LightGMEM: Lightweight Agent Graph Memory Generation

Anonymous ACL submission

Abstract

Agent memory systems have been widely proposed to equip large language model (LLM) agents with long-term memory across sessions. Compared to flat memory, graph-based memory more effectively models relationships between facts, but its construction relies on repeated LLM calls that scale poorly with conversation length. We propose LightGMEM, a lightweight entity-centric graph memory framework that addresses three key bottlenecks: (i) replacing per-episode LLM entity extraction with GLiNER2, a zero-shot named entity recognizer, and deferring entity profiling until sufficient cross-episode evidence accumulates; (ii) introducing conflict-lane partitioning for entity disambiguation, serializing only mentions with overlapping candidate sets while resolving independent ones concurrently; (iii) adopting Ego-Splitting to construct overlapping communities, allowing entities to participate in multiple retrieval contexts. On LoCoMo, LightGMEM achieves the best score on 8 of 12 QA metrics with 58.0× fewer LLM calls and 151.6× lower construction runtime than Zep. On LongMemEval, it remains competitive on single-session tasks, with trade-offs on update-heavy and temporally reasoning. These results demonstrate that graph-based memory can remain practical at scale when LLM reasoning is reserved for high-value operations.

1 Introduction

As interactions accumulate, raw conversational histories quickly become redundant, noisy, and difficult to retrieve from precisely. LLM agents extend language models toward interactive reasoning, tool use, personalization, and long-horizon decision making (Shinn et al., 2023; Park et al., 2023; Xia et al., 2025). A central requirement for such agents is long-term memory, which stores past interactions and enables coherent, consistent, and personalized behavior across sessions (Zhang et al., 2025; Zhong et al., 2024; Packer et al., 2023). As interactions accumulate, however, raw conversational histories become redundant and noisy, making precise retrieval increasingly difficult.

Graph-based memory offers a promising alternative by organizing past interactions into structured entities, attributes, and relations rather than flat text chunks, enabling explicit representation of entity connections and traversal-based retrieval over long-term context (Nan et al., 2025; Rasmussen et al., 2025). Frameworks such as GraphRAG (Edge et al., 2024) demonstrate the value of community-level organization for retrieving information from large-scale graphs. Despite this promise, building practical graph memory for long-term dialogue faces three interconnected challenges.

First, graph memory construction typically relies on LLM prompting to extract entities, attributes, and relations from every conversational episode, leading to high token costs and latency that scale poorly with conversation length. Second, mentions extracted across episodes must be resolved into canonical entities. Systems such as Zep (Rasmussen et al., 2025) process this named entity disambiguation (NED) sequentially to preserve temporal consistency, but strict serialization becomes a bottleneck as memory grows. Third, existing graph memory systems commonly employ disjoint community detection, assigning each entity to exactly one cluster. This constraint is ill-suited to conversational settings, where the same entity naturally participates in multiple contexts (e.g., family, work, preferences, travel), and forcing a single assignment conflates separable evidence.

We introduce **LightGMEM**, a lightweight entity-centric graph memory framework that addresses these three challenges through a set of coordinated design choices. For extraction, LightGMEM replaces per-episode LLM calls with GLiNER2 (Zaratiana et al., 2024, 2025a), a zero-shot named entity recognizer, and defers entity

profile construction until sufficient cross-episode evidence has accumulated, reserving LLM reasoning for high-value semantic operations. For disambiguation, LightGMEM introduces *conflict-lane partitioning*: mentions whose candidate sets overlap are serialized to preserve temporal dependencies, while independent mentions are resolved in parallel, substantially reducing NED runtime without sacrificing resolution quality. For community organization, LightGMEM adopts Ego-Splitting (Epasto et al., 2017) to construct overlapping communities, allowing entities to participate in multiple thematic retrieval contexts. These components are complemented by typed entity attributes, weak co-occurrence edges that preserve soft associations, and a multi-faceted retrieval strategy combining semantic, lexical, and graph-connectivity signals.

Our contributions are as follows:

- We propose **LightGMEM**, a lightweight entity-centric graph memory framework for long-term conversational agents that reduces LLM dependence in memory construction through lightweight extraction, selective parallel NED, and deferred profiling.
- We introduce overlapping community construction for conversational graph memory via Ego-Splitting, enabling entities to participate in multiple retrieval contexts rather than being confined to a single disjoint community.
- Experiments on LoCoMo (Maharana et al., 2024) and LongMemEval (Wu et al., 2025) demonstrate that LightGMEM preserves competitive QA performance while reducing graph-memory construction cost by over an order of magnitude.

2 Related Work

Graph-based conversational memory. Long-term memory for LLM agents spans flat stores (Zhong et al., 2024; Chhikara et al., 2025), hierarchical designs (Packer et al., 2023; Kang et al., 2025; Xu et al., 2026), and graph-based approaches that model entities, events, or relations. Systems such as AriGraph (Anokhin et al., 2025), Zep (Rasmussen et al., 2025), MAGMA (Jiang et al., 2026), and EMem-G (Zhou and Han, 2025) demonstrate that graph-structured memory supports multi-hop retrieval and cross-episode evidence accumulation, but their construction relies on repeated LLM extraction and graph updates, creating a scalability bottleneck. LightGMEM addresses this by replacing expensive LLM operations with lightweight extraction and parallelized disambiguation.

Entity extraction and disambiguation. GLiNER and GLiNER2 (Zaratiana et al., 2024, 2025a) enable zero-shot, schema-driven named entity recognition with compact encoders, offering a practical alternative to per-episode LLM extraction. For disambiguation, conversational memory imposes a temporal constraint: each resolution decision affects the candidate set for subsequent mentions, so systems such as Zep update the graph incrementally, preserving consistency but creating a runtime bottleneck as mentions accumulate. LightGMEM addresses this with conflict-lane partitioning, which serializes only mentions with overlapping candidate sets while resolving independent mentions in parallel.

Community organization for retrieval. GraphRAG (Edge et al., 2024) demonstrates the value of community summaries for retrieval, while LightRAG (Guo et al., 2025) and HippoRAG (Gutierrez et al., 2024) further show that graph organization improves retrieval beyond flat similarity search. Standard community detection algorithms such as Leiden (Traag et al., 2018) and label propagation (Raghavan et al., 2007) produce disjoint partitions; however, conversational entities routinely span multiple contexts, and forcing single-cluster assignment discards cross-community connections that retrieval could otherwise exploit. LightGMEM therefore adopts Ego-Splitting (Epasto et al., 2017), which supports overlapping memberships through persona-level decomposition.

3 LightGMEM

LightGMEM is a graph-based memory framework for long-term conversational agents (Figure 1) that combines lightweight mention extraction, selective parallel disambiguation, deferred entity profile induction, and overlapping community organization.

3.1 Memory Graph Formulation

We represent memory as a heterogeneous graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}), \quad \mathcal{V} = \mathcal{V}_{\text{ent}} \cup \mathcal{V}_{\text{epi}} \cup \mathcal{V}_{\text{com}}, \qquad (1)$$

where $\mathcal{V}_{\text{ent}}$, $\mathcal{V}_{\text{epi}}$, and $\mathcal{V}_{\text{com}}$ denote entity, episode, and community nodes. The input conversation is

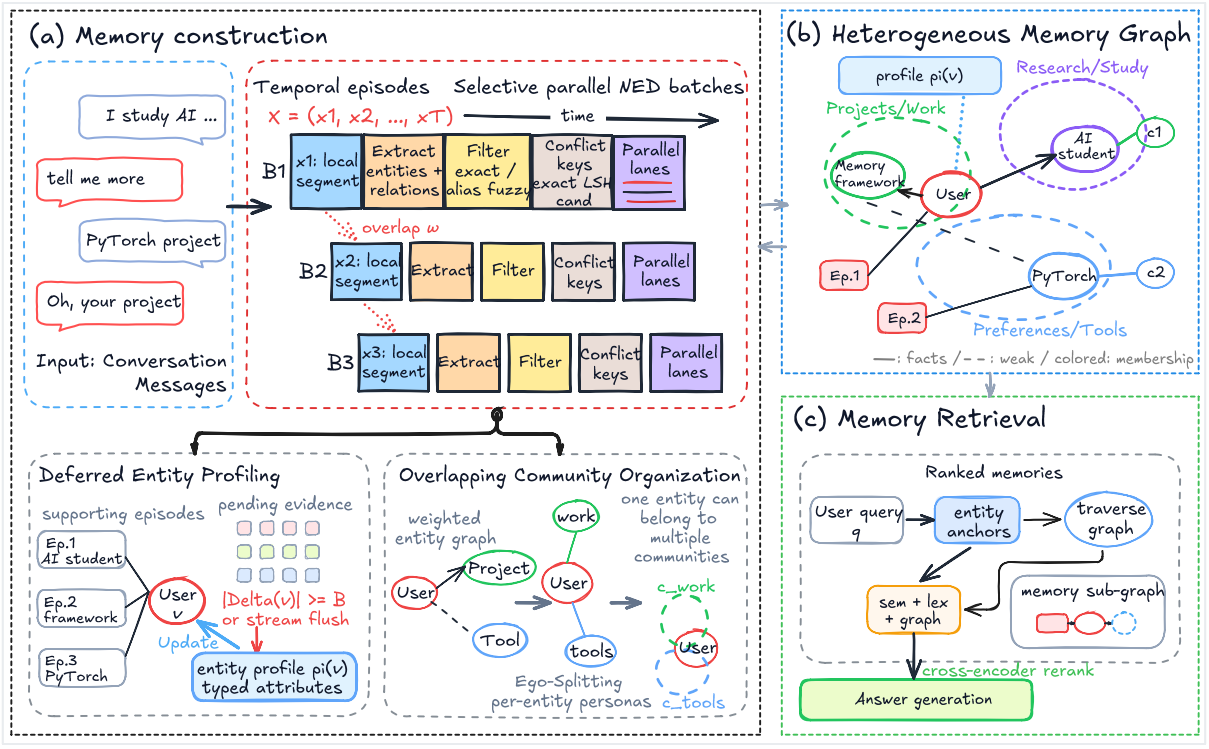

Figure 1: Overview of LightGMEM. Conversational episodes are incrementally transformed into a memory graph containing episode, entity, and community nodes.

segmented into a temporally ordered episode sequence

$$\mathcal{X} = (x_1, x_2, \ldots, x_T), \tag{2}$$

which LightGMEM incrementally maps into $\mathcal{G}$.

Episode nodes store local conversational segments, entity nodes represent canonical entities with typed attribute profiles $\pi(v) = \{a_1(v), \ldots, a_K(v)\}$, and community nodes represent higher-level shared contexts. We use four edge families: $\mathcal{E}_{\text{mem}}$ links episodes to mentioned entities, $\mathcal{E}_{\text{rel}}$ stores explicit extracted relations, $\mathcal{E}_{\text{weak}}$ captures weak co-occurrence associations, and $\mathcal{E}_{\text{com}}$ records entity–community assignments. We summarize the main notation in Table 1.

3.2 Memory Construction

Given the episode stream $\mathcal{X}$, memory construction incrementally builds $\mathcal{G}$ through four stages:

$$\mathcal{X} \to \mathcal{M} \to \mathcal{V}_{\text{ent}} \to \mathcal{P} \to \mathcal{V}_{\text{com}}, \tag{3}$$

where $\mathcal{M}$ is the set of extracted entity mentions, $\mathcal{V}_{\text{ent}}$ is the canonical entity set after disambiguation, $\mathcal{P}$ is the collection of entity profiles, and $\mathcal{V}_{\text{com}}$ is the set of induced community nodes.

3.2.1 Entity and Relation Extraction

For each episode x_t, we first extract a set of entity mentions $\mathcal{M}_t = \{m_{t,1}, \ldots, m_{t,n_t}\}$ using GLiNER2 (Zaratiana et al., 2025b) rather than invoking an LLM on every segment. We separately extract direct relations from the same episode so that relation endpoints omitted by the entity extractor can still be surfaced as candidate mentions.

Each mention m is stored as an enriched tuple

$$m = \big(s(m), \tau(m), \kappa(m), \delta(m)\big), \tag{4}$$

where $s(m)$ is the surface form of m, $\tau(m)$ is its extracted entity type, $\kappa(m)$ is its local context, and $\delta(m)$ is its timestamp. In our setting, $\kappa(m)$ includes local segment text, co-occurring entity names, and directly observed relation neighbors.

To preserve soft conversational associations beyond schema-bound relations, we also accumulate weak association counts between entities. Let $a(u, v)$ denote the accumulated weak association count between canonical entities u and v. We create a weak edge $e_{uv}^{\text{weak}} \in \mathcal{E}_{\text{weak}}$ with weight $a(u, v)$. These edges are not treated as factual predicates; they provide soft structural support for later community construction and retrieval.

Symbol	Definition
$\mathcal{X}$	Temporally ordered episode sequence.
$\mathcal{G}$	Memory graph.
$\mathcal{V}_{\text{ent}}, \mathcal{V}_{\text{epi}}, \mathcal{V}_{\text{com}}$	Entity, episode, and community node sets.
$\mathcal{E}_{\text{rel}}, \mathcal{E}_{\text{weak}}$	Explicit relation edges and weak semantic association edges.
$\mathcal{M}_t$	Mention set extracted from episode x_t.
$\mathcal{C}_\ell$	Canonical entity set after mention-processing step ℓ.
$\sigma(m, v)$, $\rho(m)$	Mention–entity compatibility score and top confidence.
γ	Confidence threshold for entering LLM-based disambiguation.
$\mathcal{K}(m)$	Conflict-key set of mention m.
$\mathcal{L}$	Conflict lanes for parallel NED.
$\mathcal{H}(v)$, $\mathcal{H}^{\text{prof}}(v)$	All supporting episodes of entity v and the subset already absorbed into its profile.
$\Delta(v)$	Pending profile evidence of entity v.
B	Profile update batch threshold.
$\mu(v)$	Community membership set of entity v.

Table 1: Main notation used in Section 3.2.

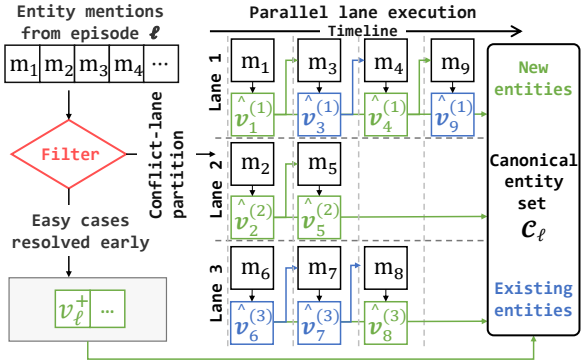

Figure 2: Selective Parallel Entity Disambiguation.

3.2.2 Selective Parallel Entity Disambiguation

The extracted mentions $\mathcal{M} = \bigcup_{t=1}^{T} \mathcal{M}_t$ must be resolved to canonical entity nodes while respecting temporal order. Let $(m_1, m_2, \ldots, m_N)$ denote the flattened mention sequence sorted by time. We call two mentions *conflicting* if resolving one may change the candidate set or final assignment of the other. We propose *Selective Parallel Entity Disambiguation*, which proceeds in three steps: pre-disambiguation filtering identifies the LLM-bound mention subset, conflict-key grouping partitions that subset into conflict lanes, and different lanes are resolved concurrently. The procedure is executed over a sequence of batches with an overlap window of size ω, so mentions near batch boundaries can still be compared across neighboring batches (Figure 2).

Pre-disambiguation Filtering Let $\mathcal{C}_{\ell-1}$ be the canonical entity set before processing mention m_ℓ. We first attempt deterministic or similarity-based resolution against $\mathcal{C}_{\ell-1}$ using exact normalization, alias matching, and fuzzy candidate retrieval. If the mention remains unresolved, we compute a heuristic compatibility score

$$\sigma(m_\ell, v) = f_{\text{conf}}\big(\phi(m_\ell, v)\big), \; v \in \mathcal{C}_{\ell-1}, \tag{5}$$

where $\phi(m_\ell, v)$ collects mention–candidate compatibility features and f_{conf} is a weighted scoring function. We then define the mention confidence

$$\rho(m_\ell) = \max_{v \in \mathcal{C}_{\ell-1}} \sigma(m_\ell, v). \tag{6}$$

Only mentions with $\rho(m_\ell) \geq \gamma$ enter LLM-based disambiguation. If $\rho(m_\ell) < \gamma$, we directly instantiate a new canonical entity v_ℓ^+ and update the canonical set as $\mathcal{C}_\ell = \mathcal{C}_{\ell-1} \cup \{v_\ell^+\}$.

Conflict-lane partitioning For each LLM-bound mention m_ℓ, we compute a conflict-key set

$$\mathcal{K}(m_\ell) = \mathcal{K}_{\text{exact}}(m_\ell) \cup \mathcal{K}_{\text{lsh}}(m_\ell) \cup \mathcal{K}_{\text{cand}}(m_\ell). \tag{7}$$

where the three components correspond to exact-name signals, MinHash-LSH bucket collisions, and top-ranked candidate identities. Two mentions are considered potentially conflicting if their conflict-key sets intersect.

We partition LLM-bound mentions into lanes $\mathcal{L} = \{L_1, \ldots, L_K\}$ such that mentions within the same lane may conflict and must remain serialized, while mentions in different lanes satisfy

$$\begin{aligned} &\mathcal{K}(m) \cap \mathcal{K}(m') = \varnothing, \\ &\forall m \in L_i, \; \forall m' \in L_j, \; i \neq j. \end{aligned} \tag{8}$$

Algorithm 1 summarizes the partitioning process.

Parallel lane execution Each lane L_k is processed in temporal order, but different lanes are executed concurrently. Let $\mathcal{C}_r^{(k)}$ denote the lane-local canonical entity set after the first r mentions in lane L_k have been resolved. If $m_i \prec_t m_j$ and both belong to the same lane, then m_i is resolved before m_j; no such ordering is imposed across disjoint lanes. For the r-th mention in lane L_k, the resolver outputs a canonical entity $\hat{v}_r^{(k)}$, which may be either an existing entity inherited from the initial canonical set or a newly created entity introduced during lane processing. The lane-local state is then updated incrementally,

$$\mathcal{C}_r^{(k)} = \mathcal{C}_{r-1}^{(k)} \cup \{\hat{v}_r^{(k)}\}, \tag{9}$$

with the understanding that if $\hat{v}_r^{(k)}$ already exists, the update merges new evidence into that entity rather than creating a duplicate node. Thus every later mention in the same lane is resolved against the canonical state produced by all preceding mentions in that lane, including newly created entities.

3.2.3 Deferred Entity Profiling

We decouple entity discovery from profile construction. Profile induction is triggered only when an entity has accumulated sufficient unprofiled episode evidence.

For each entity v, let $\mathcal{H}(v)$ denote the set of associated episodes and let $\mathcal{H}^{\text{prof}}(v) \subseteq \mathcal{H}(v)$ denote the subset already absorbed into the current profile. The pending evidence set is

$$\Delta(v) = \mathcal{H}(v) \setminus \mathcal{H}^{\text{prof}}(v). \tag{10}$$

We update the profile of v only when $|\Delta(v)| \geq B$ for a batch threshold B, or when a final flush is triggered at the end of the episode stream. The resulting profile update is

$$\pi^{(r)}(v) = \mathcal{U}\big(\pi^{(r-1)}(v), \Delta_r(v), \mathcal{R}_r(v)\big), \tag{11}$$

where $\Delta_r(v)$ is the current batch of newly incorporated episodes for entity v, $\mathcal{R}_r(v)$ is the relation evidence filtered from those episodes, and $\mathcal{U}$ is an LLM-based merge operator constrained by the schema of v.

Entities with only sparse support skip LLM profile induction and retain a lightweight profile state.

3.2.4 Overlapping Community Organization

LightGMEM constructs overlapping communities over the entity graph, allowing entities to participate in multiple retrieval contexts.

We first derive a weighted entity subgraph from the memory graph,

$$\mathcal{G}_{\text{ent}} = (\mathcal{V}_{\text{ent}}, \mathcal{E}_{\text{ent}}), \quad \mathcal{E}_{\text{ent}} = \mathcal{E}_{\text{rel}} \cup \mathcal{E}_{\text{weak}}, \tag{12}$$

where edge weights reflect the frequency of explicit or weak association evidence.

We then apply Ego-Splitting (Epasto et al., 2017). For each entity v, let $\Gamma(v)$ denote its neighborhood. Ego-Splitting partitions the ego-minus-center subgraph into local persona clusters

$$\Pi_v = \mathcal{A}_{\text{local}}\big(\mathcal{G}_{\text{ent}}[\Gamma(v)]\big), \tag{13}$$

where $\mathcal{A}_{\text{local}}$ is a local partitioning operator. Each partition in Π_v induces one persona of v. A persona graph is then formed by connecting compatible personas of adjacent entities:

$$\begin{aligned}(u^i, v^j) \in \mathcal{E}' \iff (u, v) \in \mathcal{E}_{\text{ent}}, \\ v \in S_u^i, \quad u \in S_v^j,\end{aligned} \tag{14}$$

where S_u^i is the i-th local persona neighborhood of entity u.

Standard partitioning on the persona graph yields overlapping entity memberships $\mu(v) \subseteq \mathcal{V}_{\text{com}}$. Finally, an LLM generates a short name and description for each community node.

3.3 Memory Retrieval

At inference time, given a query q, we extract query entities and match them against the canonical entity index as anchors for graph-aware retrieval.

Candidate memories are scored by combining semantic similarity, lexical matching, and graph connectivity:

$$\begin{aligned}\text{score}(d \mid q) = \lambda_{\text{sem}} s_{\text{sem}}(q, d) + \lambda_{\text{lex}} s_{\text{lex}}(q, d) \\ + \lambda_{\text{graph}} s_{\text{graph}}(q, d).\end{aligned} \tag{15}$$

Here d may be an episode, entity profile, or community description. Retrieved candidates are then re-ranked with a cross-encoder before answer generation.

4 Experiments

We evaluate LightGMEM along two axes, answer quality and memory-construction efficiency, organized around three research questions. **RQ1**: Does LightGMEM maintain competitive QA performance while reducing construction cost relative to existing graph-based memory systems? **RQ2**: How much runtime does Parallel-Pipeline NED save, and under what workload conditions is the speedup largest? **RQ3**: Does overlapping community organization provide more coherent retrieval structure and better downstream QA than disjoint community alternatives?

4.1 Experimental Setup

Datasets. We evaluate LightGMEM on two long-term conversational memory benchmarks. **LoCoMo** (Maharana et al., 2024) contains 1,540 QA pairs over 10 multi-session conversations and evaluates single-hop recall, multi-hop reasoning, temporal reasoning, and open-domain retrieval. For **LongMemEval** (Wu et al., 2025), we use the 500-question LongMemEval$_S$ subset, which covers

Method Category	Method	Multi-Hop			Open-Domain			Single-Hop			Temporal		
		F_1	B_1	LLM-J	F_1	B_1	LLM-J	F_1	B_1	LLM-J	F_1	B_1	LLM-J
RAG	FullContext	0.354	0.261	0.668	0.245	0.172	0.486	0.531	0.447	0.830	0.441	0.361	0.562
	RAG-4096	0.186	0.117	0.313	0.190	0.135	0.326	0.222	0.186	0.320	0.195	0.157	0.237
Flat Memory	LangMem	0.335	0.239	0.524	0.294	0.235	0.476	0.388	0.331	0.614	0.319	0.262	0.249
	Mem0	0.343	0.252	0.603	0.271	0.194	0.406	0.444	0.377	0.681	0.444	0.376	0.504
	NEMORI	0.365	0.256	0.653	0.208	0.151	0.448	0.544	0.432	0.821	0.567	0.466	0.710
Other Structural Memory	MemoryOS	0.412	0.308	0.567	**0.486**	**0.430**	0.458	0.353	0.252	0.671	0.200	0.165	0.402
	StructMem	-	-	0.688	-	-	0.469	-	-	0.811	-	-	**0.816**
Graph Structural Memory	Zep	0.275	0.193	0.505	0.229	0.157	0.396	0.397	0.337	0.632	0.448	0.381	0.589
	$Mem0^g$	-	-	0.657	-	-	0.472	-	-	0.757	-	-	0.581
	MAGMA	0.264	0.172	0.528	0.180	0.136	0.517	0.551	0.477	0.776	0.509	0.370	0.650
	EMem-G	0.406	0.305	0.747	0.242	0.199	0.573	0.504	0.422	0.823	**0.581**	0.468	0.760
	LightGMEM	**0.419**	**0.335**	**0.748**	0.272	0.200	**0.677**	**0.641**	**0.585**	**0.901**	0.564	**0.501**	0.760

Table 2: LoCoMo QA performance. Metrics include token-level F_1, BLEU-1 (B_1), and LLM-as-a-Judge (LLM-J) across four question types, with baselines grouped by memory paradigm.

Method	single-session -user	multi-session	single-session -assistant	single-session -preference	temporal-reasoning	knowledge -update	Overall
Full-Context	0.786	0.383	0.893	0.067	0.421	0.782	0.550
NEMORI	0.886	0.511	0.839	0.467	0.617	0.615	0.642
Zep	0.929	0.474	0.750	0.533	0.541	0.744	0.632
MAGMA	0.729	0.504	0.839	**0.733**	0.451	0.667	0.612
EMem-G	0.870	**0.736**	0.875	0.322	**0.748**	**0.944**	**0.779**
LightGMEM	**0.986**	0.511	**0.964**	**0.733**	0.459	0.782	0.675

Table 3: LongMemEval QA performance. LLM-as-a-Judge accuracy is reported across six types and overall.

Method	Tokens (k)			Calls	Time (s)
	Input	Output	Total		
LangMem	9,873	1,192	11,066	5,990	26,281
A-Mem	9,126	2,368	11,494	11,754	60,607
Mem0	10,958	1,239	12,196	9,181	30,057
MemoryOS	1,889	939	2,868	5,534	24,220
StructMem	**1,501**	436	**1,937**	1,056	22,854
$Mem0^g$	33,512	2,313	35,825	53,514	115,670
Zep	29,389	1,362	30,751	26,616	142,048
MAGMA	-	-	-	-	1,404
$LightGMEM^s$	2,422	37	2,459	**454**	1,619
$LightGMEM^p$	2,441	**36**	2,478	459	**937**

Table 4: Construction-cost on LoCoMo. ^{s} and ^{p} denote the serial and parallel LightGMEM variants.

single-session user and assistant facts, preferences, multi-session reasoning, temporal reasoning, and knowledge updates.

Evaluation Metrics. For LoCoMo, we report token-level F_1, BLEU-1 (B_1), and LLM-as-a-Judge (LLM-J), following the benchmark's combination of lexical and semantic answer-quality measures. For LongMemEval, we report LLM-J accuracy by question type and overall. To measure construction efficiency, we report the number of LLM calls, input and output tokens, total tokens, and wall-clock runtime.

Baselines. On **LoCoMo**, we compare against representative long-memory baselines from prior work, including RAG baselines (Lewis et al., 2020), flat memory systems (Chase, 2022; Chhikara et al., 2025; Ma et al., 2025), other structural memory systems (Kang et al., 2025; Xu et al., 2026), and graph structural memory systems (Rasmussen et al., 2025; Jiang et al., 2026; Zhou and Han, 2025). On **LongMemEval**, we compare with the available full-context and memory-system baselines: *Full-Context*, *NEMORI*, *Zep*, *MAGMA*, and *EMem-G*. All models use `gpt-4o-mini` for generation and `text-embedding-3-small` for embeddings. Our code is provided in the supplementary material.

4.2 Main Results

Tables 2 and 3 report QA performance across both benchmarks. LightGMEM is not uniformly best, but it consistently occupies a strong accuracy-cost region: it leads on most LoCoMo metrics and remains competitive on LongMemEval, while the cost study in Table 4 shows far fewer LLM operations during memory construction.

On LoCoMo, LightGMEM achieves the best score on eight of the twelve reported metrics. The gains are most pronounced for *single-hop* questions, where LightGMEM reaches 0.641 F_1, 0.585 B_1, and 0.901 LLM-J, outperforming both flat-memory and graph-memory baselines. It also leads all metrics on *multi-hop* questions, suggesting that entity-centric graph organization helps aggre-

gate evidence across related memories. On *open-domain* questions, LightGMEM obtains the best LLM-J score but not the best lexical-overlap scores; this mismatch indicates that semantically acceptable answers in this category need not reuse the same surface forms as the references. On *temporal* questions, LightGMEM is competitive (LLM-J of 0.760 vs. StructMem's 0.816), though StructMem's advantage suggests temporal reasoning benefits from a different organizational strategy.

LongMemEval presents a complementary picture. LightGMEM obtains the best results on *single-session-user* and *single-session-assistant* questions, and ties the best score on *single-session-preference*. Its overall score of 0.675 is second-best among the compared systems and close to NEMORI and Zep, but it trails EMem-G on *multi-session*, *temporal-reasoning*, and *knowledge-update* questions. This pattern is consistent with the design of LightGMEM: typed entity records and community retrieval are effective when evidence can be consolidated into stable entity-centric memories, while update-heavy and temporally complex cases benefit from stronger mechanisms for revising previously stored state.

4.3 Analysis: Construction Efficiency

Table 4 shows that existing graph-based memory systems suffer from a construction-efficiency bottleneck compared with other structural memory systems, especially in LLM calls, token usage, and runtime. LightGMEM substantially reduces this cost while retaining graph-based organization: compared with Zep, the parallel variant uses about $58.0\times$ fewer LLM calls, $12.4\times$ fewer total tokens, and $151.6\times$ lower runtime; compared with Mem0^{g}, it uses about $116.6\times$ fewer calls, $14.5\times$ fewer tokens, and $123.4\times$ lower runtime. LightGMEM^{s} and LightGMEM^{p} use nearly identical token budgets and call counts, but parallel execution reduces runtime from 1,619 seconds to 937 seconds, showing that the gain comes from removing unnecessary serialization in NED rather than reducing LLM volume. LightGMEM thus reduces graph-based memory-construction cost substantially while preserving memory-graph quality. Together, the QA and cost results answer RQ1 affirmatively: LightGMEM maintains competitive QA performance, leading on most LoCoMo metrics and ranking second on LongMemEval overall, while reducing construction cost by over an order of magnitude relative to existing graph-based memory systems.

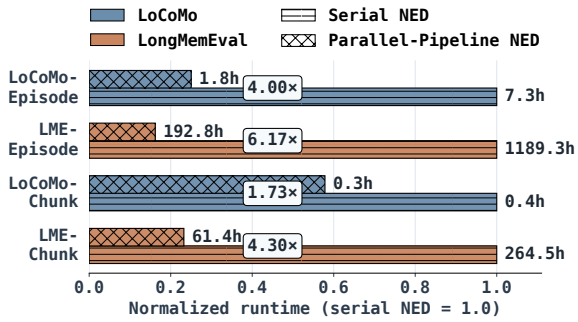

Figure 3: Runtime of serial vs. parallel NED. Bars are normalized by the serial baseline.

4.4 Ablation: Parallel-Pipeline NED

To answer RQ2, we isolate the contribution of Parallel-Pipeline NED by comparing the measured asynchronous pipeline wall clock against a serial counterpart without conflict-lane partitioning. Figure 3 shows clear gains in both settings, with larger speedups on LongMemEval than on LoCoMo. This cross-dataset ordering is consistent with the difference in benchmark scale: LongMemEval's 57.5M tokens generate far more entity mentions than LoCoMo's 90K, and more unresolved mentions mean more parallel execution opportunities. In the *chunk-based* setting, the speedup becomes smaller but the dataset ordering remains unchanged. This is consistent with how chunking interacts with our pipeline: larger local context windows allow the deterministic resolver to handle more mentions in a single pass, reducing the LLM-bound workload and therefore the parallelism headroom. In answer to RQ2, Parallel-Pipeline NED delivers a meaningful runtime reduction across both benchmarks, and the speedup is largest under high mention volume (LongMemEval, episode-level), confirming that the mechanism is most valuable for long-horizon conversational workloads where the LLM-bound disambiguation queue is nontrivial.

4.5 Analysis: Conflict Structure and Confidence Gating

We further examine the workload conditions underlying RQ2. Figure 4(a) shows the size distribution of conflict lanes after deterministic resolution. The remaining LLM-bound mentions decompose into many small conflict lanes rather than a single entangled queue. The workload is therefore structurally sparse even when nontrivial in volume, which is the regime where conflict-lane partitioning is effective: only mentions with shared conflicts must remain serialized, while unrelated mentions are processed

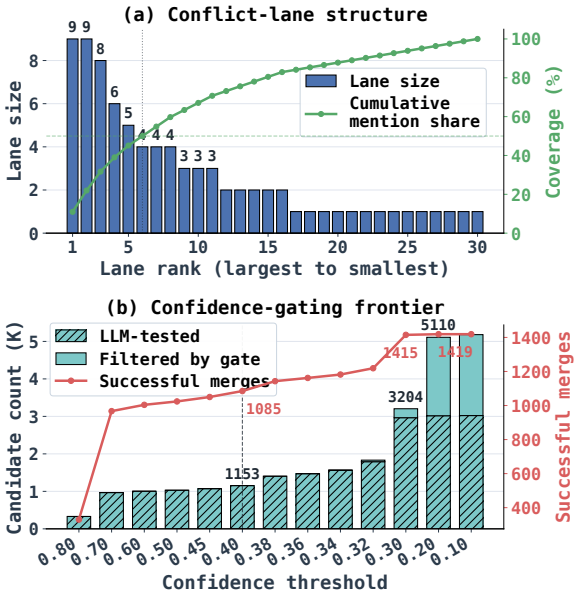

Figure 4: Conflict structure and confidence-gating frontier on LoCoMo.

in parallel. Figure 4(b) illustrates why confidence gating is complementary. Lowering γ expands the candidate frontier much faster than it recovers additional successful merges: at $\gamma = 0.40$, the system already recovers most merges at $\gamma = 0.20$ while exposing only a small fraction of the candidate frontier. Confidence gating thus serves a dual purpose: it reduces LLM usage, and, more importantly, it preserves the sparse conflict structure that Parallel-Pipeline NED exploits. Without this gate, low-confidence candidates would proliferate, erode lane independence, and force more mentions into serial execution. The two mechanisms are mutually reinforcing: conflict-lane partitioning benefits from the decomposability of genuine conflicts, while confidence gating protects that decomposability by limiting unnecessary ambiguity.

4.6 Ablation: Community Detection Algorithm

To isolate the effect of community organization, we keep all other components of LightGMEM fixed and replace only the community detection algorithm. Table 5 shows that ego-splitting yields the strongest communities by semantic coherence (0.74) and relation density (1.00), and produces the best judge score (82.99). In contrast, the disjoint alternatives yield weaker structural communities: label propagation and Leiden produce roughly half the relation density, indicating that forcing a single assignment discards substantial entity–entity connections. The downstream QA gap is modest (0.5–0.6 points) but consistent in direction, suggesting that higher-quality community structure provides a reliable, if small, retrieval advantage.

Setting	Coher. ↑	Rel Dens ↑	LLM-J ↑
No community	–	–	82.21
Ego-splitting	**0.74**	**1.00**	**82.99**
Label prop.	0.66	0.39	82.47
Leiden	0.67	0.49	82.40

Table 5: Community-layer comparison on LoCoMo.

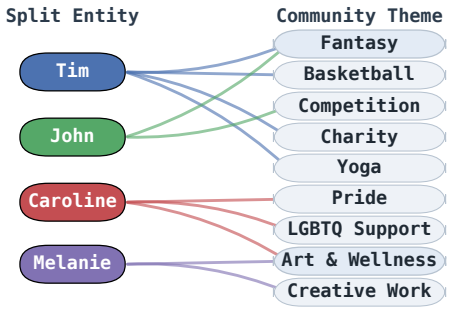

Figure 5: Illustrative split entities under ego-splitting.

Figure 5 provides qualitative intuition: entities split by ego-splitting are typically those whose conversational evidence spans distinct thematic neighborhoods, such as a colleague appearing in both work planning and personal social events. Disjoint algorithms force such entities into a single community, conflating separable contexts. Ego-splitting avoids this by preserving each neighborhood separately, maintaining local coherence while exposing retrieval handles into all relevant contexts. This answers RQ3: overlapping community organization improves retrieval by preserving thematically coherent neighborhoods that disjoint algorithms would conflate, though the QA gain is modest in absolute terms, indicating that community structure provides a reliable but supplementary retrieval signal atop the entity-centric graph.

5 Conclusion

We presented LightGMEM, a lightweight entity-centric graph memory framework that combines zero-shot entity extraction, parallel-pipeline NED via conflict-lane partitioning, and overlapping community construction via Ego-Splitting. Evaluated on LoCoMo and LongMemEval, LightGMEM achieves competitive QA performance while reducing construction cost by over an order of magnitude, demonstrating that graph-based memory can remain practical when LLM reasoning is reserved for high-value operations.

6 Limitations

LightGMEM depends on the quality of upstream extraction, and confidence thresholds may require recalibration when applied outside conversational benchmarks. Our results also indicate weaker performance on temporal-reasoning and knowledge-update questions, suggesting that the current design favors stable memory consolidation over rapid state revision. Validation on end-to-end interactive agent tasks remains an avenue for future work.

References

Petr Anokhin, Nikita Semenov, Artyom Y. Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. 2025. Arigraph: Learning knowledge graph world models with episodic memory for LLM agents. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, Montreal, Canada, August 16-22, 2025*, pages 12–20. ijcai.org.

Harrison Chase. 2022. Langchain. `https://github.com/langchain-ai/langchain`.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready AI agents with scalable long-term memory. In *ECAI 2025 - 28th European Conference on Artificial Intelligence, 25-30 October 2025, Bologna, Italy - Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025)*, Frontiers in Artificial Intelligence and Applications, pages 2993–3000. IOS Press.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph RAG approach to query-focused summarization. *CoRR*, abs/2404.16130.

Alessandro Epasto, Silvio Lattanzi, and Renato Paes Leme. 2017. Ego-splitting framework: from non-overlapping to overlapping clusters. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 145–154. ACM.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2025. Lightrag: Simple and fast retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 10746–10761. Association for Computational Linguistics.

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Dongming Jiang, Yi Li, Guanpeng Li, and Bingzhe Li. 2026. MAGMA: A multi-graph based agentic memory architecture for AI agents. *CoRR*, abs/2601.03236.

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 25961–25970, Suzhou, China. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Wenquan Ma, Jiayan Nan, Wenlong Wu, and Yize Chen. 2025. What Deserves Memory: Adaptive Memory Distillation for LLM Agents. *arXiv e-prints*, arXiv:2508.03341.

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), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 13851–13870. Association for Computational Linguistics.

Jiayan Nan, Wenquan Ma, Wenlong Wu, and Yize Chen. 2025. Nemori: Self-organizing agent memory inspired by cognitive science. *CoRR*, abs/2508.03341.

Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. 2023. Memgpt: Towards llms as operating systems. *CoRR*, abs/2310.08560.

Joon Sung Park, Joseph C. 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, UIST 2023, San Francisco, CA, USA, 29 October 2023- 1 November 2023*, pages 2:1–2:22. ACM.

Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, 76(3):036106.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. Zep: A temporal knowledge graph architecture for agent memory. *CoRR*, abs/2501.13956.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Vincent A. Traag, Ludo Waltman, and Nees Jan van Eck. 2018. From louvain to leiden: guaranteeing well-connected communities. *CoRR*, abs/1810.08473.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. 2025. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *CoRR*, abs/2511.16043.

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. *CoRR*, abs/2604.21748.

Urchade Zaratiana, Gil Pasternak, Oliver Boyd, George Hurn-Maloney, and Ash Lewis. 2025a. Gliner2: An efficient multi-task information extraction system with schema-driven interface. *CoRR*, abs/2507.18546.

Urchade Zaratiana, Gil Pasternak, Oliver Boyd, George Hurn-Maloney, and Ash Lewis. 2025b. GLiNER2: Schema-driven multi-task learning for structured information extraction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 130–140, Suzhou, China. Association for Computational Linguistics.

Urchade Zaratiana, Nadi Tomeh, Pierre Holat, and Thierry Charnois. 2024. Gliner: Generalist model for named entity recognition using bidirectional transformer. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 5364–5376. Association for Computational Linguistics.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025. A survey on the memory mechanism of large language model-based agents. *ACM Trans. Inf. Syst.*, 43(6):155:1–155:47.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 19724–19731. AAAI Press.

Sizhe Zhou and Jiawei Han. 2025. A simple yet strong baseline for long-term conversational memory of LLM agents. *CoRR*, abs/2511.17208.

A Potential Risks

Because LightGMEM is designed for long-term conversational memory, errors in entity extraction, entity disambiguation, or profile consolidation may lead to incorrect or outdated stored memories. In addition, applying the system to real user conversations may involve privacy-sensitive information. We therefore frame LightGMEM as a research prototype and emphasize the need for careful deployment safeguards, privacy controls, and validation beyond benchmark settings.

B Method Supplement

Algorithm 1 Conflict-Lane Partitioning

Require: Ordered mentions $m_1, \ldots, m_N$, initial canonical set $\mathcal{C}$
Ensure: Conflict lanes $\mathcal{L} = \{L_1, \ldots, L_K\}$

Build candidate index $\mathcal{I}$ from $\mathcal{C}$; $\mathcal{L} \leftarrow []$
for $\ell = 1$ to N **do**
  if m_ℓ is deterministically resolved under $\mathcal{I}$ **then**
    Update $\mathcal{C}$ and $\mathcal{I}$; **continue**
  end if
  if $\rho(m_\ell) < \gamma$ **then**
    Promote m_ℓ to a new canonical entity; update $\mathcal{C}$ and $\mathcal{I}$; **continue**
  end if
  Compute conflict keys $\mathcal{K}(m_\ell)$
  Find all lanes whose key sets intersect $\mathcal{K}(m_\ell)$
  if no such lane exists **then**
    Create a new lane containing m_ℓ
  else
    Insert m_ℓ into one matched lane and merge all intersecting lanes
  end if
end for
return $\mathcal{L}$

C Hyperparameter Details

This appendix summarizes the hyperparameters and implementation settings used by the main LightGMEM pipeline. In our experiments, the base memory graph is built in `chunk` mode with `entity_profile` profiling, and the community layer used for retrieval is constructed with Ego-Splitting.

D Experimental Details

D.1 Dataset Details

LoCoMo (Maharana et al., 2024) is a multi-session dialogue benchmark comprising 10 conversations with very long interaction histories. The benchmark averages 27.2 sessions, 588.2 turns, and 16,618.1 tokens per conversation, with conversations spanning up to 32 sessions.

LongMemEval-S (Wu et al., 2025) contains 500 evaluation instances. Concatenating the full history is approximately 115K tokens, or roughly 40 history sessions, for Llama 3.

D.2 Baseline Details

Full-Context and RAG-4096. These two baselines are simple retrieval settings rather than standalone memory architectures. *Full-Context* directly feeds the entire available conversation history to the answer model without a separate memory layer. *RAG-4096* follows the standard retrieval-augmented generation paradigm of Lewis et al. (2020), treating the history as a document collection and retrieving semantically relevant chunks; in our experiment naming, the history is chunked at 4096 tokens.

LangMem. LangMem is a framework-style long-term memory baseline from the LangChain ecosystem rather than a standalone research paper (Chase, 2022). LangMem provides a memory API that stores long-term memories as JSON documents in a persistent store, supports search over those memories, and lets agents write memories during interaction or through a background memory manager.

Mem0 and Mem0^g. Mem0 is a memory-centric architecture that extracts salient memories from conversations, compares them against existing memories, and then applies memory operations such as add, update, delete, or no-op (Chhikara et al., 2025). Mem0^g is a graph-enhanced variant that augments the base memory design with graph-based memory representations to capture richer relational structure among conversational elements.

NEMORI. NEMORI is a self-organizing memory framework inspired by cognitive science (Nan et al., 2025). Its central idea is that the agent should not retain all observations uniformly; instead, memory formation is guided by adaptive distillation, so the system preferentially stores and reorganizes information that is estimated to be more useful for future tasks.

MemoryOS. MemoryOS organizes agent memory using a hierarchical storage architecture with short-term, mid-term, and long-term personal memory (Kang et al., 2025). It uses four core modules, namely storage, updating, retrieval, and generation, with dynamic movement of information across memory levels rather than a single flat store.

StructMem. StructMem is a structured memory framework for long-horizon agent behavior (Xu et al., 2026). It employs temporally anchored structured memories, dual perspectives over user and agent behavior, and periodic semantic consolidation, with the goal of preserving long-range behavioral and temporal dependencies while controlling memory cost.

Zep. Zep formulates agent memory as a temporal knowledge graph built by its Graphiti engine (Rasmussen et al., 2025). Zep uses a bi-temporal design and a three-level graph organization over episodes, semantic entities, and higher-level communities, so retrieval can combine raw interaction evidence with structured and summarized memory.

MAGMA. MAGMA is a multi-graph memory architecture for AI agents (Jiang et al., 2026). Rather than storing all structure in a single graph, it models different relational views explicitly and frames retrieval as policy-guided traversal over those graphs, enabling query-adaptive selection of memory evidence.

EMem-G. EMem uses an event-centric memory representation for long-term conversational memory (Zhou and Han, 2025). On top of that representation, EMem provides retrieval variants based on dense search and LLM filtering; EMem-G is the graph-based variant, which adds graph propagation to connect and aggregate evidence across related event units.

Stage	Hyperparameter	Value
Segmentation	Segment unit	chunk
Segmentation	Chunk size	500 tokens
Segmentation	Chunk overlap	50 tokens
Extraction	Entity and relation extractor	GLiNER2
Extraction	Maximum concurrent extraction budget	12,000 tokens in flight
Preprocessing	Coreference resolution	Disabled in the default main pipeline
Deduplication	Processing mode	Parallel pipeline
Deduplication	Dedup batch size	50 chunks
Deduplication	Batch-boundary overlap	3 chunks
Deduplication	Maximum parallel conflict lanes	10
Deduplication	LLM fallback confidence threshold γ	0.40
Profiling	Build mode	`entity_profile`
Profiling	Profile update threshold B	15 episodes
Profiling	Profile prompt context radius	1 neighboring episode on each side
Profiling	Sparse-entity shortcut	Skip LLM profiling on first profile if total evidence ≤ 2 episodes
Community	Community algorithm	Ego-Splitting (`ego_splitting`)
Community	Minimum entities per community	2
Community	Minimum supporting episodes per community	2
Community	Maximum supporting episodes for topic generation	12

Table 6: Main graph-construction hyperparameters used by LightGMEM.

Stage	Hyperparameter	Value
Entity anchoring	Embedding-based entity match top-k	3
Entity anchoring	Entity embedding similarity threshold	0.75
Context retrieval	Profile context nodes retained	5
Context retrieval	Community context nodes retained	2
Episode expansion	Supporting episodes added from each selected profile node	12
Episode retrieval	Initial episode recall depth	30
Rank fusion	BM25 + dense retrieval fusion	Reciprocal rank fusion, rank constant $= 1$
Candidate scoring	Profile-context prior weight	0.20
Candidate scoring	Matched-entity bonus	0.05 per overlapping entity
Candidate scoring	Matched-relation bonus	0.10 per overlapping relation pair
Reranking	Cross-encoder rerank depth	25 episodes
Final answer context	Returned memories	10 episodes

Table 7: Main retrieval hyperparameters used by LightGMEM QA. Episode candidates are first recalled lexically and semantically, then expanded with graph signals and re-ranked before answer generation.

LoCoMo scale	Value
Released conversations	10
Average sessions per conversation	27.2
Average turns per conversation	588.2
Average tokens per conversation	16,618.1
Maximum sessions in one conversation	32

Table 8: Scale statistics of the LoCoMo benchmark.

LongMemEval-S scale	Value
Evaluation instances	500
Approximate context depth per problem	115K tokens
Approximate history length per problem	$\sim$40 sessions

Table 10: Scale statistics of LongMemEval-S.

LoCoMo type	Count	Reported in Table 2
Multi-Hop	282	282
Temporal Reasoning	321	321
Open-Domain	96	96
Single-Hop	841	841
Total	1,540	1,540

Table 9: LoCoMo question categories reported in our main results.

LongMemEval-S type	Share in Figure 9(a)
single-session-user	14%
single-session-assistant	11%
single-session-preference	6%
multi-session	27%
temporal-reasoning	27%
knowledge-update	16%

Table 11: LongMemEval-S question-type distribution.

D.3 Evaluation Metrics

We use three answer-quality metrics in the QA experiments: LLM-as-a-Judge (LLM-J), token-level F_1, and BLEU-1. For the lexical metrics, the evaluation script first lowercases both prediction and reference, removes articles (*a*, *an*, *the*), maps non-alphanumeric characters to spaces, collapses repeated whitespace, and then tokenizes on spaces. Let $\hat{y}$ and y denote the resulting prediction and reference token sequences.

LLM-J. LLM-J is the accuracy of a binary LLM grader that labels each answer as `CORRECT` or `WRONG`. For N evaluated questions,

$$\text{LLM-J} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[g_i = \texttt{CORRECT}],$$

where g_i is the grader output for question i.

Token-level F_1. Let $m = \sum_w \min(c_{\hat{y}}(w), c_y(w))$ be the multiset token overlap, where $c_{\hat{y}}(w)$ and $c_y(w)$ count occurrences of token w in $\hat{y}$ and y. Precision and recall are

$$P = \frac{m}{|\hat{y}|}, \qquad R = \frac{m}{|y|},$$

and the reported score is

$$\text{F}_1 = \frac{2PR}{P+R}.$$

This metric rewards lexical overlap after normalization while remaining less strict than exact-match judging.

BLEU-1. BLEU-1 uses clipped unigram precision with a brevity penalty:

$$p_1 = \frac{m}{|\hat{y}|},$$

$$\text{BP} = \begin{cases} 1, & |\hat{y}| > |y|, \\ \exp(1 - |y|/|\hat{y}|), & |\hat{y}| \le |y|, \end{cases}$$

$$\text{BLEU-1} = \text{BP} \cdot p_1.$$

Compared with F_1, BLEU-1 penalizes overly short generations more directly through the brevity term.

D.4 Additional LongMemEval-S Metrics

Table 12 reports the token-level F_1 and BLEU-1 scores of LightGMEM on LongMemEval-S. These metrics supplement the LLM-as-a-Judge results reported in Table 3.

Type	LLM-J	F_1	B_1
knowledge-update	0.782	0.606	0.557
multi-session	0.511	0.423	0.378
single-session-assistant	0.964	0.808	0.718
single-session-preference	0.733	0.113	0.026
single-session-user	0.986	0.844	0.795
temporal-reasoning	0.459	0.382	0.289

Table 12: Additional LongMemEval-S metrics for LightGMEM. LLM-J denotes LLM-as-a-Judge accuracy, B_1 denotes BLEU-1.

D.5 Retrieval Hit Rate by Category

Tables 13 and 14 summarize QA retrieval hit rates by question category and top-k. For LoCoMo, retrieved memories are chunk-level spans, so a prediction is counted as a hit when a gold evidence turn falls inside a retrieved chunk range. For LongMemEval-S, the retrieved memories are directly compared against the stored gold evidence turn identifiers in the result file.

LoCoMo type	Hit@1	Hit@3	Hit@5
Multi-Hop	0.521	0.780	0.865
Temporal Reasoning	0.567	0.801	0.875
Open-Domain	0.312	0.458	0.562
Single-Hop	0.709	0.899	0.926

Table 13: LoCoMo retrieval hit rates by category and top-k.

LongMemEval-S type	Hit@1	Hit@3	Hit@5
knowledge-update	0.410	0.782	0.846
multi-session	0.466	0.774	0.872
single-session-assistant	0.357	0.464	0.464
single-session-preference	0.400	0.667	0.733
single-session-user	0.586	0.743	0.786
temporal-reasoning	0.594	0.805	0.857

Table 14: LongMemEval-S retrieval hit rates by category and top-k.

D.6 Community Ablation Metrics

The community-ablation study in Table 5 reports semantic coherence, relation density, and downstream LLM-J. The evaluation script in `run_community_evaluation.py` computes a broader set of structural metrics on the community overlay, which we summarize here.

Let the detected communities be $\mathcal{C} = \{C_1, \ldots, C_M\}$. For each community C, let V_C be its entity set, E_C its supporting episodes, $\mathbf{z}_e$ the embedding of episode $e \in E_C$, $\bar{\mathbf{z}}_C = \frac{1}{|E_C|}\sum_{e \in E_C} \mathbf{z}_e$ the mean episode embedding, $\mathbf{d}_C$ the embedding

of the generated community description, and R_C^{in} the set of original entity-relation edges whose two endpoints both lie in V_C.

Semantic coherence and fidelity. Community coherence is the average pairwise cosine similarity among supporting-episode embeddings within a community, averaged over all communities with at least two supporting episodes:

$$\text{Coherence} = \frac{1}{|\mathcal{C}^E_{\geq 2}|} \sum_{C \in \mathcal{C}^E_{\geq 2}} \frac{2}{|E_C|(|E_C|-1)} \cdot \sum_{e_i < e_j \in E_C} \cos(\mathbf{z}_{e_i}, \mathbf{z}_{e_j}).$$

Description fidelity measures how well the generated topic description matches the mean episode semantics:

$$\text{Fidelity} = \frac{1}{|\mathcal{C}'|} \sum_{C \in \mathcal{C}'} \cos(\mathbf{d}_C, \bar{\mathbf{z}}_C),$$

where $\mathcal{C}'$ contains communities with a non-empty description and at least one supporting episode.

Separation, relation density, and temporal coherence. Inter-community separation is defined as one minus the mean pairwise cosine similarity between community mean embeddings:

$$\text{Separation} = 1 - \frac{1}{\binom{M}{2}} \sum_{1 \leq a < b \leq M} \cos(\bar{\mathbf{z}}_{C_a}, \bar{\mathbf{z}}_{C_b}).$$

Relation density measures how many of the possible intra-community entity pairs are supported by original graph edges:

$$\text{RelDens} = \frac{1}{|\mathcal{C}^V_{\geq 2}|} \sum_{C \in \mathcal{C}^V_{\geq 2}} \frac{|R_C^{\text{in}}|}{|V_C|(|V_C|-1)/2}.$$

Temporal coherence is reported as the average standard deviation of supporting-episode timestamps, converted to days, so lower is better:

$$\text{TempStd} = \frac{1}{|\mathcal{C}^T_{\geq 2}|} \sum_{C \in \mathcal{C}^T_{\geq 2}} \text{std}(\{t_e : e \in E_C\}).$$

Community-size inequality. Let $x_{(1)} \leq \cdots \leq x_{(M)}$ be the sorted community entity counts. The reported Gini coefficient is

$$\text{Gini} = \frac{2\sum_{i=1}^{M} i x_{(i)}}{M \sum_{i=1}^{M} x_{(i)}} - \frac{M+1}{M}.$$

Lower Gini indicates a more even size distribution across communities.

Table 15 adds the structural metrics that were not shown in the main paper for the three community algorithms compared in Table 5.

Setting	Fidelity ↑	Separation ↑	Temp Std ↓	Gini ↓
Ego-Splitting	0.656	0.067	57.75	0.000
Label Propagation	0.621	0.052	61.16	0.047
Leiden	0.618	0.109	59.30	0.057

Table 15: Supplementary community-ablation metrics on LoCoMo. Coherence and relation density are already reported in Table 5; this table adds the remaining structure metrics computed by the community evaluation script. Lower Temp Std and Gini are better.

E Prompt Templates

This appendix lists the prompt templates used in the main LightGMEM pipeline for entity disambiguation, entity-profile construction, and community description generation. For readability, runtime values are replaced with uppercase placeholders such as <MESSAGES> and <EXISTING_ENTITIES>.

E.1 Entity Disambiguation Prompt

Prompt name: dedupe_nodes.nodes

```
SYSTEM
You are a helpful assistant that determines whether or not
     ENTITIES extracted
from a conversation are duplicates of existing entities.

USER
<MESSAGES>
<SERIALIZED_MESSAGES>
</MESSAGES>

Each of the following ENTITIES was extracted from the MESSAGES
      block.
Each entity in ENTITIES is represented as a JSON object with
     the following
structure:
{
    id: integer id of the entity,
    name: "name of the entity",
    entity_type: ["Entity", "<optional additional label>",
      ...],
    entity_type_description: "Description of what the entity
      type represents"
}

<ENTITIES>
<SERIALIZED_EXTRACTED_ENTITIES>
</ENTITIES>

<EXISTING ENTITIES>
<SERIALIZED_EXISTING_ENTITIES>
</EXISTING ENTITIES>

Each entry in EXISTING ENTITIES is an object with the
     following structure:
{
    candidate_id: integer id of the candidate entity,
    name: "name of the candidate entity",
    aliases: ["known alias", ...],
    dialog_contents: ["dialog content where this candidate
      entity appeared", ...],
    cooccurring_entity_names: [
        "entity name seen in the same message/dialog context",
       ...
```

```
    ],
    relation_neighbor_names: [
        "entity name directly connected by extracted relations
      ", ...
    ],
    ...<selected supporting attributes>
}

For each of the above ENTITIES, determine if the entity is a
      duplicate of any
of the EXISTING ENTITIES.

Entities should only be considered duplicates if they refer to
       the same
real-world object or concept.

Do NOT mark entities as duplicates if:
- They are related but distinct.
- They have similar names or purposes but refer to separate
      instances or
  concepts.

Task:
ENTITIES contains <NUM_ENTITIES> entities with IDs 0 through
<NUM_ENTITIES_MINUS_1>. Your response MUST include EXACTLY <
      NUM_ENTITIES>
resolutions with IDs 0 through <NUM_ENTITIES_MINUS_1>. Do not
      skip or add IDs.

For every entity, return an object with the following keys:
{
    "id": integer id from ENTITIES,
    "name": the best full name for the entity (preserve the
      original name unless
           a duplicate has a more complete name),
    "duplicate_id": the exact candidate_id of the EXISTING
      ENTITY that is the
                   best duplicate match, or -1 if there is no
       duplicate,
    "duplicate_name": the exact top-level name field of that
      same EXISTING
                    ENTITY, or empty string if there is no
      duplicate
}

Respond with a JSON object in exactly this shape:
{
    "entity_resolutions": [
        {
            "id": 0,
            "name": "entity name",
            "duplicate_id": -1,
            "duplicate_name": "matching existing entity name
      or empty string"
        }
    ]
}

- Always use the exact candidate_id of an EXISTING ENTITY for
      duplicate_id.
- Only use the exact top-level name field of that same
      EXISTING ENTITY for
  duplicate_name.
- Never return an alias, a cooccurring entity name, a relation
       neighbor name,
  or any string copied from dialog_contents.
- Use duplicate_id: -1 and empty string if there is no
      duplicate.
- Never fabricate entity names.
```

E.2 Entity Profile Construction Prompt

Prompt name: `my_tests.new_graph_paradigm.extract_<entity_type>_profile`

This template is instantiated separately for each profiled entity type. The placeholder <ENTITY_TYPE> is replaced by the current type (e.g., Person, Organization, Event), and <SINGLE_VALUE_ATTRIBUTES> is filled with the type-specific attributes that must take at most one value. The final three safety rules are only added for Person.

```
SYSTEM
You are a helpful assistant that extracts <ENTITY_TYPE>
      properties from the
provided messages and structured relationship evidence.

USER
Given the MESSAGES, ENTITY, and RELATIONSHIP_EVIDENCE, update
      the
<ENTITY_TYPE> attributes.

Guidelines:
1. Do not hallucinate values that are not supported by the
      MESSAGES or
   RELATIONSHIP_EVIDENCE.
2. Use RELATIONSHIP_EVIDENCE as reference when deciding
      attributes that involve
   other entities.
3. Use the existing ENTITY attributes as prior context, but
      only return values
   grounded in the current evidence.
4. Do not overwrite, delete, or weaken an existing attribute
      unless the current
   MESSAGES or RELATIONSHIP_EVIDENCE provide clear support for
       the change.
5. If the current evidence is insufficient to refine a field,
      preserve the
   prior value by leaving that field empty in your update
      instead of guessing.
6. description must never be empty. Always return a short,
      evidence-grounded
   description of what this ENTITY is.
7. RELATIONSHIP_EVIDENCE provides candidate values for
      attributes that involve
   other entities. Use the MESSAGES as the primary evidence
      when deciding
   whether those candidates should affect the profile.
8. When generating profileAttributeFacts, the source entity is
       the current
   ENTITY name, and the target entity name must be chosen only
       from the
   corresponding candidates in RELATIONSHIP_EVIDENCE.
9. Each generated fact must connect two distinct entities. Do
      not generate
   duplicate or semantically redundant facts.
10. Each generated fact should closely paraphrase the
      supporting message
    evidence instead of quoting it verbatim. If time
      expressions are vague or
    relative, resolve them using the message timestamps only
      when the evidence
    supports that resolution.
11. The following attributes are SINGLE-VALUE attributes for
      this entity type:
    <SINGLE_VALUE_ATTRIBUTES>
12. For each SINGLE-VALUE attribute, choose at most one best
      value. If
    RELATIONSHIP_EVIDENCE contains multiple candidates, use
      the MESSAGES to
    choose the single best-supported value. Do not combine
      multiple candidates
    into one field.
13. If a SINGLE-VALUE attribute does not have a clearly
      supported best value,
    keep the prior value when it is still plausible; otherwise
       leave the field
    empty.
14. For Person entities, do not infer a personal name from
      nearby context
    unless the MESSAGES explicitly identify this ENTITY as
      that named person.
15. If the ENTITY is a generic role, descriptor, or group
      label, leave
    givenName, familyName, and additionalName empty unless the
       MESSAGES
    explicitly rename that same ENTITY.
16. Do not copy another person's name onto this ENTITY just
      because they appear
    in the same message or conversation.

<MESSAGES>
<TIMESTAMPED_MESSAGE_BLOCK>
</MESSAGES>

<ENTITY>
```

```
{
  "name": "<ENTITY_NAME>",
  "attributes": <SERIALIZED_EXISTING_ATTRIBUTES>
}
</ENTITY>

<RELATIONSHIP_EVIDENCE>
<SERIALIZED_RELATIONSHIP_EVIDENCE>
</RELATIONSHIP_EVIDENCE>
```

E.3 Community Description Prompt

Prompt name: `my_tests.new_graph_paradigm.topic_description`

```
SYSTEM
You are a helpful assistant that names graph topics and
      describes them using
only the provided entities and supporting episodes. Your
      description should
cover the community broadly rather than focusing on only the
      most salient
entity.

USER
Create a concise topic node for this entity community.

Requirements:
- name: 3 to 8 words, noun-phrase style, specific and readable.

- description: 2 to 3 sentences, grounded only in the provided
       entities and
  episodes.
- Do not invent facts that are not supported by the context.
- Prefer a community-level summary that reflects the main
      shared activities,
  relationships, roles, or themes across the entity set.
- The description should cover as many community entities as
      possible, not just
  one or two prominent members. But make sure the description
      still represents
  the overall community composition.

<COMMUNITY ENTITIES>
[
  {
    "name": "<ENTITY_NAME>",
    "entity_type": "<ENTITY_TYPE>",
    "mention_count": <MENTION_COUNT>
  }
]
</COMMUNITY ENTITIES>

<SUPPORTING EPISODES>
[
  {
    "dia_id": "<DIA_ID>",
    "session_id": "<SESSION_ID>",
    "valid_at": "<TIMESTAMP>",
    "matched_entities": ["<ENTITY_1>", "<ENTITY_2>"],
    "text": "<EPISODE_TEXT>"
  }
]
</SUPPORTING EPISODES>
```

F Runtime Ablation Details

We report runtime ablations on LoCoMo by decomposing graph construction into four stages: entity and relation extraction, entity disambiguation, entity profile construction, and community detection. The default setting is the main system used in the paper, with GLiNER2 extraction, parallel conflict-lane disambiguation, entity profile construction, and Ego-Splitting community detection. We then replace GLiNER2 with LLM-based extraction, and separately replace parallel disambiguation with its matched serial version. Table 16 reports absolute stage time and normalized stage share for each setting.

Setting	Metric	Ext.	Dedup.	Prof.	Comm.
Default	Sec.	120.4	294.9	526.5	32.8
(969.3s)	Share	12.35%	30.26%	54.03%	3.37%
LLM Ext.	Sec.	3051.8	555.6	462.9	291.2†
(3887.2s)	Share	69.97%	12.74%	10.61%	6.68%
Serial Dedup	Sec.	110.2	991.4	498.2	35.6†
(1654.4s)	Share	6.74%	60.62%	30.47%	2.18%

Table 16: Stage-wise runtime composition on LoCoMo. Each setting is shown with one row for stage time and one row for normalized stage share.

Under the default pipeline, profile construction is the dominant stage, followed by disambiguation. Replacing GLiNER2 with LLM extraction shifts the main bottleneck to extraction, while replacing parallel disambiguation with its serial counterpart makes disambiguation the dominant cost. GLiNER2 reduces extraction-stage time from 3051.789 seconds to 120.368 seconds, yielding a 25.354× speedup over LLM-based extraction. Parallel conflict-lane execution reduces deduplication time from 991.388 seconds to 294.899 seconds, yielding a 3.362× deduplication-stage speedup. Including the community stage, the default pipeline is 4.010× faster end-to-end than the LLM-extraction variant and 1.707× faster than serial deduplication.