KeyMem: Graph-Augmented Vector Retrieval for Persistent Conversational Memory in LLM Agents

Yiang Lu*

March 26, 2026

Abstract

Long-term memory is a critical capability gap in LLM-based agents. Existing approaches either compress conversation history into structured facts(sacrificing information fidelity) or rely on pure vector retrieval that fails on multi-hop and temporal reasoning questions. This paper presents **KeyMem**, a persistent memory system that stores raw conversational turns in a knowledge graph and retrieves them through a dual-path architecture combining keyword vector search with graph traversal. The system introduces three key contributions: (1) a Fragment-based multi-hop expansion mechanism that aggregates topically related memories for complex queries; (2) a source-aware scoring formula with query-local normalization that requires no preset hyperparameters; and (3) a lightweight reference resolution state machine for anaphora disambiguation during storage. On the LoCoMo-10 benchmark, KeyMem achieves an overall token F1 of 0.609 and Recall of 0.847, outperforming mem0 (F1=0.372) by 63.7% while preserving full information fidelity through raw turn storage. The code is available at https://github.com/g0dA/KeyMem.

1. Introduction

Conversational agents powered by large language models face a fundamental limitation: the context window. Although SOTA models offer context windows of hundreds of thousands of tokens, long-running personal assistants, customer service agents, and companion systems accumulate conversation histories that far exceed these limits. The challenge is not merely storage, it is *retrieval*: given a new query, which past memories are relevant, and how can they be ranked and surfaced accurately?

Current approaches fall into two broad categories. **Compression-based systems** such as mem0 (Chhikara et al., 2025) extract structured facts from conversations, like "User likes hiking" into a vector store, enabling fast retrieval but discarding the contextual details, temporal information, and nuanced expressions that are necessary for accurate question answering. **Full-context systems** avoid compression but are limited by context window length and become prohibitively expensive as histories grow.

We argue that neither approach is satisfactory, and identify four design principles that a production memory system should satisfy:

*Corresponding author. Email: lyi4ng@gmail.com

1. **Stateless store/recall interface:** Memory management should be decoupled from session context, enabling the system to be deployed as an independent service usable by any application layer without binding to a particular dialogue manager.

2. **No automatic forgetting:** Forgetting policy is an application layer concern, not a storage layer concern. Automatic forgetting causes irreversible information loss.

3. **No automatic conflict resolution:** When newer memories contradict older ones, temporal ordering is often itself the answer. The decision of how to reconcile conflicting information should be delegated to the LLM at query time, not resolved destructively at store time.

4. **Raw memories over compressed summaries:** Compression is lossy encoding. As token costs decline and context windows expand, the benefit of compression diminishes while the information loss is permanent. The experiments confirm that KeyMem (F1=0.609) substantially outperforms mem0's fact-compression approach (F1=0.372).

Based on these principles, we present **KeyMem**, a system that stores each conversational QA turn as a raw text node in a FalkorDB knowledge graph, enriched with extracted keywords, named entities, time points, locations, and inter-turn topic clusters (Fragments). Retrieval combines two complementary paths: Path B performs keyword vector search augmented by entity/time/location graph traversal with intersection filtering; Path C expands Fragment clusters to retrieve topically related turns that keyword search alone would miss. A source-aware scoring formula with query-local normalization then ranks all candidates without requiring preset thresholds.

We evaluate on LoCoMo-10 (Maharana et al., 2024), a challenging benchmark of 10 long conversations totaling ~1,590 questions across five categories including multi-hop, temporal, open-domain, single-hop, and adversarial questions. KeyMem achieves:

Table 1: Comparison of KeyMem and mem0 on representative evaluation metrics.

Metric	KeyMem	mem0	Improvement
Overall F1	**0.609**	0.372	+63.7%
Recall	**0.847**	0.480	+76.5%
MultiHop F1	**0.452**	0.262	+72.5%
Temporal F1	**0.570**	0.080	+612.0%

2. Related Work

2.1. Long-term Memory for LLM Agents

MemGPT (Packer et al., 2023) introduces a hierarchical memory architecture inspired by operating system paging, with main context and external storage. It tightly couples memory management with the conversation loop, requiring the LLM itself to issue memory operations. This coupling limits deployment flexibility and makes it difficult to use MemGPT's memory component independently.

mem0 (Chhikara et al., 2025) takes a fact-extraction approach: each conversation turn is processed by an LLM to extract structured facts, which are stored in a vector database. Retrieval is a single vector search over these facts. The key limitation is information loss during extraction, temporal details, emotional context, and nuanced expressions are discarded.

2.2. Retrieval-Augmented Generation

RAG systems (Lewis et al., 2020) retrieve relevant documents using dense retrieval (Karpukhin et al., 2020; Izacard et al., 2021) and pass them to a generator. Conversational memory differs from document retrieval in several ways: turns are short and highly contextual, references (pronouns, demonstratives) span multiple turns, and multi-hop reasoning requires aggregating information from non-adjacent turns.

GraphRAG (Edge et al., 2024) builds community summaries over extracted entity graphs for large document corpora. **LightRAG** (Guo et al., 2024) combines graph-based and vector-based retrieval. Our system shares the graph+vector hybrid philosophy but is designed specifically for conversational memory rather than document corpora, with dedicated mechanisms for temporal indexing, reference resolution, and inter-turn fragment clustering.

2.3. Conversational Memory Benchmarks

LoCoMo (Maharana et al., 2024) is a benchmark of long social conversations with questions testing multi-hop, temporal, open-domain, single-hop, and adversarial reasoning. It is currently the most comprehensive benchmark for long-term conversational memory evaluation, and we use its full 10-conversation dataset as our primary evaluation.

3. System Design

3.1. Design Philosophy

KeyMem is built around four principles that distinguish it from prior work (Section 1), These principles have concrete architectural implications:

- the stateless interface requirement leads to session isolated engine instances;
- the no-forgetting principle means never automatically deleting or merging memories;
- the no-conflict-resolution principle means storing all turns independently without deduplication;
- the raw-memory principle means storing original conversation text rather than extracted summaries.

Figure 1 provides an overview of the system. The store pipeline processes each conversational turn through reference resolution, LLM extraction, and batch embedding before writing to the FalkorDB knowledge graph. The recall pipeline performs keyword vector search, graph traversal, and fragment-based multi-hop expansion, followed by source-aware scoring to return the top-k ranked memories.

3.2. Knowledge Graph Schema

Each conversational turn is stored as a pair of nodes in FalkorDB:

- **QNode**: the human utterance text, with fields `memory_id`, `text`, `embedding`, `created_at` (wall time of the conversation), `event_time` (ISO 8601 time extracted from text, or anchor time if not explicit).

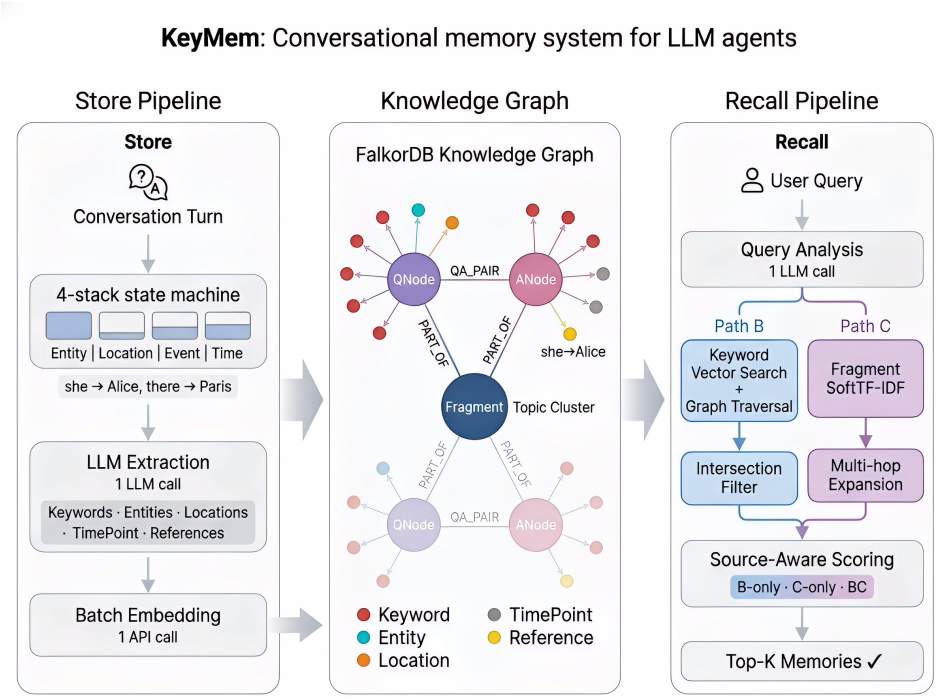

Figure 1: Overview of the KeyMem architecture.

- **ANode**: the assistant response, linked to its QNode via `QA_PAIR` edge.

These turn nodes are connected to the following structural nodes through typed edges:

Node Type	Purpose	Edge
Keyword	Content word with embedding	`HAS_KEYWORD`
Entity	Named entity	`INVOLVES`
Location	Place name	`AT_LOCATION`
TimePoint	ISO 8601 datetime	`AT_TIME`
Reference	Resolved anaphora	`HAS_REFERENCE`
Fragment	Topic cluster of turns	`PART_OF`

Table 2: Node types and edges in the memory graph.

This schema supports both semantic search (via Keyword embeddings) and structured traversal (via Entity/Location/TimePoint exact and contains matching).

To illustrate the schema concretely, we simulate 5-turn conversation:

Turn 1 — Q: *"Hey Emma, how was your weekend in Paris?"* / A: *"It was amazing! I visited the Louvre on Saturday and had dinner near the Eiffel Tower. My cat Mochi stayed with my friend Sarah while I was away."*

Turn 2 — Q: *"That sounds wonderful! Did you go alone?"* / A: *"No, I went with my colleague James. We've been planning this trip since March 2024. James is a huge art fan — he studies Renaissance paintings."*

Turn 3 — Q: *"What was the highlight of the trip?"* / A: *"Definitely the Mona Lisa. We waited two hours to get close enough for a photo. James said it was the best day of his life."*

Turn 4 — Q: *"When did you get back?"* / A: *"We flew back last Sunday, April 14th. The flight was delayed by three hours though."*

Turn 5 — Q: *"How is Mochi doing now that you're back?"* / A: *"She was so happy to see me! She slept on my lap all evening. Sarah said Mochi barely ate while I was gone."*

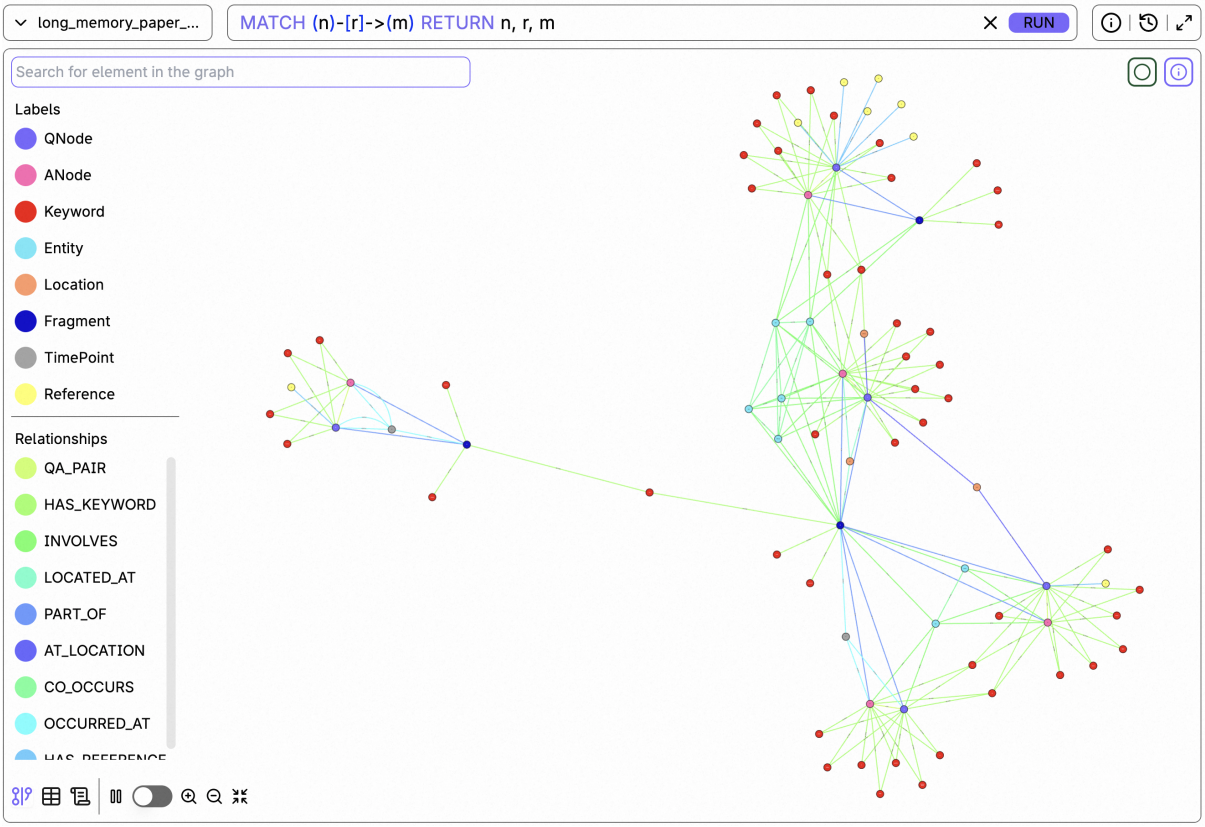

Figure 2: Knowledge graph of a 5-turn conversation stored in KeyMem, visualized in FalkorDB Browser. Node colors: QNode (purple), ANode (pink), Keyword (red), Entity (cyan), Location (orange), TimePoint (gray), Reference (yellow), Fragment (dark blue). Fragment nodes aggregate topically related turns via `PART_OF` edges, enabling Path C multi-hop expansion during recall.

3.3. Store Pipeline

When a conversational turn `(question, answer)` is stored, the following steps execute:

Step 1: Reference Resolution. Before extraction, the current state of the reference resolution state machine (Section 3.5) is serialized as a `state_summary` string containing the recent entity, location, event, and time focus stacks. This summary is prepended to the extraction prompt.

Step 2: LLM Extraction (1 LLM call). A single call to the extraction LLM (gpt-4.1-mini) processes the turn text together with the `state_summary`. The LLM performs two tasks simultaneously:

- **Reference detection and resolution**: identifies pronouns, demonstratives, relative time expressions, and zero anaphora, resolving each to a value from the state summary or computing absolute dates from the anchor time.
- **Metadata extraction**: extracts keywords,named entities, events, locations, and an explicit `event_time`(using anchor time if the turn describes a current or recent event).

Step 3: Embedding (1 batch call). All extracted keywords and the QA turn text are embedded in a single batch API call using `text-embedding-3-small` (1536 dimensions).

Step 4: Graph Construction. The QNode/ANode pair is created. For each extracted keyword, a `MERGE` operation finds or creates a Keyword node by exact name match (no similarity deduplication). Entity, Location, and TimePoint nodes are similarly merged. Resolved Reference nodes are created with links to their source Entity nodes.

Step 5: State Machine Update. The reference resolution state machine is updated with the entities, locations, events, and times extracted in this turn (Section 3.5).

Step 6: Consolidation Check. If the turn count since the last consolidation reaches the window size (N=10), the sliding window consolidation process runs asynchronously (Section 3.6).

The entire store pipeline requires **1 LLM call + 1 embedding batch call** per turn.

3.4. Recall Pipeline

Given a query string, the recall pipeline (Figure 3) proceeds as follows:

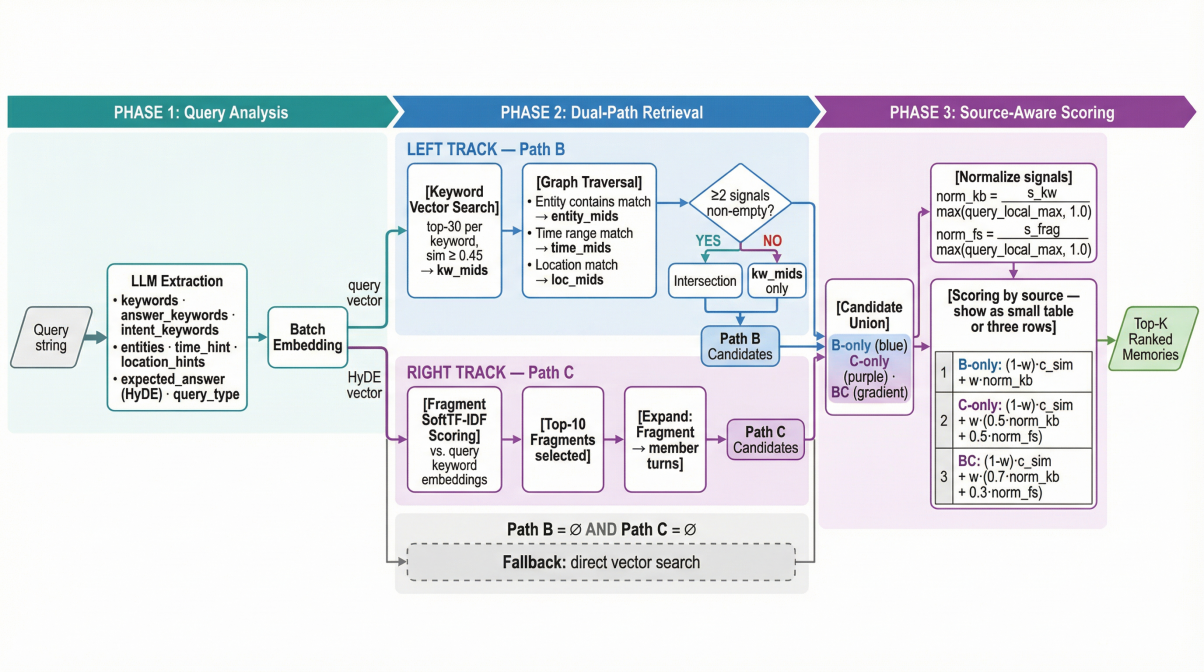

Figure 3: KeyMem recall pipeline.

Phase 1: Query Analysis (1 LLM call + 2 embedding calls). The query LLM call extracts: `keywords` (2–5 content words), `answer_keywords` (2–5 expected-answer direction words), `intent_keywords` (3–8 implicit reasoning words for non-factual queries), `entities`, `time_hint` (exact/before/after/range with ISO 8601 bounds), `location_hints`, `expected_answer` (HyDE-style hypothetical answer), and `query_type`(factual/reasoning). The query text and HyDE answer are embedded in parallel.

The `state_summary` from the reference resolution state machine (or from an `AttentionState` provided by the application layer for stateful recall) is passed to the LLM to enable anaphora resolution in the query itself.

Phase 2: Dual-Path Retrieval. Path B and Path C execute in parallel:

- *Path B — Direct Graph Retrieval.* All query and answer keywords are embedded in a batch call. For each keyword embedding, the top-30 nearest Keyword nodes are retrieved by cosine similarity (threshold ≥ 0.45) using FalkorDB's vector index. The union of all matched Keyword nodes' linked memories forms the keyword candidate set `kw_mids`. Simultaneously, entity names are matched against Entity nodes use case-insensitive bidirectional contains matching. Time hints are matched against TimePoint nodes use before/after/range comparison. Location hints are matched against Location nodes via contains matching. These form `entity_mids`, `time_mids`, and `loc_mids`. if two or more of `{kw_mids, entity_mids, time_mids, loc_mids}` are non-empty, only their intersection is retained. This reduces noise while preserving recall for queries with multiple strong signals.

- *Path C — Fragment Multi-hop Expansion.* Candidate Fragment nodes are first collected by traversing from Path B's matched Keyword nodes, query entities, time hints, and location hints to their associated Fragments via graph edges. Each candidate fragment f is then scored by SoftTF-IDF similarity against the query keywords $\{q_1, \ldots, q_m\}$. Let K_f denote the union of keywords from all turns belonging to fragment f (i.e., all Keyword nodes reachable via `PART_OF` $\rightarrow$ `HAS_KEYWORD` edges). The score is:

$$\text{SoftTF-IDF}(q, f) = \sum_{i=1}^{m} \hat{s}_i \cdot \text{IDF}(\hat{k}_i)$$

  where $\hat{s}_i = \max_{k \in K_f} \cos(\mathbf{e}_{q_i}, \mathbf{e}_k)$ is the best cosine similarity for query keyword q_i against all fragment keywords, and $\hat{k}_i$ is the corresponding best-matching keyword. Terms with $\hat{s}_i < \tau$ ($\tau = 0.45$) are excluded from the sum. Each fragment keyword may match at most one query keyword. The IDF weight is:

$$\text{IDF}(k) = \min\left(1,\ \frac{\log(N/\text{df}(k))}{\log N}\right)$$

  where $\text{df}(k)$ is the number of memories containing keyword k among N total memories. The top-10 Fragments by score are expanded, and all turns belonging to these Fragments become Fragment candidates `frag_mids`. This path recovers memories that share a topic cluster with query-relevant turns but whose own keywords do not directly match the query, making it particularly important for multi-hop questions.

- **Fallback**: if both paths return no candidates, direct vector search over all QNode embeddings is used.

Phase 3: Source-Aware Scoring. The union of Path B and Path C candidates is classified into three sources:

- **B-only**: retrieved by Path B but not Path C
- **C-only**: retrieved by Path C but not Path B
- **BC**: retrieved by both paths

For each candidate, two signals are computed: s_{kw} (SoftTF-IDF between query keywords and the candidate's Keyword nodes) and s_{frag} (SoftTF-IDF between query keywords and the candidate's Fragment's Keyword nodes; zero for B-only candidates).

Both signals are normalized query-locally with a floor of 1.0 to avoid hyperparameter dependence:

$$\tilde{s}_{\text{kw}} = \frac{s_{\text{kw}}}{\max(\max_{c \in \mathcal{C}} s_{\text{kw}}(c),\ 1.0)} \qquad \tilde{s}_{\text{frag}} = \frac{s_{\text{frag}}}{\max(\max_{c \in \mathcal{C}} s_{\text{frag}}(c),\ 1.0)}$$

where $\mathcal{C}$ is the full candidate set for the current query.

The adaptive keyword weight w is computed dynamically from the raw keyword signal to prevent low-signal queries from over-relying on keyword matching:

$$w = \min(w_{\text{cap}},\ w_{\text{base}} + 0.15 \cdot s_{\text{kw}})$$

with $w_{\text{cap}} = 0.50$, $w_{\text{base}} = 0.30$ for factual queries and $w_{\text{cap}} = 0.75$, $w_{\text{base}} = 0.60$ for reasoning queries.

Let c_{sim} denote the cosine similarity between the candidate's text embedding and the HyDE answer embedding. The final score depends on the candidate's retrieval source:

$$\text{score}(c) = \begin{cases} (1-w)\, c_{\text{sim}} + w\, \tilde{s}_{\text{kw}} & \text{if } c \in \text{B-only} \\ (1-w)\, c_{\text{sim}} + w\ (0.5\, \tilde{s}_{\text{kw}} + 0.5\, \tilde{s}_{\text{frag}}) & \text{if } c \in \text{C-only} \\ (1-w)\, c_{\text{sim}} + w\ (0.7\, \tilde{s}_{\text{kw}} + 0.3\, \tilde{s}_{\text{frag}}) & \text{if } c \in \text{BC} \end{cases}$$

The BC blend weights (0.7/0.3) reflect the moderate positive correlation ($r = 0.51$) observed between s_{kw} and s_{frag} for dual-path candidates, favouring the more precise keyword signal while retaining fragment context.

The top-`k` candidates by final score are returned (default k=30).

3.5. Reference Resolution State Machine

Conversational text is rich with anaphoric references: pronouns (he/she/they), demonstratives (there, that event), and relative time expressions (last time, that day). Without resolution, storing "she went there" loses the identity of both entities.

KeyMem maintains a lightweight state machine with four focus stacks:

- `entity_stack` (limit 10)
- `location_stack` (limit 4)
- `event_stack` (limit 4)
- `time_stack` (limit 4)

Each stack stores the most recently mentioned items of that type, in recency order.

An additional `EntityBinding` map stores associations between entities and their most recently co-occurring location, event, and time, with a TTL of 3 turns.

At store time, the state machine's current contents are serialized as a natural language `state_summary` and provided to the extraction LLM as reference context. The LLM identifies all references in the turn and resolves each to a value from the summary. Code-level validation confirms that resolved entity/location/event values exist in the corresponding stacks before accepting the resolution. Time references are handled in two modes: calendar-relative expressions (yesterday, last week) are resolved to absolute ISO 8601 dates using the anchor time; context-relative expressions (last time, back then) are resolved to the most recent value in `time_stack`.

After extraction, the state machine is updated with the new entities, locations, events, and times from the current turn.

For recall, the application layer may optionally provide an `AttentionState`, a lightweight variant of the state machine maintained at the application level across a user session, to enable pronoun resolution in queries (e.g., "What did she say about that?" → "What did Alice say about the meeting?").

3.6. Memory Consolidation

Individual turns are too granular for topic-level reasoning. To support Fragment-based multi-hop retrieval (Path C), turns are periodically grouped into topic clusters called Fragments.

Every N=10 turns, a consolidation pass runs asynchronously. The 10-turn window is processed by an LLM that groups turns by shared topic (single-turn groups are permitted). Each group becomes a Fragment node in the graph, with a summary description and all member turns linked via `PART_OF` edges. Fragment nodes accumulate the combined keyword vocabulary of their member turns, enabling the SoftTF-IDF matching in Path C.

4. Experiments

4.1. Dataset

The evaluation uses **LoCoMo-10**, a subset of 10 conversations from the LoCoMo benchmark (Maharana et al., 2024). LoCoMo consists of long, multi-session social conversations between two speakers, annotated with question-answer pairs spanning five categories:

Category	n	Description
MultiHop (Cat 1)	179	Requires aggregating information from multiple turns
Temporal (Cat 2)	261	Time-related questions requiring date arithmetic or temporal ordering
OpenDomain (Cat 3)	69	Open-ended reasoning ("Would X likely…")
SingleHop (Cat 4)	705	Direct factual lookup from a single turn
Adversarial (Cat 5)	380	Questions about information not present in the conversation

Total: 1,594 questions across 10 conversations.

4.2. Metrics

- **Token F1**: the harmonic mean of token-level precision and recall between prediction $\hat{y}$ and ground truth y, used for Cat 1–4:

$$\text{F1} = \frac{2 \cdot P \cdot R}{P + R}, \quad P = \frac{|\hat{y} \cap y|}{|\hat{y}|}, \quad R = \frac{|\hat{y} \cap y|}{|y|}$$

  where token sets are computed after lower-casing and whitespace tokenization.

- **Adversarial Accuracy** (Cat 5): fraction of questions correctly answered as "Not mentioned in the conversation."

- **Recall@K**: fraction of evidence turns present in the top-K retrieved memories.

- **Hit@K**: fraction of questions for which at least one evidence turn appears in the top-K.

- **MRR**: mean reciprocal rank of the first evidence turn in the ranked list.

Note on QA prompt: We augment the official LoCoMo QA prompt with three formatting constraints:

1. use absolute dates rather than relative expressions;
2. infer country/region from city names when asked about geography;
3. make reasonable inferences for opinion/preference questions based on context.

Without constraint (1), Temporal F1 drops from 0.592 to 0.355 due to date format mismatches between predictions and ground truth, a QA formatting issue unrelated to retrieval quality. All systems use the identical augmented prompt.

4.3. Baselines

mem0 (Chhikara et al., 2025) represents the fact-compression paradigm. It extracts structured facts from each conversation turn using an LLM, stores them in a Qdrant vector database, and retrieves them via cosine similarity search. We use the open-source implementation with gpt-4.1-mini for both fact extraction and QA, and text-embedding-3-small for embeddings, matching KeyMem's configuration.

SimpleMem (Liu et al., 2025) is a recent memory system that introduces a three-stage pipeline: semantic lossless compression (converting dialogue turns into atomic memory units with absolute timestamps and resolved references), online synthesis, and intent-aware retrieval planning. Crucially, SimpleMem's lossless restatement explicitly encodes absolute timestamps at store time, making it stronger on temporal questions than pure fact-compression approaches. SimpleMem uses a local Qwen3-Embedding-0.6B model for embeddings and gpt-4.1-mini for LLM operations.

4.4. Implementation Details

KeyMem and mem0 both use:

- **Memory LLM**: gpt-4.1-mini
- **QA LLM**: gpt-4.1-mini
- **Embedding model**: text-embedding-3-small (1536 dimensions, OpenAI)
- **top-k**: 30

SimpleMem uses gpt-4.1-mini for LLM operations and Qwen3-Embedding-0.6B (local, 1024 dimensions) for embeddings, following its default configuration.

KeyMem additionally uses FalkorDB as the graph database backend. Store and recall are served via gRPC. Each conversation uses an isolated graph namespace (session_id).

5. Results

5.1. Main Results

Table 3 compares KeyMem, SimpleMem, and mem0 across all five categories. Note that Recall, Hit, and MRR are not available for SimpleMem as its API bundles retrieval and answer generation into a single call.

Table 3: Main Results on LoCoMo-10

System	MultiHop	Temporal	OpenDomain	SingleHop	Adversarial	Overall F1	Recall	Hit	MRR
mem0	0.262	0.080	0.149	0.266	**0.861**	0.372	0.480	0.541	0.311
SimpleMem	0.429	**0.629**	0.339	0.554	0.016	0.415	—	—	—
KeyMem	**0.452**	0.570	**0.343**	**0.659**	0.666	**0.609**	**0.847**	**0.887**	**0.563**

KeyMem achieves the highest overall F1(0.609), outperforming SimpleMem(0.415) and mem0(0.372). The results reveal distinct trade-offs across architectures.

Temporal reasoning is where SimpleMem excels (F1=0.629), slightly outperforming KeyMem (0.570). This is attributable to SimpleMem's lossless restatement mechanism, which explicitly encodes absolute timestamps at store time ("On 2023-07-20, Alice told Bob..."), making temporal questions directly answerable. KeyMem achieves competitive Temporal performance(0.570) through its TimePoint graph nodes and event_time indexing, while mem0 nearly fails(0.080) due to discarding temporal metadata during fact compression.

MultiHop and SingleHop are where KeyMem's graph-based retrieval shows its clearest advantage. KeyMem achieves 0.452 on MultiHop vs SimpleMem's 0.429, both substantially above mem0's 0.262. On SingleHop, KeyMem(0.659) outperforms SimpleMem(0.554) by a larger margin, suggesting that KeyMem's keyword vector search combined with entity graph traversal is more precise for direct factual lookup.

Adversarial reveals a fundamental challenge for fact preserving systems. mem0 leads strongly(0.861) because its poor retrieval quality naturally leads the QA LLM to answer "Not mentioned." KeyMem achieves a balanced 0.666, while SimpleMem nearly fails (0.016), its retrieval planning mechanism apparently always surfaces some context, making it very difficult to identify unanswerable questions. This is a significant limitation for real-world deployment where agents must recognize the boundaries of their knowledge.

5.2. Ablation Study

Table 4 shows the contribution of each system component, evaluated with full (top-k=0) retrieval to isolate retrieval quality from truncation effects.

Table 4: Ablation Study (top-k=0 full recall)

Variant	MultiHop	Temporal	OpenDomain	SingleHop	Adversarial	Overall	ΔOverall	Recall@K	MRR
Full system	**0.518**	**0.597**	0.357	**0.680**	0.611	**0.618**	—	**0.939**	0.558
w/o Path C	0.415	0.574	0.297	0.566	**0.718**	0.575	−0.043	0.735	0.502
w/o Reference Resolution	0.437	0.577	0.349	0.655	0.668	0.608	−0.010	0.847	0.557
w/o Graph Traversal	0.439	0.592	0.335	0.679	0.621	0.609	−0.009	0.868	**0.563**
w/o Intersection Filter	0.444	0.604	0.342	0.665	0.618	0.605	−0.013	0.865	0.559
w/o Source Scoring	0.444	0.581	0.349	0.648	0.653	0.602	−0.015	0.819	0.521

Path C is the most critical component (ΔF1=−0.043, ΔRecall=−0.204). Removing it causes the largest single-component degradation, particularly on MultiHop (−0.103), confirming

that Fragment-based multi-hop expansion is essential for questions requiring aggregation across multiple turns. The Recall@K drop from 0.939 to 0.735 shows that 20% of evidence turns are only reachable via Path C's topic-cluster expansion.

Source-aware scoring ranks second (ΔF1=-0.015, ΔMRR=-0.037). The MRR impact is notably larger than the F1 impact, suggesting that the scoring formula primarily improves the rank of correct answers rather than changing which answers are included.

Intersection filtering (ΔF1=-0.013) reduces noise by requiring multiple retrieval signals to agree. Its removal allows more false positives to enter the candidate set.

Reference resolution contributes ΔF1=-0.010. Its effect is modest in aggregate but matters for conversations with heavy pronoun use.

Graph traversal (entity/time/location matching) contributes the smallest individual effect (ΔF1=-0.009), but its removal interacts with all other components; the combined effect of removing all graph structure would be larger.

Notably, removing any component *improves* Adversarial accuracy, a consistent pattern indicating that higher retrieval quality makes it harder for the QA LLM to identify unanswerable questions. This is a trade-off inherent to improving recall quality.

5.3. Retrieval Depth Sensitivity

Table 5 shows how the retrieval cutoff k affects performance.

Table 5: top-k Sensitivity

top-k	MultiHop	Temporal	OpenDomain	SingleHop	Adversarial	Overall	Recall@K	MRR	Avg tokens/q
10	0.358	0.583	0.296	0.615	**0.713**	0.590	0.745	0.563	1,252
30	**0.452**	0.570	**0.343**	**0.659**	0.666	**0.609**	0.847	0.563	3,472
∞ (all)	0.518	**0.597**	0.357	0.680	0.611	0.618	**0.939**	0.559	15,616

Three observations are notable. First, **MRR is nearly constant across all k values** (0.559–0.563), confirming that the scoring formula consistently ranks correct answers near the top regardless of how many candidates are retained. The k parameter controls coverage, not ranking quality.

Second, **MultiHop is the most k-sensitive category**: increasing from k=10 to k=∞ raises F1 from 0.358 to 0.518 (+0.160), because multi-hop questions require evidence from multiple non-adjacent turns that may be scattered across the rank list.

Third, k=30 offers the best cost-quality tradeoff: it achieves 98.5% of the F1 achievable with full retrieval (0.609 vs 0.618) while consuming only 22.2% of the context tokens (3,472 vs 15,616 per query).

5.4. Cost Analysis

Table 6: Computational Cost Comparison

System	LLM calls/turn (store)	LLM calls/query (recall)	Avg recall latency	Avg context tokens/query
mem0	1	0	0.65s	1,167
SimpleMem	1	1+ (planning)	—	—
KeyMem	1	1	~10s	3,472

KeyMem's recall requires one additional LLM call (for query analysis) and two embedding calls compared to mem0's zero-LLM recall. This translates to a ~15× higher recall latency (10s vs 0.65s). For latency-critical applications, this is a significant trade-off. However, the quality gains are substantial (F1: 0.609 vs 0.372, Recall: 0.847 vs 0.480).

SimpleMem bundles retrieval and answer generation into a single `ask()` call with optional multi-step retrieval planning, making it infeasible to isolate recall latency independently. Store latency is comparable across all three systems (~9–10s per turn), as each requires one LLM call for extraction or fact generation.

5.5. Error Analysis

Error analysis on KeyMem (top-k=30) reveals that **retrieval failure is not the primary bottleneck**: across 1,594 questions, recall failure (Hit@K=0) accounts for fewer than 5% of errors in Cat 1–4. The dominant failure mode is QA-stage errors, which fall into four patterns:

1. **Token F1 evaluation limitations** (largest source): semantically correct predictions that use different surface forms, "3" vs "Three children", "excited" vs "excitement", score F1=0 under strict token matching. A manual review of 30 such cases found approximately 33% to be semantically correct.
2. **Incomplete multi-hop aggregation**: the QA LLM retrieves relevant context but provides a partial answer ("His family" instead of "family, fitness tracker, and love of adventure"). Evidence for these questions is distributed across multiple turns, and the LLM tends to answer from the highest-ranked single turn.
3. **Wrong fact selection**: the QA LLM correctly identifies relevant memories but selects incorrect facts when multiple similar entities or events are present in the retrieved context.
4. **Temporal format mismatches** (Temporal category): despite prompt constraints, occasional date format differences (e.g., "14 October, 2023" vs "October 14, 2023") cause F1=0 for correct answers.

These observations suggest that future work on improving KeyMem should focus on QA-stage improvements (answer aggregation, format normalization) rather than retrieval.

6. Conclusion

This paper presented KeyMem, a graph-augmented vector retrieval system for persistent conversational memory in LLM agents. The system embodies four principled design choices: stateless interfaces, no automatic forgetting, no automatic conflict resolution, and raw memory over compression. On the LoCoMo-10 benchmark, KeyMem achieves an overall token F1 of 0.609, outperforming the fact-compression baseline mem0 by 63.7%.

The ablation study identifies Fragment-based multi-hop expansion (Path C) as the most critical architectural innovation (ΔRecall@K$=-0.204$ when removed), followed by source-aware scoring with query-local normalization (ΔMRR$=-0.037$). These components together enable KeyMem to retrieve the distributed, temporally-ordered evidence required for complex multi-hop and temporal reasoning questions.

As LLM context windows continue to expand and token costs continue to fall, the case for raw memory over compressed summaries will only strengthen. We believe KeyMem represents a principled foundation for the next generation of persistent, accurate, and information-preserving conversational memory systems.

References

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

Darren Edge et al. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*, 2024.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning, 2021. URL https://arxiv.org/abs/2112.09118.

Vladimir Karpukhin et al. Dense passage retrieval for open-domain question answering. In *Proceedings of EMNLP 2020*, 2020.

Patrick Lewis et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems (NeurIPS 2020)*, 2020.

Jiaqi Liu, Yaofeng Su, Peng Xia, Yiyang Zhou, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for llm agents. *arXiv preprint arXiv:2601.02553*, 2025. URL https://github.com/aiming-lab/SimpleMem.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*, 2024.

Charles Packer et al. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.