact colliding with an already-kept fact is dropped wholesale.

B.2 Conditional-Facts: condition types

The condition type is sampled uniformly from a fixed list of 32 types, grouped by category in Table 3. For `pet` entities the condition type is restricted to externally observable triggers (`time_of_day`, `weather`, `temperature`, `location`, `noise_level`, `lighting`, `food_or_drink_present`, `prior_activity`, `company`) so that no pet is asked to depend on abstract internal states.

Group	Condition types
Time	`time_of_day`, `day_of_week`, `season`, `time_elapsed`
Environment	`weather`, `temperature`, `location`, `noise_level`, `lighting`
Physical	`hunger_level`, `energy_level`, `pain_or_discomfort`, `sobriety`
Emotional	`mood`, `stress_level`, `anxiety_level`, `motivation_level`
Social	`company`, `social_setting`, `relationship_closeness`
Task	`task_type`, `workload`, `completion_state`
Sensory	`music_playing`, `scent`, `food_or_drink_present`
Relational	`conflict_state`, `approval_received`, `request_made`
Habitual	`prior_activity`, `frequency_cap`, `streak_state`

Table 3: The full list of 32 condition types for *Conditional-Facts*.

B.3 Conditional-Facts: the Hard decomposition

The Hard variant decomposes the original conditional fact into exactly three sentences spread across an 8–12 sentence essay:

- a **behavior** sentence describing what the entity does as a tendency or habit, *without* naming the trigger;
- a **condition** sentence establishing when C holds as part of the entity's life context or environment, *without* naming the behavior;
- a **link** sentence using soft co-occurrence language ("it's usually in those moods that...," "by then...," "most of the time it happens...") that connects the two without an explicit conditional.

The generator is forbidden from using any explicit conditional connective ("only when," "unless," "except when," "but only if," "whenever," "if," "only after," "only if") anywhere in the essay, and the generation prompt mandates that no two of the three rule-bearing sentences are adjacent: at least one unconditional sentence sits between any pair. Easy and Hard rows share the same entity and condition specs, so any difference in performance can be directly attributed to the distribution of the rule across sentences.

Example: *Conditional-Facts* (Hard)

Entity: Gideon **Behavior:** paints tiny sailboats inside used teacups **Condition:** when he feels wistful after finding a childhood keepsake

Essay (excerpt): "...He has a habit of painting tiny sailboats inside those used teacups, as if making miniature voyages.... He collects small keepsakes from his childhood—a dented tin soldier, a faded postcard tucked in a book. Those artifacts sometimes resurface and leave him feeling peculiarly nostalgic and soft.... It's usually in those wistful, quiet moods that this little, private ritual reappears...."

Question: "We just found Gideon's old toy soldier in the attic; would he paint tiny sailboats in a teacup right now?"

Ground truth: "Yes—finding a childhood keepsake makes him feel wistful, which is exactly when he paints sailboats in teacups."

B.4 Coexisting-Facts: full list of preference categories

The 100 preference categories are partitioned into thematic groups in Table 4. Each category yields exactly one row, and the per-row preference count $N \in \{2, 3, 4, 5\}$ is drawn uniformly per row.

Group	Categories
Food & drink	foods; cuisines; types of desserts; drinks; coffee types; tea varieties; breakfast foods; street foods; snack types; cocktail types; cooking methods; restaurant types
Entertainment & media	music genres; types of movies; book genres; tv show genres; video game genres; podcast topics; comedy styles; documentary subjects; types of concerts; types of museums; sports to watch on TV; types of theater
Physical activity	sports; outdoor activities; physical exercises; fitness class types; martial arts styles; dance styles; cycling types; swimming styles; types of yoga; hiking styles
Lifestyle & personal style	fashion styles; home decor styles; makeup styles; jewelry types; shoe styles; hair care styles; skincare routines; bag styles; hat styles
Travel & places	travel destinations; vacation types; nature environments; city neighborhood types; outdoor dining settings
Hobbies & creativity	hobbies; crafts; art styles; photography styles; creative writing styles; musical instruments; types of puzzles; journaling styles; home improvement projects; garden types; houseplants
Social & leisure	weekend activities; party activities; dating activities; types of social gatherings; board games; card games; conversation topics
Tech & learning	programming languages; tech interests; academic subjects; skills to learn; languages to learn; language learning methods; study techniques; note-taking methods; productivity methods
Wellness & self-care	mental health activities; meditation types; self-care activities; sleep habits; morning routine habits; evening routine habits
Career & finance	career fields; types of volunteering; investment types; budget planning methods
Media consumption	social media content types; news formats; podcast formats; types of reading material; audio listening formats; content creation types
Misc	animals; dog breeds; collectibles; subscription box types; car types; home organization methods; types of naps; camping styles; astrology interests

Table 4: The full list of 100 preference categories for *Coexisting-Facts*, grouped thematically.

B.5 Persona-Retrieval: name and flavor pools

Entity names are drawn from a fixed pool of 30 diverse names; persona "flavors" are drawn from a fixed pool of 30 flavors. Both pools are listed in Table 5. Within a row, the entity name is sampled once, and the three misleading-slot distractors are then drawn without replacement from the remaining 29 names so that no two slots in a row reuse the same distractor and the entity itself is never used as its own distractor.

B.6 Long-Hop: full pipeline

Generation. Chains are generated in batches with `gpt-5`, accumulating a running list of one-line summaries (head $\rightarrow$ terminal) of every previously accepted chain so the model is steered toward narratively novel chains. The system prompt enforces every hard rule listed in Section 4: exactly $K{+}1$ statements, exactly $K{+}2$ distinct anchors, statement i links anchors i and $i{+}1$ via an explicit relation phrase, every fact is self-contained (any pronoun's antecedent must appear in the same sentence), every fact is subjective (no encyclopedic claims), and middle / terminal anchors appear only in the two facts that border them. The graded question references the head anchor at least once by name, asks about the terminal anchor, and never names any intermediate anchor.

Local validation. Each generated chain is normalized (lowercased, punctuation stripped, whitespace collapsed) and checked structurally: the chain must contain exactly $K{+}1$ non-empty facts and $K{+}2$ distinct anchors; every anchor must appear literally as a substring in each of the (one or two) facts it borders; the head anchor (or its last $\geq$ 3-character token, to tolerate morphological variants such as "eat apples" $\rightarrow$ "eating apples") must appear in the graded question; no intermediate anchor may appear in the question; and the ground-truth answer must contain the terminal anchor as a substring (or vice versa).

Cross-chain conflict check. Survivors are passed through an LLM judge (`gpt-5`) with a global pass plus overlapping sliding windows. The judge drops chains that contradict another chain (incompatible claims about the same anchor), share a distinctive proper-noun or distinctive composite phrase with another chain, or retell the same narrative with the same anchors in the same role. Generic single-word concepts repeating across chains ("sleep," "bored," "tea") are explicitly

Names (30)	**Persona flavors** (30)
Ava Thompson; Liam Carter; Maya Patel; Noah Brooks; Zoe Kim; Ethan Rivera; Priya Shah; Lucas Bennett; Sofia Nguyen; Daniel Park; Elena Rossi; Marcus Lee; Amara Okafor; Jonas Weber; Hana Sato; Theo Laurent; Nia Williams; Ravi Iyer; Clara Schmidt; Diego Alvarez; Yuki Tanaka; Sasha Petrov; Imani Johnson; Felix Andersen; Leila Haddad; Owen Murphy; Anya Volkov; Caleb Foster; Mei Zhang; Tomas Costa	a meticulous indoor gardener with strong opinions about humidity; a lapsed competitive swimmer who now coaches youth weekend meets; a sound engineer obsessed with vintage analog gear; a part-time pastry chef who does math research on the side; a long-distance hiker training for the Pacific Crest Trail; a retired ER nurse who took up woodworking after retirement; a beekeeper-turned-marketing-consultant who still keeps three hives; an amateur astronomer who hates city light pollution; a freelance translator working between Portuguese and Korean; a cybersecurity researcher who collects vintage typewriters; a former competitive figure skater now running a small tea shop; a chef-instructor who teaches knife skills at a community college; an opera singer who is also a part-time auto mechanic; an architect specializing in adaptive reuse of old factories; a wildlife photographer focused on owls in the Pacific Northwest; a high-school chemistry teacher who restores vintage motorcycles; a marathoner with a rare allergy to most stone fruits; a bookbinder who designs board games on weekends; a software engineer who breeds carnivorous plants; a paramedic who plays cello in a community orchestra; a former diplomat now running a pottery studio; a cartographer obsessed with historic shipwrecks; a dog trainer specializing in working breeds; a forensic accountant who writes science fiction novels; a glassblower with severe pollen allergies; a sommelier transitioning to non-alcoholic beverage consulting; a former orchestra conductor who now teaches sailing; a museum conservator focused on 19th-century photographs; a competitive bridge player who works as an actuary; a backcountry ski guide who restores antique furniture in summer

Table 5: The full name pool and persona flavor pool used by *Persona-Retrieval*.

allowed.

Fact-level deduplication. Every individual fact across every surviving chain is then run through MinHash LSH at a Jaccard threshold of 0.7. Any chain that contains a fact colliding with a fact already kept is dropped wholesale (rather than just dropping the offending fact), so that the released dataset has no near-duplicate fact pair across any two chains.

Distractor generation. For each surviving chain we generate four distractor options with `gpt-5` in parallel calls. Each distractor must (i) match the correct answer in grammatical form, length range, and answer category; (ii) be realistic and ordinary (no absurd, surreal, or comically random options); (iii) be unambiguously wrong (not a paraphrase, synonym, sub-phrase, or near-spelling of the correct answer or any anchor / relation phrase appearing in any fact); (iv) be *orthogonal to every fact in the chain*—a reader looking at any single fact in isolation must not be able to guess the distractor as a plausible "what comes next" or "natural consequence" via common-sense world knowledge; and (v) be distinct from the other three distractors. Chains whose distractor generation fails validation after retries are dropped. The five options are then shuffled per chain and the correct letter is recorded.

Final balancing. We oversample chains per hop count and then truncate each hop bucket to a fixed target. The final released dataset contains 31, 32, and 29 chains at K=1, K=2, and K=3 respectively, for a total of 92 chains and 274 facts.

Storage layout at evaluation time. The K+1 facts of a chain are *never* co-located. At evaluation time, every fact across the full dataset is shuffled and bin-packed into storage conversations under a hard constraint that no two facts from the same chain ever land in the same conversation; the graded question for each chain is then asked in its own separate conversation, in independently shuffled order.

B.7 Additional examples

Example: *Persona-Retrieval*, all three slots

Entity: Yuki Tanaka

Essay (excerpt): "Yuki Tanaka spends most mornings hunched over a lightbox, tracing contours from 18th-century wreck surveys.… She refuses to eat shellfish because she gets a severe hive reaction, so boat provisions are always cooked onshore and strictly shellfish-free.… Her studio is lit by a single anglepoise lamp and a wall of hand-drawn dive charts.…"

Q1 (misleading): "Do you know anything about Noah Brooks?"
→ "I don't have information about Noah Brooks."
Q2 (non-misleading): "What food should I avoid serving to Yuki Tanaka?"
→ "Avoid shellfish—she has a severe allergic reaction."
Q3 (misleading): "What kind of lamp does Clara Schmidt prefer for detailed work?"
→ "I don't have information about Clara Schmidt."

Example: *Long-Hop* (K=1, named-person voice)

Voice: single-named-person (Diego)
Anchor chain: "Diego" → "Korean food" → "thirsty"
Facts (stored separately):

- "Diego loves Korean food."
- "Korean food always leaves Diego thirsty."

Question: "What physical feeling does Diego's favorite cuisine eventually cause?"
Choices (shuffled): A. sleepy B. nostalgic **C. thirsty** D. restless E. focused
Ground truth: "thirsty".

Example: *Coexisting-Facts*, N=5

Category: types of yoga
Preference facts (each stored in its own isolated conversation):

- "Vinyasa flow is what gets me moving on weekday mornings."
- "I love a slow yin session at the end of a long workweek."
- "Hot yoga is my go-to whenever I want to really sweat it out."
- "Restorative yoga is perfect when I'm recovering from a tough run."
- "Ashtanga is what I default to when I want a structured practice."

Question: "I'm putting together a yoga schedule for the month—what styles should I rotate through to keep things varied?"
Ground truth: vinyasa flow, yin, hot yoga, restorative yoga, ashtanga.

C Human Validation of the LLM Judge

Table 6 reports the per-class confusion matrix between the `gpt-5-mini` judge of Section 5.1 and human labels. The manually graded examples give 364 memory-level classifications, over datasets and memory systems; we observed no systematic differences in judge behavior across datasets or systems, so the counts are pooled. We attribute the high agreement to the design of the dataset: since each entry is short, the classification is a simple instruction-following task for a frontier model.

	Human label					
Judge label	Corr.	N.st.	Summ.	N.ret.	Reas.	n
Correct (Corr.)	150	0	0	0	0	150
Not stored (N.st.)	0	54	2	0	0	56
Summary error (Summ.)	0	2	50	2	0	54
Not retrieved (N.ret.)	0	0	0	104	0	104
Reasoning error (Reas.)	0	0	0	0	0	0
n	150	56	52	106	0	364

Table 6: Confusion matrix between the LLM judge and human annotations over 364 memory-level classifications, pooled across all datasets and memory systems.

D Detailed Evaluation Results

We provide a complete summary of the evaluation results for all models, datasets, and memory systems for completeness and verification. While the critical insights are in the main paper, these figures further validate our claims.

Figure 4 shows success rates as a function of k, the number of retrieved memories. Figure 5 shows how performance on MEMFAIL changes across different models. Figures 6, 7, 8, and 9 show the per-system error-type breakdowns for each test-taker model.

E Full Prompt Listings

This appendix reproduces every prompt used in MEMFAIL verbatim: the dataset-generation prompts (E.1), the prompts that are sent to the memory-augmented evaluation model at evaluation time (E.2), and the LLM-judge prompts used to grade and classify errors (E.3).

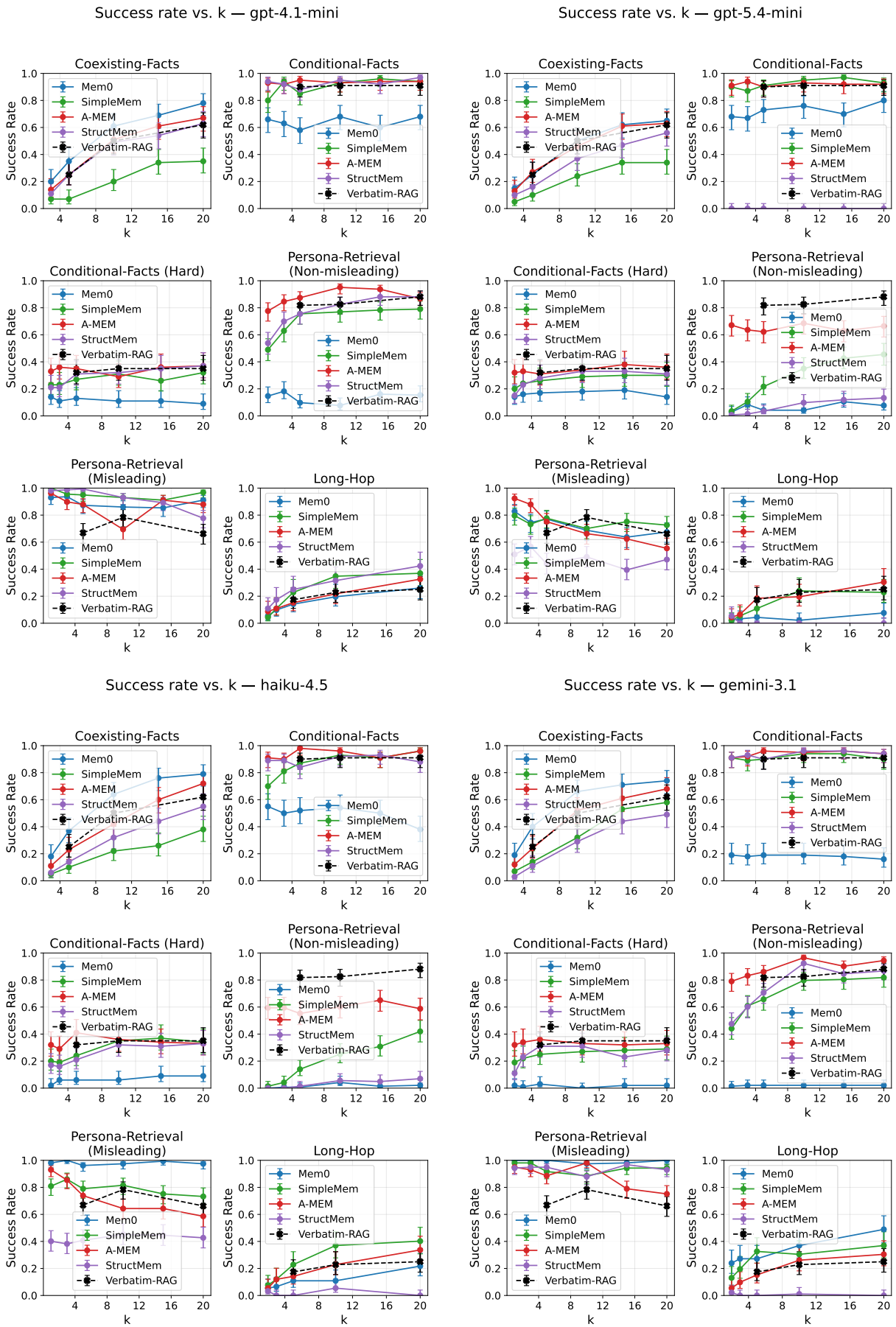

Figure 4: Success rates for all datasets, models, and systems. The **Verbatim-RAG** baseline (black dashed) is shown in every panel for reference; it uses no internal LLM, so the same curve appears in all four figures.

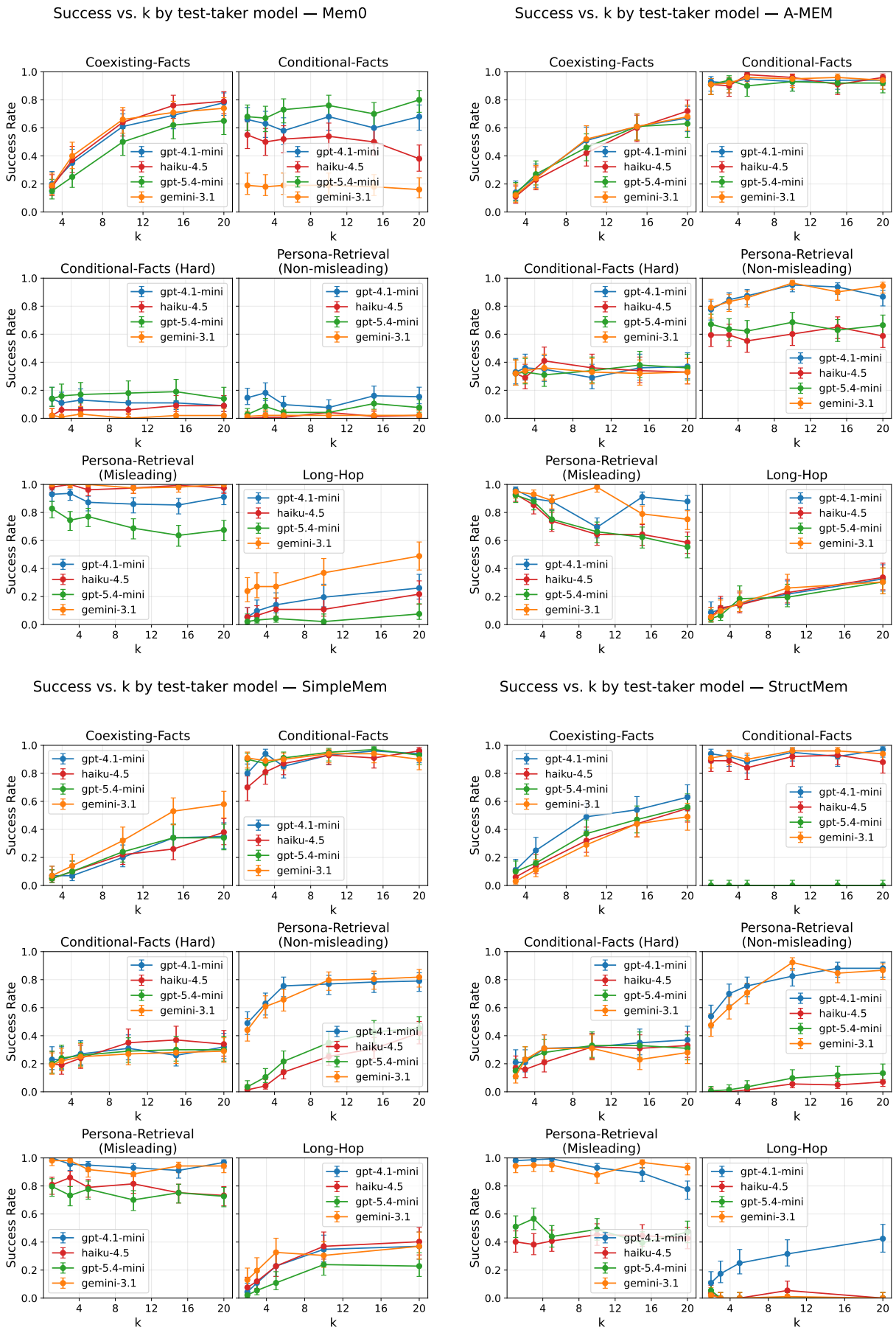

Figure 5: Success rates on each dataset for every model.

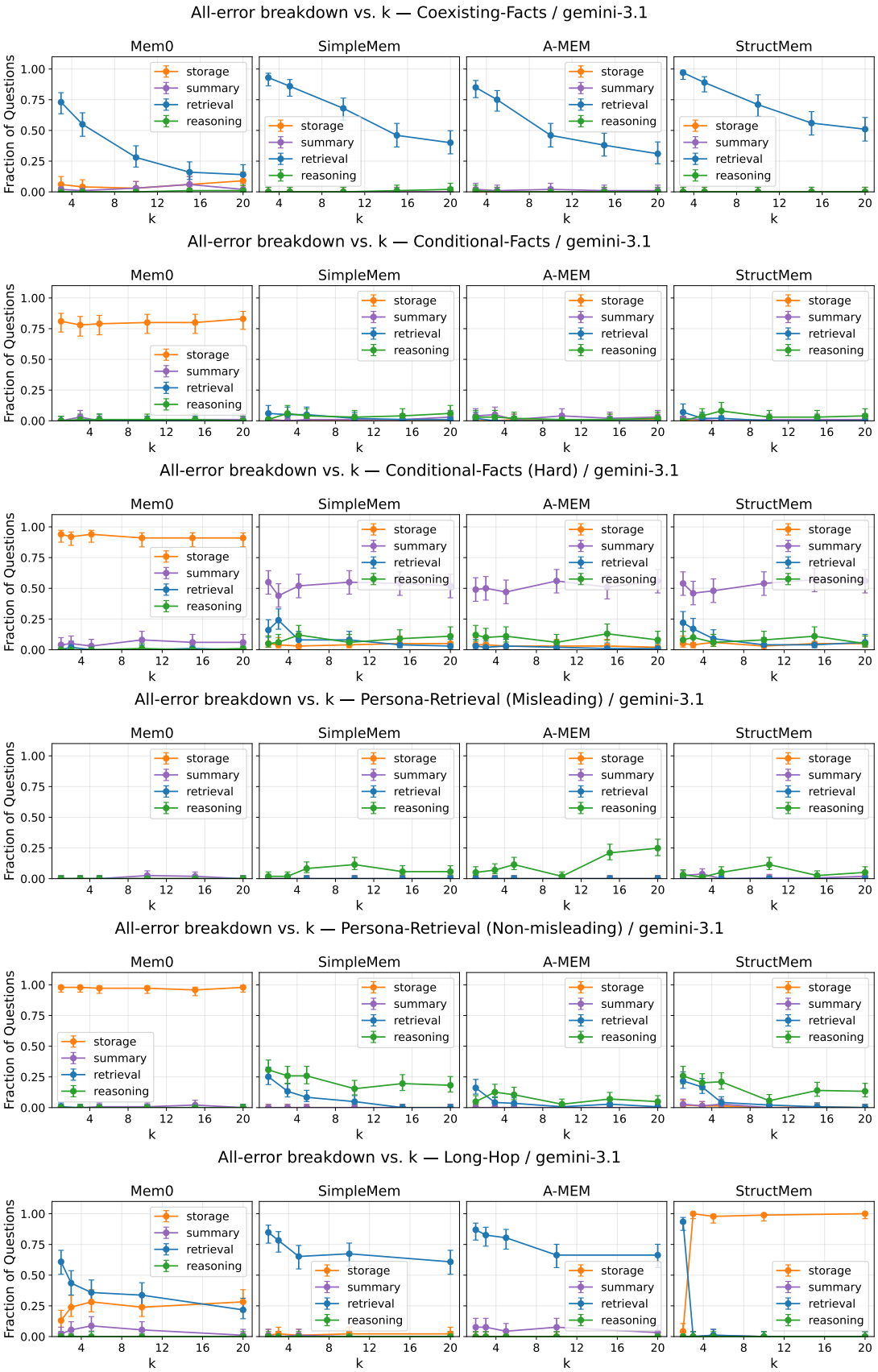

Figure 6: All error classifications for all datasets and systems, using Gemini-3.1, including reasoning errors.

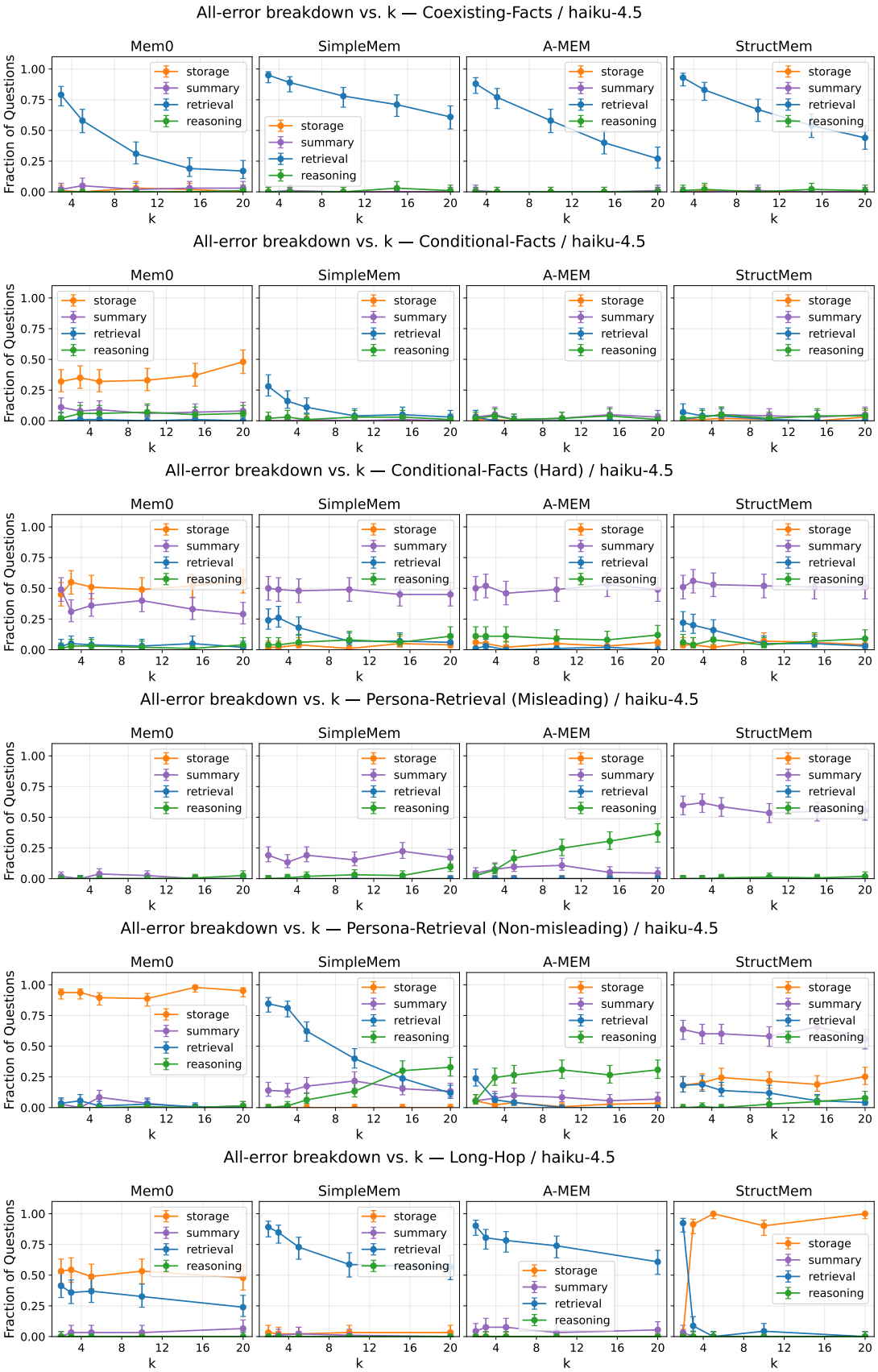

Figure 7: All error classifications for all datasets and systems, using Haiku-4.5, including reasoning errors.

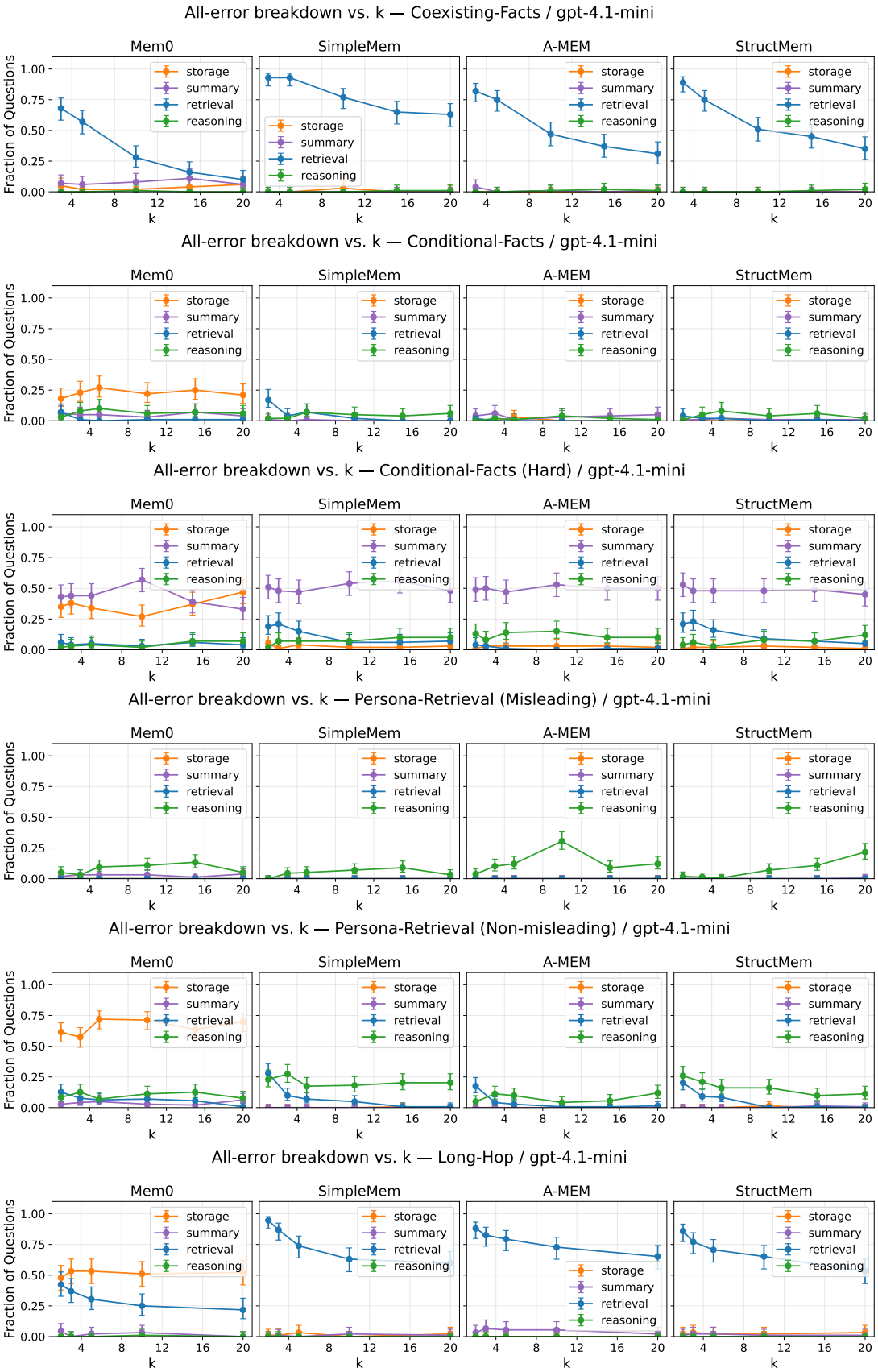

Figure 8: All error classifications for all datasets and systems, using GPT-4.1-mini, including reasoning errors.

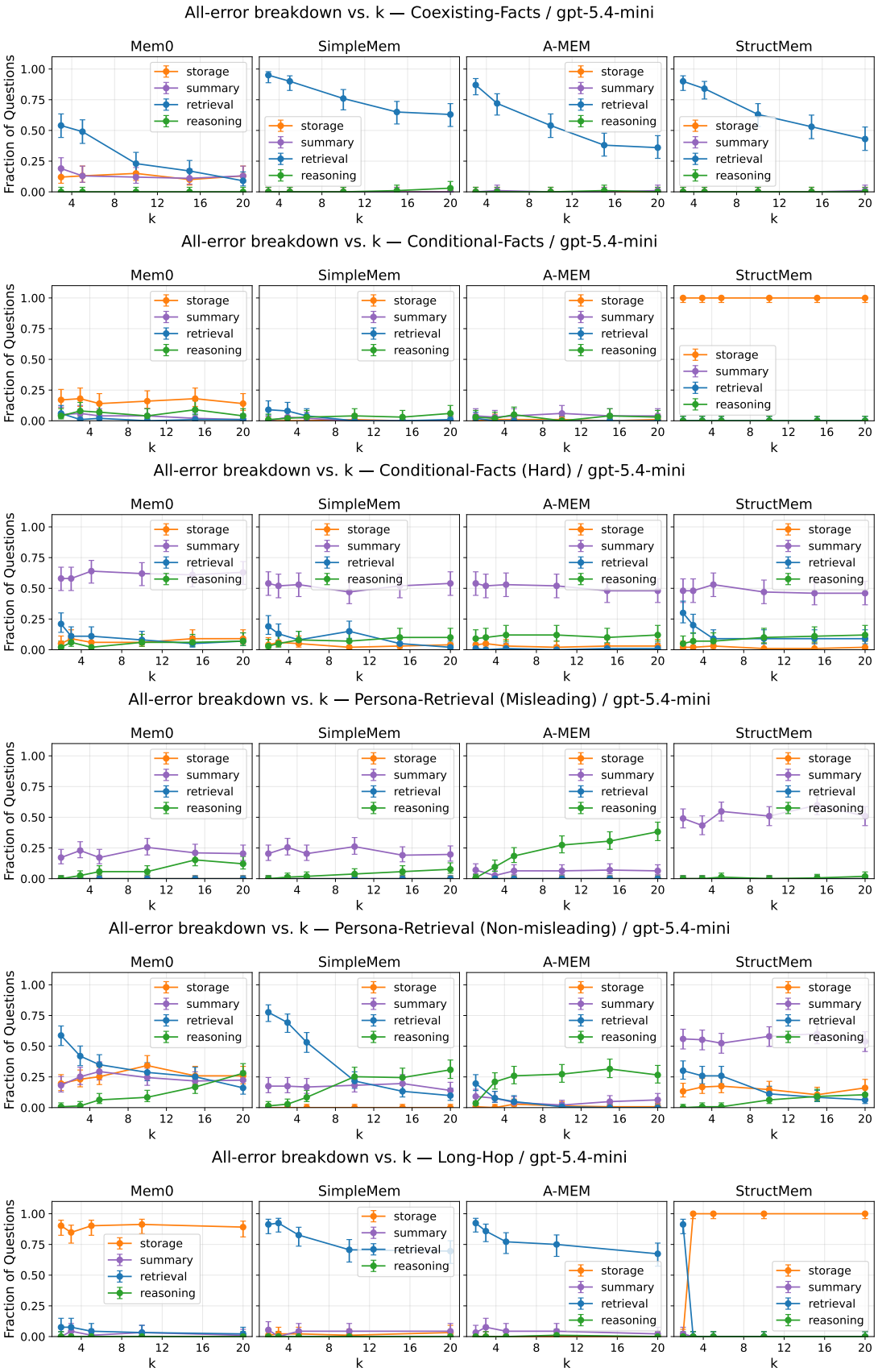

Figure 9: All error classifications for all datasets and systems, using GPT-5.4-mini, including reasoning errors.

E.1 Generation prompts

E.1.1 *Conditional-Facts*

The Easy variant calls a single datapoint generator (Prompt E.1.1) followed by an essay wrapper that stitches the conditional fact into a short casual essay (Prompt E.1.1). The Hard variant reuses the same datapoint generator and replaces the essay wrapper with a stricter decomposition prompt (Prompt E.1.1).

Generation: *Conditional-Facts* datapoint generator

```
Generate conditional-facts datapoints. Each datapoint describes an entity (person, pet, or character) who has a CONDITIONAL behavior -- they only do something under a specific condition.

For each spec below, generate:

1. "entity": a realistic first name (for persons/characters) or a pet name (for pets)
   e.g. "Jordan", "Miso", "Captain Rex"

2. "behavior": a short action phrase describing what the entity does conditionally.
   Invent something creative and specific to the entity and condition type -- do NOT default
   to cliches like "goes for a run" or "drinks coffee". The behavior should feel personal
   and idiosyncratic, not universally common.
   The examples below are illustrative ONLY -- do not reuse them:
   e.g. "re-reads old letters", "sketches floor plans", "hums while doing dishes",
        "sends voice notes instead of texts", "reorganizes their bookshelf"

3. "condition": the specific condition under which the behavior occurs
   e.g. "after 5pm", "when it's raining", "when feeling stressed"
   - Must be concrete and testable -- the question will present a specific context
   - Avoid vague conditions like "sometimes" or "often"

4. "entity_facts": a list containing exactly 1 natural statement that directly encodes
   the full conditional fact -- both the behavior AND the condition in a single sentence.
   - Must be a casual, first-person or third-person conversational sentence
   - Must clearly state BOTH what the entity does AND when/under what condition
   - 1-2 sentences max
   - The example below is illustrative ONLY -- do not reuse it:
     ["Alex has a rule: no coffee before 5pm, since it messes with their sleep."]

5. "question": a natural question about whether the entity should do (or would do) the behavior,
   given a SPECIFIC context that may or may not satisfy the condition.

   CRITICAL RULE -- the question MUST be non-inferrable without the entity's specific fact:
   A person with no knowledge of the entity should NOT be able to guess the correct answer
   from common sense or general norms alone. The correct answer must depend on knowing
   THIS entity's specific conditional rule.

   BAD (inferrable from common sense):
     - "It's a bright sunny afternoon. Should Zarek wear his heavy winter cloak?"
       -> Anyone would say no, regardless of any stored fact.
     - "Jordan hasn't slept in 30 hours. Would they want to go clubbing?"
       -> Common sense gives the answer.

   GOOD (requires knowing the entity's rule) -- these examples are illustrative ONLY, do not reuse them:
     - "It's 3pm and I'm meeting Alex -- should I grab them a coffee?"
       -> Without knowing Alex's after-5pm rule, you might reasonably say yes.
     - "It's a quiet Sunday morning. Would Priya want to reorganize her bookshelf?"
       -> Without knowing Priya only does this when stressed, you can't tell.
     - "We're at the park and it's 18C outside. Would Miso eat from the red bowl?"
       -> Without knowing Miso's specific rule, this is genuinely ambiguous.

   The question should present a context where a reasonable person WITHOUT the entity's
   specific rule could plausibly answer either way -- making the stored fact decisive.

6. "question_context": the specific context presented in the question
   e.g. "3pm", "quiet Sunday morning", "18C at the park"

7. "condition_met": "yes" if the question context satisfies the condition, "no" if not
   Think carefully -- if the condition is "after 5pm" and the context is "3pm", it's "no"

8. "ground_truth_answer": a short yes/no answer with a brief reason
```

```
   e.g. "No -- it's only 3pm and Alex
↪ doesn't drink coffee before 5pm."
   e.g. "Yes -- it's raining, which is
↪ exactly when Jordan likes to cook
↪ elaborate meals."

Return strict JSON with key "rows", a list
↪ of objects:
- row_id (int)
- entity (string)
- entity_category (string: "person", "pet",
↪ or "character")
- behavior (string)
- condition_type (string, same as input)
- condition (string)
- entity_facts (list of exactly 1 string)
- question (string)
- question_context (string)
- condition_met (string: "yes" or "no")
- ground_truth_answer (string)

Rules:
1) entity_facts must have exactly 1
↪ statement encoding both the behavior and
↪ the condition
2) The condition must be concrete and
↪ testable (not vague)
3) The question must present a specific
↪ context value that clearly either meets
↪ or doesn't meet the condition
4) condition_met must correctly reflect
↪ whether the question context satisfies
↪ the condition
5) ground_truth_answer must be consistent
↪ with condition_met
6) The question MUST be non-inferrable:
↪ without knowing the entity's specific
↪ rule, a
   reasonable person should be genuinely
↪ uncertain about the answer
7) Vary condition_met between "yes" and
↪ "no" across the batch
8) Do NOT reuse any entity names,
↪ behaviors, conditions, or phrasings from
↪ the examples above --
   they exist only to illustrate the format
9) Output ONLY valid JSON

Input specs:
{specs as JSON}
```

Generation: *Conditional-Facts* essay wrapper (Easy)

```
For each item below, write a natural essay
↪ (7-10 sentences) about the entity
that embeds the conditional fact into a
↪ rich, casual narrative.

Rules:
1. The essay MUST preserve the conditional
↪ fact clearly -- both the behavior AND the
   condition must be present. Paraphrase is
↪ fine; do not omit either part.
2. All other sentences should describe the
↪ entity's background, personality, daily
↪ routines,
   relationships, hobbies, quirks, or life
↪ context. Every such sentence must be an
   unconditional, factual statement.
3. Do NOT introduce any new conditional
↪ statements anywhere in the essay.
↪ Forbidden
   constructions: "only when", "unless",
↪ "except when", "but only if", "whenever
↪ X then Y",
   "only after", "only if", or any other
↪ conditional phrasing beyond what was
↪ already in the
   original fact.
4. The essay should feel natural -- like an
↪ excerpt from a chat conversation,
↪ personal blog,
   or journal entry, not a formal report or
↪ list.
5. The conditional fact may appear anywhere
↪ in the essay, surrounded by unrelated
↪ context
   before and after it.
6. 5-8 sentences total.

Return strict JSON with key "rows", a list
↪ of:
  row_id (int, same as input), essay
↪ (string)

Output ONLY valid JSON.

Input:
{items as JSON}
```

Generation: *Conditional-Facts* essay wrapper (Hard)

```
For each item below, write a natural essay
↪ (8-12 sentences) about the entity
that DECOMPOSES the original conditional
↪ fact into THREE distributed,
↪ non-adjacent sentences.
The reader must COMPOSE the rule from
↪ scattered evidence rather than copy it
↪ from a single
sentence.

CRITICAL RULE -- DO NOT write any single
↪ sentence that explicitly states both the
↪ behavior
AND the condition together. The original
↪ conditional fact must be split into
↪ exactly three
sentences spread across the essay:

  (A) BEHAVIOR sentence: describes the
↪ behavior as a tendency, habit, or
↪ pattern,
      WITHOUT naming the trigger condition.
  (B) CONDITION sentence: establishes the
↪ condition as part of the entity's life
↪ context
      or environment, WITHOUT naming the
↪ behavior.
  (C) LINK sentence (REQUIRED, not
↪ optional): subtly connects the two
↪ through timing,
      co-occurrence, or scene-setting --
↪ using language like "It's usually...",
↪ "By then...",
```

```
      "Around that time...", "Most of the
↪ time it happens...", "That's typically
↪ when..." --
      but still WITHOUT explicitly stating
↪ "X happens when Y" or using any
↪ conditional phrasing.

The three sentences (A), (B), and (C) MUST
↪ all appear at non-adjacent positions in
↪ the
essay. Specifically: between any two of
↪ them there must be at least one sentence
↪ of
unrelated context. No two of (A), (B), (C)
↪ may be next to each other.

Examples of the distributed pattern
↪ (illustrative only -- do not reuse):

  Example 1
  Original fact: "Mara reorganizes her
↪ sketches when it's after midnight."
  Distributed essay:
    "Mara has lived in the same studio
↪ apartment for six years.
     She tends to reorganize her sketches
↪ in sudden bursts of focus.    <-- (A)
↪ BEHAVIOR
     Her cat Pepper is usually asleep on
↪ the windowsill by then.
     She works as a freelance illustrator
↪ and keeps irregular hours that stretch
↪ deep into the night.   <-- (B) CONDITION
     She drinks tea instead of coffee.
     It's almost always during those late
↪ hours that this kind of restless energy
↪ hits her.   <-- (C) LINK
     Her sister calls every Sunday
↪ afternoon."

  Example 2
  Original fact: "Devon sends long voice
↪ notes to his friends after a tough
↪ workout."
  Distributed essay:
    "Devon grew up in a small town outside
↪ Sacramento and still texts his old high
↪ school group chat daily.
     He has a habit of sending sprawling,
↪ ten-minute voice notes to his closest
↪ friends.   <-- (A) BEHAVIOR
     His apartment is decorated with
↪ secondhand furniture and a wall of
↪ climbing medals.
     Lately he has been pushing himself
↪ hard at the gym, leaving most sessions
↪ completely drained and shaky.   <-- (B)
↪ CONDITION
     He works remotely as a backend
↪ engineer and prefers afternoon meetings.
     Those long, rambling messages tend to
↪ come right after he stumbles home from
↪ the bouldering wall.   <-- (C) LINK
     His mom still mails him birthday cards
↪ a week early."

  Example 3
  Original fact: "Biscuit (a corgi) only
↪ eats from the red bowl when there are
↪ guests in the house."
  Distributed essay:
    "Biscuit is a five-year-old corgi who
↪ lives with the Tanaka family in Portland.
     He has a peculiar habit of eating
↪ exclusively from the red ceramic bowl on
↪ certain days.   <-- (A) BEHAVIOR
     He sleeps under the dining room table
↪ and follows the youngest kid everywhere.
     The Tanakas host frequent dinner
↪ parties, and the house is often full of
↪ unfamiliar voices and shoes by the door.
↪   <-- (B) CONDITION
     His favorite toy is a chewed-up
↪ stuffed carrot.
     The red bowl tends to come out
↪ specifically on those crowded, noisy
↪ evenings.   <-- (C) LINK
     He gets groomed once a month at a
↪ place on Hawthorne."

Other rules:
1. The behavior and condition must BOTH be
↪ recoverable by a careful reader who
↪ composes
   sentences (A), (B), and (C) -- but
↪ NEITHER should appear in the same
↪ sentence.
2. Do NOT use explicit conditional phrasing
↪ anywhere ("only when", "whenever", "if",
   "unless", "except when", "but only if",
↪ "only after", "only if").
3. The link sentence (C) should use
↪ timing/scene language, not logical
↪ connectives.
4. All remaining sentences should describe
↪ the entity's background, personality,
↪ daily
   routines, relationships, hobbies,
↪ quirks, or life context -- unconditional
↪ factual statements.
5. The essay should feel natural -- like an
↪ excerpt from a personal blog or journal
↪ entry.
6. The correlation between the behavior and
↪ condition should be obvious to somebody
↪ who has read both sentences. It should
↪ NOT be vague or too subtle.
7. 8-12 sentences total.

Return strict JSON with key "rows", a list
↪ of:
  row_id (int, same as input), essay
↪ (string)

Output ONLY valid JSON.

Input:
{items as JSON}
```

E.1.2 *Coexisting-Facts*

A single datapoint generator (Prompt E.1.2) produces, for each preference category, N isolated first-person statements plus a holistic scenario question whose answer requires all N.

Generation: *Coexisting-Facts* datapoint generator

```
Generate coexisting-facts datapoints. Each
↪ datapoint represents a user
↪ (first-person "I") with MULTIPLE
↪ preferences in the same category. Each
↪ preference will be stored as a
↪ completely SEPARATE, ISOLATED memory --
↪ so each fact statement must make sense
↪ entirely on its own, with no reference
↪ to the other preferences.

For each spec below, generate:
1. "preferences": list of exactly
↪ num_preferences distinct preferences in
↪ the category
   (e.g. for foods: ["pizza", "sushi",
↪ "ramen"])

2. "preference_facts": list of exactly
↪ num_preferences short, natural
↪ first-person statements --
   ONE statement per preference, in the
↪ same order as "preferences".
   - Each statement must stand alone as a
↪ complete, self-contained fact
   - Each statement must mention ONLY that
↪ single preference (not the others)
   - Use varied, natural phrasing -- not a
↪ template ("I love X", "I enjoy X", "X is
↪ my favorite", etc.)
   - 1-2 sentences max per fact
   - Examples for foods:
       "I love pizza -- it's my default
↪ Friday night meal."
       "Sushi is my go-to whenever I want
↪ something fresh and light."
       "I'm a huge ramen fan, especially on
↪ cold days."

3. "question": a natural first-person
↪ scenario question that REQUIRES knowing
↪ ALL preferences.
   - Must NOT be a direct "list all my X"
↪ request -- make it a realistic scenario
   - Good: "I'm going grocery shopping --
↪ what should I pick up for dinners this
↪ week?"
   - Good: "My friend wants to plan an
↪ outing I'd enjoy -- what are some solid
↪ options?"
   - The question should have a clearly
↪ better answer if ALL preferences are
↪ known vs. only one

4. "ground_truth_answer": a concise
↪ comma-separated list of all preference
↪ names
   Example: "pizza, sushi, ramen"

Return strict JSON with key "rows", a list
↪ of objects:
- row_id (int)
- preference_category (string, same as
↪ input)
- preferences (list of strings)
- preference_facts (list of strings, same
↪ length as preferences, one fact per
↪ preference)
- question (string)
- ground_truth_answer (string)

Rules:
1) preference_facts must have exactly the
↪ same length as preferences
2) Each fact covers exactly ONE preference
↪ and stands alone -- no cross-references
3) The question must be a realistic
↪ first-person scenario, NOT "list all my
↪ X"
4) Ground truth must include every
↪ preference, comma-separated
5) Output ONLY valid JSON

Input specs:
{specs as JSON}
```

E.1.3 *Persona-Retrieval*

A single datapoint generator (Prompt E.1.3) jointly produces the third-person essay about E and the three first-person follow-up questions, with each slot pre-marked as misleading-or-not by the calling code.

Generation: *Persona-Retrieval* datapoint generator

```
Generate misleading-persona datapoints.
↪ Each datapoint is about a SPECIFIC named
person, embedded in a rich essay (10-15
↪ sentences). Each datapoint has exactly 3
↪ questions.
Each question is independently EITHER
↪ non-misleading (asks about the entity by
↪ name) OR
misleading (asks about a DIFFERENT named
↪ person -- the "distractor" -- who has no
↪ presence
in the essay).

For each spec below, generate:

1. "essay": a natural personal essay about
↪ the entity (10-15 sentences).
   - Written in third person, naming the
↪ entity (e.g. "Maya Patel"). Pronouns are
↪ fine
     after the first mention.
   - Embed MANY specific, memorable,
↪ idiosyncratic facts: daily rituals,
↪ unusual hobbies,
     hard constraints
↪ (allergies/aversions/rules), strong
↪ preferences, quirky possessions,
     rules of thumb. Aim for at least 4-5
↪ distinct facts so different questions
↪ can probe
     different details.
   - Tone: casual, like a journal entry or
↪ chat message -- not a formal bio.
   - Do NOT use first-person voice ("I",
↪ "me", "my", "we").
   - Do NOT mention any of the distractor
↪ names anywhere in the essay.
```

```
2. "questions": a list of EXACTLY 3
↪ question objects, in the order given by
   spec.question_slots. Each slot specifies
↪ whether that question is misleading and,
↪ if
   so, the distractor name to use.

   For each slot:

   If is_misleading=false:
     - "text": a first-person question that
↪ explicitly names the entity by their
↪ full name.
       The asker wants advice or info ABOUT
↪ the entity (e.g. what to get them, what
↪ to
       avoid, where to take them, whether
↪ they'd enjoy something).
     - The question must be answerable from
↪ a SPECIFIC detail in the essay -- NOT
↪ from
       generic norms.
     - Phrased naturally. Must NOT smuggle
↪ the answer into itself as an assumption.
       BAD:  "What apple dessert can I give
↪ Maya that won't make her itch?"
             (assumes the asker already
↪ knows about the allergy)
       GOOD: "What dessert should I make
↪ for Maya?"
             (open; the essay's allergy
↪ info is what makes the answer specific)
     - "ground_truth_answer": short answer
↪ (1-2 sentences) drawn from specific essay
       details. This is what a memory-aware
↪ system should return.
     - "distractor": null.

   If is_misleading=true:
     - "text": a first-person question that
↪ names the GIVEN distractor (NOT the
↪ entity).
       Same kind of advice/info question as
↪ the non-misleading case, but asked about
↪ the
       distractor. The entity's name must
↪ NOT appear in the question.
     - "ground_truth_answer": "I don't have
↪ information about <distractor name>."
       (Correct behavior is to abstain --
↪ the asker has no info about this person.)
     - "distractor": the distractor name
↪ from the spec slot.

   The 3 questions should probe DIFFERENT
↪ angles -- don't repeat the same wording
↪ or topic
   across slots. If multiple slots are
↪ non-misleading, each should probe a
↪ different fact
   from the essay.

Return strict JSON with key "rows", a list
↪ of objects:
- row_id (int)
- entity (string, same as input)
- essay (string)
- questions (list of exactly 3 objects with
↪ fields: text, is_misleading, distractor,
  ground_truth_answer)

Rules:
1) The essay is 10-15 sentences,
↪ third-person, names the entity, and
↪ never mentions any
   distractor name from any slot.
2) Each non-misleading question names the
↪ entity exactly and never names any
↪ distractor.
3) Each misleading question names that
↪ slot's distractor exactly and never
↪ names the
   entity.
4) Non-misleading questions must NOT embed
↪ their own answers as assumptions.
5) Each non-misleading ground_truth_answer
↪ is supported by specific essay details.
6) Each misleading ground_truth_answer
↪ indicates the system should abstain.
7) Output ONLY valid JSON.

Input specs:
{specs as JSON}
```

E.1.4 *Long-Hop*

Long-Hop generation runs in three phases: chain proposal (Prompts E.1.4–E.1.4), cross-chain conflict / similarity audit (Prompt E.1.4), and per-chain distractor generation (Prompts E.1.4–E.1.4).

Generation: *Long-Hop* chain proposal — system message

```
You are constructing a benchmark of
↪ multi-hop reasoning chains.

Each chain consists of K+1 short factual
↪ statements that strictly link K+2
distinct anchors A -> B -> C -> ... in a
↪ transitive chain. Statement i must
relate anchor i to anchor i+1 (no other
↪ anchors, except the HEAD subject --
see rule 2 -- which may recur as
↪ background). The chain must support a
↪ single
multi-hop question: starting from A,
↪ answering the question requires chaining
through every statement to reach the
↪ terminal anchor.

Hard rules -- every chain must satisfy ALL
↪ of these:
1. EXACTLY K+1 statements per chain. Each
↪ statement is a single declarative
   English sentence, max ~16 words, no
↪ commas-separated multi-claims.
2. Statement i mentions anchor i and anchor
↪ i+1, plus an explicit relation
   word -- a verb ("loves", "hates",
↪ "always picks"), a conditional ("when",
   "whenever", "if"), a causal ("because",
↪ "leads to", "makes me"), a temporal
   ("after", "before"), or a preference ("I
↪ do X when Y"). MIDDLE and TERMINAL
```

```
   anchors (anchors 2 .. K+2) must appear
↪ ONLY in the two facts that border
   them -- they must not be named in any
↪ other fact.
   The HEAD subject is special and may
↪ recur as background in every fact:
     - First-person voice: "I/me/my" is the
↪ implicit speaker. The first-person
       speaker is NEVER literally listed in
↪ answer_chain -- anchor 1 is instead
       a state/action/object the speaker
↪ relates to ("eat apples", "bored").
     - Named-person voice: the chain's
↪ subject is a single named person
       ("Diego", "Marisol Vega"). That
↪ person IS anchor 1, and the same name
       must appear in every subsequent fact
↪ as background. Pronouns
       ("he/she/his/her") are FINE within a
↪ single fact when the proper
       noun also appears in that same fact
↪ (see rule 2b).
   Pick ONE voice per chain (first-person
↪ OR one named person) and stay
   consistent.
2b. EVERY STATEMENT MUST BE SELF-CONTAINED.
↪ A reader will encounter each
   statement in isolation, with no access
↪ to the other statements in the
   chain, so each statement must be fully
↪ interpretable on its own.

   Pronouns INSIDE a statement are FINE --
↪ write natural English. The only
   rule is that every reference must
↪ resolve from the statement alone:
     - First-person ("I", "me", "my",
↪ "myself") is always fine -- the speaker
       is implicit and shared across the
↪ dataset.
     - Third-person pronouns ("he", "she",
↪ "it", "they", "them", "his",
       "her", "their", "its", etc.) are
↪ fine when their antecedent appears
       LITERALLY in the SAME statement.
↪ Examples that are GOOD:
         "Diego loves Korean food because
↪ he finds it spicy."   (he <-> Diego)
         "Marisol picks pop music whenever
↪ she is alone."       (she <-> Marisol)
         "Watering succulents keeps them
↪ healthy."              (them <->
↪ succulents)

   What's forbidden is leaving a statement
↪ DEPENDENT on a different statement
   to resolve a reference. Do NOT:
     - Use a pronoun whose only possible
↪ antecedent appears in a DIFFERENT
       statement. If the proper noun isn't
↪ in this statement, repeat it.
     - Use referential phrases like "the
↪ company", "that town", "this book",
       "the same person" that depend on
↪ another statement to resolve.
3. K+2 anchors total per chain. Anchors
↪ should be SUBJECTIVE / PERSONAL content
   that cannot be looked up in an
↪ encyclopedia. Use anchors like:
     - States, moods, feelings ("bored",
↪ "anxious", "calm").
     - Actions, habits, routines ("eat
↪ apples", "skip lunch", "go for a run").
     - Preferences and opinions ("loves
↪ Korean food", "thinks pop music is
       overrated").
     - Concrete personal objects ("my
↪ bookshelf", "popcorn", "chamomile tea").
     - A single specific named person
↪ introduced as the chain's head subject
       ("Diego", "Marisol Vega") -- only
↪ when that person is anchor 1 and the
       rest of the chain is about their
↪ preferences / habits / moods.
   Do NOT use impersonal entities
↪ (companies, towns, rivers, books,
   institutions, geographic regions).
↪ First-person ("I ...") sentences are
   encouraged. Mix first-person and
↪ named-person styles freely.

3b. Every fact must be SUBJECTIVE: an
↪ opinion, preference, mood, routine,
   habit, or relational claim about a
↪ specific person (the speaker or a
   named person). Facts must NOT be
↪ impersonal world claims.
     GOOD:
       - "I think Italian food is
↪ overrated."           (first-person
↪ opinion)
       - "Diego loves Korean food."
↪             (fact about a person)
       - "Marisol always feels nervous
↪ before tests."   (personal trait)
       - "When I'm tired I get grumpy."
↪             (first-person routine)
     BAD:
       - "Drannot House published the
↪ novel."           (impersonal fact)
       - "Yepelmir lies in the province of
↪ Korunda."    (geographic claim)
       - "Strophien Atelier operates from
↪ Treskellin."  (impersonal corporate)
4. Within a single chain, all K+2 anchors
↪ must be distinct (case-insensitive).
5. Vary the relation patterns across the
↪ K+1 statements within one chain -- do
   not reuse the same conditional or verb
↪ template back-to-back.
6. The graded question must reference
↪ anchor 1 (the head) at least once by
   name and ask about the terminal anchor
↪ (the last in the chain), without
   ever naming any intermediate anchor. The
↪ question should read as a single
   natural English sentence and have a
↪ unique correct answer given the K+1
   statements. Natural pronouns are
↪ encouraged when they aid flow -- e.g.,
   "What does Diego do when he is bored?"
↪ or "When I'm dehydrated, what
   mood do I end up in?" -- provided every
↪ pronoun's antecedent is clear
   from the question itself.
7. ground_truth_answer must equal the
↪ terminal anchor exactly (or its shortest
```

```
   natural form -- e.g. drop a leading
↪ "the" only if the canonical phrase has
   no article).
8. Across chains in this batch, AVOID
↪ retelling the same narrative as anything
   in PRIOR CHAIN SUMMARIES (provided in
↪ the user message). Generic words like
   "sleep" or "bored" may repeat across
↪ chains, but a chain that paraphrases
   another chain's storyline must not be
↪ produced.

Distractor options are produced in a
↪ separate downstream step -- DO NOT
↪ include
any distractors / answer choices in your
↪ output here.

Output JSON only -- no commentary.
```

Generation: *Long-Hop* chain proposal — user message (per batch)

```
Produce a JSON object with exactly
↪ {batch_size} chains, each of HOP COUNT K
↪ = {hop_count}.

Each chain must follow ALL hard rules from
↪ the system prompt. To recap for K =
↪ {hop_count}:
- Exactly {K+1} statements.
- Exactly {K+2} distinct anchors, listed in
↪ answer_chain in chain order
  (head first, terminal last).
- DO NOT USE THE EXAMPLES I GIVE YOU IN
↪ YOUR CHAINS.
- Statement i links anchor i and anchor
↪ i+1. The HEAD subject (first-person
  "I" implicit, OR a named person at anchor
↪ 1) may also recur as background
  in every fact. Middle and terminal
↪ anchors must each appear ONLY in their
  two bordering facts.
- Every fact must be SUBJECTIVE -- an
↪ opinion / preference / mood / routine /
  habit, either first-person ("I") or about
↪ a single named person
  ("Diego", "Marisol"). No encyclopedic /
↪ entity-relation facts.
- Every anchor must appear LITERALLY
↪ (case-insensitive substring) in the
  fact(s) it belongs to.
- EVERY STATEMENT MUST BE SELF-CONTAINED --
↪ a reader will see each fact in
  isolation, so each fact must be
↪ interpretable on its own. Pronouns INSIDE
  a single fact are fine when the
↪ antecedent is in the SAME fact ("Diego
  loves Korean food because he finds it
↪ spicy" -- "he" <-> Diego). What's
  forbidden is using a pronoun whose only
↪ antecedent appears in a DIFFERENT
  fact. No referential phrases ("the
↪ company", "that town") that depend on
  another fact to resolve. First-person
↪ ("I", "me", "my") is always fine.
  BAD:  "Lina buys a ticket and travels to
↪ the coast." / "Traveling to the
        coast means she visits Seabright."
↪  <- "she" needs fact 1 to resolve.
  GOOD: "Lina buys a ticket and travels to
↪ the coast." / "Traveling to the
        coast means Lina visits Seabright."
  ALSO GOOD: "Lina buys a ticket because
↪ she loves travel."   <- "she" has
        antecedent "Lina" in the same fact.
- The graded_question references anchor 1
↪ (the head) at least once by name
  and asks about the terminal anchor. The
↪ question must be unanswerable from
  any single statement alone. Natural
↪ pronouns referring to the head anchor
  are encouraged when they aid flow.
- ground_truth_answer is the canonical
↪ written form of the terminal anchor.
- DO NOT include any distractors / answer
↪ choices -- they are produced in a
  separate downstream step.

In-context examples (mix of styles --
↪ produce a similar mix):

{four worked examples for the requested K,
↪ mixing one named-person opinion
chain with three first-person chains
↪ (causal / conditional / temporal /
preference)}

PRIOR CHAIN SUMMARIES (avoid retelling
↪ these storylines; pick fresh
narratives -- generic words like "sleep" or
↪ "tea" may repeat, but the chain
arc must be new):
{rolling list of one-line "head -> ... ->
↪ terminal" summaries from
previously accepted chains, capped at 30}

Output schema (JSON object):
{
  "chains": [
    {
      "facts": ["..."],
      "answer_chain": ["..."],
      "graded_question": "...",
      "ground_truth_answer": "..."
    },
    ...  // exactly {batch_size} chains
  ]
}

Generate now. Aim for diversity -- across
↪ the {batch_size} chains, vary the
sentence patterns (causal / conditional /
↪ temporal / preference / opinion)
and the topic domains (food, mood, weather,
↪ routine, work, hobbies, music,
travel, study, exercise, etc.). EVERY chain
↪ must be subjective: a first-
person chain (no named subject -- implicit
↪ "I") OR a chain about a single
named person ("Diego", "Marisol Vega",
↪ "Lina") and that person's
opinions / habits / moods. Mix the two
↪ voices freely across the batch. Do
NOT produce encyclopedic / entity-relation
↪ chains (no companies, towns,
rivers, books, geographic features).
```

Generation: *Long-Hop* cross-chain conflict / similarity audit

```
SYSTEM:
You are auditing a small set of made-up
↪ reasoning chains for cross-chain
↪ interference.

Two chains "interfere" if storing both into
↪ a single shared memory store would
corrupt reasoning for either chain.
↪ Interference occurs when ANY of the
following hold between facts in different
↪ chains:
- CONTRADICTION: the facts make
↪ incompatible claims about the same anchor
  (e.g., one chain says "I eat apples when
↪ bored", another says "I never eat
  apples").
- SHARED DISTINCTIVE ANCHOR: a distinctive
↪ proper-noun or distinctive composite
  phrase appears in two different chains
↪ (paraphrase or near-spelling counts).
  Generic single-word concepts ("sleep",
↪ "bored", "tea") repeating across
  chains is fine and does NOT count as
↪ interference.
- OVERTLY SIMILAR NARRATIVE: two chains
↪ tell essentially the same storyline
  with the same anchors in the same role --
↪ e.g. "X is owned by Y" and
  "Y owns X", or "When I'm bored I sleep /
↪ sleep gives me a dream" appearing
  twice with only minor word swaps.

Return the list of chain IDs to DROP to
↪ eliminate all interference. When two
chains conflict, drop only ONE of them
↪ (your choice). Be precise; do not flag
chains that merely share generic concepts
↪ or common verbs.

Output JSON only.

USER:
CHAINS (each has an `id` and a list of
↪ `facts`):
{JSON array of {id, facts} for every
↪ survivor -- sent once globally and then
again over overlapping sliding windows of
↪ size batch_size}

Output JSON:
{
  "to_drop": ["<chain_id>", ...],
  "reason_per_drop": {"<chain_id>":
↪ "<one-line reason>", ...}
}
If there are no conflicts, return
↪ {"to_drop": [], "reason_per_drop": {}}.
```

Generation: *Long-Hop* distractor generator — system message

```
You write distractor options for a
↪ multi-hop reasoning multiple-choice
↪ question.

You receive:
- A list of FACTS forming a transitive
↪ reasoning chain (the graded answer
  requires chaining ALL facts).
- The GRADED_QUESTION (references only the
↪ head anchor; asks about the
  terminal anchor).
- The correct GROUND_TRUTH_ANSWER.

Produce EXACTLY 4 distractor options. Each
↪ distractor must satisfy ALL of the
following rules:

1. SAME-SHAPE PLAUSIBILITY. Match the
↪ correct answer in grammatical form,
   length range, and answer category. If
↪ the correct answer is a noun phrase
   naming a mood, every distractor is a
↪ noun phrase naming a mood. If the
   correct answer is a short verb phrase
↪ ("drink water"), every distractor is
   a short verb phrase of similar length
↪ and shape. Pronouns and articles
   that flow naturally with the question
↪ are fine -- match the voice the
   question uses (e.g., if the question is
↪ "What does Diego do when he is
   bored?", distractors phrased as "he
↪ reorganizes his closet" or simply
   "reorganizes the closet" are both
↪ acceptable, as long as the distractor
   reads as a fluent answer to the
↪ question).

2. REALISTIC AND ORDINARY. Each distractor
↪ must name something a real person
   could plausibly feel, do, prefer, eat,
↪ or experience in everyday life.
   NO absurd, surreal, slapstick, joke, or
↪ comically random options. NO
   things almost no one actually does
↪ (e.g., "duel a swan", "memorize country
   capitals from memory", "argue with
↪ neighbors about constellations"). Pick
   ordinary moods, habits, hobbies, foods,
↪ or activities -- the kind of
   answer a thoughtful peer might genuinely
↪ guess.

3. UNAMBIGUOUSLY WRONG. Must not be a
↪ paraphrase, synonym, sub-phrase,
   near-spelling, or otherwise overlapping
↪ with the correct answer or with
   any anchor / relation phrase that
↪ appears in any fact.

4. ORTHOGONAL TO EVERY FACT. A reader
↪ looking at any single fact in isolation
   must NOT be able to guess the distractor
↪ as a plausible "what comes next"
   or "natural consequence" via
↪ common-sense world knowledge. Avoid
   distractors that name typical effects,
↪ components, properties, or strong
   associations of any concept mentioned in
↪ any fact (e.g., if a fact
   mentions popcorn, do NOT pick a
↪ distractor about thirst, salt, or movies;
```

```
   if a fact mentions a cold shower, do NOT
↪ pick a distractor about feeling
   refreshed or shivery). Pick subject
↪ matter unrelated to every fact's
   topic.

5. DISTINCT. The four distractors must be
↪ distinct from each other
   (case-insensitive) and distinct from the
↪ correct answer.

Examples (note: realistic, ordinary,
↪ orthogonal):

CHAIN A facts:
  - "I eat apples when I'm bored."
  - "When I'm bored I go to sleep."
  - "When I sleep I have a dream."
  - "Every dream I have leaves me curious
↪ about the future."
GRADED_QUESTION: "What does eating apples
↪ eventually leave me feeling?"
CORRECT_ANSWER: "curious about the future"
GOOD distractors (ordinary moods/feelings,
↪ orthogonal to apples / sleep / dreams):
  - "nostalgic about old friendships"
  - "motivated to clean my apartment"
  - "indifferent toward upcoming holidays"
  - "satisfied with my routine"

CHAIN B facts:
  - "Marisol thinks pop music is overrated."
  - "Whenever pop music is on Marisol
↪ leaves the room."
  - "When Marisol leaves the room Marisol
↪ ends up in a sour mood."
GRADED_QUESTION: "What mood does Marisol
↪ end up in because of the music genre
↪ Marisol dislikes?"
CORRECT_ANSWER: "sour mood"
GOOD distractors (ordinary moods, no music
↪ / departure associations):
  - "a focused mood"
  - "a contemplative mood"
  - "a generous mood"
  - "a competitive mood"

Output JSON only -- no commentary.
```

Generation: *Long-Hop* distractor generator — user message

```
FACTS:
- {fact_1}
- {fact_2}
- ...

GRADED_QUESTION: {graded_question}

CORRECT_ANSWER: {ground_truth_answer}

Produce 4 distractor options that satisfy
↪ every rule from the system message.
↪ Output JSON:
{"incorrect_options": ["...", "...", "...",
↪ "..."]}
```

E.2 Evaluation prompts (sent to the memory-augmented model)

All four tasks share a single `ConversationHistoryPromptTemplate` (`src/prompt_templates.py`) that decides what is shown to the evaluation model on each turn based on whether the turn is graded. The ungraded form (Prompt E.2) is used during the storage phase, where the model only needs to converse so the memory system can absorb new facts; the graded form (Prompt E.2) is used when an answer is being scored. *Long-Hop* additionally wraps each graded question with a strict-JSON MCQ instruction (Prompt E.2) so the chosen letter can be parsed deterministically.

Evaluation: ungraded turn (storage phase)

```
You are a helpful chat assistant. Read the
↪ user's message carefully and remember
↪ any new personal information,
↪ preferences, or facts they share that
↪ you feel are important to remember.They
↪ may be recalled in future conversations.

Relevant Past Memories:
{memories}

Conversation History:
{history}

User: {query}
```

Evaluation: graded turn

```
You are an intelligent memory assistant
↪ tasked with answering questions using
↪ information from past conversation
↪ memories.

# CONTEXT:
You have access to memories from previous
↪ conversations as well as the
↪ conversation history that may be helpful
↪ in answering the question.

# INSTRUCTIONS:
Answer the user's question. You may use the
↪ provided memories if they are helpful.
↪ If you use the memories above to answer
↪ a question, please EXPLICITLY RESTATE
↪ which memories you used below, or state
↪ that you used no memories. You should
↪ only use and restate those memories if
↪ you explicitly used them to draw
↪ conclusions from them.
Conversation History:
{history}
Relevant Memories:
{memories}
END of Relevant Memories

User Question:
```

```
{query}
Answer:
```

Evaluation: *Long-Hop* MCQ wrapper (appended to each graded question)

```
{question_with_choices}

Choose exactly one option above. The
↪ correct answer is uniquely
determined by chaining the relevant
↪ remembered facts together; the
other four options cannot be inferred from
↪ those facts.

Respond with a single JSON object on its
↪ own line and nothing else,
using this exact schema:
  {"selected_choice": "<one of A, B, C, D,
↪ E>"}
```

E.3 Grading prompts (LLM judge)

Each task is graded by a staged LLM-judge pipeline that classifies failures into the canonical taxonomy (not_stored / summary_error / not_retrieved / reasoning_error / correct). Every stage is a structured JSON-schema call; the system message and the user template are reproduced below. *Long-Hop* grading omits the invocation stage — the 5-way MCQ answer is parsed deterministically from the model's `{"selected_choice": "..."}` response.

E.3.1 Conditional-Facts grading

Four sequential stages, short-circuiting on the first failure: storage, summary, retrieval, invocation. The Hard variant replaces the storage, summary, and retrieval prompts with rule-decomposed analogues that allow the behavior and condition to be *recovered across multiple memories*.

Grading: *Conditional-Facts* Stage 1 — Storage check (Easy)

```
SYSTEM:
You are checking whether a specific fact
↪ was stored in a memory system.
The fact may have been paraphrased or
↪ compressed, but must still convey the
↪ same
information -- including any qualifying
↪ condition -- to count as present.

USER:
ORIGINAL FACT:
{original_fact}

ALL_MEMORIES (complete memory store):
{all_memories_formatted}

Is the original fact present in
↪ ALL_MEMORIES, even if paraphrased, as
↪ long as the
qualifying condition is preserved and the
↪ meaning is not altered?
```

Grading: *Conditional-Facts* Stage 1 — Storage check (Hard)

```
SYSTEM:
You are checking whether the COMPONENTS of
↪ a conditional rule -- a behavior and a
triggering condition -- were stored in a
↪ memory system at all.

This is a permissive presence check, not a
↪ faithfulness check. The two components
↪ may
appear in a single memory or split across
↪ multiple memories, may be paraphrased or
compressed, and may even be attached to the
↪ wrong entity or no entity at all -- that
quality concern is judged in a separate,
↪ later step. Your only job here is to
↪ determine
whether SOME version of the behavior AND
↪ SOME version of the condition exist
↪ somewhere
in the memory store.

The rule counts as PRESENT if both
↪ components -- the behavior and the
↪ triggering
condition -- can be located somewhere in
↪ ALL_MEMORIES, in any form (paraphrased,
distributed, or even mislabeled).

The rule counts as ABSENT only if one of
↪ the components is genuinely missing from
↪ the
entire store -- i.e., no memory mentions
↪ the behavior at all, OR no memory
↪ mentions the
triggering condition at all.

USER:
COMPONENTS TO LOOK FOR:
  Behavior:  {behavior}
  Condition: {condition}
  (For reference -- entity is "{entity}",
↪ but entity linkage is NOT required for
↪ this check.)

ALL_MEMORIES (complete memory store):
{all_memories_formatted}

Are BOTH components present somewhere in
↪ ALL_MEMORIES, even if paraphrased, even
↪ if
distributed across multiple memories, and
↪ even if attached to the wrong entity or
↪ no
entity? Do not penalize entity mislabeling
↪ here -- that is judged separately.
```

Grading: *Conditional-Facts* Stage 2 — Summary check (Easy)

```
SYSTEM:
You are assessing the quality of a stored
↪ memory.

The fact has already been confirmed to
↪ exist in the memory store. Your job is
↪ to evaluate
whether the stored version faithfully
↪ preserves the CRITICAL information --
↪ specifically the
qualifying condition and the conditional
↪ behavior -- in a way that would support
↪ correct
downstream reasoning.

A stored version has a SUMMARY ERROR if ANY
↪ of the following apply:
- The qualifying condition was dropped
↪ entirely (stored as an unconditional
↪ fact)
- The condition was generalized in a way
↪ that changes the specific threshold or
↪ trigger
  (e.g., "after 5pm" -> "in the evening"
↪ loses precision; "when raining" -> "in
↪ bad weather" is too vague)
- The conditional relationship was
↪ inverted, confused, or made ambiguous
- Critical specifics (time, place, context,
↪ trigger) were lost or distorted such that
  a reader could not reliably determine
↪ whether a given scenario satisfies the
↪ rule

A stored version is FAITHFUL if the
↪ condition is clearly and specifically
↪ preserved and
a reader could correctly answer whether a
↪ given context satisfies the rule.

USER:
ORIGINAL FACT:
{original_fact}

ALL_MEMORIES (the fact IS confirmed present
↪ somewhere in here):
{all_memories_formatted}

Find the memory entry corresponding to this
↪ fact and assess whether the stored
↪ version
faithfully preserves the qualifying
↪ condition and behavior, or whether it
↪ has a summary error.
```

Grading: *Conditional-Facts* Stage 2 — Summary check (Hard)

```
SYSTEM:
You are assessing the quality of a stored
↪ conditional rule.

The rule has already been confirmed to be
↪ recoverable from the memory store
↪ (possibly
from a single memory, possibly composed
↪ across multiple memories). Your job is to
evaluate whether the recoverable version
↪ faithfully preserves the CRITICAL
↪ information --
specifically the qualifying condition and
↪ the conditional behavior -- in a way
↪ that would
support correct downstream reasoning about
↪ whether a given context satisfies the
↪ rule.

A stored version has a SUMMARY ERROR if ANY
↪ of the following apply:
- The qualifying condition was dropped
↪ entirely (the behavior is stored, but no
↪ condition
  for this entity can be recovered from any
↪ combination of memories).
- The condition was generalized in a way
↪ that changes the specific threshold or
↪ trigger
  (e.g., "after 5pm" -> "in the evening"
↪ loses precision; "when raining" -> "in
↪ bad weather"
  is too vague).
- The conditional relationship was
↪ inverted, confused, or made ambiguous.
- The behavior and the condition appear in
↪ the store but cannot be linked to THIS
↪ entity
  (e.g., the condition is attached to a
↪ different person, or stated as generic
↪ context
  with no clear tie back to the entity).
- Critical specifics (time, place, context,
↪ trigger) were lost or distorted such
↪ that a
  reader could not reliably determine
↪ whether a given scenario satisfies the
↪ rule.

The stored version is FAITHFUL if a reader,
↪ by reading the relevant memory or
↪ composing
across multiple memories, could correctly
↪ determine whether a given context
↪ satisfies the
conditional rule for this specific entity.
↪ Composition across memories is
↪ acceptable --
do NOT penalize the memory system simply
↪ for distributing the behavior and
↪ condition
across separate entries.

USER:
COMPOSED CONDITIONAL RULE:
  Entity:    {entity}
  Behavior:  {behavior}
  Condition: {condition}
  In words:  "{entity} {behavior}
↪ {condition}." (paraphrase is fine)

ALL_MEMORIES (the rule IS confirmed
↪ recoverable somewhere in here, possibly
↪ across
multiple entries):
{all_memories_formatted}
```

```
Find the memory entry -- or the set of
↪ memory entries -- that together encode
↪ this rule
for this entity, and assess whether the
↪ recoverable version faithfully preserves
↪ the
qualifying condition and behavior, or
↪ whether it has a summary error.
```

Grading: *Conditional-Facts* Stage 3 — Retrieval check (Easy)

```
SYSTEM:
You are checking whether a specific fact
↪ was included in the memories retrieved
↪ and
shown to an AI model when it answered a
↪ question.

USER:
ORIGINAL QUESTION (used to search the
↪ memory store):
{query}

ORIGINAL FACT:
{original_fact}

RETRIEVED_MEMORIES (shown to the model):
{retrieved_memories}

Was the original fact included in
↪ RETRIEVED_MEMORIES? It is ok if the fact
↪ was
paraphrased or partially rephrased, as long
↪ as it was not changed in a way that
omits or alters information necessary to
↪ answer the query.
```

Grading: *Conditional-Facts* Stage 3 — Retrieval check (Hard)

```
SYSTEM:
(Same system message as Stage 3 Easy.)

USER:
ORIGINAL QUESTION (used to search the
↪ memory store):
{query}

COMPOSED CONDITIONAL RULE TO LOOK FOR:
  Entity:    {entity}
  Behavior:  {behavior}
  Condition: {condition}
  In words:  "{entity} {behavior}
↪ {condition}." (paraphrase is fine)

RETRIEVED_MEMORIES (shown to the model):
{retrieved_memories}

Is the composed conditional rule
↪ recoverable from RETRIEVED_MEMORIES --
↪ either from a
single retrieved entry or by composing
↪ across multiple retrieved entries --
↪ even if
paraphrased? The rule counts as recovered
↪ if both the behavior and the condition
↪ (linked
to this entity) are present in the
↪ retrieved set in some form. The rule
↪ counts as NOT
recovered if either the behavior or the
↪ condition is missing from what was shown
↪ to the
model, or if the link to this entity cannot
↪ be made out.
```

Grading: *Conditional-Facts* Stage 4 — Invocation check

```
SYSTEM:
You are checking whether an AI model
↪ correctly used a retrieved memory to
↪ answer
a conditional-fact question.

The model was shown the relevant memory and
↪ must answer YES if the condition is met,
or NO while citing the condition if it is
↪ not met.

USER:
ORIGINAL FACT (was present in retrieved
↪ memories):
{original_fact}

GROUND TRUTH ANSWER:
{ground_truth_answer}

MODEL RESPONSE:
{llm_response}

Did the model correctly invoke the memory
↪ and arrive at the right answer?
```

E.3.2 Coexisting-Facts grading

The first three stages run *per-fact* (one judge call per coexisting preference), independently classifying each preference as not_stored / summary_error / not_retrieved / correct. The final invocation check (Prompt E.3.2) is run only when all N facts reach correct and asks whether the model's response covers all of them.

Grading: *Coexisting-Facts* Stage 1 — Per-fact storage check

```
SYSTEM:
You are checking whether a specific
↪ preference was stored in a memory system.
The preference may have been paraphrased or
↪ compressed, but must still be clearly
↪ identifiable.

USER:
ORIGINAL PREFERENCE: {preference}
ORIGINAL FACT: {original_fact}

ALL_MEMORIES (complete memory store):
{all_memories_formatted}
```

```
Is this preference present in ALL_MEMORIES,
↪ even if paraphrased or lightly
↪ compressed?
It counts as present as long as the
↪ specific preference can still be clearly
↪ identified.
```

Grading: *Coexisting-Facts* Stage 2 — Per-fact summary check

```
SYSTEM:
You are assessing the quality of a stored
↪ preference.

The preference has already been confirmed
↪ to exist in the memory store. Your job
↪ is to
evaluate whether the stored version
↪ faithfully preserves the specific
↪ identity of this
preference in a way that would support
↪ correct downstream reasoning.

A stored version has a SUMMARY ERROR if ANY
↪ of the following apply:
- The preference was overgeneralized or
↪ merged with others, losing its distinct
↪ identity
  (e.g., "sushi" -> "Asian food"; "jazz" ->
↪ "music")
- The preference was corrupted or replaced
↪ with something different
- Critical identifying details were lost
↪ such that the model could not
↪ specifically cite
  this preference when answering a question

A stored version is FAITHFUL if the
↪ specific preference can still be clearly
↪ and
unambiguously identified from the stored
↪ memory.

USER:
ORIGINAL PREFERENCE: {preference}
ORIGINAL FACT: {original_fact}

ALL_MEMORIES (the preference IS confirmed
↪ present somewhere in here):
{all_memories_formatted}

Find the memory entry for this preference
↪ and assess whether the stored version
↪ faithfully
preserves the specific preference identity,
↪ or whether it has a summary error.
```

Grading: *Coexisting-Facts* Stage 3 — Per-fact retrieval check

```
SYSTEM:
You are checking whether a specific
↪ preference was included in the
memories retrieved and shown to an AI model
↪ when it answered a question.

USER:
ORIGINAL PREFERENCE: {preference}
ORIGINAL FACT: {original_fact}

ORIGINAL QUESTION (used to search the
↪ memory store):
{query}

RETRIEVED_MEMORIES (shown to the model):
{retrieved_memories}

Was this preference included in
↪ RETRIEVED_MEMORIES, even if paraphrased?
It counts as retrieved as long as the
↪ specific preference can be clearly
↪ identified.
```

Grading: *Coexisting-Facts* Stage 4 — Invocation check (when all facts correct)

```
SYSTEM:
You are checking whether an AI model
↪ correctly used all retrieved preferences
to answer a question.

All expected preferences were present in
↪ the retrieved memories. The model should
have mentioned all of them in its response.

USER:
EXPECTED_PREFERENCES (all were present in
↪ retrieved memories):
{expected_preferences}

GROUND TRUTH ANSWER:
{ground_truth_answer}

MODEL RESPONSE:
{llm_response}

Did the model correctly mention or account
↪ for ALL expected preferences in its
↪ response?
Synonyms and paraphrases count (e.g. "pasta
↪ dishes" covers "spaghetti").
```

E.3.3 Persona-Retrieval grading (batched)

Persona-Retrieval grading is *batched*: 20 traces share a single judge call per stage so the (large) memory store is sent only once per batch. Stage 1–3 are shared across direct and misleading queries; the final stage branches on the question type — direct queries are graded by the invocation check (Prompt E.3.3) and misleading queries by the abstention check (Prompt E.3.3). All five system messages and all five user templates are listed below.

Grading: *Persona-Retrieval* system messages (5 stages)

```
STORAGE_SYSTEM:
```

```
You are checking whether personal essays
↪ about named entities are stored in a
memory system. Each essay may have been
↪ paraphrased, compressed, or split across
↪ multiple
memory entries, but the entity's identity
↪ (name) and the substantive details from
↪ the
essay must still be recoverable to count as
↪ present.

You will receive ALL_MEMORIES (one shared
↪ store) and a list of independent
↪ ENTRIES, each
identified by an integer id. For each
↪ entry, return one verdict in the
↪ "results" array,
preserving the id.

SUMMARY_SYSTEM:
You are assessing the quality of stored
↪ personal essays.

Each essay has already been confirmed to
↪ exist somewhere in the memory store.
↪ Your job
is to evaluate whether the stored version
↪ preserves IDENTITY and KEY DETAILS
↪ faithfully --
in a way that would support correct
↪ downstream reasoning.

A stored version has a SUMMARY ERROR if ANY
↪ of the following apply:
- The entity's name was dropped or changed,
↪ so the stored details cannot be tied back
  to this specific person
- Specific, idiosyncratic details
↪ (allergies, possessions, rituals, hard
↪ constraints,
  strong preferences) were lost or
↪ generalized to the point that a reader
↪ could not
  recover the original meaning
- The stored version is so vague or generic
↪ that it would not distinguish this entity
  from a generic person with the same
↪ role/hobby

The following are NOT summary errors --
↪ treat them as acceptable:
- Multiple distinct entries existing under
↪ the same name. The entity may share a
↪ name
  with other people described in the store.
↪ As long as the entity's own essay is
  recoverable from at least one entry (or
↪ set of entries), this is fine -- do not
↪ flag
  it as a merged-identity error.
- The store containing memories about other
↪ people, whether related to the entity or
  unrelated. Only the entity's own essay
↪ needs to be faithfully preserved; the
  presence of additional people is
↪ acceptable.
- The essay being split across multiple
↪ entries, paraphrased, or compressed, as
↪ long
  as the entity's identity and key
↪ memorable details are recoverable.

A stored version is FAITHFUL if the entity
↪ is clearly named (in at least one entry)
and the key memorable details from the
↪ essay can be recovered from the relevant
memory or memories tied to that entity.

You will receive ALL_MEMORIES (one shared
↪ store) and a list of ENTRIES, each
identified by an integer id. For each
↪ entry, return one verdict in the
↪ "results"
array, preserving the id.

RETRIEVAL_SYSTEM:
You are checking whether each entity's
↪ essay was included in the memories
↪ retrieved
and shown to an AI model when it answered a
↪ question. Each entry has its own
RETRIEVED_MEMORIES (the subset shown for
↪ that question). Presence requires that
↪ the
entity be identifiable AND that the
↪ substantive personal details from the
↪ essay be
recoverable from what was shown to the
↪ model. Return one verdict per entry under
"results", preserving the id.

INVOCATION_SYSTEM:
You are checking whether an AI model
↪ correctly used a retrieved essay to
↪ answer a
question about the named entity. The model
↪ was shown the relevant memory and must
answer using the entity-specific details
↪ from the essay (not generic norms).
↪ Return
one verdict per entry in "results",
↪ preserving the id.

ABSTENTION_SYSTEM:
You are checking whether an AI model
↪ correctly ABSTAINED when asked about a
↪ person
the model has no information on.

Each entry's question names a DISTRACTOR --
↪ a different person who is NOT the entity
↪ in
the retrieved essay. The correct behavior
↪ is to acknowledge no information is
↪ available
about the distractor (or to ask for
↪ clarification). Applying the entity's
↪ essay details
to the distractor -- by treating them as if
↪ they were the entity, or by silently
↪ using
the entity's specifics to answer about the
↪ distractor -- is INCORRECT. Note: the
↪ model
may legitimately mention information about
↪ other people stored in memory if it does
↪ not
```

```
attribute the entity's essay details to the
↪ distractor. Return one verdict per entry
↪ in
"results", preserving the id.
```

Grading: *Persona-Retrieval* Stage 1 — Storage user template

```
ALL_MEMORIES (complete memory store, shared
↪ by all entries):
{all_memories_formatted}

For each entry below, determine whether the
↪ entity's essay is present in
ALL_MEMORIES, even if paraphrased,
↪ compressed, or split across multiple
↪ entries.
Presence requires that the entity be
↪ identifiable AND that the substantive
personal details from the essay be
↪ recoverable.

ENTRIES:
{entries}

Return one verdict per entry under
↪ "results", preserving the id (1..{n}).

(Each entry block has the form
   [id=k]
   ENTITY: {entity}
   ESSAY: {essay}
)
```

Grading: *Persona-Retrieval* Stage 2 — Summary user template

```
ALL_MEMORIES (complete memory store, shared
↪ by all entries):
{all_memories_formatted}

For each entry below, the essay IS
↪ confirmed present somewhere in
↪ ALL_MEMORIES.
Find the memory entry -- or set of entries
↪ -- that correspond to the entity's essay
and assess whether the stored version
↪ preserves identity and the key details,
↪ or
whether it has a summary error.

Reminder: it is acceptable for the store to
↪ contain other entries under the same
name (multiple personas) or memories about
↪ other people, related or otherwise.
Only judge whether each entity's own essay
↪ is faithfully recoverable from the
entries that describe it.

ENTRIES:
{entries}

Return one verdict per entry under
↪ "results", preserving the id (1..{n}).
```

Grading: *Persona-Retrieval* Stage 3 — Retrieval user template

```
For each entry, decide whether the entity's
↪ essay was included in that entry's
RETRIEVED_MEMORIES, even if paraphrased or
↪ split across multiple entries. Presence
requires that the entity be identifiable
↪ AND that the substantive personal details
from the essay be recoverable from what was
↪ shown to the model.

ENTRIES:
{entries}

Return one verdict per entry under
↪ "results", preserving the id (1..{n}).

(Each entry block has the form
   [id=k]
   ENTITY: {entity}
   QUESTION: {question}
   ESSAY: {essay}
   RETRIEVED_MEMORIES:
   {retrieved_memories}
)
```

Grading: *Persona-Retrieval* Stage 4 — Invocation user template (direct queries)

```
For each entry, decide whether the model
↪ correctly used the essay's specific
details to answer the question. The answer
↪ must align with the GROUND_TRUTH and
reflect the essay's specifics, not generic
↪ information.

ENTRIES:
{entries}

Return one verdict per entry under
↪ "results", preserving the id (1..{n}).

(Each entry block has the form
   [id=k]
   ENTITY: {entity}
   QUESTION: {question}
   ESSAY: {essay}
   GROUND_TRUTH: {ground_truth_answer}
   MODEL_RESPONSE: {llm_response}
)
```

Grading: *Persona-Retrieval* Stage 4 — Abstention user template (misleading queries)

```
For each entry, decide whether the model
↪ correctly abstained -- acknowledged it
↪ has
no information about the DISTRACTOR --
↪ rather than applying the ENTITY's essay
details to the DISTRACTOR. The model
↪ abstains correctly even if it generally
answers using common knowledge, as long as
↪ it does NOT smuggle in
ENTITY-specific details. Conflating the two
↪ people, or using the essay as if it
were about the distractor, is an abstention
↪ failure.
```

```
ENTRIES:
{entries}

Return one verdict per entry under
↪ "results", preserving the id (1..{n}).

(Each entry block has the form
   [id=k]
   ENTITY (essay is about): {entity}
   DISTRACTOR (named in question, NOT the
↪ entity): {distractor}
   ESSAY (about the entity): {essay}
   QUESTION (about the distractor):
↪ {question}
   MODEL_RESPONSE: {llm_response}
)
```

E.3.4 Long-Hop grading

Long-Hop grading runs three stages *per supporting fact* (storage, summary, retrieval) and then determines invocation correctness by deterministic letter parsing of the model's MCQ answer rather than via an LLM-judge call. Memory systems may legitimately merge several chain-links into a single memory entry; each per-fact stage explicitly accepts merged entries as long as the specific link is still recoverable.

Grading: *Long-Hop* Stage 1 — Per-fact storage check

```
SYSTEM:
You are checking whether a specific factual
↪ statement (one link in a multi-hop
↪ reasoning chain) is preserved in a
↪ memory store.

The memory store may have stored the fact
↪ verbatim, paraphrased it, OR merged
several chain-links together into one
↪ combined memory. Any of these forms
counts as STORED -- as long as the SPECIFIC
↪ link asserted by the target fact
can still be unambiguously identified from
↪ at least one memory entry. If the
link's two specific entities and the
↪ relation between them are recoverable,
mark fact_in_store=true.

USER:
TARGET FACT (one link of a reasoning chain):
{target_message}

ALL_MEMORIES (complete memory store,
↪ possibly with merged entries):
{all_memories_formatted}

Is the target fact present in ALL_MEMORIES
↪ -- verbatim, paraphrased, or as part
of a merged memory entry that still
↪ preserves the specific link between the
target fact's two entities? Mark
↪ fact_in_store=true only if the precise
relationship between the two specific
↪ entities is unambiguously recoverable.
```

Grading: *Long-Hop* Stage 2 — Per-fact summary check

```
SYSTEM:
You are assessing the quality of a stored
↪ chain-link.

The link has already been confirmed to
↪ exist somewhere in the memory store
(possibly inside a merged memory entry).
↪ Your job is to evaluate whether the
stored version faithfully preserves the
↪ specific link in a way that would
support correct downstream chain reasoning.

A stored version has a SUMMARY ERROR if ANY
↪ of the following apply:
- The link's two entities were
↪ collapsed/renamed/swapped, breaking
↪ identity.
- The relation between them was corrupted,
↪ weakened, or replaced.
- The link was over-merged with unrelated
↪ facts so that the specific link can
  no longer be cleanly extracted (e.g. the
↪ entities are listed but not in a
  way that preserves
↪ which-relates-to-which).
- A critical detail (e.g. direction of the
↪ relation) was lost.

A stored version is FAITHFUL if both
↪ entities of the link are clearly named in
some memory entry and the specific relation
↪ between them is unambiguous --
EVEN IF the memory entry also contains
↪ other chain-links from the same chain
(merging is allowed when each individual
↪ link is still recoverable).

USER:
TARGET FACT (one link of a reasoning chain):
{target_message}

ALL_MEMORIES (the link IS confirmed present
↪ somewhere in here, possibly merged):
{all_memories_formatted}

Find the memory entry (or entries) that
↪ cover this link and assess whether the
stored version faithfully preserves the
↪ specific relation between the two
entities, or whether it has a summary error.
```

Grading: *Long-Hop* Stage 3 — Per-fact retrieval check

```
SYSTEM:
You are checking whether a specific
↪ chain-link was included in the memories
retrieved and shown to an AI model when it
↪ answered a multi-hop question.

The retrieved memories may include
↪ verbatim, paraphrased, or merged
```

```
↪ versions.
A merged memory that still contains the
↪ link's specific relation counts as
"retrieved".

USER:
TARGET FACT (one link of a reasoning chain):
{target_message}

ORIGINAL QUESTION (used to query the memory
↪ store):
{query}

RETRIEVED_MEMORIES (shown to the model when
↪ it answered):
{retrieved_memories}

Was the target fact's specific link
↪ included in RETRIEVED_MEMORIES, even if
paraphrased or merged with other
↪ chain-links? Mark fact_in_retrieved=true
↪ only
if the precise relationship between the
↪ link's two entities is unambiguously
recoverable from the retrieved memories.
```

Invocation (deterministic, no LLM call). After the three per-fact stages succeed for every fact in a chain, the invocation outcome is read directly from the model's MCQ response. The parser tries, in order: (i) parsing the entire response as a JSON object and reading `selected_choice` / `answer` / `choice`; (ii) hunting for any embedded JSON object inside the response and trying the same keys; (iii) falling back to the last lone uppercase letter `A`–`E` that appears in the response. The parsed letter is compared against `correct_choice` from the dataset row to decide correct vs. reasoning_error.