MEMGUARD: Preventing Memory Contamination in Long-Term Memory-Augmented Large Language Models

Hyeonjeong Ha[1], **Jeonghwan Kim**[1], **Cheng Qian**[1],
Jiayu Liu[1], **William M. Campbell**[3], **Yue Wu**[3],
Yuji Zhang[1], **Kathleen McKeown**[2], **Dilek Hakkani-Tür**[1], **Heng Ji**[1]
[1]University of Illinois Urbana-Champaign, [2]Columbia University, [3]Capital One
{hh38, hengji}@illinois.edu

Abstract

Memory-augmented large language models extend reasoning beyond a fixed context window by maintaining long-term memory across interactions. However, existing memory systems often collapse stable user facts, episodic events, and behavioral rules into a shared space, allowing functionally distinct memories to be retrieved and used as interchangeable evidence. We identify this failure mode as *heterogeneous memory contamination*, where context-specific events become overgeneralized claims, or semantically relevant but functionally incompatible memories mislead generation. To this end, we introduce **MEMGUARD**, a type-aware memory framework that preserves functional memory boundaries during memory construction and retrieval. It assigns each memory an explicit functional role at write time, maintains relations across type-isolated memories, and selectively composes evidence only from necessary memory types, reducing contamination from irrelevant or functionally incompatible evidence. Across hallucination and long-horizon conversation benchmarks, MEMGUARD improves memory reliability by up to 28.27% while retrieving up to 5.8× fewer memory tokens than prior methods. These results suggest that reliable long-term reasoning depends on principled organization and selective use of heterogeneous memory.

1 Introduction

Recent memory-augmented large language models (LLMs) move beyond single-context reasoning by persisting knowledge across interactions and reusing it over long time horizons (Wu et al., 2024; Du et al., 2024; Ma et al., 2024; Park et al., 2023; Shridhar et al., 2020; Maharana et al., 2024; Liu et al., 2025). This capability is central to personalization and long-horizon reasoning (Zhong et al., 2024; Packer et al., 2023; Chhikara et al., 2025; Wang and Chen, 2025; Xu et al., 2025; Yan et al., 2025), but it also creates a new reliability risk: once noisy, unsupported, or misstructured knowledge is written to memory, it can be repeatedly retrieved and reused. Thus, hallucinations do not arise only at generation time; they can accumulate across the memory cycle, including writing and retrieval.

A key source of risk is that conversational memory is functionally heterogeneous: stable facts, event-specific observations, and behavioral rules may be topically related while serving different evidential roles. For example, a memory system may store: (i) a semantic constraint, *[Ibuprofen can trigger symptoms for people with asthma.]*; (ii) an episodic observation, *[User's friend took ibuprofen for headache and felt better.]*; and (iii) a procedural recommendation, *[Given headaches, recommend common over-the-counter pain relievers.]*. Although these memories concern the same topic, they should not be used interchangeably within the same retrieval context. When we disregard the functional distinctions among the evidence, heterogeneous memories can interfere during memory writing and retrieval, a failure mode we refer to as **heterogeneous memory contamination** (§3). As illustrated in Figure 1, the contamination can occur when an episodic observation is overgeneralized into an unsupported fact for the user, such as *[Ibuprofen is effective for headaches.]*, while omitting the asthma-related constraint at write-time. At retrieval time, when a query such as *[I have a headache now. Should I take ibuprofen?]* retrieves the episodic success case and procedural recommendation due to topical similarity, but under-ranks the semantic constraint required for safe reasoning. The model may then compose these memories into an unsupported answer (Hong et al., 2024; Ha et al., 2025; Jin et al., 2025; Park and Lee, 2024).

Our preliminary analysis on LoCoMo (Maharana et al., 2024) supports this view (§3): *unverifiability errors*, where the model provides responses despite the presence of insufficient evidence, rather

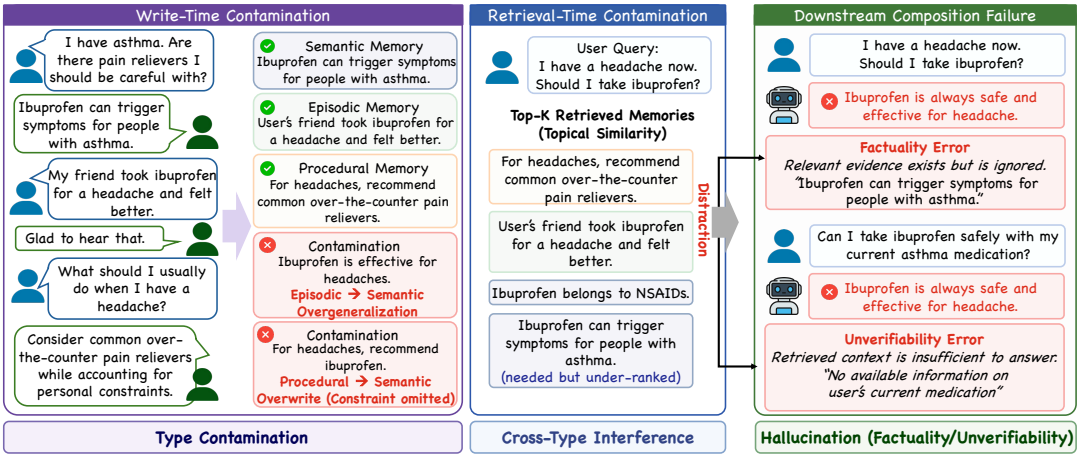


Figure 1: **Heterogeneous memory contamination.** Weak functional boundaries cause heterogeneous memories, including semantic constraints, episodic observations, and procedural guidance, to be stored, retrieved, and composed as interchangeable evidence. This contamination propagates across the memory writing and retrieval, leading to persistent hallucinations and degraded reasoning quality.

than abstaining, are predominantly associated with write-time contamination (97.7%), while *factuality errors*, where the answer is supported by the conversation history but the model generates an incorrect response, are frequently associated with retrieval-time contamination (63.8%). However, existing memory-augmented LLMs rarely treat memory-type boundaries as a reliability mechanism. They either treat it as a monolithic memory store and retrieve via semantic similarity (Packer et al., 2023; Zhong et al., 2024), or introduce structural variations to the memory (Ye et al., 2025; Hu et al., 2026; Jiang et al., 2026; Xu et al., 2025; Li et al., 2025; Yu et al., 2026; Yan et al., 2025) to stop at optimizing memory utility or retrieval effectiveness.

To address this gap, we propose **MemGuard**, a type-aware memory framework that treats functional boundaries as reliability constraints across memory writing, retrieval, and evidence composition. This design is motivated by cognitive theories of long-term memory, which posit functionally specialized systems that can be selectively recruited and composed according to the reasoning goal (Tulving et al., 1972; Eustache and Desgranges, 2008). MemGuard operationalizes this principle for memory-augmented LLMs by separating memories according to their functional roles and controlling how they are later retrieved and composed. At write time, it performs *type-aware memory reorganization*, decomposing conversational content into type-specific atomic memories and recording their dependencies in a relational knowledge graph (§4.1). At retrieval time, it performs *dynamic memory routing*, selecting memory types compatible with the query and composing evidence through the relational graph (§4.1). By disentangling functional distinctions during writing and enforcing type-compatible evidence composition during retrieval, MemGuard reduces type contamination and cross-type interference, thereby mitigating persistent hallucinations in memory-augmented LLMs.

Experiments on memory hallucination benchmarks and long-horizon conversational tasks show that MemGuard substantially improves memory reliability. On HaluMem (Chen et al., 2025), MemGuard achieves 89.53% (+28.27%) anti-hallucination accuracy and 71.49% (+9.38%) memory update correctness. On LoCoMo (Maharana et al., 2024), our framework retains competitive performance against state-of-the-art methods, while retrieving *21% fewer memory tokens*. These results indicate that selectively retrieving the right memory types by preserving functional boundaries is more effective than scaling retrieval indiscriminately. To summarize, our contributions are threefold:

- We identify *heterogeneous memory contamination* as a key source of persistent hallucination, caused by weak functional memory boundaries.
- We propose MemGuard, a type-aware memory framework that uses memory types to structure memory writing, route retrieval, and guide evidence composition.
- We show that MemGuard improves hallucination robustness while maintaining utility and with fewer retrieved memory tokens.

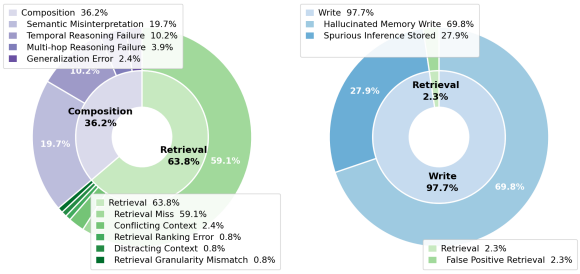
2 Related Work

2.1 Memory-Augmented LLMs

Memory-augmented LLMs support sustained interactions and long-horizon reasoning, with existing methods broadly falling into four paradigms (Table 4). *Flat semantic memory* stores conversational chunks or memories in vector storage and retrieves them via semantic similarity in retrieval-augmented generation (RAG) (Zhong et al., 2024; Chhikara et al., 2025; Ye et al., 2025). Although efficient, these methods treat history as an unordered set of propositions, making heterogeneous facts prone to inadvertent merging and write-time interference. *Structured and graph-based memory* models the relationships between events, entities, and concepts (Jiang et al., 2026; Rasmussen et al., 2025; Gutiérrez et al., 2024; Xu et al., 2026), but often retrieves mixed episodic and semantic nodes via similarity or topological search, leaving systems vulnerable to retrieval-time contamination. *Cognitive-inspired and hierarchical memory* adopts human-like divisions (Packer et al., 2023; Kang et al., 2025; Hu et al., 2026; Li et al., 2025; Sumers et al., 2023), such as working, episodic, semantic, and procedural memories, yet commonly relies on static type assignment and mixed retrieval, weakening boundaries between memory types. *Agentic and dynamic memory* transforms static storage with context-adaptive, model-driven decisions (Xu et al., 2025; Yan et al., 2025; Yu et al., 2026), but without explicit boundary preservation, it may amplify write-time contamination and uncontrolled associative loops. In contrast, MEMGUARD addresses the above-mentioned limitations by preserving the functional boundaries throughout the memory lifecycle.

2.2 Reliability in Memory-Augmented LLMs

Hallucination remains a central challenge in memory-augmented LLMs on long-horizon conversation tasks. Prior work on RAG shows that noisy, conflicting, or weakly relevant retrieved contexts can degrade consistency and induce hallucinations (Park and Lee, 2024; Hong et al., 2024; Ha et al., 2025; Liu et al., 2026). These risks amplify when retrieval operates over accumulated histories and persistent memory stores. HaluMem (Chen et al., 2025) shows that hallucinations can emerge across the memory lifecycle, including writing, retrieval, and reasoning, where incorrect knowledge can persist and reinforce erroneous response over time. Others examine retrieval interference and conversational drift from noisy or conflicting memory retrieval (Wu et al., 2025; Xu et al., 2025). However, existing methods mostly mitigate hallucination after unreliable knowledge has been stored or retrieved, overlooking the *heterogeneity* of knowledge items that differ in memory types.



(a) Factuality error. (b) Unverifiability error.

Figure 2: Error analysis across distinct hallucinations.

3 Heterogeneous Memory Contamination

Formulation Conversational memory is inherently heterogeneous, spanning episodic events, semantic facts, and procedural behaviors that serve distinct roles in downstream reasoning. When these functional boundaries are weak, topically related but functionally incompatible memories may be incorrectly updated, retrieved, or composed together. As a result, transient events may be stored as stable facts, or anecdotal evidence can override explicit constraints. We refer to this failure mode as **heterogeneous memory contamination**: the degradation of memory reliability caused by insufficient functional separation among memory types.

We characterize this contamination across the memory lifecycle (Figure 1): **Write-time contamination** occurs when memory construction stores incomplete, outdated, fabricated, or overgeneralized knowledge, causing unsupported content to persist. **Retrieval-time contamination** occurs when semantically related but functionally unsuitable memories are retrieved together, introducing cross-type noise that obscures the evidence needed for the query. *Composition failures* are typically downstream consequences of write- or retrieval-time contamination: the model incorrectly integrates contaminated or insufficient evidence, allowing irrelevant, conflicting, or weakly grounded memories to distort the final response.

Analysis We conduct a systematic error analysis on LoCoMo (Maharana et al., 2024). We categorize failures into (i) *factuality errors*, where the correct answer is recoverable from the conversa-

tion but the system generates an incorrect response, and (ii) *unverifiability errors*, where the conversation lacks sufficient evidence, and the model should abstain but instead answers, following prior work (Huang et al., 2025). Each failure is annotated by its memory-lifecycle source using `GPT-5.2`; full taxonomy definitions and annotation details are provided in Appendix B.

The results reveal distinct stage-wise patterns (Figure 2). Unverifiability errors are predominantly associated with write-time contamination (97.7%), indicating that unsupported or overgeneralized knowledge is often introduced before retrieval occurs. In contrast, factuality errors are associated with retrieval-time contamination (63.8%), where relevant evidence exists but is missed or underranked by memories from incompatible types. Across both error types, the common mechanism is *cross-type interference*: semantically plausible but functionally incompatible memories can make unanswerable queries appear answerable, or can compete with the evidence needed for correct answers. These findings show that persistent memory failures arise from weak functional separation among heterogeneous memory types. This motivates MEMGUARD's design: preserving functional boundaries through write-time memory reorganization and dynamic retrieval-time routing.

4 MEMGUARD

4.1 Write-Time Memory Reorganization

To mitigate heterogeneous memory contamination, MEMGUARD reorganizes conversational memory before storage. Given a conversation D, the write-time procedure converts raw dialogue into type-specific memory atoms, verifies their coverage, links related atoms through a relational graph, and stores them in type-isolated memory stores. This design enforces functional separation at the storage level while preserving explicit dependencies needed for downstream retrieval and reasoning. Formally, MEMGUARD maintains type-isolated memory stores: $\mathcal{M} = \{\mathcal{M}_\tau\}_{\tau \in \mathcal{T}}$, where $\mathcal{T} = \{\texttt{semantic}, \texttt{episodic}, \texttt{procedural}\}$, and a relational knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where nodes correspond to memory atoms and edges encode typed dependencies among them.

Type-Aware Knowledge Decomposition The LLM decomposes D into a set of non-overlapping memory atoms $A = \text{Decompose}(D) = \{a_j\}_{j=1}^{N}$. Each atom captures a single memory unit and is assigned exactly one functional type:

$$a_j = (\text{title}_j, \text{details}_j, \tau_j, t_j), \quad \tau_j \in \mathcal{T},$$

where title_j is the short description of a_j, details_j includes actual memory content, and t_j is the absolute timestamp derived from the conversation. The single-type constraint prevents heterogeneous knowledge from being compressed into a shared memory representation.

Self-Verified Extraction Because a single extraction pass may miss relevant information, MEMGUARD applies a self-verification step before writing. Given the conversation D and the initial atom set A, the verifier identifies information in D that is not covered by any atom in A. Let $\Delta(D, A)$ denote the recovered missing atoms. The final atom set is $A' = A \cup \Delta(D, A)$. Recovered atom must satisfy the same constraints as the initial decomposition: each must be atomic, non-overlapping, and tied to a single memory type.

Relational Knowledge Graph Although atoms are isolated by type, downstream reasoning often requires composing related facts, events, and procedures. To preserve such dependencies without merging heterogeneous content, MEMGUARD constructs a directed typed graph over the atom set: $\mathcal{V} = \{v_j \mid v_j \equiv a_j,\ a_j \in A'\}$. Edges encode typed semantic relations: $\mathcal{E} = \{(v_i, v_j, r) \mid v_i, v_j \in \mathcal{V},\ r \in \mathcal{R}\}$, where the relation type r is selected from the relation taxonomy. A directed edge (v_i, v_j, r) is added when atom a_i relates to atom a_j under relation r. For retrieval-time traversal, MEMGUARD also stores inverse edges:

$$(v_i, v_j, r) \in \mathcal{E} \Rightarrow (v_j, v_i, \texttt{inverse\_}r) \in \mathcal{E}.$$

Thus, $\mathcal{G}$ preserves cross-atom dependencies, while the atom contents remain type-isolated.

Type-Isolated Memory Writing Finally, each atom $a_j \in A'$ is routed to its corresponding typed stores $\mathcal{M}_{\tau_j}$. The atom is compared only with existing top-N (we set $N = 20$) relevant memories in the same store (i.e., type-local comparison), and the model assigns one write operation: $o_j \in \{\texttt{ADD}, \texttt{UPDATE}, \texttt{SKIP}\}$. `ADD` inserts a new memory, `UPDATE` revises an existing memory within the same type-specific store, and `SKIP` discards redundant or low-value atoms. By restricting deduplication and updates to $\mathcal{M}_{\tau_j}$, MEMGUARD prevents functionally distinct memories from overwriting

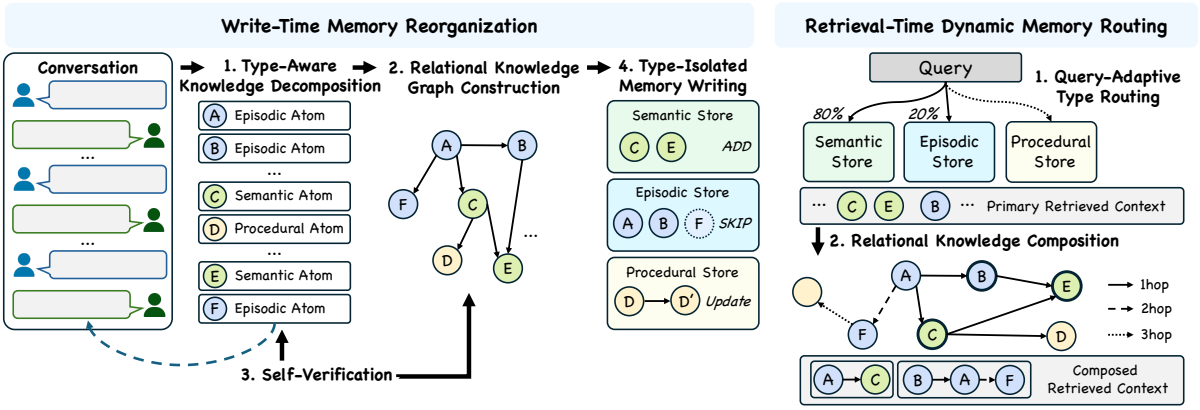


Figure 3: **Overview of MEMGUARD.** At write time, MEMGUARD reorganizes a conversation into atomic knowledge units, constructs directed relations among them, verifies missing information, and writes each atom to a type-isolated memory store. At retrieval time, the model routes queries adaptively to relevant memory types and selectively composes retrieved atoms via a relational knowledge graph, reducing cross-type interference. By preserving functional boundaries, MEMGUARD prevents heterogeneous memory contamination.

or absorbing one another. Cross-type relations are instead maintained only through $\mathcal{G}$, enabling controlled relational expansion during retrieval without compromising storage-level functional boundaries. We set K=20. Details are in Appendix C.1.

4.2 Retrieval-Time Dynamic Memory Routing

At retrieval time, MEMGUARD retrieves memories through two stages: query-adaptive type routing and relational composition. The model allocates the primary retrieval budget across type-isolated stores, restricting retrieval to memory types likely to be useful for the query. The retrieved memories are then expanded over the relational knowledge graph constructed at write time, allowing the system to recover relevant cross-memory dependencies without collapsing functional boundaries.

Query-Adaptive Type Routing Given a query q, MEMGUARD uses a prompt-based soft router to estimate the relevance of each memory type and output a confidence distribution over memory types: $\mathbf{w} = \rho_{\text{soft}}(q)$, $w_\tau \geq 0$, $\sum_{\tau\in\mathcal{T}} w_\tau = 1$. Each w_τ denotes the estimated utility of type τ for answering q. Given a retrieval budget K, we allocates a type-specific budget k_τ proportional to $\mathbf{w}$: $k_\tau = \lfloor w_\tau K \rfloor$, $\sum_{\tau\in\mathcal{T}} k_\tau = K$. For each type τ, retrieval is performed only over its corresponding store $\mathcal{M}_\tau$:

$$P_\tau = \text{Top}_{k_\tau}\left(\mathcal{M}_\tau, \text{sim}(q, m)\right),$$

where sim is the cosine similarity between L2-normalized embeddings $\phi(q)$ and $\phi(m)$. We use `text-embedding-3-small` as $\phi(\cdot)$. The primary retrieval set is: $P = \bigcup_{\tau\in\mathcal{T}} P_\tau, \quad |P| = K$. Unlike uniform retrieval over all stores, this routing mechanism makes retrieval capacity query-adaptive while preserving the type-isolation introduced at write time. Details are included in Appendix C.2.

Relational Knowledge Composition Type routing improves retrieval precision, but primary retrieval may still miss memories that are weakly similar to the query yet essential through relationships. To recover such memories, MEMGUARD expands each primary result over the relational graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. For each $p_i \in P$, let v_i be its corresponding graph node. Starting from v_i, breadth-first search (BFS) is performed up to $h_{\max}$ hops and collects reachable nodes:

$$\mathcal{N}_i = \left\{ (v', r, d) \;\middle|\; \begin{array}{l} v' \in \text{BFS}(v_i, h_{\max}), \\ r \in \mathcal{R},\ d \leq h_{\max} \end{array} \right\}.$$

where d is the hop distance and r is the relation label along the traversal path. Each reachable node is composed into a relation-aware context entry:

$$c_{i,v'} = p_i \oplus ([\rightarrow r], v'), \qquad (v', r, d) \in \mathcal{N}_i,$$

where $\oplus$ denotes concatenation with an explicit relation label. The composed entry is scored by query relevance with hop decay:

$$\text{score}(c_{i,d,v'}) = \text{sim}(q, c_{i,d,v'}) \cdot \lambda^{d-1}, \qquad \lambda \in (0, 1),$$

where λ is a hop-decay factor. We set $\lambda = 0.85$. The final retrieval context is obtained by reranking all graph-expanded entries and selecting the top-K:

$$C = \text{Top}_K\left(\{c_{i,d,v'} \mid p_i \in P,\ (v', r, d) \in \mathcal{N}_i\}, \text{score}\right).$$

The answer-generation LLM is conditioned on the query q and the composed context C. Overall, query-adaptive routing prevents irrelevant memory types from dominating retrieval, while graph-guided composition restores cross-memory dependencies through explicit typed relations. This provides controlled cross-type reasoning without weakening the functional boundaries enforced during write-time memory reorganization.

5 Experiment

5.1 Experimental Setup

Baselines To evaluate our method, we compare against memory-augmented LLM baselines spanning diverse memory management paradigms: (1) *flat semantic memory* methods, including Mem0 (Chhikara et al., 2025) and Memobase (Ye et al., 2025), (2) *structured/graph memory* methods, including Mem0-Graph (Chhikara et al., 2025), Zep (Rasmussen et al., 2025), and Supermemory (Shah et al., 2025), (3) *cognitive-inspired/hierarchical memory* approaches, including MIRIX (Wang and Chen, 2025) and MemOS (Li et al., 2025), and (4) *agentic memory* methods, such as A-Mem (Xu et al., 2025). Together, these baselines cover diverse memory structures, representations, and management.

Datasets We use HaluMem (Chen et al., 2025), which diagnoses hallucinations in memory-augmented LLMs by testing whether models avoid ungrounded memory writes, recognize insufficient evidence, and resist propagating erroneous information during memory writing and generation. This makes it well-suited for measuring contamination across the memory pipeline. We further evaluate long-term memory in realistic conversational settings, i.e., LongMemEval (Wu et al., 2024), LoCoMo (Maharana et al., 2024), and PerLTQA (Du et al., 2024), all requiring personalized memory retention and retrieval. Together, these benchmarks measure hallucination robustness and long-horizon memory reasoning across diverse settings.

Implementation Details For HaluMem, we follow the original protocol and evaluate on HaluMem-Medium due to computational cost. MEMGUARD uses `GPT-4.1-mini` as the base LLM, `GPT-4.1` as the LLM-as-a-judge. Baseline results are taken from HaluMem, which uses the stronger `GPT-4o` as both base LLM and judge; thus, the comparison is conservative for MEMGUARD while reducing cost. We exclude A-Mem and MIRIX because they are not reported in HaluMem and require method-specific APIs.

For utility evaluation on LoCoMo, PerltQA, and LongMemEval, we follow MemOS (Li et al., 2025), and use `GPT-4o-mini` as both base LLM and judge. We also report results under the HaluMem-consistent setting, using `GPT-4.1-mini` as the base LLM and `GPT-4.1` as the judge. Baseline numbers are taken from MemOS *(PerltQA results are unavailable)*, with A-Mem (Xu et al., 2025) reproduced under our utility setting.

Evaluation Metrics Following HaluMem (Chen et al., 2025), we evaluate memory systems across three tasks in the memory lifecycle: memory extraction, memory updating, and memory question answering. For **memory extraction**, we report recall, weighted recall, target memory precision, memory accuracy (anti-hallucination), and F1, measuring coverage, factuality, and overall extraction quality. For **memory updating**, we classify each required update as correct, hallucinated, or omitted. For **memory question answering**, we evaluate the end-to-end reliability after extraction, updating, retrieval, and generation. Each response is judged against the reference answer and key memory points, and categorized as correct, hallucinated, or omitted. We report the correctness, hallucination, and omission rates using LLM-as-a-Judge for memory updating and question answering.

For utility evaluation, we report answer accuracy using an LLM-as-a-judge, following prior work (Li et al., 2025). Accuracy on adversarial queries in LoCoMo is not available for existing works. Details are provided in the Appendix D.

5.2 Main Results

Hallucination Evaluation Table 1 shows that MEMGUARD reduces hallucination by improving the upstream memory construction and update. In extraction, MEMGUARD achieves the highest anti-hallucination accuracy (`Acc.`=89.53) and F1 score (94.15), indicating that type-aware writing produces cleaner and more complete memory units. In update, it obtains the highest correctness rate (70.79) and the lowest omission rate (28.86), indicating more reliable memory state management. These gains directly support our hypothesis: preserving functional boundaries reduces the risk of heterogeneous knowledge overwriting, obscuring, or conflicting with one another.

Since HaluMem does not directly evaluate retrieval hallucination, we further assess retrieval

Table 1: **Hallucination evaluation on HaluMem.** Hallucination is evaluated at three stages: memory extraction, update, and answer generation. R/P denote recall/precision; C/H/O denote Correct/Hallucination/Omission rates; and Acc. denotes anti-hallucination accuracy. Higher is better for R, P, Acc., F1, and C; lower is better for H and O. † indicates results taken from the original paper.

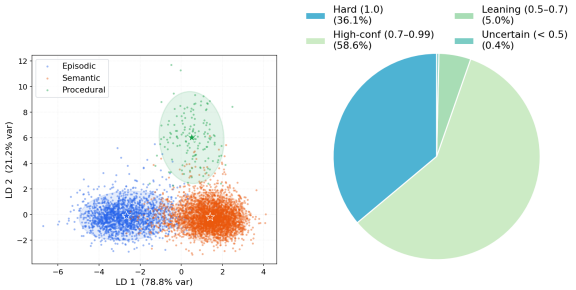
Method	Extraction					Update			Generation		
	R ↑	Weighted R ↑	Target P ↑	Acc. (%) ↑	F1 ↑	C ↑	H (↓)	O (↓)	C ↑	H (↓)	O (↓)
Mem0†	42.91	65.03	86.26	60.86	57.31	25.50	0.45	74.02	53.02	19.17	27.81
Mem0-Graph†	43.28	65.52	87.20	61.86	57.85	24.50	**0.26**	75.24	54.66	19.28	26.06
Zep†	–	–	–	–	–	47.28	0.42	52.31	55.47	21.92	22.62
MemOS†	74.07	84.81	86.25	59.55	79.70	62.11	0.42	37.48	**67.23**	**15.17**	**17.59**
Memobase†	14.55	25.88	92.24	32.29	25.13	5.20	0.55	94.25	35.33	29.97	34.71
Supermemory†	41.53	64.76	90.32	60.83	56.90	16.37	1.15	82.47	54.07	22.24	23.69
MEMGUARD	**90.47**	**93.95**	98.13	**89.53**	**94.15**	70.79	0.35	**28.86**	59.50	16.32	24.17
MEMGUARD(2 hop)	90.38	93.83	**98.16**	89.49	94.11	**71.49**	0.51	27.99	58.26	16.61	25.12

Table 2: **Utility evaluation on general long-horizon conversation benchmarks.** # Avg. Token refers to the average number of tokens in retrieved memories. Accuracy (%) is measured through LLM-as-a-Judge. †, ∗ indicates results taken from the original paper, while "–" indicates results not reported in the original paper.

Method	LoCoMo								
	# Avg. Token	Single Hop	Multi Hop	Open Domain	Temporal	Adversarial	Avg.	PerltQA	LongMemEval
Base LLM: GPT-4o-mini, Judge LLM: GPT-4o-mini									
Mem0†	1172	73.33	58.75	45.83	52.34	–	64.57	–	66.40
Mem0-Graph*	3616	65.71	47.19	75.71	58.13	–	68.44	–	–
Zep†	2701	66.23	52.12	33.33	54.82	–	59.22	–	63.80
MemOS	1589	81.09	67.49	55.90	75.18	–	75.80	–	77.80
Memobase†	2102	73.12	64.65	53.12	81.20	–	72.01	–	72.40
Supermemory†	500	67.30	51.12	42.67	31.77	–	55.34	–	58.40
MIRIX†	-	68.22	54.26	46.88	68.54	–	64.33	–	43.49
A-Mem	7244	60.64	38.54	62.78	23.68	68.83	56.34	77.04	47.00
MEMGUARD	1297	73.40	48.96	75.98	64.80	89.46	75.53	75.31	60.00
MEMGUARD (2 hop)	1250	73.76	51.04	76.58	62.93	90.13	75.78	75.64	60.00
Base LLM: GPT-4.1-mini, Judge LLM: GPT-4.1									
A-Mem	7244	58.87	45.83	64.80	44.24	64.35	59.62	80.62	62.25
MEMGUARD	1605	70.92	59.38	83.12	78.01	75.11	77.29	79.44	73.50
MEMGUARD (2 hop)	1540	75.89	58.33	84.54	77.57	76.46	79.10	80.07	72.25

quality by comparing the retrieved context with gold memory points for each query q (details in Appendix D.4). MEMGUARD achieves 89.87% and 90.24% retrieval accuracy with 1-hop and 2-hop retrieval, respectively, comparable to its extraction recall ($R = 90.47$). This suggests that type-aware writing and retrieval improve both memory construction and evidence accessibility. The improvement from 1-hop to 2-hop retrieval further suggests that graph-guided composition can recover relationally relevant memories missed by direct semantic similarity. While MEMGUARD does not lead on every generation metric, this is consistent with our scope: the method targets contamination before generation, rather than enforcing generation-time grounding. Moreover, MEMGUARD uses `GPT-4.1-mini`, whereas baselines use the stronger `GPT-4o`; despite this conservative setting, it maintains competitive generation hallucination and omission rates, showing that reducing write- and retrieval-time contamination can improve downstream memory faithfulness.

Utility Evaluation Table 2 shows the results on long-horizon memory benchmarks. On LoCoMo, MEMGUARD achieves 75.53% average accuracy under the `GPT-4o-mini` setting, within 0.27% of MemOS while using fewer retrieved tokens. Under the setting of `GPT-4.1-mini`/`GPT-4.1` for base LLM/judge, it reaches to 77.29%, surpassing A-Mem while using about 4.5× fewer retrieved tokens. The gain is most pronounced on adversarial queries, which test whether the system can abstain when evidence is insufficient. MEMGUARD outperforms A-mem by 20.63% under the `GPT-4o-mini` and by 10.76% under the `GPT-4.1-mini` setting. These results suggest that preserving functional memory boundaries improves answerability calibration. Type-aware writing reduces unsupported or functionally unsuitable memories from becoming reusable evidence, while query-adaptive routing limits retrieval to memory types relevant to the query, enabling MEMGUARD to answer when sufficient evidence exists and abstain more reliably when it does not.



(a) Memory Type Separation. (b) Query Routing Results.

Figure 4: Analysis of MEMGUARD at writing and retrieval.

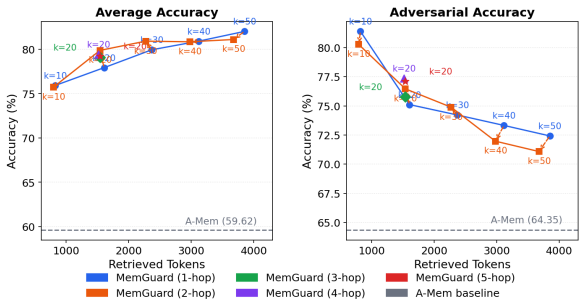


Figure 5: Results with different retrieval budgets and relational hop depth in relational knowledge composition.

6 Analysis

Type-Aware Memory Reorganization Keeps Functional Boundaries. To examine whether MEMGUARD mitigates write-time heterogeneous memory contamination, we visualize LoCoMo memories using Linear Discriminant Analysis (LDA) with `text-embedding-3-small` as an embedding. Figure 4a shows that semantic, episodic, and procedural memories form separable clusters, indicating that MEMGUARD stores memories in type-consistent regions rather than mixing functionally different knowledge in a shared representation space. The result supports our design: type-aware reorganization preserves functional boundaries.

Relational Composition Reduces Hallucination Without Sacrificing Utility. We analyze relational knowledge composition on LoCoMo by varying retrieval hop depth and top-K budget. As shown in Figure 5, increasing K improves average accuracy but lowers adversarial accuracy, indicating that larger contexts recover more answer-supporting evidence while also introducing weakly relevant memories that discourage abstention on ungrounded queries. In contrast, with K fixed at 20, deeper relational expansion maintains utility comparable to the best 2-hop setting while achieving stronger adversarial accuracy. This shows that relational composition improves evidence coverage more selectively than simply increasing retrieval volume, supporting our design of controlled structural expansion for reliable memory retrieval.

Table 3: Ablations of different components.

Relational Composition	Query-Adaptive Routing	LoCoMo	PerltQA	LongMemEval
✓	✓	**77.29**	**79.44**	**73.50**
✗	✓	75.53	74.18	70.50
✗	✗	68.83	69.71	60.75

Functional Boundaries Require Both Structure and Routing. Table 3 shows that relational knowledge composition and query-adaptive routing both contribute to MEMGUARD's effectiveness. Removing the relational knowledge graph while retaining query routing consistency degrades performance across all benchmarks, indicating that adaptive budget allocation alone is insufficient when retrieval relies mainly on semantic similarity. Removing both components causes the largest drop, showing that unstructured retrieval without type-aware routing is less reliable for long-horizon memory reasoning. These results suggest that the two components are complementary: the relational graph enables controlled evidence expansion over type-aware memory atoms, while query routing allocates retrieval capacity across memory stores based on routing confidence. Together, they preserve functional boundaries, reduce cross-memory interference, and improve downstream reasoning.

7 Conclusions and Future Work

We introduced MEMGUARD, a framework that improves long-term memory reliability by treating hallucination as a memory-governance failure across writing and retrieval. Our results show that persistent hallucinations often stem from heterogeneous memory contamination, where functionally distinct memories are stored or accessed without sufficient boundary preservation. By enforcing type-aware memory organization and query-adaptive retrieval, it reduces cross-memory interference while enabling structured evidence composition across memory types. Experiments across long-term memory benchmarks show that our method improves memory reliability, strengthens abstention on ungrounded queries, and maintains strong utility with fewer retrieved tokens, suggesting that scalable memory-augmented LLMs should treat memory as a structured and selectively accessed substrate for grounded reasoning. Future work can extend MEMGUARD with generation-time grounding to further reduce composition failures.

Limitation

MEMGUARD improves memory reliability by preserving functional boundaries during writing and retrieval, but it does not directly control generation-time behavior. As a result, although the retrieved memory context is correct and reliable, composition errors may still occur when the base LLM misinterprets or insufficiently grounds its response in the given context. In addition, MEMGUARD is implemented as an agentic memory framework built on LLM-based memory construction and retrieval, which may incur additional inference cost compared with fully trained end-to-end memory policies. Finally, our evaluation focuses on long-horizon conversational memory and hallucination benchmarks; future work should validate boundary-preserving memory management in broader settings, such as embodied agents and long-horizon decision-making tasks.

References

Ding Chen, Simin Niu, Kehang Li, Peng Liu, Xiangping Zheng, Bo Tang, Xinchi Li, Feiyu Xiong, and Zhiyu Li. 2025. Halumem: Evaluating hallucinations in memory systems of agents. *arXiv preprint arXiv:2511.03506*.

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

Francis Eustache and Béatrice Desgranges. 2008. Mnesis: towards the integration of current multisystem models of memory. *Neuropsychology review*, 18(1):53–69.

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in neural information processing systems*, 37:59532–59569.

Hyeonjeong Ha, Qiusi Zhan, Jeonghwan Kim, Dimitrios Bralios, Saikrishna Sanniboina, Nanyun Peng, Kai-Wei Chang, Daniel Kang, and Heng Ji. 2025. Mm-poisonrag: Disrupting multimodal rag with local and global poisoning attacks. *arXiv preprint arXiv:2502.17832*.

Kostas Hatalis, Despina Christou, Joshua Myers, Steven Jones, Keith Lambert, Adam Amos-Binks, Zohreh Dannenhauer, and Dustin Dannenhauer. 2023. Memory matters: The need to improve long-term memory in llm-agents. In *Proceedings of the AAAI Symposium Series*, volume 2, pages 277–280.

Giwon Hong, Jeonghwan Kim, Junmo Kang, Sung-Hyon Myaeng, and Joyce Jiyoung Whang. 2024. Why so gullible? enhancing the robustness of retrieval-augmented models against counterfactual noise. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2474–2495.

Chuanrui Hu, Xingze Gao, Zuyi Zhou, Dannong Xu, Yi Bai, Xintong Li, Hui Zhang, Tong Li, Chong Zhang, Lidong Bing, and 1 others. 2026. Evermemos: A self-organizing memory operating system for structured long-horizon reasoning. *arXiv preprint arXiv:2601.02163*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and 1 others. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55.

Dongming Jiang, Yi Li, Guanpeng Li, and Bingzhe Li. 2026. Magma: A multi-graph based agentic memory architecture for ai agents. *arXiv preprint arXiv:2601.03236*.

Bowen Jin, Jinsung Yoon, Jiawei Han, and Sercan Arik. 2025. Long-context llms meet rag: Overcoming challenges for long inputs in rag. In *International Conference on Learning Representations*, volume 2025, pages 37784–37822.

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

Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, and 1 others. 2025. Memos: An operating system for memory-augmented generation (mag) in large language models. *arXiv preprint arXiv:2505.22101*.

Jiayu Liu, Cheng Qian, Zhaochen Su, Qing Zong, Shijue Huang, Bingxiang He, and Yi R Fung. 2025. Costbench: Evaluating multi-turn cost-optimal planning and adaptation in dynamic environments for llm tool-use agents. *arXiv preprint arXiv:2511.02734*.

Jiayu Liu, Rui Wang, Qing Zong, Qingcheng Zeng, Tianshi Zheng, Haochen Shi, Dadi Guo, Baixuan Xu, Chunyang Li, and Yangqiu Song. 2026. Naacl: Noise-aware verbal confidence calibration for llms in rag systems. *arXiv preprint arXiv:2601.11004*.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362.

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.

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

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Seong-Il Park and Jay-Yoon Lee. 2024. Toward robust ralms: Revealing the impact of imperfect retrieval on retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 12:1686–1702.

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*.

Dhravya Shah, Mahesh Sanikommu, Yash, and 1 others. 2025. supermemory. https://supermemory.ai/. Accessed: 2025-11-05.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *Transactions on Machine Learning Research*.

Endel Tulving and 1 others. 1972. Episodic and semantic memory. *Organization of memory*, 1(381-403):1.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. 2023. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36:74530–74543.

Yu Wang and Xi Chen. 2025. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*.

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

Yaxiong Wu, Yongyue Zhang, Sheng Liang, and Yong Liu. 2025. Sgmem: Sentence graph memory for long-term conversational agents. *arXiv preprint arXiv:2509.21212*.

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. *arXiv 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. *arXiv preprint arXiv:2502.12110*.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, and 1 others. 2025. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*.

Gustavo Ye, Jinjia Gener, and 1 others. 2025. Memobase. https://github.com/memodb-io/memobase. Accessed: 2025-11-05.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. 2026. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. *arXiv preprint arXiv:2601.01885*.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 19724–19731.

A Related Work

A.1 Memory-Augmented LLMs

Memory-augmented LLMs have evolved to support sustained, multi-turn interactions and long-horizon reasoning (Wang et al., 2023; Hatalis et al., 2023; Wu et al., 2024; Du et al., 2024; Ma et al., 2024; Park et al., 2023; Shridhar et al., 2020; Maharana et al., 2024; Liu et al., 2025), with existing methods broadly falling into four paradigms. *Flat semantic memory* stores conversational chunks or memories in vector databases and retrieves them via semantic similarity via retrieval-augmented generation (RAG) (Zhong et al., 2024; Chhikara et al., 2025; Ye et al., 2025). While being computationally efficient, they treat conversational history as an unordered set of propositions and disparate pieces of knowledge are inadvertently merged, making it prone to interference from merged heterogeneous facts at write-time. *Structured and graph-based memory* explicitly models the relationships between events, entities, and concepts (Jiang et al., 2026; Rasmussen et al., 2025; Gutiérrez et al., 2024; Xu et al., 2026), yet still retrieves mixed episodic and semantic nodes indiscriminately via similarity or topological search, leaving them susceptible to retrieval-time contamination. *Cognitive-inspired and hierarchical memory* replicates the layered storage divisions of the human brain (Packer et al., 2023; Kang et al., 2025; Hu et al., 2026; Li et al., 2025), such as working, episodic, semantic, and procedural memories, but often relies on static type assignment and mixed retrieval, leaving weak boundaries between memory types that lead to cross-type interference. *Agentic and dynamic memory* transforms static database by employing active, context-adaptive decision processes optimized by the model (Xu et al., 2025; Yan et al., 2025; Yu et al., 2026); however, without explicit boundary preservation, they suffer from exacerbating write-time contamination and raising the risk of uncontrolled associated loops. In contrast, MEMGUARD targets the shared limitation across these paradigms: heterogeneous memory contamination. By preserving semantic boundaries throughout the memory lifecycle, MEMGUARD aims to reduce cross-type interference and downstream hallucination. Comparison between existing works and MEMGUARD is provided in Table 4.

B Heterogeneous Memory Contamination

B.1 Definition of Heterogeneous Memory Contamination

Formulation Memory-augmented LLMs continuously write, retrieve, and compose information from prior interactions to support long-context reasoning and personalization. However, conversational memory naturally contains heterogeneous forms of knowledge, including episodic events, semantic facts, and procedural behaviors, that differ in structure, temporal scope, and intended use. While existing memory systems improve scalability and organization through hierarchical architectures, semantic categories, or adaptive memory management, they still largely rely on shared retrieval spaces where heterogeneous memory types are stored and retrieved together. As a result, semantically distinct knowledge can interfere throughout the memory lifecycle, leading to what we term **heterogeneous memory contamination**: a failure mode in which weak semantic knowledge boundaries produce noisy memory interactions and degraded reasoning over long interaction horizons.

To better characterize this phenomenon, we organize contamination into three stages of the memory lifecycle: *write-time contamination*, *retrieval-time contamination*, and *composition-time contamination* (Figure 1). Write-time contamination occurs when heterogeneous knowledge is improperly constructed or updated during memory writing. This includes incomplete abstraction, incorrect updates, unsupported generalization, or fabrication beyond conversational evidence, causing contaminated knowledge to persist in memory over time. Retrieval-time contamination occurs when semantically mismatched memories are retrieved together, introducing noisy or cross-type information that interferes with identifying the relevant evidence for a query. Composition-time contamination occurs when retrieved memories are incorrectly integrated during reasoning, allowing irrelevant episodic details, procedural artifacts, or semantically unrelated memories to distort final answer generation. Although these failures occur at different stages, they frequently propagate throughout the memory pipeline, where early contamination during memory construction degrades downstream retrieval and reasoning. Fine-grained categories of contamination type within each stage are described in Table 5.

Table 4: Comparison with existing works. Unlike prior methods, MEMGUARD explicitly preserves functional knowledge boundaries via type-aware memory reorganization at writing and dynamic memory routing at retrieval.

Method	Memory Structure	Representation	Memory Sharing	Write-time Strategy	Retrieval Strategy	Routing Mechanism
Flat Retrieval–based Memory						
MemoryBank (Zhong et al., 2024)	Flat / Profile-based	Text summaries + user portrait	Shared	Summarization + memory updating	Similarity-based recall	✗
Mem0 (Chhikara et al., 2025)	Flat / Hybrid	Text	Shared	Extract + consolidate salient facts	Similarity search	✗
MemoBase (Ye et al., 2025)	Profile-based	Structured text profile	Shared	Profile extraction + update	Profile retrieval / similarity search	✗
Structured / Graph-based Memory						
Mem0-Graph (Chhikara et al., 2025)	Graph-enhanced	Entity-relation graph + text	Shared	Extract + link entities	Graph traversal + similarity search	✗
MAGMA (Jiang et al., 2026)	Multi-graph	Semantic/temporal/causal/entity graphs	Shared	Multi-graph construction	Policy-guided graph traversal	Partial
Zep (Rasmussen et al., 2025)	Temporal KG	Text + Entities/Relations	Shared	Incremental extraction & entity resolution	Graph traversal + similarity	✗
Agentic / Dynamic Memory						
A-MEM (Xu et al., 2025)	Network/Graph-like	Structured notes + links	Shared	Note construction + link generation + evolution	Link-aware retrieval + vector search	✗
Memory-R1	Flat	Text entries	Shared	RL-guided operations	RAG + Memory Distillation	✗
Cognitive-Inspired / Hierarchical Memory						
MemOS (Li et al., 2025)	System-level typed memory	Parametric / activation / plaintext memory	Partially shared	Memory governance via MemCube	Managed memory access	Partial
EverMemOS (Hu et al., 2026)	Hierarchical / Self-organizing	Text / structured memory units	Shared	Engram-inspired consolidation	Hierarchical / lifecycle-aware retrieval	✗
MemGPT (Packer et al., 2023)	Hierarchical	Text	Shared	LLM-controlled memory edits	Context paging + archival search	Partial
MIRIX (Wang and Chen, 2025)	Typed / Modular	Text memories by type	Partially shared	Static type assignment / consolidation	Mixed type-wise retrieval	✗
MemGuard (Ours)	**Typed + Dynamic**	**Atomic units + relational knowledge graph**	**Isolated**	**Dynamic routing**	**Query-aware memory routing**	✓

Stage	Type	Category	Definition
Write	F	`memory_missing`	Relevant information appears in the conversation but is never written to memory.
		`abstraction_error`	Information is stored with distorted, incomplete, or overly compressed details, losing important specificity.
		`update_error`	A previously stored memory is not updated when new information supersedes or corrects it, resulting in stale knowledge.
	U	`hallucinated_memory_write`	A memory entry is fabricated without grounding in the conversation.
		`spurious_inference_stored`	An unjustified inference is stored as a fact based on weak or indirect evidence.
Retrieval	F	`retrieval_miss`	A correct memory exists in storage but is not retrieved.
		`retrieval_ranking_error`	The correct memory is retrieved but ranked below less relevant or incorrect memories.
		`retrieval_granularity_mismatch`	Retrieved memories are too coarse or too fine-grained, preventing correct reasoning.
		`conflicting_context`	Retrieved memories contain contradictory information, and the correct one is not properly selected.
		`distracting_context`	Irrelevant or weakly related memories are retrieved and interfere with reasoning.
	U	`false_positive_retrieval`	Retrieved memories are topically similar but do not contain the required information.
		`context_overextension`	Partial or weak evidence from retrieved memory leads the model to generate unsupported conclusions.
Composition	F	`temporal_reasoning_failure`	The model fails to correctly handle temporal order, recency, or updates across time.
		`multi_hop_reasoning_failure`	The model fails to combine multiple retrieved facts to reach a correct conclusion.
		`semantic_misinterpretation`	The model misinterprets the meaning or implication of retrieved content.
		`generalization_error`	The model incorrectly applies or fails to apply knowledge beyond its valid scope.
	U	`parametric_memory_intrusion`	The model relies on pretrained knowledge instead of retrieved memory, overriding relevant context.
		`unanswerable_recognition_failure`	The model fails to recognize insufficient information and generates an answer instead of abstaining.
		`hallucinated_reasoning_chain`	The model fabricates a multi-step reasoning process without sufficient grounding in retrieved memory.

Table 5: Fine-grained taxonomy of memory contamination errors across write-time, retrieval-time, and composition-time, covering two types of hallucinations: factuality errors (F) and unverifiability errors (U).

Analysis Based on this formulation, we conduct a systematic error analysis on existing memory frameworks (Chhikara et al., 2025) using the LoCoMo (Maharana et al., 2024) benchmark. We divide hallucinations into two categories: *factuality errors*, where the correct answer is recoverable from the conversational history but the system fails to generate it correctly, and *unverifiability errors*, where the conversational history does not contain sufficient evidence for answering and the model should abstain but instead generates unsupported content. Using GPT-5.2 as an LLM-as-a-Judge, we annotate failures across the memory lifecycle and categorize them according to the contamination taxonomy described above, including write-time, retrieval-time, and composition-time failures.

Our annotation results reveal distinct stage-wise failure patterns (Figure 2). Unverifiability errors are predominantly associated with *write-time contamination* (97.7%), where unsupported, overgeneralized, or improperly abstracted knowledge becomes persistently stored in memory and later reinforced across future interactions. In contrast, factuality errors are more strongly associated with *retrieval-time contamination* (63.8%), where the correct knowledge exists in memory but is not reliably retrieved or appropriately prioritized during reasoning.

To further understand the underlying causes of these failures, we analyze memory interactions across retrieved evidence and identify a common mechanism behind many errors: *cross-type retrieval and collision*. Because heterogeneous memory types are often stored and retrieved within shared memory spaces, semantically distinct knowledge can become entangled during retrieval and reasoning. As a result, topically related but semantically incompatible memories, such as transient episodic observations, procedural instructions, or loosely associated contextual details, are frequently retrieved together and compete during reasoning. We observe that many factuality errors arise not because the correct memory is absent, but because it is overshadowed by irrelevant cross-type memories during retrieval. Similarly, unverifiability errors often emerge when weakly grounded or unsupported memories collide with relevant evidence and become reinforced during generation. Together, these findings suggest that persistent memory failures stem not only from retrieval quality itself, but more fundamentally from weak semantic boundaries between heterogeneous memory types throughout the memory lifecycle.

B.2 Annotation Pipeline

To analyze heterogeneous memory contamination, we annotate each incorrect model response with an LLM-based stepwise pipeline. We first distinguish between two failure types: *factuality errors*, where the question has a valid ground-truth answer but the model answers incorrectly, and *hallucinations*, where the question is unanswerable but the model produces a non-abstaining answer. We randomly

sample 300 incorrect cases from existing methods for annotation. All annotations are produced by LLM-as-a-judge calls with temperature set to 0, using GPT-5.2 as the LLM.

Factuality Error Pipeline For factuality errors, we identify the earliest stage at which the correct answer is available. We first check whether the ground-truth answer can be derived from the retrieved context. If so, the failure is attributed either to *retrieval errors*, where the retrieved context is insufficient or misleading despite containing relevant information, or to *composition errors*, where sufficient evidence is retrieved but the model fails to reason over it correctly. If the answer is not recoverable from the retrieved context, we check the full memory storage. When the answer exists in storage but is not retrieved, we label the failure as a *retrieval miss*. If the answer is absent from memory storage, we inspect the original conversation and classify the failure as a write-time error, including *memory missing*, *abstraction error*, or *update error*.

Step 1: Is Ground-Truth Answerable from Retrieved Context?

```
You are an expert evaluator of memory-augmented language
 models.

## Task
Determine whether the **ground-truth answer** can be
correctly derived from the **retrieved context**
provided to the model.

## Retrieved Context
{{retrieved_context}}

## Ground-Truth Answer
{{ground_truth}}

## Question
{{question}}

## Instructions
- Answer YES if a careful reader of the retrieved
context could produce the ground-truth answer.
- Answer NO if key information needed to arrive at the
ground-truth answer is absent or insufficient.
- Do NOT consider whether the model *chose* to use the
context correctly - only whether the context *contains*
what is needed.

Return JSON only:
```json
{"answerable": true_or_false, "reasoning": "<1-2
sentences citing specific evidence>"}
```
```

Step 1b: Disambiguation - retrieval error vs. composition error

```
You are an expert evaluator of memory-augmented language
 models.

The ground-truth answer **can** be derived from the
retrieved context, yet the model produced an incorrect
answer.
Determine whether the root cause is a **retrieval-
quality problem** or a **composition/reasoning problem
**.

## Retrieved Context
{{retrieved_context}}

## Model Response
{{model_response}}

## Ground-Truth Answer
{{ground_truth}}

## Question
{{question}}

---

## Distinction

**Retrieval-quality problem** - The composition of *what
 was retrieved* caused the error:
- A correct memory is present but ranked lower than a
misleading one.
- Memories are at the wrong granularity (too coarse or
too fine).
- The retrieved set contains two directly contradictory
statements.
- An irrelevant memory in the retrieved set distracted
the model.
In all these cases, improving the *retriever* - not the
*reasoner* - would fix the error.

**Composition/reasoning problem** - The retrieval
content was sufficient; the model's *reasoning* failed:
- The model failed to reason about time or sequence
correctly.
- The model had all pieces but failed to chain them.
- The model misinterpreted the meaning of a retrieved
phrase.
- The model applied a specific fact beyond its intended
scope.
In these cases, improving the *reasoner* - not the *
retriever* - would fix the error.

Return JSON only:
```json
{"is_retrieval_quality_error": true or false, "reasoning
": "<1-2 sentences explaining the deciding signal>"}
```
```

Fine-Grained Retrieval Error Annotation

```
You are an expert evaluator of memory-augmented language
 models.

The ground-truth answer **can** be derived from the
retrieved context, yet the model's error
stems from a **retrieval-quality problem** - the
composition of retrieved memories caused the failure.
Classify the specific retrieval-quality error type.

## Conversation Context
{{conversation}}

## Retrieved Context
{{retrieved_context}}

## Model Response
{{model_response}}

## Ground-Truth Answer
{{ground_truth}}

## Question
{{question}}

---

## ERROR TAXONOMY (retrieval-quality errors)

### retrieval_ranking_error
```

```
The correct memory IS present in the retrieved context
but is outranked by a less-relevant item
that the model preferentially uses.
- The correct answer is retrievable from the set; model'
s answer aligns with a weaker, lower-ranked item.
- Example: Retrieved ["User prefers vegetarian meals  (
rank 2)", "User ate a burger at a company event (rank 1)
"] -> model recommends burger.

### retrieval_granularity_mismatch
Retrieved memories are at the wrong level of detail -
too coarse (bundling unrelated facts) or too fine
(splitting linked facts so the inference chain breaks).
- Individual items are not wrong per se, but their
granularity prevents the correct inference.
- Example: Chunk A "User takes medication X" + Chunk B "
X interacts with alcohol" - only A retrieved ->
incomplete picture.

### conflicting_context
The retrieved set contains two or more directly
contradictory statements; the model picks the wrong one
or fails to reconcile them.
- Both a correct and an incorrect (or two incompatible)
claims appear in the retrieved set.
- Example: Retrieved ["Goal weight is 70 kg", "Target is
 80 kg"] -> model uses wrong value.

### distracting_context
The retrieved set contains irrelevant information that
is not directly contradictory but misleads the model.
- No direct contradiction; an off-topic or loosely-
related fact pulls the answer in the wrong direction.
- Example: "User prefers Python " + "User's team uses
Java at work" -> model answers "Java".

---

## Tiebreaker Rules
| Tie | Resolution |
|-----|------------|
| retrieval_ranking_error vs. distracting_context | If
the correct answer IS in the set (just lower-ranked) ->
ranking_error. If irrelevant noise wins without a
correct-answer counterpart -> distracting_context. |
| conflicting_context vs. retrieval_ranking_error | If
there is a direct factual contradiction ->
conflicting_context. If correct item is simply
outweighed by a less-relevant one without direct
contradiction -> ranking_error. |

---

## Proposing a Novel Sub-type
If none of the four categories adequately describe the
failure:
1. Explain specifically why each existing type fails.
2. Use `snake_case`; describe the mechanism, not the
symptom.
3. Set `"is_novel_type": true` and fill `"
novel_type_definition"` with one sentence.
4. Set `"confidence"` to `"low"` or `"medium"` - never
`"high"` for novel types.

---

Return JSON only:
```json
{
  "error_type": "<retrieval_ranking_error |
retrieval_granularity_mismatch | conflicting_context |
distracting_context | or new name>",
  "is_novel_type": false,
  "novel_type_definition": null,
  "confidence": "<high | medium | low>",
  "explanation": "<2-4 sentences: what went wrong and
why this category fits>",
  "alternative_considered": "<second-most-likely
category and why it was ruled out>",
  "evidence": {
    "retrieved_context": "<relevant retrieved snippet>",
    "model_output": "<key incorrect phrase>",
    "ground_truth": "<correct answer>"
  }
}
```
```

Fine-Grained Composition Error Annotation

```
You are an expert evaluator of memory-augmented language
 models.

The ground-truth answer **can** be derived from the
retrieved context AND the retrieval quality
is adequate. The model's error is a **composition or
reasoning failure** - it had what it needed
but failed to produce the correct answer through its
reasoning process.
Classify the specific reasoning error.

## Conversation Context
{{conversation}}

## Retrieved Context
{{retrieved_context}}

## Model Response
{{model_response}}

## Ground-Truth Answer
{{ground_truth}}

## Question
{{question}}

---

## ERROR TAXONOMY (composition / reasoning errors only)

### temporal_reasoning_failure
Retrieved context contains time-stamped or sequenced
information; model fails to correctly interpret
recency, order, or duration.
- All needed facts are retrieved; error arises from
mishandling the time dimension.
- Example: Retrieved ["Lived in NYC in 2021", "Moved to
Austin in 2023"] -> model answers "NYC".

### multi_hop_reasoning_failure
All required facts are present in retrieved context;
model fails to chain them together to derive
the correct answer.
- No retrieval issue; answer requires combining 2+ facts
 the model has access to.
- Example: "Alice is Bob's manager" + "Bob works in
London" -> "Which office does Alice's report work in?"
-> model: "Unknown".

### semantic_misinterpretation
Retrieved context is complete and correctly retrieved;
model misunderstands the meaning of a word,
phrase, or implicit reference.
- Error is lexical, pragmatic, or referential - not a
retrieval issue.
- Example: "User said they are 'done' with keto" ->
model interprets "done" as "finished successfully"
rather than "quitting".

### generalization_error
Model correctly retrieves a specific fact but
incorrectly generalizes or over-applies it beyond
its intended scope, or fails to apply a general rule to
a clear instance.
- Memory and retrieval are correct; error is about scope
 application.
- Example: "User dislikes cilantro in savory dishes" ->
model: "User has no food restrictions."

---

## Tiebreaker Rules
| Tie | Resolution |
|-----|------------|
| multi_hop_reasoning_failure vs.
semantic_misinterpretation | Model has all pieces but
fails to connect them -> multi_hop. Model misreads the
meaning of a single retrieved phrase ->
semantic_misinterpretation. |
| generalization_error vs. temporal_reasoning_failure |
Scope error is about time/recency ->
temporal_reasoning_failure. Scope error is about
category/generality, not time -> generalization_error. |

---

## Proposing a Novel Sub-type
If none of the four categories adequately describe the
failure:
1. Explain specifically why each existing type fails.
2. Use `snake_case`; describe the mechanism, not the
symptom.
3. Set `"is_novel_type": true` and fill `"
novel_type_definition"` with one sentence.
4. Set `"confidence"` to `"low"` or `"medium"`.

---

Return JSON only:
```json
{
  "error_type": "<temporal_reasoning_failure |
multi_hop_reasoning_failure | semantic_misinterpretation
 | generalization_error | or new name>",
  "is_novel_type": false,
  "novel_type_definition": null,
  "confidence": "<high | medium | low>",
  "explanation": "<2-4 sentences: what went wrong and
why this category fits>",
  "alternative_considered": "<second-most-likely
category and why it was ruled out>",
  "evidence": {
    "retrieved_context": "<relevant retrieved snippet>",
    "model_output": "<key incorrect phrase>",
    "ground_truth": "<correct answer>"
  }
}
```
```

Step 2: Is Ground-Truth in Constructed Memory?

```
You are an expert evaluator of memory-augmented language
 models.

## Task
Determine whether the **ground-truth answer** can be
correctly derived from the **full memory storage**
(all stored memories, not just what was retrieved).

## Full Memory Storage
{{memory_storage}}

## Ground-Truth Answer
{{ground_truth}}

## Question
{{question}}

## Instructions
- Answer YES if any entry in memory storage contains the
 fact(s) needed to produce the ground-truth answer.
- Answer NO if the needed fact is absent from memory
storage entirely.
- Do NOT consider whether retrieval surfaced it - only
whether it exists anywhere in storage.

Return JSON only:
```json
{"answerable": true_or_false, "reasoning": "<1-2
sentences citing the specific memory entry or noting
absence>"}
```
```

Step 3: Is Ground-Truth in Original Conversation History?

```
You are an expert evaluator of memory-augmented language
 models.

## Task
Determine whether the **ground-truth answer** can be
correctly derived from the **original conversation**.
```

```
## Original Conversation
{{conversation}}

## Ground-Truth Answer
{{ground_truth}}

## Question
{{question}}

## Instructions
- Answer YES if the conversation contains the fact(s)
needed to produce the ground-truth answer.
- Answer NO if the conversation does not contain
sufficient information.

Return JSON only:
```json
{"answerable": true_or_false, "reasoning": "<1-2
sentences citing the relevant conversation turn or
noting absence>"}
```
```

Fine-Grained Write-Time Error Annotation

```
You are an expert evaluator of memory-augmented language
 models.

The ground-truth answer **cannot** be derived from
either the retrieved context or the full memory
storage, but **can** be derived from the original
conversation.
Classify which write-time memory failure caused the
information to be lost or distorted.

## Conversation Context
{{conversation}}

## Memory Storage
{{memory_storage}}

## Model Response
{{model_response}}

## Ground-Truth Answer
{{ground_truth}}

## Question
{{question}}

---

## ERROR TAXONOMY (write-time / memory-origin errors)

### memory_missing
Relevant information was present in the conversation but
 was **never written to memory at all**.
- The correct answer can be derived from the
conversation.
- That information is entirely absent from memory
storage.
- This is an **omission** failure, not contamination of
existing content.
- Example: Conversation "I'm vegetarian and have been
for 10 years." Memory: (no dietary info stored) ->
memory_missing.

### abstraction_error
Information was stored in memory, but **incorrectly,
incompletely, or with distorted meaning** -
without a conflicting update event.
- Memory exists for the topic but key details are wrong,
 missing, or negated incorrectly.
- There is no temporal conflict - this is a one-time
write error.
- Example: Conversation "I'm allergic to shellfish, not
just sensitive - I carry an EpiPen."
  Memory: "User is mildly sensitive to shellfish." ->
abstraction_error.

### update_error
A new piece of information should have updated an old
entry, but the memory system failed -
keeping only the outdated version, only the new version
when the old was still valid,
or creating a corrupt merged entry.
- At least **two temporally distinct** conversation
turns address the same fact.
- Memory fails to reflect the correct final state.
- Example: Turn 1 "I live in Berlin." Turn 5 "I just
relocated to Amsterdam."
  Memory: "User lives in Berlin." -> update_error.

---

## Disambiguation Table
| memory_missing | abstraction_error | update_error |
|---------------|-------------------|-------------|
| Info absent from storage entirely | Info in storage
but distorted | Info in storage but stale |
| No memory entry exists for topic | Entry exists,
content is wrong | Two competing temporal versions exist
 |
| Single write never happened | Single write happened
incorrectly | Multi-turn update failed |

**Tiebreaker:** Later conversation turn that should
overwrite an earlier memory -> update_error.
Single write was simply wrong with no competing version
-> abstraction_error.
No write happened at all -> memory_missing.

---

## Proposing a Novel Sub-type
If none of the three categories adequately describe the
failure:
1. Explain specifically why each existing type fails.
2. Use `snake_case`; describe the mechanism, not the
symptom.
3. Set `"is_novel_type": true` and fill `"
novel_type_definition"` with one sentence.
4. Set `"confidence"` to `"low"` or `"medium"`.

---

Return JSON only:
```json
{
  "error_type": "<memory_missing | abstraction_error |
update_error | or new name>",
  "is_novel_type": false,
  "novel_type_definition": null,
  "confidence": "<high | medium | low>",
  "explanation": "<2-4 sentences: what went wrong and
why this category fits>",
  "alternative_considered": "<second-most-likely
category and why it was ruled out>",
  "evidence": {
    "conversation": "<relevant conversation snippet>",
    "memory_storage": "<relevant stored entry or 'N/A -
not present'>",
    "model_output": "<key incorrect phrase>",
    "ground_truth": "<correct answer>"
  }
}
```
```

Unverifiability Error Annotation Pipeline For hallucinations, we trace the origin of the fabricated answer. We first check whether the hallucinated content can be linked to memory storage. If so, we label it as a *memory-time contamination*, distinguishing between hallucinated memory writes with no conversational grounding and spurious inferences that over-extend loosely related evidence. If the hallucination is not traceable to storage, we check the retrieved context. Cases grounded in retrieved but non-answering or only partially relevant context are labeled as *retrieval-time contamina-*

tion, including false positive retrieval and context overextension. If neither memory storage nor retrieved context explains the hallucination, we attribute the error to *generation-time contamination*, such as parametric memory intrusion, unanswerable recognition failure, or hallucinated reasoning chains.

Step 1: Is Hallucination traceable to Constructed Memory?

```
You are an expert evaluator of memory-augmented language
 models.

## Task
Determine whether the model's **hallucinated answer**
can be traced back to any entry in
**memory storage** - even if that entry is fabricated,
inferred, or only loosely related.

This question has **no correct answer** derivable from
the original conversation.
The model generated an answer anyway.

## Memory Storage
{{memory_storage}}

## Model's Hallucinated Answer
{{model_response}}

## Question
{{question}}

## Instructions
- Answer YES if any memory entry - even a fabricated,
over-inferred, or distorted one -
  could plausibly be the source the model drew on to
produce this answer.
- Answer NO if the model's answer has no traceable
connection to any entry in memory storage.

Return JSON only:
```json
{"traceable_to_memory": true_or_false, "reasoning":
"<1-2 sentences identifying the relevant memory entry or
 confirming absence>"}
```
```

Fine-Grained Write-Time Annotation

```
You are an expert evaluator of memory-augmented language
 models.

## Task
Determine whether the model's **hallucinated answer**
can be traced back to anything in the
**retrieved context** - even loosely related or only
partially grounding content.

This question has **no correct answer** derivable from
the original conversation.

## Retrieved Context
{{retrieved_context}}

## Model's Hallucinated Answer
{{model_response}}

## Question
{{question}}

## Instructions
- Answer YES if any retrieved item - even weak,
tangential, or only superficially relevant -
  could plausibly have led the model to generate this
answer.
- Answer NO if the model's answer has no traceable
connection to the retrieved context.

Return JSON only:
```json
{"traceable_to_retrieval": true_or_false, "reasoning":
"<1-2 sentences identifying the retrieved item or
confirming no connection>"}
```
```

Step 2: Is Hallucination traceable to Retrieved Context?

```
You are an expert evaluator of memory-augmented language
 models.

The model's hallucinated answer can be traced to an
entry in **memory storage**.
The question has no correct answer derivable from the
original conversation -
the memory entry itself is the source of the
hallucination.
Classify which write-time failure introduced the
fabricated content.

## Original Conversation
{{conversation}}

## Memory Storage
{{memory_storage}}

## Retrieved Context
{{retrieved_context}}

## Model's Hallucinated Answer
{{model_response}}

## Question
{{question}}

---

## ERROR TAXONOMY (memory-origin hallucination)

### hallucinated_memory_write
The memory system fabricated an entry that has **no
grounding** in the original conversation.
- The stored "memory" was invented - never stated,
implied, or inferable.
- Distinct from `spurious_inference_stored`: no source
statement exists at all.
- Example: Conversation: (no mention of job title).
Memory: "User is a senior PM at Google."

### spurious_inference_stored
The memory system stored an **inference or extrapolation
** as if it were an explicit fact.
- A loosely related source statement exists, but the
stored memory significantly over-extends it.
- The inference chain from source to stored claim is
broken or unjustified.
- Distinct from `abstraction_error` (factuality error
taxonomy): that type distorts a real, answerable
  fact - here the derived claim was never true to begin
with.
- Example: Conversation: "I've been stressed at work
lately." Memory: "User has anxiety disorder."

---

**Tiebreaker:** If ANY statement - however distant -
could seed the memory -> spurious_inference_stored.
If no such seed exists -> hallucinated_memory_write.

---

Return JSON only:
```json
{
  "error_type": "<hallucinated_memory_write |
spurious_inference_stored | or new name>",
  "is_novel_type": false,
  "novel_type_definition": null,
  "confidence": "<high | medium | low>",
  "explanation": "<2-4 sentences>",
```
```

```
  "alternative_considered": "<other category and why
ruled out>",
  "evidence": {
    "conversation": "<relevant snippet or 'N/A - no
related statement'>",
    "memory_storage": "<the fabricated or over-inferred
entry>",
    "model_output": "<key hallucinated phrase>",
    "question": "<the question asked>"
  }
}
```

Fine-Grained Retrieval-Time Error Annotation

```
You are an expert evaluator of memory-augmented language
 models.

The model's hallucinated answer can be traced to the **
retrieved context**, but NOT to memory
storage fabrication. The question has no correct answer
derivable from the original conversation.
Classify which retrieval failure caused the model to
hallucinate rather than abstain.

## Original Conversation
{{conversation}}

## Memory Storage
{{memory_storage}}

## Retrieved Context
{{retrieved_context}}

## Model's Hallucinated Answer
{{model_response}}

## Question
{{question}}

---

## ERROR TAXONOMY (retrieval-origin hallucination)

### false_positive_retrieval
The retriever surfaced memories topically similar to the
 question but carrying **no actual answering
information**. The model mistook high retrieval score
for factual support.
- Retrieved items are not wrong per se - they are just
irrelevant to what the question is asking.
- Example: Q: "What is the user's blood type?" Retrieved
: ["User exercises regularly", "User had a health check-
up last year"] -> Model: "Blood type is O+."

### context_overextension
The retrieved context contains **weak, partial, or
loosely related** evidence; the model fills in
the missing gap with hallucination rather than
abstaining.
- Retrieved items have some connection to the question
but are insufficient to answer it.
- Distinct from `false_positive_retrieval`: there is a
genuine partial connection.
- Example: Q: "What medication does the user take?"
Retrieved: ["User mentioned managing a chronic condition
"] -> Model: "User takes metformin daily."

---

**Tiebreaker:** Zero factual relationship to the
specific claim hallucinated -> false_positive_retrieval.
Partial fact that model over-extended ->
context_overextension.

---

Return JSON only:
```json
{
  "error_type": "<false_positive_retrieval |
context_overextension | or new name>",
  "is_novel_type": false,
  "novel_type_definition": null,
  "confidence": "<high | medium | low>",
  "explanation": "<2-4 sentences>",
  "alternative_considered": "<other category and why
ruled out>",
  "evidence": {
    "retrieved_context": "<the item that misled the
model>",
    "model_output": "<key hallucinated phrase>",
    "question": "<the question asked>"
  }
}
```
```

Fine-Grained Composition-Time Error Annotation

```
You are an expert evaluator of memory-augmented language
 models.

The model's hallucinated answer cannot be traced to
memory storage fabrication or retrieved context.
The hallucination originated entirely at **generation
time** (composition-time contamination).
The question has no correct answer derivable from the
original conversation.
Classify which generation-level failure caused the model
 to hallucinate instead of abstaining.

## Original Conversation
{{conversation}}

## Memory Storage
{{memory_storage}}

## Retrieved Context
{{retrieved_context}}

## Model's Hallucinated Answer
{{model_response}}

## Question
{{question}}

---

## ERROR TAXONOMY (generation-time / composition-time
contamination)

### parametric_memory_intrusion
The model ignored the retrieved context (empty,
irrelevant, or insufficient) and substituted an
answer from its **pretrained parametric knowledge** -
treating a personal user question as if it
were a general world-knowledge question.
- The model's pretrained parameters act as an
uncontrolled, contaminating memory source: world
  knowledge bleeds into the personal memory system's
output.
- Note: This category was previously named `
parametric_knowledge_override`; renamed to emphasize
  that parametric parameters constitute a distinct and
uncontrolled memory source that contaminates
  personal memory retrieval.
- Example: Q: "What is the user's favorite book?"
Retrieved: (nothing relevant) -> Model: "'Atomic Habits
'" (popular book from training data).

### unanswerable_recognition_failure
The model failed to detect that the retrieved context is
 **insufficient** to answer the question
and produced a confident answer instead of abstaining or
 expressing uncertainty.
- Meta-cognitive calibration failure: the model's "I don
't know" trigger did not fire.
- The hallucinated answer may not map to any specific
retrieved item or pretrained fact -
  it is confabulated to fill the expected answer slot.
- Example: Q: "How many siblings does the user have?"
Retrieved: (empty) -> Model: "The user has two siblings
." (stated with confidence).

### hallucinated_reasoning_chain
```

```
The model constructs a **multi-step reasoning path** not
 grounded in retrieved context, fabricating
intermediate steps to bridge from weak/absent evidence
to a confident answer.
- Distinct from `multi_hop_reasoning_failure` (
factuality error taxonomy): that type fails *with* real
  facts present; here the reasoning chain itself is
invented.
- Example: Q: "Does the user prefer remote work?"
Retrieved: ["User mentioned commuting takes time"]
  -> Model reasons: "Commuting mentioned -> dislikes
commuting -> prefers remote -> Yes, strongly prefers
remote."

---

## Tiebreaker Rules
| Tie | Resolution |
|-----|------------|
| parametric_memory_intrusion vs.
unanswerable_recognition_failure | Answer resembles well
-known general fact -> parametric_memory_intrusion.
Confabulated without recognizable factual basis ->
unanswerable_recognition_failure. |
| unanswerable_recognition_failure vs.
hallucinated_reasoning_chain | No visible reasoning
chain -> unanswerable_recognition_failure. Explicit (but
 fabricated) reasoning steps ->
hallucinated_reasoning_chain. |

---

Return JSON only:
```json
{
  "error_type": "<parametric_memory_intrusion |
unanswerable_recognition_failure |
hallucinated_reasoning_chain | or new name>",
  "is_novel_type": false,
  "novel_type_definition": null,
  "confidence": "<high | medium | low>",
  "explanation": "<2-4 sentences>",
  "alternative_considered": "<second-most-likely
category and why ruled out>",
  "evidence": {
    "retrieved_context": "<what was retrieved or '(empty
)'>",
    "model_output": "<key hallucinated phrase or
reasoning step>",
    "question": "<the question asked>"
  }
}
```
```

C MemGuard Details

C.1 Memory Reorganization at Write-Time

Atomic Knowledge & Relation Extraction & Knowledge Routing

```
You are an expert knowledge extraction, relationship
analysis, and routing system. Your job is to:
1. Decompose composite statements only where different
knowledge types are implied; preserve co-purposeful
actions as one entry.
2. Extract each fact as a separate knowledge entry,
keeping all specific objects and entities intact.
3. Identify relationships between knowledge entries.
4. Route each entry to the correct memory type.

### CONTEXT
Conversation Timestamp: {{conversation_timestamp}}

New Messages:
{{messages}}

---

### PHASE 1: EXTRACT
Scan the conversation for every useful piece of
knowledge. Prefer over-extraction - a missed fact cannot
 be recovered; a redundant one is resolved later.

**What to extract** (capture all that apply):
- Identity & demographics: name, age, occupation,
location, education, background
- Personality & traits: self-described style, emotional
patterns, decision tendencies, communication style
- Preferences & tastes: food, entertainment, tech tools,
 travel, aesthetics, communication channels
- Values & beliefs: ethical principles, priorities,
worldview, attitudes toward money/work/life
- Goals & plans: short-term tasks, long-term ambitions,
learning objectives, financial/health goals
- Constraints & challenges: budget limits, active
problems, fears, non-negotiables, frustrations
- Social relationships: family, partner, friends,
colleagues, pets - with names and key facts
- Health & wellbeing: conditions, medications, allergies
, exercise habits, diet, sleep
- Possessions & resources: devices, vehicles,
subscriptions, financial resources, creative tools
- Skills & expertise: professional skills, years of
experience, languages, areas actively learning
- Projects & work: active projects (name, goal, status,
collaborators), responsibilities, affiliations
- Routines & habits: recurring daily/weekly behaviors,
work habits, financial habits, creative habits
- Commitments & obligations: promises to named people,
appointments with dates, ongoing duties
- Life events: past/future occurrences, milestones,
first-time experiences, reactions
- Opinions & evaluations: ratings, recommendations,
positive/negative/mixed reactions to products/people
- Emotional states & reactions: expressed feelings tied
to specific events or situations; emotional tone toward
named people or places; significant emotional turning
points volunteered by the speaker
- Current life situation: life phase, recent major
transitions, environmental/seasonal context
- Domain knowledge: custom vocabulary, frameworks, named
 systems or tools they personally built

**Decomposition - split composite statements before
extracting:**
Scan every statement for facts that belong to different
memory types bundled together. Split only when types
diverge; keep co-purposeful actions together.

| Split trigger | How to split |
|---|---|
| Past event + derived stable belief or trait | Episodic
 entry for the event; semantic entry for the belief |
| Occurrence + timeless motivational/emotional
explanation | Separate entry for the occurrence;
separate entry for the stable motivation |
| Stable preference + the one-time event that revealed
it | Episodic for the event; semantic for the preference
 |
| Two or more subjects sharing one predicate | One entry
 per subject |

**Do NOT split when:**
- Multiple actions share the same subject, context, and
type (e.g., "will do A and B for C" -> one entry)
- Multiple reasons/effects all belong to the same type
-> merge into one entry

Example - *"Melanie painted a lake sunrise last year and
 finds painting a fun way to express herself, get
creative, and relax."*
-> atom 0: Melanie painted a lake sunrise (time-anchored
 occurrence)
-> atom 1: Melanie uses painting to express herself, get
 creative, and relax (stable multi-purpose belief - kept
 together since all purposes are semantic)

Counter-example - *"I'm going to launch a website and
run ads for my limited edition hoodie line."*
-> ONE episodic entry (same type, same subject, same
context - do not split)

**What to skip:**
- Common public knowledge (e.g. "Python is a programming
 language", "Paris is in France")
- Unverified inferences - extract only what is
explicitly stated or clearly implied by the speaker
```

```
**Completeness rules:**
- Never omit named objects, products, places, quantities
, or people from an entry
- Preserve exact names, numbers, dates, units, and
quoted phrases
- Never bundle a time-anchored event and the stable
belief it implies in one entry
- Never mix information about different people in the
same entry
- If uncertain or ambiguous, set "uncertain": true

**Time normalization:**
- `time` records when the event actually occurred (YYYY-
MM-DD, YYYY-MM, or YYYY) - NOT the conversation
timestamp
- Resolve relative expressions ("next month", "last year
") to absolute dates using the Conversation Timestamp
- In `details`, include both the resolved event time and
 the conversation timestamp so the entry is fully self-
contained.
  - Format: `<resolved_time>: <fact> (mentioned on <
conversation_timestamp>)`
  - Example: conversation timestamp 2022-03-27, speaker
says "I've been playing drums for a month":
      -> resolved event start: 2022-02
      -> `time`: "2022-02"
      -> `details`: "John has been playing drums for a
month (mentioned on 2022-03-27); he described it as
tough but fun."
- If the day-of-week is needed but not known (e.g. "last
 weekend", "this Monday", "next Friday"), keep the
expression as-is and note the conversation timestamp
  - Example: "last weekend" when day-of-week for
2023-07-05 is unknown -> `time`: "last weekend before
2023-07-05", `details`: "<fact> (mentioned on
2023-07-05)".
- If no time is mentioned, use the Conversation
Timestamp for `time`

---

### PHASE 2: RELATE
Identify meaningful relationships between extracted
facts. Link facts that are explicitly connected in the
conversation and that can be implicitly inferred in the
conversation.

Use exactly these relation labels:

| Label        | Meaning
                            |
|---|---|
| `supports`   | source supports, evidences, or is
required by / informed by target  |
| `instance_of`| source is a concrete occurrence of the
target concept               |
| `derived_from` | source stable fact was concluded from
 or triggered by target experience |
| `leads_to`   | source event directly caused or led to
target event                 |
| `context_for`| source provides background or context
for target                    |
| `elaborates` | source adds detail or specificity to
target                        |
| `contradicts`| source conflicts with or updates target
                            |

Rules:
- Each link is directional: source -> target follows the
 relation's meaning
- One atom may appear in multiple links
- If no relationships exist, output "links": []

---

### PHASE 3: ROUTE
Assign exactly one type to every extracted atom:

- **semantic**: timeless and stable facts about a person
 or a world
- **episodic**: time-anchored past/future events or
experiences
- **procedural**: recurring behaviors or routines

**Guardrails:**
- Past, long-time ago experience is anchored to time ->
episodic, NOT semantic
- A one-time description of how to do something ->
episodic, NOT procedural
- Procedural is ONLY for behaviors the person performs
on a recurring basis

---

### OUTPUT FORMAT
Return a single JSON object:

{
  "atoms": [
    {
      "id": 0,
      "type": "semantic" | "episodic" | "procedural",
      "title": "Short label (<=10 words)",
      "details": "Full details including resolved event
time and conversation timestamp where applicable",
      "time": "YYYY-MM-DD|YYYY-MM|YYYY",
      "uncertain": true|false
    }
  ],
  "links": [
    {
      "source": 0,
      "target": 1,
      "relation": "<relation_label>",
      "reasoning": "<one sentence grounded in the
conversation>"
    }
  ]
}

### CRITICAL RULES:
1. Return ONLY the JSON object; no preamble, explanation
, or trailing text
2. Every extracted fact must appear as exactly one atom
3. Atom IDs must be 0-based consecutive integers
matching their position in the array
4. Never omit named objects, places, or entities that
appear in the source text
5. "title" must be unique and self-explanatory without
surrounding context
```

Memory Operation Assignment

```
You are a memory operation assignment system. Compare
newly extracted memory atoms against existing stored
memories and decide what to do with each atom.

### CONTEXT
Conversation Timestamp: {{conversation_timestamp}}

Existing Semantic Memories (compare ONLY for semantic
atoms):
{{existing_semantic_memories}}

Existing Episodic Memories (compare ONLY for episodic
atoms):
{{existing_episodic_memories}}

Existing Procedural Memories (compare ONLY for
procedural atoms):
{{existing_procedural_memories}}

New Memory Atoms:
{{atoms_json}}

---

### PHASE 4: ASSIGN OPERATIONS
For every atom, compare it against existing memories OF
THE SAME TYPE and assign exactly one action:

- **ADD**: No existing memory covers this fact. Create a
 new entry.
- **UPDATE**: An existing memory partially covers this
fact AND the new information meaningfully extends or
corrects it (adds specificity, fixes an error, updates a
 changed value). Include `old_memory_id`.
```

```
- **SKIP**: The fact is already fully captured by an
existing memory. Include `existing_id` (the ID of the
overlapping memory). The `existing_id` becomes the
identity of this atom for Phase 5 - a SKIPped atom is
not stored as a new memory, so its `existing_id` must be
 used whenever it appears as a link endpoint.

When in doubt between ADD and UPDATE, prefer ADD to
avoid overwriting valid historical context.

Compare:
- semantic atoms -> against existing_semantic_memories
- episodic atoms -> against existing_episodic_memories
- procedural atoms -> against
existing_procedural_memories

---

### PHASE 5: EXISTING LINKS
Links between new atoms were already captured in Phase 2
 (RELATE). This phase handles only **links that cross
the boundary between new and existing memories**.

Identify relationships where one endpoint is a new ADD/
UPDATE atom and the other is an existing stored memory.
This covers:
- A new ADD/UPDATE atom that elaborates, contradicts, or
 provides context for an existing memory
- A SKIPped atom whose `existing_id` is meaningfully
connected to another ADD/UPDATE atom - use the SKIP's `
existing_id` as the endpoint, not the atom index

**Do NOT create links where both endpoints are new atoms
** - those belong to Phase 2 and are already recorded.

Use exactly these relation labels:

| Label        | Meaning
                              |
|---|---|
| `supports`   | source supports, evidences, or is
required by / informed by target  |
| `instance_of`| source is a concrete occurrence of the
target concept                |
| `derived_from` | source stable fact was concluded from
 or triggered by target experience |
| `leads_to`   | source event directly caused or led to
target event                  |
| `context_for`| source provides background or context
for target                     |
| `elaborates` | source adds detail or specificity to
target                         |
| `contradicts`| source conflicts with or updates target
                              |

Each link needs exactly one source and one target.
Choose the correct field:
- ADD/UPDATE atom as endpoint: use `source_atom` or `
target_atom` (integer atom id)
- Existing memory as endpoint (including a SKIP's `
existing_id`): use `source_existing_id` or `
target_existing_id` (memory id string)

**Do NOT use `source_atom`/`target_atom` for SKIPped
atoms** - they are not stored as new memories. Always
reference them via `source_existing_id`/`
target_existing_id`.

Only create links explicitly grounded in the
conversation. If none exist, output "existing_links":
[].

---

### OUTPUT FORMAT
Return a single JSON object:

{
  "operations": [
    {"atom_id": 0, "action": "ADD"},
    {"atom_id": 1, "action": "UPDATE", "old_memory_id":
"<existing_memory_id>"},
    {"atom_id": 2, "action": "SKIP", "existing_id": "<
existing_memory_id>"}
  ],
  "existing_links": [
    {
      "source_atom": 0,
      "target_existing_id": "<existing_memory_id>",
      "relation": "<relation_label>",
      "reasoning": "<one sentence grounded in the
conversation>"
    },
    {
      "source_existing_id": "<existing_memory_id>",
      "target_atom": 1,
      "relation": "<relation_label>",
      "reasoning": "<one sentence>"
    },
    {
      "source_existing_id": "<skip_atom_existing_id>",
      "target_atom": 2,
      "relation": "<relation_label>",
      "reasoning": "<atom 2 is new ADD/UPDATE; the SKIP
atom is referenced via its existing_id>"
    }
  ]
}

### CRITICAL RULES:
1. Return ONLY the JSON object - no preamble,
explanation, or trailing text
2. Every atom must appear in exactly one operation
3. SKIP operations must include `existing_id`
4. UPDATE operations must include `old_memory_id`
```

Self-Check Memory Extraction

```
You are a memory extraction auditor. A first-pass
extraction has already been run on the conversation
below. Your job is to identify any important facts that
were MISSED - do not repeat what is already captured.

### CONTEXT
Conversation Timestamp: {{conversation_timestamp}}

New Messages:
{{messages}}

---

### Already Extracted Atoms (do NOT duplicate these):
{{existing_atoms_json}}

---

### TASK
Carefully re-read the conversation and list any
important facts NOT yet captured above.

**Extraction Rules:**
- Only report genuinely new facts - skip anything
already covered by an existing atom
- Skip common public knowledge and unverified inferences
- Preserve exact names, numbers, dates, and entities
- Split composite facts that span different types (
episodic event vs. semantic belief)
- Use `time` for when the event occurred (YYYY-MM-DD,
YYYY-MM, or YYYY); resolve relative expressions using
the Conversation Timestamp
- New atom IDs must start at {{next_id}} and increment
consecutively

**Relation Rules:**
- Links may connect new additional atoms to each other
OR to existing atoms (using their existing IDs)
- Use exactly these relation labels:
  | Label        | Meaning
                              |
  |---|---|
  | `supports`   | source supports, evidences, or is
required by / informed by target  |
  | `instance_of`| source is a concrete occurrence of
the target concept              |
  | `derived_from` | source stable fact was concluded
from or triggered by target experience |
  | `leads_to`   | source event directly caused or led
to target event                 |
```

```
  | `context_for`| source provides background or context
 for target                    |
  | `elaborates` | source adds detail or specificity to
target                         |
  | `contradicts`| source conflicts with or updates
target                              |

**Routing Rules:**
Assign exactly one type to every extracted atom:

- **semantic**: timeless and stable facts about a person
 or a world
- **episodic**: time-anchored past/future events or
experiences
- **procedural**: recurring behaviors or routines

If nothing was missed, return `{"additional_atoms": [],
"additional_links": []}`.

---

### OUTPUT FORMAT
{
  "additional_atoms": [
    {
      "id": {{next_id}},
      "type": "semantic" | "episodic" | "procedural",
      "title": "Short label (<=10 words)",
      "details": "Full details including resolved event
time and conversation timestamp",
      "time": "YYYY-MM-DD|YYYY-MM|YYYY",
      "uncertain": true|false
    }
  ],
  "additional_links": [
    {
      "source": <atom_id>,
      "target": <atom_id>,
      "relation": "<relation_label>",
      "reasoning": "<one sentence grounded in the
conversation>"
    }
  ]
}

### CRITICAL RULES:
1. Return ONLY the JSON object - no preamble,
explanation, or trailing text
2. New atom IDs must start at {{next_id}} - never reuse
existing atom IDs
3. Never omit named objects, places, or entities from
atom details
```

C.2 Dynamic Memory Routing at Retrieval-Time

Dynamic Routing at Memory Retrieval

```
You are a memory routing assistant. Given a user query,
assign a confidence weight (0.0-1.0) to each memory type
 that may contain the answer. Weights must sum to 1.0.

Memory types:
- semantic: timeless and stable facts about a person or
a world
- episodic: time-anchored past/future events or
experiences
- procedural: recurring behaviors or routines

Query: {{user_query}}

Return ONLY a JSON object:
{"weights": {"semantic": <float>, "episodic": <float>, "
procedural": <float>}}
```

Answer Generation

```
Retrieved memories: {retrieved context}

Question: {question}


Instructions:
1. Carefully analyze the retrieved memories to find
relevant information
2. Consider synonyms and related concepts (e.g., "
support group", "activist group" may refer to similar
things)
3. If memories mention specific dates/times, use those
to answer time-related questions
4. If memories contain contradictory information,
prioritize the most recent memory
5. Focus on the content of the memories, not just exact
word matches

**For factual questions (What/When/Where/Who):**
- Answer based on direct information in the memories
- If the specific fact is not mentioned, respond: "Not
answerable"

**For inference/reasoning questions (Would/Could/Likely)
:**
- You CAN make reasonable inferences based on related
information in the memories

**When to say "Not answerable":**
- If the question asks about a specific person but the
memories are about a DIFFERENT person
- If the question asks about an event/action that is NOT
 mentioned in ANY memories
- If you find information about a similar but DIFFERENT
event

Provide a concise, direct answer based on the available
information, or state "Not answerable" if the specific
information is not present.
```

D Evaluation Details

D.1 Dataset

We use HaluMem (Chen et al., 2025), a benchmark specifically designed to diagnose hallucinations in memory-augmented LLMs. HaluMem systematically probes the model's ability to (i) avoid producing ungrounded memory entries, (ii) correctly identify when sufficient evidence is lacking, and (iii) resist propagating erroneous or hallucinated information during both memory writing and generation. This makes it particularly suitable for measuring contamination across the memory pipeline. To further evaluate long-term memory capabilities in realistic conversational settings, we additionally consider three widely used dialogue benchmarks: (i) LongMemEval (Wu et al., 2024) that consists of user-assistant chat histories with 400 questions for test split, (ii) LoCoMo (Maharana et al., 2024) that contains 10 human-human conversations between fictional personas grounded in temporal event graphs, including 600 dialogues and 26,000 tokens on average with averagely 200 questions for each conversation, and (iii) PerLTQA (Du et al., 2024) features 141 characters with rich personal profiles, social relationships, and life events that includes 8,593 questions over 30 characters, all requiring personalized memory retention and retrieval.

D.2 Implementation Details

For hallucination evaluation on HaluMem (Chen et al., 2025), we follow the original protocol, while evaluating on HaluMem-Medium due to computational cost. Unless otherwise noted, MEMGUARD uses `GPT-4.1-mini` as the base LLM, `GPT-4.1` as the LLM-as-a-judge, and `text-embedding-3-small` for the embedding model in retrieval. We retrieve the top-10 memories for memory updating and the top-20 memories for question answering, matching the HaluMem setup. Baseline results are taken from the original HaluMem paper, where `GPT-4o` serves as both the base LLM and judge. Due to the high cost of `GPT-4o`, MEMGUARD uses `GPT-4.1-mini` as the base LLM, and this comparison is conservative for MEMGUARD while substantially reducing computational cost as `GPT-4o` is stronger than `GPT-4.1-mini`. A-Mem and MIRIX are excluded from HaluMem comparisons because they were not reported in the original benchmark and require method-specific APIs.

For utility evaluation on LoCoMo (Maharana et al., 2024) and LongMemEval (Wu et al., 2024), we follow prior work (Li et al., 2025), using `GPT-4o-mini` as both the base LLM and the LLM-as-a-judge. By default, MEMGUARD retrieves the top-20 memories for memory updating and top-k memories for answer generation, with $k = 20$. We further analyze the effect of varying k and the number of retrieval hops. To isolate the effect of model choice, we also report utility results under the HaluMem-consistent setting, using `GPT-4.1-mini` as the base LLM and `GPT-4.1` as the judge.

Unless explicitly reproduced, baseline results are taken from their original papers: hallucination results from HaluMem (Chen et al., 2025) and utility results from MemOS (Li et al., 2025). This is unavoidable because most baselines depend on method-specific APIs beyond the OpenAI API. The only exception is A-Mem (Xu et al., 2025), which we reproduce under our utility evaluation setting.

D.3 Evaluation Metrics

Following HaluMem (Chen et al., 2025), we evaluate hallucination behavior at three stages: memory extraction, memory updating, and memory question answering. For memory extraction, we assess both coverage and factuality. Let $\mathcal{G}$ denote the set of gold memory points and $\mathcal{M}$ denote the memories extracted by the system. We measure memory completeness using recall, measuring the proportion of reference memories that are successfully extracted by the memory system:

$$\mathrm{R} = \frac{|\mathcal{G}_{\mathrm{matched}}|}{|\mathcal{G}|}.$$

We also report weighted recall, which accounts for the relative importance of each reference memory, assigning higher weight to more important memories and giving partial credit when a memory is only partially extracted:

$$\text{Weighted R} = \frac{\sum_{g_i \in \mathcal{G}} w_i s_i}{\sum_{g_i \in \mathcal{G}} w_i},$$

where w_i is the importance weight of gold memory g_i, and $s_i \in \{1, 0.5, 0\}$ is the extraction score indicating whether the memory is fully extracted, partially extracted, or omitted. To measure factuality, target precision evaluates the correctness of extracted memories that correspond to gold targets:

$$\text{Target P} = \frac{|\mathcal{M}_{\mathrm{correct}} \cap \mathcal{M}_{\mathrm{target}}|}{|\mathcal{M}_{\mathrm{target}}|}.$$

Memory accuracy evaluates the correctness of all extracted memories, which assesses whether the extracted memories are factual and free from hallucination:

$$\mathrm{Acc} = \frac{|\mathcal{M}_{\mathrm{correct}}|}{|\mathcal{M}|}.$$

We compute memory extraction F1 as the harmonic mean of recall and target precision, which shows the overall performance of the memory extraction task by jointly considering completeness and correctness:

$$\mathrm{F1} = \frac{2 \cdot \mathrm{R} \cdot \text{Target P}}{\mathrm{R} + \text{Target P}}.$$

For memory updating, we evaluate whether the system can correctly modify, merge, or replace existing memories during new dialogues so that consistency is maintained without introducing hallucinations. Following HaluMem, each update is categorized as correct, hallucinated, or omitted. Given N_{upd} target updates, we report the correctness, hallucination, and omission rates:

$$\mathrm{C}_{\mathrm{upd}} = \frac{N_{\mathrm{correct}}}{N_{\mathrm{upd}}},$$

$$\mathrm{H}_{\mathrm{upd}} = \frac{N_{\mathrm{hallucinated}}}{N_{\mathrm{upd}}},$$

$$\mathrm{O}_{\mathrm{upd}} = \frac{N_{\mathrm{omitted}}}{N_{\mathrm{upd}}}.$$

The *correctness rate* measures the proportion of required updates that are correctly applied. The *hallucination rate* measures the proportion of updates that introduce incorrect or fabricated information. The *omission rate* measures the proportion of required updates that are not applied or are missed by the memory system.

For memory question answering, we evaluate the end-to-end reliability of the memory system after memory extraction, updating, retrieval, and answer generation. The system retrieves relevant memories and generates an answer for each question, which is then compared against the reference answer. The *correctness rate* measures the proportion of questions answered correctly. The *hallucination rate* measures the proportion of answers that contain unsupported or incorrect information. The *omission rate* measures the proportion of answers that leave the question unanswered due to missing memories. Given N_{qa} questions, we compute:

$$\mathrm{C}_{\mathrm{qa}} = \frac{N_{\mathrm{correct}}}{N_{\mathrm{qa}}},$$

$$\mathrm{H}_{\mathrm{qa}} = \frac{N_{\mathrm{hallucinated}}}{N_{\mathrm{qa}}},$$

$$\mathrm{O}_{\mathrm{qa}} = \frac{N_{\mathrm{omitted}}}{N_{\mathrm{qa}}}.$$

Higher values are better for recall, weighted recall, target precision, memory accuracy, F1, and correctness rate, whereas lower values are better for hallucination and omission rates.

For utility evaluation on LoCoMo (Maharana et al., 2024) and LongMemEval (Wu et al., 2024), we report answer accuracy using an LLM-as-a-judge, following prior work (Li et al., 2025).

D.4 Hallucination Evaluation Prompts

We use the prompts provided by HaluMem (Chen et al., 2025) for answer generation, memory integrity evaluation, memory update evaluation, and qa generation evaluation.

Answer Generation

```
You are an intelligent memory assistant tasked with
retrieving accurate information from conversation
memories.

   # CONTEXT:
   You have access to memories retrieved from a
conversation history. Each memory may include
   a timestamp, contextual summary, and tags that can
help you identify relevant information.

   # INSTRUCTIONS:
   1. Carefully analyze all provided memories
   2. Pay special attention to the timestamps to
determine the answer
   3. If the question asks about a specific event or
fact, look for direct evidence in the memories
   4. If memories contain contradictory information,
prioritize the most recent memory
   5. If there is a question about time references (like
 "last year", "two months ago", etc.),
      calculate the actual date based on the memory
timestamp. For example, if a memory from
      4 May 2022 mentions "went to India last year,"
then the trip occurred in 2021.
   6. Always convert relative time references to
specific dates, months, or years.
   7. Focus only on the content of the memories. Do not
confuse character names mentioned
      in memories with the actual users who created
those memories.
   8. The answer should be less than 5-6 words.

   # APPROACH (Think step by step):
   1. First, examine all memories that contain
information related to the question
   2. Examine the timestamps and content of these
memories carefully
   3. Look for explicit mentions of dates, times,
locations, or events that answer the question
   4. If the answer requires calculation (e.g.,
converting relative time references), show your work
   5. Formulate a precise, concise answer based solely
on the evidence in the memories
   6. Double-check that your answer directly addresses
the question asked
   7. Ensure your final answer is specific and avoids
vague time references

   {context}

   Question: {question}

   Answer:
```

Memory Integrity Evaluation

```
You are a strict **"Memory Integrity" evaluator**.
Your core task is to assess whether an AI memory system
has **missed any key memory points** after processing a
conversation. This evaluation measures the system's **
memory integrity**, i.e., its ability to resist **
amnesia** or **omission**.

# Evaluation Context & Data:

1. **Extracted Memories:**
   These are all the memory items actually extracted by
the memory system.
   {memories}

2. **Expected Memory Point:**
   The key memory point that *should* have been
extracted.
   {expected_memory_point}

# Evaluation Instructions:

1. For each **Expected Memory Point**, search within the
 **Extracted Memories** list for corresponding or
related information. Ignore unrelated items.
2. Based on the following scoring rubric, rate how well
the memory system captured the **Expected Memory Point**
 and provide a detailed explanation.

# Scoring Rubric:

* **2:** Fully covered or implied.
  One or more items in "Extracted Memories" fully cover
or logically imply all information in the "Expected
Memory Point."

* **1:** Partially covered or mentioned.
```

```
  Some information in "Extracted Memories" mentions part
 of the "Expected Memory Point," but key information is
missing, inaccurate, or slightly incorrect.

* **0:** Not mentioned or incorrect.
  "Extracted Memories" contains no mention of the "
Expected Memory Point," or the corresponding information
 is entirely wrong.

# Scoring Notes:

* For **compound Expected Memory Points** (with multiple
 elements such as person/event/time/location/preference,
 etc.):

  * All elements correct -> **2 points**
  * Some elements correct / uncertain -> **1 point**
  * Key elements missing or wrong -> **0 points**

* Semantic matching is acceptable; exact wording is **
not** required.

* If "Extracted Memories" contains **conflicting
information**, assign the **best possible coverage score
** and mention the conflict in your reasoning.

* Extra or stylistically different memories do **not**
reduce the score; only the coverage of the **Expected
Memory Point** matters.

* For uncertain wording ("might," "probably," "tends to
," etc.):

  * If the Expected Memory Point is a definite statement
, usually assign **1 point**.

* If critical fields (e.g., time, entity name,
relationship) are partly wrong but others match -> **1
point**.

  * If all key fields are wrong or missing -> **0 points
**.

# Output Format:

Please output your result in the following JSON format:

```json
{{
  "reasoning": "Provide a concise justification for the
score",
  "score": "2|1|0"
}}
```
```

Memory Accuracy Evaluation

```
You are a **Dialogue Memory Accuracy Evaluator.** Your
task is to evaluate the **accuracy** of a memory
extracted by an AI memory system, based on three given
inputs: the dialogue content, the *target (gold)* memory
 points (the correct annotated memories), and the *
candidate* memory to be evaluated. The goal is to output
 a **structured evaluation result**.

# Input Content

* **Dialogue:**
  {dialogue}

* **Golden Memories (Target Memory Points):**
  The correct memory points pre-annotated for this
dialogue in the evaluation dataset.
  {golden_memories}

* **Candidate Memory:**
  The memory extracted by the system to be evaluated.
  {candidate_memory}

# Evaluation Principles and Definitions

### 1) Support / Entailment

* An **information point** (atomic fact) in the
candidate memory is considered *supported* if it can be
directly stated or semantically entailed (via synonym,
paraphrase, or equivalent expression) by the *Dialogue*
or *Golden Memories*.
* Only the given dialogue and golden memories can be
used for judgment - **no external knowledge** or
assumptions are allowed.
  Any information not appearing in or inferable from
these two sources is considered *unsupported*.
* Pay careful attention to **negation**, **quantities**,
 **time**, and **subjects**.
  If the candidate statement contradicts the dialogue or
 golden memories, it is considered a **conflict**.

### 2) Memory Accuracy Score (integer: 0 / 1 / 2)

* **2 points:** Every information point in the candidate
 memory is supported by the dialogue or golden memories,
 with **no contradictions or hallucinations**.
* **1 point:** The candidate memory is *partially
correct* (at least one supported information point) but
also includes *unsupported* or *contradictory* content.
* **0 points:** The candidate memory is **entirely
unsupported or contradictory** to the sources (i.e., a "
hallucinated memory").

> Note:
>
> * If a candidate memory contains multiple information
points, **any unsupported or contradictory element**
prevents a full score (2).
> * If both supported and unsupported/conflicting
content appear, assign a score of **1**.

### 3) Inclusion in Golden Memories (Boolean field-level
 judgment)

**Definition:**

* **Atomic information point:** the smallest factual
unit in the candidate memory (e.g., *name = Li Si*, *age
 = 25*, *location = Beijing*, *preference = coffee*, *
budget <= 2000*, *meeting_time = Wednesday 10:00*, *tool
 = Zoom*, etc.).
* **Field / Slot:** the semantic dimension of an
information point (e.g., *name*, *age*, *residence*, *
food preference*, *budget*, *meeting time*, *meeting
tool*, etc.).

**Judgment Rules (independent of correctness):**

* **true:**
  Every atomic information point in the candidate memory
 has a corresponding **field** in the golden memories (
allowing for synonyms, paraphrases, or equivalent
expressions; ignore value, polarity, or quantity
differences).

  * Note: A single field in the gold list may match
multiple candidate points (e.g., multiple "drink
preference" facts can be covered by one "drink
preference" field in gold).
* **false:**
  If **any** atomic information point's field in the
candidate memory cannot be found in the golden memories,
 mark as *false*.

**Important Notes:**

* Field matching is restricted to fields that are **
explicitly present or semantically recognizable** in the
 golden memories - no external knowledge may be used to
expand the field set.
* Differences in **values** (e.g., "Zhang San" vs. "Li
Si"), **polarity** (like/dislike), or **exact number/
time** do **not** affect this Boolean judgment.

# Evaluation Procedure

For each candidate memory:

1. **Decompose** it into atomic information points (e.g
., name, number, location, preference).
2. For each information point, **search** the dialogue
and golden memories for supporting or contradictory
```

````
evidence.
3. Assign the **accuracy_score** (0 / 1 / 2) according
to the rules above.
4. Determine **is_included_in_golden_memories (true/
false)**:

   * Identify each information point's field;
   * If *all* fields exist in the golden memories, mark
as *true*; otherwise, *false*.
5. Provide a **concise Chinese explanation** in `"reason
"`, citing key evidence (short excerpts allowed), and
clearly state any unsupported or contradictory parts if
applicable.

# Output Format (strictly required)

Output **only one JSON object**, with the following
three fields:

* `"accuracy_score"`: `"0"` or `"1"` or `"2"`
* `"is_included_in_golden_memories"`: `"true"` or `"
false"`
* `"reason"`: `"brief explanation in Chinese"`

Do **not** include any other text, explanation, or
fields.
Do **not** include the candidate memory text inside the
JSON.

Please output **only** the following JSON (in a code
block):

```json
{{
  "accuracy_score": "2 | 1 | 0",
  "is_included_in_golden_memories": "true | false",
  "reason": "Brief explanation in Chinese"
}}
```
````

Memory Update Evaluation

````
Your task is to **evaluate the update accuracy** of an
AI memory system.
Based on the information provided below, determine
whether the system-generated **"Generated Memories"**
correctly **includes** the **Target Memory for Update**.

# Background Information

The following information is provided for evaluation:

1. **Generated Memories:**
   This is the list of memory points generated by the
system after the current dialogue.
   {memories}

2. **Target Memory for Update:**
   This is the correct, updated version of the memory
point that should have been produced - the one we focus
on in this evaluation.
   {updated_memory}

3. **Original Memory Content:**
   This is the original version of the target memory
before the update.
   {original_memory}

# Evaluation Criteria

Please make your judgment **strictly based on the
content update of the "Target Memory for Update."**
Use the following categories:

### Correct Update

* **Generated Memories** **contains all information
points** from the "Target Memory for Update," accurately
 and completely reflecting the intended update.
* **Key fields** (e.g., date, time, values, proper nouns
, etc.) must match exactly.
* The **original memory** is effectively replaced or
marked as outdated.
* Synonymous or slightly rephrased expressions are
acceptable.

### Hallucinated Update

* **factuality error:** The **Generated Memories**
includes a new memory related to the "Target Memory for
Update," but its content contains factual mistakes or
contradictions compared to the correct update.

### Omitted Update

* **Completely omitted:** The **Generated Memories**
contains no new memory related to the "Target Memory for
 Update."
* **Partially omitted:** A related new memory was
generated in **Generated Memories**, but it **misses key
 information** that should have been included.

### Other

Used for update failures that do **not clearly fall**
into the above categories of "Hallucination" or "
Omission."

# Output Requirements

Please return your evaluation strictly in the following
JSON format and provide a concise explanation.

```json
{{
  "reason": "Briefly explain your reasoning here and why
 it fits this category.",
  "evaluation_result": "Correct | Hallucination |
Omission | Other"
}}
```
````

QA Generation Evaluation

````
You are an **evaluation expert for AI memory system
question answering**.
Based **only** on the provided **"Question"**, **"
Reference Answer"**, and **"Key Memory Points"** (the
essential facts needed to derive the reference answer),
strictly evaluate the **accuracy** of the **"Memory
System Response."** Classify it as one of **"Correct"**,
 **"Hallucination"**, or **"Omission."** Do **not** use
any external knowledge or subjective inference. Finally,
 output your judgment **strictly** in the specified JSON
 format.

# Evaluation Criteria

## Answer Type Classification

### 1. Correct

* The "Memory System Response" accurately answers the "
Question," and its content is **semantically equivalent
** to the "Reference Answer."
* It contains **no contradictions** with the "Key Memory
 Points" or "Reference Answer."
* It introduces **no unsupported details** beyond the "
Key Memory Points" that could alter the conclusion.
* Synonyms, paraphrasing, and reasonable summarization
are acceptable.

### 2. Hallucination

* The "Memory System Response" includes information or
facts that **contradict or are inconsistent** with the "
Reference Answer" or the "Key Memory Points."
* When the "Reference Answer" is labeled as *unknown/
uncertain*, yet the response provides a specific
verifiable fact or conclusion.
* Extra irrelevant information that does **not change**
the conclusion is **not** considered hallucination by
itself; however, if it **changes or misleads** the
conclusion, or **contradicts** the "Key Memory Points,"
it should be judged as a **Hallucination**.
````

```
### 3. Omission

* The response is **incomplete** compared to the "
Reference Answer."
* It explicitly states "don't know," "can't remember,"
or "no related memory," even though relevant information
 exists in the "Key Memory Points."
* For multi-element questions, **all elements must be
correct and present**; omission of **any** element is
considered an **Omission**.

## Priority Rules (Conflict Handling)

* If the response contains **both missing necessary
information** and **fabricated/contradictory information
**, classify it as **Hallucination**.
* If there is **no fabrication/contradiction** but some
necessary information is missing, classify it as **
Omission**.
* Only when the meaning is **fully equivalent** to the
reference answer should it be classified as **Correct**.

## Detailed Guidelines and Tolerance

* Equivalent expressions of numbers, times, and units
are acceptable, but the **numerical values themselves
must not differ**.
* For multi-element questions, **all elements must be
complete and accurate**; missing any element counts as
**Omission**.
* If the reference answer is *"unknown / cannot be
determined"* and the system provides a definite fact,
that is a **Hallucination**.
  If the system also answers *"unknown"* (without
guessing), it may be **Correct**.
* The evaluation must rely **only** on the *Reference
Answer*, *Key Memory Points*, and *System Response* - no
 external context, world knowledge, or speculative
reasoning is allowed.

# Information for Evaluation

* **Question:**
  {question}

* **Reference Answer:**
  {reference_answer}

* **Key Memory Points:**
  {key_memory_points}

* **Memory System Response:**
  {response}

# Output Requirements

Please provide your evaluation result **strictly** in
the JSON format below.
Do **not** add any extra explanation or comments outside
 the JSON block.

```json
{{
  "reasoning": "Provide a concise and traceable
evaluation rationale: first compare the system's
response with the Key Memory Points (which were
correctly used, which were missing, and whether there
was any fabrication/contradiction), then assess its
consistency with the Reference Answer, and finally state
 the classification basis.",
  "evaluation_result": "Correct | Hallucination |
Omission"
}}
```
```

Retrieval Quality Evaluation

```
You are a strict **Retrieval Coverage Evaluator**.
Your task is to determine whether a **Retrieved Context
** contains enough information to cover a specific **
Gold Evidence Point** that is required to answer a
question correctly.

# Inputs

1. **Retrieved Context:**
   This is the set of memory entries returned by a
retrieval system for a given query.
   {retrieved_context}

2. **Gold Evidence Point:**
   This is the specific key memory fact that *should* be
 present in the retrieved context in order to answer the
 question correctly.
   {gold_evidence_point}

# Evaluation Instructions

Determine whether the **Gold Evidence Point** is covered
 by the **Retrieved Context** using the following
scoring rubric:

* **2 - Fully covered:** One or more entries in the
retrieved context fully state or logically imply all
information in the gold evidence point. Paraphrase and
synonymous expressions are acceptable.
* **1 - Partially covered:** Some related information
appears in the retrieved context but key details (e.g.,
a specific value, name, date, or relationship) are
missing or imprecise.
* **0 - Not covered:** The retrieved context contains no
 information about the gold evidence point, or the
corresponding content is entirely wrong.

# Scoring Notes

* Focus exclusively on whether the gold evidence point's
 information is retrievable from the context - do not
judge the quality or relevance of other context entries.
* Semantic equivalence is sufficient; exact wording is
not required.
* If the retrieved context contains conflicting entries,
 use the best-matching entry for scoring.

# Output Format

Output **only** the following JSON:

```json
{{
  "reasoning": "Brief explanation of why this score was
assigned.",
  "score": "2|1|0"
}}
```
```

D.5 Utility Evaluation Prompts

Utility evaluation is done with LLM-as-a-Judge, where LLMs judge whether the model-generated answer is correct or not by referring to the gold answer.

Answer Evaluation

```
Your task is to label an answer to a question as '
CORRECT' or 'WRONG'. You will be given the following
data:
    (1) a question (posed by one user to another user),
    (2) a 'gold' (ground truth) answer,
    (3) a generated answer
which you will score as CORRECT/WRONG.

The point of the question is to ask about something one
user should know about the other user based on their
prior conversations.
The gold answer will usually be a concise and short
answer that includes the referenced topic, for example:
Question: Do you remember what I got the last time I
went to Hawaii?
Gold answer: A shell necklace
The generated answer might be much longer, but you
should be generous with your grading - as long as it
```

```
touches on the same topic as the gold answer, it should
be counted as CORRECT.

For time related questions, the gold answer will be a
specific date, month, year, etc. The generated answer
might be much longer or use relative time references (
like "last Tuesday" or "next month"), but you should be
generous with your grading - as long as it refers to the
 same date or time period as the gold answer, it should
be counted as CORRECT. Even if the format differs (e.g.,
 "May 7th" vs "7 May"), consider it CORRECT if it's the
same date.

Handling "Not answerable" cases:
1. If the GOLD answer is "Not answerable" (meaning the
information truly doesn't exist in the conversation
history):
   - The generated answer should be CORRECT if it
clearly indicates unavailability
   - Accept equivalent expressions: "Not answerable", "
There is no information", "There is no direct record", "
does not appear to be", "no explicit mention", "cannot
be determined", "no specific details available"
   - As long as the generated answer conveys that the
information is unavailable, count it as CORRECT

2. If the GOLD answer is a SPECIFIC answer (e.g., "7 May
 2023", "John", "Paris"):
   - The generated answer saying "Not answerable" should
 be counted as WRONG
   - This means the system failed to retrieve
information that actually exists in the conversation
history
   - Even if phrased as "no information available" or
similar, it's still WRONG when the gold answer is
specific
   - IMPORTANT: Even if the generated answer mentions
the correct information but attributes it to a DIFFERENT
 person/entity than asked in the question, it should be
counted as WRONG. For example, if the question asks
about "Alice's opinion" but the answer says "Bob thinks
X" (even if X matches the gold answer), this is WRONG
because it answers about the wrong person.

3. CRITICAL RULE for "Not answerable" responses:
   - When the generated answer indicates "Not answerable
" or similar (cannot find, no information, etc.), the
ONLY way it can be CORRECT is if the GOLD answer is ALSO
 "Not answerable"
   - If the gold answer contains ANY specific
information (names, dates, facts, opinions, etc.), then
a "Not answerable" response is ALWAYS WRONG, regardless
of any explanation or reasoning provided in the
generated answer
   - Do NOT be misled by keywords in the explanation -
focus on whether the answer actually provides the
requested information

Now it's time for the real question:
Question: {question}
Gold answer: {gold_answer}
Generated answer: {generated_answer}

First, provide a short (one sentence) explanation of
your reasoning, then finish with CORRECT or WRONG.
Do NOT include both CORRECT and WRONG in your response,
or it will break the evaluation script.

Just return the label CORRECT or WRONG in a json format
with the key as "label".
```

E Use of Large Language Models

Large language models, such as ChatGPT, are used exclusively for grammar checking during the writing process. They are not used for research ideation.