NEXMEM: EFFICIENT MULTI-GRANULAR CONVERSATIONAL MEMORY USING N-ARY LINK SCORING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-term conversational memory methods must preserve specific facts and connect experiences across sessions. Many existing approaches support this through costly, iterative memory refinement steps. We propose NEXMEM, which jointly extracts memory records and their indexing metadata in a single pass, without subsequent iterative refinements. Typed n-ary links, called *nexus points*, organize each memory at two granularities: broad *aboutness* links capture topical context, while fine-grained *event-fact* links capture entity–phrase associations. Retrieval first scores conventional memory, entity, and phrase graph nodes against the query, then propagates relevance between nexus points using personalized PageRank within a bounded local neighborhood. Across three long-term conversational-memory benchmarks, NEXMEM achieves the highest average accuracy among evaluated methods on each benchmark, including a 28-percentage-point advantage over the strongest evaluated baseline on the benchmark with the longest histories. On the benchmark where we profile construction cost, NEXMEM requires half as many LLM calls as the next-lowest profiled memory system and generates 5–10$\times$ fewer construction output tokens than the two strongest memory baselines.

1 INTRODUCTION

Conversational AI agents are increasingly expected to sustain long-term, complex multi-session interactions with users. This requires recalling relevant facts, preferences, and events from prior conversations to provide personalized and contextually grounded responses. To support this capability, memory systems have been developed to store, organize, and retrieve information extracted from conversation history. These systems range from simple context-window (Zhong et al., 2024) or retrieval-augmented approaches to more structured representations such as knowledge graphs (Rasmussen et al., 2025) and hierarchical summaries (Fang et al., 2025). As conversations accumulate over time, however, the scale and complexity of stored memory grows substantially, making both the construction and retrieval of memory increasingly demanding in terms of computation and cost.

Many existing memory systems incur substantial computational overhead, relying on numerous LLM calls for memory construction and costly search or reasoning at query time. Building practical memory systems requires balancing accuracy with efficiency: agents must recall relevant experiences precisely, without flooding the context window with irrelevant information or incurring prohibitive construction and retrieval costs. Existing approaches fall short in one of two ways: methods using flat, session- or turn-level retrieval struggle to capture fine-grained relationships across long conversation histories (Wang & Chen, 2025; Chhikara et al., 2025), while methods relying purely on pairwise graph relations (Xu et al., 2025b) find it hard to capture the n-ary co-occurrence patterns natural to human dialogue. Moreover, structures that do capture such relationships, such as hierarchical memory organizations, compound this cost further, requiring substantially more LLM calls during construction (Yue et al., 2026). We address these limitations with NEXMEM, a construction-efficient memory framework that targets a strong accuracy–cost trade-off. It extracts entities and phrases from conversational memory and organizes them into two levels of relational granularity through n-ary connections we term *nexus points*[1]: a broad *aboutness* point linking the memory to all its entities and phrases, and *event-fact* points linking it to individual entity–phrase pairs.

[1] *nexus*: a connection linking two or more things.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

The name NEXMEM reflects this design: memory retrieval in our framework centers on identifying and propagating relevance through these nexus points [2]. A nexus point is the single structural primitive we use to represent a memory at both levels of granularity: the same construct can bind a memory's event node to every entity and phrase it mentions at once, a wide, memory-level view, or to just one entity-phrase pair at a time, a narrow, fact-level view, rather than requiring separate graph structures for each granularity. Retrieval then scores and propagates relevance directly over these nexus points, letting evidence at either granularity contribute to ranking memories for a query. Specifically, our contributions in NEXMEM are as follows: (i) a multi-granular conversational-memory index in which each memory is represented by a broad *aboutness* nexus point and fine-grained *event-fact* nexus points; (ii) a query-driven, two-stage retrieval method that converts node-level evidence into a bounded local transition graph over nexus points, then applies type-aware personalized PageRank with multi-round reseeding to rank memories; and (iii) a single-pass multi-granular memory-construction method that jointly creates memory records and their complementary nexus-point views in a single pass, avoiding costly multi-stage construction and continuous restructuring of existing memory; evaluations across three long-term conversational-memory benchmarks show that this efficiency does not compromise accuracy.

We evaluate NEXMEM on three long-term conversational-memory benchmarks: LoCoMo, LongMemEval, and LifeBench. NEXMEM achieves the best average accuracy on all three, with its largest margin on LifeBench, the longest-context benchmark, where it exceeds the strongest evaluated baseline by 28 points. On LoCoMo, where we profile LLM usage in detail, NEXMEM requires 272 memory- construction calls, the fewest among the evaluated systems and half as many as the next-lowest methods. It also generates 5–$10\times$ fewer construction output tokens than the two systems closest to it in accuracy. Some simpler RAG-style baselines use fewer tokens in absolute terms, but trail NEXMEM by 11–22 points in average accuracy. Thus, our efficiency claim concerns the accuracy–construction-cost trade-off rather than minimum token usage in isolation.

2 RELATED WORK

Retrieval augmented generation (RAG) (Gao et al., 2023; Saxena et al., 2025; Edge et al., 2024; Feng et al., 2025) has shown great utility when working with limited resources and context lengths of LLMs. However, managing long-term conversations as memory to support downstream applications of agent-based systems demands greater capabilities than naive RAG systems can provide, hence investments in memory systems (Wang & Chen, 2025; Tan et al., 2025; Zhang et al., 2026b). Initially, MemoryBank (Zhong et al., 2024) proposed to compress rather than merely store by summarizing past interactions into denser records. A-Mem (Xu et al., 2025b) clusters related memories together so retrieval operates over groups rather than individual turns. Similarly, LightMem (Fang et al., 2025) proposed to cluster memories using lightweight annotations and operations so that the memory system is more efficient compared to systems taking long time to process and retrieve memories. Mem0 (Chhikara et al., 2025) pushes further toward explicit fact management, treating memory as a set of atomic claims that can be added, revised, or retracted as new information arrives. However, Zhang et al. (2026a) cautions against aggressive consolidation as memory utility first rises and then degrades as LLM-driven consolidation continues. NEXMEM's one-pass memory construction design follows this evidence. On the other hand, learning-based approaches (Wang et al., 2025; Yan et al., 2026; Zhang et al., 2026c) attempt to tune language models to this complex problem.

A parallel line of works exist creating structured representations (Xu et al., 2026; Sarthi et al., 2024; Rezazadeh et al., 2025) and inter-memory relationships by building a graph (Gutiérrez et al., 2025). HippoRAG (Gutiérrez et al., 2024) constructs a knowledge graph and retrieves via Personalized PageRank seeded from query-matched nodes; GraphRAG (Edge et al., 2024) builds community-level summaries over an entity graph for higher-level queries; Zep (Rasmussen et al., 2025) maintains a temporally-aware knowledge graph designed for continuously updating agent memory; and Mem0-graph (Chhikara et al., 2025) extends Mem0's fact store with graph edges between related facts. These systems recover exactly the cross-memory structure that flat fact stores lack — but a graph edge connects exactly two nodes. Complementing these approaches, more recent works have focused on modeling different levels of representation granularity to support accurate retrieval of stored memories for complex questions. MemGas (Xu et al., 2025a) models memory in different

[2] We plan to release code after internal legal review/approval.

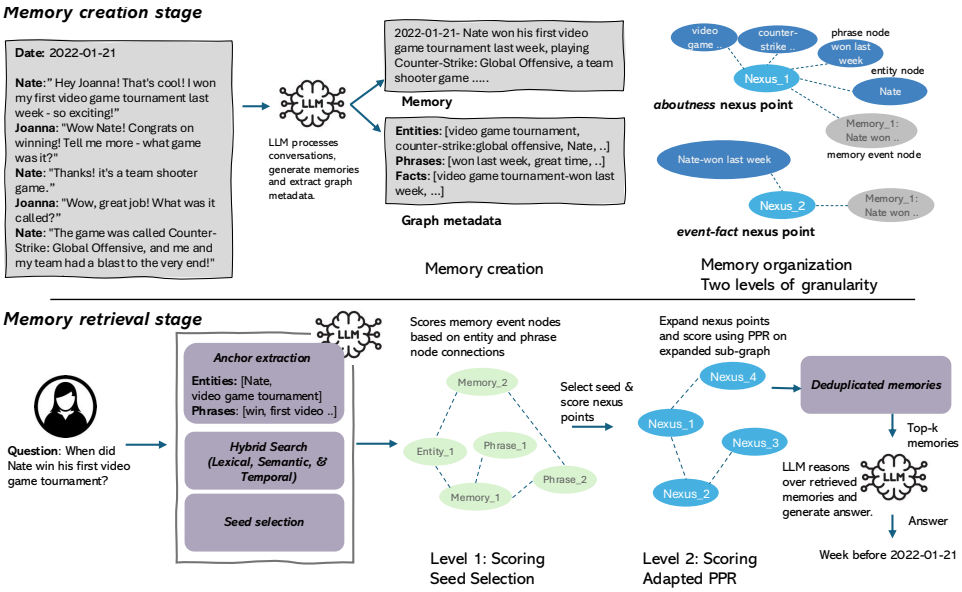


Figure 1: NexMem memory creation and retrieval overview. Extracted memory and graph metadata are organized into nexus points, linked primarily through shared entity/phrase nodes, which also determine link type. Retrieval applies a two-level scoring process adapting PPR.

levels (e.g., keyword, summary, session, etc.) whereas Memora (Xia et al., 2026) addresses this by separating a memory's content from a lightweight indexing layer of cue anchors, so the same underlying fact can be reached through several different access paths without duplicating or restructuring the content itself. Moreover, some approaches model memory in a hierarchy in-order to be able to access it from different perspectives and granularity (Yue et al., 2026). However, maintaining a hierarchy is hard where simple insertion to a hierarchy/tree can invoke many updates to the memory structure, hence can be quite expensive (see our evaluation).

Our approach, NEXMEM, uses a lightweight, non-hierarchical memory index. Each memory yields a broad *aboutness* nexus point over entities and phrases, plus fine-grained *event-fact* nexus points for entity–phrase associations. Shared connectors induce query-specific local neighborhoods without having a global hierarchy. NEXMEM then combines query-driven node scoring with bounded local PPR over the nexus points, enabling single-pass memory construction and multi-granular retrieval.

3 APPROACH

NEXMEM represents memory at two levels of granularity and performs retrieval through a two-level scoring process over this representation. Our central design goal is efficiency: we invoke LLMs only where they are strictly necessary, relying on lightweight metadata extraction at memory-creation time and shifting the bulk of the computational effort to retrieval — while still keeping retrieval itself free of excessive LLM calls. Concretely, NEXMEM requires only a single LLM call per conversation segment for memory creation, and only two LLM calls per query at inference time — one for retrieval (query anchor extraction) and one for answer generation — with all graph scoring and propagation performed without any additional LLM involvement.

At the core of NEXMEM is the *nexus point*: an n-ary link that binds a set of entity and phrase nodes extracted from a memory into a single connective unit. We construct nexus points at two levels of relational granularity. An *aboutness* nexus point binds all entities and phrases extracted from a memory together with the memory itself, capturing its overall topical content, while an *event-fact* nexus point binds the memory with a single entity-phrase pair, modeling one discrete fact. See Figure 1. Entities and phrases serve as index keys into an adjacency structure over nexus points, allowing a query to enter the memory structure through its constituent entities and phrases. Critically, two nexus points are themselves *adjacent* if they share at least one common entity or phrase,

forming a connected network that supports multi-hop retrieval: relevance propagates from query-matched nexus points to adjacent ones through shared entities and phrases, analogous to transferring between connected hubs in a transit network. This adjacency structure underlies our two-level scoring mechanism, though each level uses nexus points differently. In the first level, entity and phrase nodes mediate a one-hop exchange of evidence between memory nodes: a memory node's score can reach other memories sharing an entity or phrase with it, refining an initial ranking at the memory level, computed jointly over memory, entity, and phrase node scores, without yet treating nexus points as retrieval targets themselves. In the second level, nexus points become the unit of propagation directly: seeded from this refined ranking together with direct query-anchor matches, relevance spreads across the adjacency network, expanding beyond the initial candidates.

3.1 Memory creation and graph metadata extraction

Constructing nexus points requires first extracting memories and their entities, phrases, and facts from raw conversational text. Given a raw conversational segment x, a single extraction function Φ (one LLM call) maps it to a set of memories, each paired with its extracted graph elements: $\Phi(x) = \{(m_i, E_{m_i}, P_{m_i}, F_{m_i})\}_{i=1}^{k}$. Here k is determined by the content of x rather than fixed in advance, so a segment may yield one or several memories depending on how many independent facts it contains. Each m_i is a normalized, third-person memory text grounded in x, with entity set E_{m_i}, phrase set P_{m_i}, and entity-phrase fact set F_{m_i}.

For the remainder of this section, we fix a single memory $m = m_i$ and drop the index for readability, writing E_m, P_m, and F_m for its entities, phrases, and facts. Each fact $(e, p) \in F_m$ pairs an entity $e \in E_m$ with a phrase $p \in P_m$ that acts as its predicate, i.e., a phrase strongly associated with that entity. Letting n_m denote the memory (event) node representing m itself, memory m is represented by the node set $V_m = \{n_m\} \cup E_m \cup P_m$.

From these graph elements, we define two nexus-point types that represent memory at two levels of granularity. The ***aboutness*** nexus point binds the memory node to every entity and phrase extracted from the memory, and, for each fact $(e, p) \in F_m$, an ***event-fact*** nexus point binds only the memory node, the entity, and its predicate phrase:

$$h_m^{\text{about}} = V_m = \{n_m\} \cup E_m \cup P_m, \qquad h_{m,(e,p)}^{\text{fact}} = \{n_m, e, p\}.$$

Each memory thus has exactly one *aboutness* nexus point, whose arity $1 + |E_m| + |P_m|$ grows with the number of entities and phrases mentioned, and $|F_m|$ *event-fact* nexus points of arity 3, $H_m^{\text{fact}} = \{h_{m,(e,p)}^{\text{fact}} : (e, p) \in F_m\}$: a single wide link versus many narrow ones. Because *aboutness* captures everything a memory contains rather than a specific fact, it contributes mainly to recall, while *event-fact* links contribute precise, fact-level evidence.

Memory m contributes the nexus-point set $H_m = \{h_m^{\text{about}}\} \cup H_m^{\text{fact}}$, and aggregating over all memories yields the graph $\mathcal{G} = (V, H)$ with $V = \bigcup_m V_m$, $H = \bigcup_m H_m$, and entity and phrase node sets $E = \bigcup_m E_m$ and $P = \bigcup_m P_m$. Two nexus points $h_i \neq h_j$ are adjacent iff they share at least one entity or phrase node, $h_i \cap h_j \cap (E \cup P) \neq \emptyset$.

3.2 Retrieval

Retrieval must be both fast and accurate: slow retrieval limits practical deployment, while inaccurate retrieval degrades response quality. Algorithm 1 overviews retrieval in NexMem. We first extract query anchors from the input query, then score entity, phrase, and memory nodes to rank and select seed nodes (Level 1). These seed nodes then seed the nexus points that contain them, and we perform sub-graph expansion from the seeded nexus points, scoring them via an adapted personalized PageRank (Level 2).

Query anchors are extracted through a single LLM call (ExtractQueryAnchors), returning a set of anchors A_q, entities and phrases, in the same vocabulary used to construct nexus points, together with a temporal filter judgment. This shared vocabulary enables direct token-level matching against nexus points throughout both levels of scoring, as detailed below.

Level 1 — Seed Memory Scoring To identify which memories are most relevant to query q, we first perform *hybrid search*, scoring each memory node $n_m \in V_m$ using a weighted blend of

semantic, lexical, and temporal relevance. Let $\phi(\cdot)$ denote the dense embedding function used to represent both memory-node text and the query in a shared vector space; in our implementation, we use Qwen3-Embedding-8B. Semantic similarity is the cosine similarity between $\phi(n_m)$ and $\phi(q)$, rescaled to $[0, 1]$, combined with BM25 lexical relevance and, when the query is judged temporal by the same LLM call that extracts the query anchors, a proximity term based on the day-distance Δ between the memory's date and the query's resolved temporal pattern, passed through a Gaussian kernel, $\exp\big(-\frac{1}{2}(\Delta/\sigma)^2\big)$. The resulting score is:

$$s(n_m) = \alpha \cdot \text{sem}(n_m, q) + \eta \cdot \text{bm25}(n_m, q) + \theta \cdot \text{temp}(n_m, q),$$

where $\alpha + \eta = 1$ balance semantic against lexical evidence (0.8 and 0.2), and $\theta = 0.5$ scales temporal proximity, which enters as an additive bonus so that date evidence can lift a memory.

Starting from these hybrid-search scores, we retain a small set of top-scoring candidates as initial seeds for Level 1 graph scoring, which uses entity and phrase nodes as bridges between memories. Our ultimate goal is to score and rank multi-granular memory representations, the nexus points, over a bounded sub-graph in order to keep retrieval latency low, which requires first selecting a small set of well-suited seed nexus points from which that sub-graph can be expanded. Level 1 serves as this bootstrapping step, scoring only memory, entity, and phrase nodes rather than nexus points directly: because a single memory participates in multiple nexus points, one *aboutness* link and several *event-fact* links, scoring at that granularity would duplicate the same memory's evidence across several nexus points and score it inconsistently depending on which one was considered.

Before any propagation, memory-event nodes receive a bounded anchor-based boost from their overlap with the query anchors A_q (APPLYANCHORBOOST). Let k_a denote the anchor evidence reaching n_m through its nexus points, where each matching anchor is weighted by its inverse document frequency across memories, so rare anchors count for more than hub entities such as the user's own name. This evidence saturates and is capped at c: $\text{anchor}(n_m) = \min(c,\, 1 - e^{-k_a})$, so a few strong anchors suffice while further matches add progressively less. It is merged as a floor, $\tilde{s}(n_m) = \max(s(n_m),\, \text{anchor}(n_m))$, raising a memory's score only when the anchor evidence exceeds it, and bringing anchor-connected memories that missed the initial seeds into the candidate pool. Entity and phrase nodes receive no direct score, so $\tilde{s}(v) = 0$ for them until propagation. We set $c = 0.90$.

Algorithm 1 NexMem Query-Driven Retrieval

Require: query q, graph $G = (V, H)$ with memory-event nodes V_m, entity/phrase nodes $V_{ep} := E \cup P$ ($V = V_m \cup V_{ep}$), and nexus points H
Ensure: top-k retrieved memory nodes $M^* \subset V_m$
$A_q \leftarrow$ EXTRACTQUERYANCHORS(q)
Level 1
$s(\cdot) \leftarrow$ HYBRIDSEARCH(V_m, q)
$\tilde{s}(\cdot) \leftarrow$ APPLYANCHORBOOST($s(\cdot), A_q$)
$s'(\cdot) \leftarrow$ PROPAGATEVIANEXUSPOINTS($\tilde{s}(\cdot)$) {via incident nexus points}
Level 2
$S \leftarrow$ scored nodes from $s'(\cdot)$ over $V_m \cup V_{ep}$
for $h \in H$ **do**
  $\text{score}_1(h) \leftarrow$ SEEDSCORE(h, S, A_q)
end for
$H_0 \leftarrow$ top-scoring nexus points by $\text{score}_1(\cdot)$
$\hat{H} \leftarrow$ BUILDNEXUSTRANSITIONGRAPH(H_0)
$\pi(\cdot) \leftarrow$ LOCALPUSHPPR($\hat{H}, \text{score}_1|_{H_0}$)
for $n_m \in V_m$ **do**
  $s^*(n_m) \leftarrow \max_{h\,:\,n_m \in N(h)} \pi(h)$, normalized over V_m
  $\text{final}(n_m) \leftarrow$ FUSESCORES($s(n_m), s^*(n_m)$)
end for
$M^* \leftarrow$ top-k memory nodes ranked by final($\cdot$)
return M^*

Entity and phrase nodes still play a central role in Level 1, just not as scored retrieval targets: since they are shared across memories, they let evidence propagate from one memory to related memories through the common nodes (PROPAGATEVIANEXUSPOINTS). Concretely, evidence from scored nodes accumulates at each nexus point they belong to, weighted by a node-type weight $w(\cdot)$ that is highest for memory nodes and discounted for entity and phrase nodes. The accumulated evidence is normalized by arity and weighted by a nexus-point type prior $\tau(h)$ reflecting how directly each nexus-point type encodes event evidence, where $N(h)$ denotes the set of nodes bound by nexus point h:

$$\text{ev}(h) = \frac{\tau(h)}{\sqrt{|N(h)|}} \sum_{v \in N(h)} \tilde{s}(v)\, w(v)$$

Only the nexus points with the highest $\mathrm{ev}(h)$ then redistribute their score back to the nodes they touch, again weighted by $w(\cdot)$; we write $\mathrm{prop}(v)$ for the evidence node v receives. Propagation is therefore selective: a shared entity or phrase transfers evidence between two memories only when it sits inside a well-supported nexus point, not through every shared node indiscriminately. Each node merges this with its direct score, $s'(v) = \tilde{s}(v) + \beta_1 \,\mathrm{prop}(v)$ when $\tilde{s}(v) > 0$ and $s'(v) = \beta_2 \,\mathrm{prop}(v)$ otherwise. Since $\beta_1 < \beta_2 < 1$, direct evidence stays dominant, while the second case admits nodes carrying no direct score: entity and phrase nodes, and memories that missed both the initial seeds and the anchor boost.

Nexus points participate in this process only as an internal propagation medium; in Level 2 they themselves become scored, seeded, and expanded retrieval units. The scored nodes form the seed set $S \subseteq V_m \cup E \cup P$ that bootstraps Level 2. Bounding the candidate set before propagation keeps S small, so Level 2 expands a sub-graph whose size does not grow with the memory store.

Level 2 — Nexus-Point Scoring Level 1 produces the seed set S and node scores $s'(v)$ described above. Level 2 uses these, together with query anchor tokens A_q, to score the nexus points themselves, the retrieval units Level 1 deliberately left unscored, and propagate that evidence across the graph. Seeding from Level 1's bootstrapped nodes and expanding outward through the nexus-point structure lets us discover additional relevant memories beyond the initial candidates, while nexus points' differing granularity, wide *aboutness* links versus narrow *event-fact* links, lets us balance precision and recall over the resulting candidate set.

Each nexus point $h \in H$ first receives an initial score (SEEDSCORE in Algorithm 1) from two overlap terms computed over its node set $N(h)$. Seed overlap $\mathrm{ov}_s(h)$ is the mean Level 1 score of its nodes in S, averaged rather than summed so that nexus points with many weakly scored overlaps are not favored over those with a few strongly scored ones. It treats memory, entity, and phrase nodes identically, so a nexus point can score highly because its memory node was strongly matched in Level 1, because an entity or phrase it shares with other memories was, or both. Anchor overlap $\mathrm{ov}_a(h)$ is the fraction of query anchors covered by the anchor tokens $A(h)$ of its nodes:

$$\mathrm{ov}_s(h) = \frac{1}{|N(h) \cap S|} \sum_{v \in N(h) \cap S} s'(v), \qquad \mathrm{ov}_a(h) = \frac{|A(h) \cap A_q|}{\max(|A_q|, 1)}.$$

The two terms are combined with a nexus-point type prior $\tau(h)$ reflecting how directly each nexus-point type encodes event evidence, ranking *event-fact* above *aboutness* so that precise matches outweigh broader, noisier ones. The result is normalized by the square root of arity so that large nexus points are not favored by size alone, with $\lambda > \frac{1}{2}$ weighting seed overlap above anchor overlap. As a result, a nexus point can be seeded without its own memory node matching the query, provided one of the entity or phrase nodes it binds carries a score, whether from a query anchor or from evidence propagated out of a matched memory.

$$\mathrm{score}_1(h) = \frac{\tau(h) \cdot \big(\lambda \cdot \mathrm{ov}_s(h) + (1 - \lambda) \cdot \mathrm{ov}_a(h)\big)}{\sqrt{|N(h)|}}$$

Rather than exploring the underlying graph directly, we first select the top-scoring nexus points H_0 by $\mathrm{score}_1(\cdot)$ as seeds, then build a second, derived graph $\hat{H}$ (BUILDNEXUSTRANSITIONGRAPH) by expanding locally outward from H_0, up to K hops and capped at C candidate nexus points. In $\hat{H}$, two nexus points h, h' are connected whenever they share one or more connector nodes (type entity or phrase), and each shared connector contributes its own transition weight:

$$w(h \to h') = \tau(h)\,\tau(h') \cdot \sum_{v \in N(h) \cap N(h')} w^{\mathrm{conn}}(v)$$

This reflects how directly each nexus-point type carries event evidence and how strongly the shared connector type ties them together. We set $w^{\mathrm{conn}}_{\mathrm{entity}} > w^{\mathrm{conn}}_{\mathrm{phrase}}$, since phrases can be fuzzier or more generic in what they refer to. Relevance then spreads over $\hat{H}$ via an ϵ-approximate personalized PageRank (LOCALPUSHPPR), computed with the standard local-push procedure: each seed nexus

point's residual mass ($r(h)$), initialized from $\mathrm{score}_1(h)$ for $h \in H_0$, is repeatedly split into a retained fraction ($\pi(h)$) and a pushed fraction until residual mass falls below a threshold everywhere,

$$\pi(h) \mathrel{+}= \gamma \cdot r(h),\ r(h') \mathrel{+}= (1-\gamma)\, r(h)\, \frac{w(h \to h')}{\sum_{h''} w(h \to h'')}$$

This is repeated for R rounds: after each round, the highest-scoring nexus points reseed the next round, with each round's contribution decayed by δ so that direct, early-round evidence remains dominant over evidence accumulated from farther exploration.

Each memory node n_m inherits the highest PPR mass among the nexus points containing it, normalized by the highest such mass over all memory nodes; we denote this normalized score $s^*(n_m)$. Finally, $s^*(n_m)$ is fused with the original hybrid score $s(n_m)$ from the start of Level 1, before the anchor boost and propagation, so that graph-based evidence adjusts rather than replaces the initial relevance signal (FUSESCORES). Retrieved memories are ranked by $\mathrm{final}(\cdot)$.

$$\mathrm{final}(n_m) = s(n_m) + \beta_{\text{graph}} \cdot s^*(n_m)$$

Implementation. Relevance propagation uses restart probability $\gamma = 0.15$ and push threshold $\epsilon = 10^{-4}$, initialized from the top $|H_0| = 250$ nexus points. The local neighborhood uses two-hop connector expansion, capped at 500 candidate nexus points and adaptively expanded to at most 750 when too few memories are retrieved. PPR is run for $R = 2$ rounds with round-decay factor $\delta = 0.85$. The graph-fusion weight is $\beta_{\text{graph}} = 0.3$. Nexus-point type priors are $\tau(\textit{event-fact}) = 1.0$ and $\tau(\textit{aboutness}) = 0.9$; connector-type priors are $w^{\text{conn}}_{\text{entity}} = 0.9$ and $w^{\text{conn}}_{\text{phrase}} = 0.8$. We selected these values once based on their intended semantic roles (e.g., assigning a higher prior to entity connectors than phrase connectors) and sanity-checked them on a small internal development dataset. No parameters were tuned on, or separately selected for, any reported benchmark; all settings were fixed across benchmarks. Full hyperparameter and prompt details are provided in the appendix. We used an 8-GPU node with NVIDIA A100 GPUs (80 GB each) to host the Qwen models.

4 EVALUATION AND DISCUSSION

Datasets We evaluate NEXMEM against comparable baselines on three benchmarks: LoCoMo (Maharana et al., 2024), LongMemEval (Wu et al., 2024), and LifeBench (Cheng et al., 2026). LoCoMo comprises multi-turn dialogues averaging 600 turns (∼20k tokens), with question-answer pairs spanning single-hop, multi-hop, temporal, and open-domain (selected similarly as in HyperMem). We use the LongMemEval_S split (115k context length), following recent work (Xia et al., 2026). LifeBench simulates a year of user activity across multiple sources (SMS, email, chat, calendar), averaging 360 sessions per user, with 312k–1.83M tokens per user and 13.6M tokens in total, reflecting the scale and heterogeneity of real-world usage.

Baselines We select multiple comparable memory baselines: HippoRAG 2 (Gutiérrez et al., 2025) and LightMem (Fang et al., 2025) focus more on efficient management of memory where Memora (Xia et al., 2026) and MemGAS (Xu et al., 2025a) are recent approaches that attempt at modeling memory in multi-level granularity. HyperGraphRAG (Luo et al., 2025) models multi-link modeling and HyperMem (Yue et al., 2026) focuses on hierarchical and graph level modeling. We also present naive RAG as a simpler baseline. We use Qwen 3.5-27B large language model as both the memory agent and the judge and the same judge prompt for all methods.

Accuracy Table 1 presents evaluation results for all baselines on LoCoMo. Across all three benchmarks, NEXMEM achieves the best average accuracy despite its lightweight memory organization, which we attribute mainly to coupling a lightweight memory representation with a two-level multi-granular retrieval method. On LoCoMo it ranks first on single-hop, temporal, and average accuracy, exceeding even the full-context upper bound on average and leading the strongest memory baseline, HyperMem, by 3.76 points. Among memory systems it is also strongest on multi-hop, ahead of Memora by 3.90 points, with only full context scoring higher. Open-domain and Multi-hop, NEXMEM places second.

Table 2a and Table 2b present evaluation results on LongMemEval and LifeBench, respectively; for both, we selected the best-performing memory systems from the LoCoMo results as baselines. HyperMem is omitted from LifeBench because its repeated, sequential LLM-based memory updates

Method	Single-hop	Multi-hop	Temporal	Open-domain	Average
Full Context	92.15	**83.33**	77.88	51.04	85.00
RAG	60.40	38.65	34.89	26.04	48.96
HippoRAG 2	77.52	42.90	53.58	48.95	64.41
LightMem	82.64	69.50	73.21	45.83	75.97
MemGAS	70.92	71.70	74.45	**68.75**	71.66
Memora	89.06	74.82	77.26	52.08	81.69
HyperGraphRAG	88.35	60.64	44.24	45.83	71.43
HyperMem	90.01	69.86	82.55	59.38	82.86
NexMem	**92.63**	78.72	**85.67**	60.42	**86.62**

Table 1: LoCoMo results (LLM-as-judge) with Qwen3.5-27B as memory agent and judge. **Bold** = best, underline = second best. NEXMEM achieves the best average score.

Type	MemGAS	LgtMem	Memora	HMem	HGRAG	NEXMEM
s-sn-pref.	70.00	63.33	86.67	**93.33**	83.33	86.67
s-sn-asst.	94.64	17.86	87.50	87.50	**96.43**	94.64
temp-reasn.	73.68	69.17	**89.47**	69.92	45.86	73.68
m-session	56.39	80.45	78.20	78.20	68.42	**90.98**
k-update	78.21	79.49	93.59	92.31	61.54	**96.15**
s-sn-user	77.14	91.43	95.71	**98.57**	92.86	97.14
Average	72.40	70.80	87.60	83.00	68.80	**88.20**

(a) LongMemEval

Type	MemGAS	LgtMem	Memora	HGRAG	NEXMEM
IE	52.37	43.04	28.97	57.10	**88.30**
multi-hop	29.31	37.52	24.96	49.41	**76.05**
temp-know.	48.25	40.33	28.90	44.76	**64.19**
nondec.	16.16	38.86	23.14	45.41	**79.49**
unansw.	83.33	86.67	**90.00**	73.33	86.67
Average	40.94	40.99	28.01	51.07	**79.98**

(b) LifeBench

Table 2: Results on (a) LongMemEval and (b) LifeBench. Accuracy is measured with an LLM-as-a-judge protocol. Qwen3.5-27B serves as both the memory agent and the judge. **Boldface** marks the best result and underline the second-best result in each row. LgtMem, HMem, and HGRAG represents LightMem, HyperMem, and HyperGraphRAG, respectively.

projected weeks of additional processing (approx. 28 days), whereas other baselines completed within 2–5 days. NEXMEM leads on average on both datasets, and on four of the five LifeBench categories. On LongMemEval its margin comes from multi-session and knowledge-update questions, where it leads the next-best system by 10.5 and 2.6 points. LifeBench appears especially challenging overall: existing methods such as LightMem and MemGAS reach only about 40% average score, nearly 40 points behind NEXMEM, while Memora performs substantially worse. This difficulty likely stems from LifeBench's much longer context, spanning up to one year and around 1.8M tokens (13.2M total), as well as its complex multi-hop and multi-intent questions, which require combining several related memories to produce a complete answer.

Efficiency Table 3a reports LLM usage for memory creation and retrieval. NEXMEM constructs its memory in a single pass, without iterative refinement or multi-stage reprocessing, requiring 272 calls in total: the fewest of any system evaluated and half the next-lowest, HippoRAG 2 and MemGAS at 544. Baselines that revisit or refine memory across stages pay for it directly, with HyperMem and Memora, the two strongest competitors on LoCoMo, using $53\times$ and $30\times$ more calls. Figure 2 plots accuracy against both LLM calls and token usage. On tokens, NEXMEM sits mid-pack rather than lowest, because it generates memory text rather than indexing or lightly annotating retrieved chunks as the RAG-style baselines do. That position is the informative one: every system that decodes fewer tokens than NEXMEM trails it by 11 to 22 points in average accuracy, while the two systems closest to it in accuracy decode far more, $5.3\times$ for HyperMem and $10\times$ for Memora. No evaluated baseline reaches higher accuracy at lower cost on any measured axis.

Token counts follow the same pattern against the systems nearest in accuracy. NEXMEM consumes 1.3M input and 1.3M output tokens during construction, $4.8\times$ and $10\times$ below Memora and $26\times$ and $5.3\times$ below HyperMem. Every system that decodes fewer tokens than NEXMEM trails it by 11 to 22 points in average accuracy; the cheapest among them, MemGAS and HippoRAG 2, index or lightly annotate retrieved chunks rather than generating structured memory. At retrieval time NEXMEM uses two calls and 2.92K input tokens per query, four to six times fewer tokens than the graph-based systems MemGAS, HyperGraphRAG, and Memora.

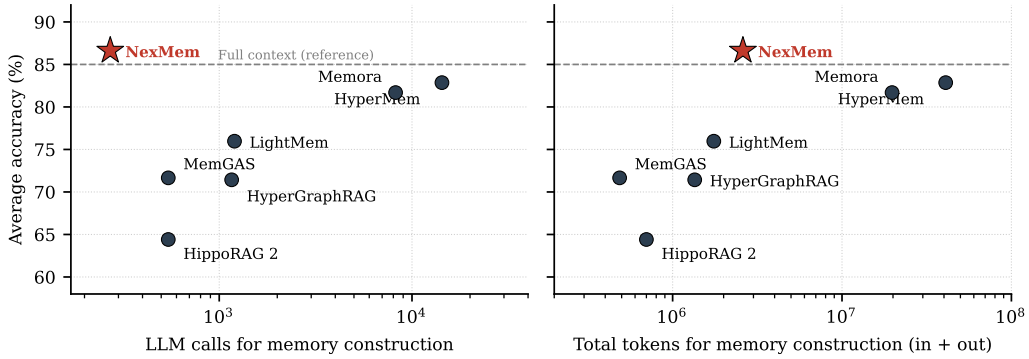


Figure 2: Accuracy vs. memory-construction cost on LoCoMo. No baseline is both cheaper and more accurate than NexMem on either axis.

	Creation (total)			Retrieval	
Method	#Calls	In tok.	Out tok.	#Calls	In tok.
HippoRAG 2	544	0.61M	0.09M	2	2.8K
LightMem	1.2K	1.19M	0.56M	1	2.6K
MemGAS	544	0.45M	36.52K	2	13.61K
Memora	8.22K	6.22M	13.53M	5.14	16.83K
HGRAG	1.16K	0.22M	1.13M	2	13.63K
HyperMem	14.3K	33.98M	6.95M	1	3.01K
NEXMEM	272	1.30M	1.30M	2	2.92K

(a) Memory creation and retrieval cost on LoCoMo

Method	Accuracy
NEXMEM	**86.62**
(-) *event-fact* nexus points	85.97
(-) *aboutness* nexus points	84.48
(-) level 1 scoring	85.78
(-) level 2 scoring	85.71

(b) LoCoMo ablation.

Table 3: Efficiency and ablation on LoCoMo. (a) Total LLM calls and input/output tokens for memory creation over the full corpus, and per-question LLM calls and input tokens at retrieval (including answer generation). HGRAG denotes HyperGraphRAG. (b) Accuracy when removing *event-fact* or *aboutness* nexus points, or the first (level 1) or second (level 2) retrieval scoring level.

Ablation Table 3b reports our LoCoMo ablation study. Removing *aboutness* nexus points produces the largest decrease in accuracy, from 86.62 to 84.48, while removing *event-fact* nexus points lowers it to 85.97. Both granularities therefore contribute to the model's performance. Removing Level 1 or Level 2 scoring reduces average accuracy to 85.78 and 85.71, respectively, suggesting that initial node-level scoring and subsequent nexus-point propagation provide complementary benefits.

5 CONCLUSION

We presented NEXMEM, an efficient conversational memory agent that represents memory at multiple levels of granularity through nexus points and retrieves through a two-level scoring mechanism: an initial node-level bootstrapping stage followed by an adaptation of personalized PageRank over typed, multi-way relations rather than pairwise entities. Across LoCoMo, LongMemEval, and LifeBench, NEXMEM achieves the best average accuracy among existing memory systems, with its largest margin on LifeBench, the longest-context and most complex benchmark. On LoCoMo it does so while constructing memory in a single pass with the fewest LLM calls of any system evaluated, over 30× fewer than Memora, and no evaluated baseline reaches higher accuracy at lower cost. Ablations confirm that both scoring levels and both nexus-point granularities contribute to this performance. These results show that accurate, scalable long-term conversational memory does not require expensive, LLM-heavy memory operations.

AI USE STATEMENT

We used generative AI tools for language editing of the manuscript and for debugging code in this work. We did not use generative AI tools to develop the conceptual framework, formulate mathematical claims, propose or refine hypotheses, design experiments, generate or clean datasets, conduct qualitative analysis, or interpret results. All AI-assisted text was reviewed and revised by the authors.

ETHICS STATEMENT

The authors are not aware of any ethical concerns associated with this work.

REPRODUCIBILITY STATEMENT

We describe the proposed method and its design choices in Approach and Evaluation Sections. Evaluation Section details the experimental setup, including datasets, evaluation metrics, baselines, and implementation settings. The appendix provides additional implementation details, and hyperparameters used across all experiments. We clearly explained in the Approach section how we represent memories and how our two-level retrieval process works. PPR we used in our approach is a standard algorithm that we adapted to work with our retrieval mechanism. The implementation is currently undergoing internal legal review; we intend to release the code publicly once the required approval is obtained.

REFERENCES

Zihao Cheng, Weixin Wang, Yu Zhao, Ziyang Ren, Jiaxuan Chen, Ruiyang Xu, Shuai Huang, Yang Chen, Guowei Li, Mengshi Wang, et al. Lifebench: A benchmark for long-horizon multi-source memory. *arXiv preprint arXiv:2603.03781*, 2026.

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, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, et al. Lightmem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*, 2025.

Yifan Feng, Hao Hu, Xingliang Hou, Shiquan Liu, Shihui Ying, Shaoyi Du, Han Hu, and Yue Gao. Hyper-rag: Combating llm hallucinations using hypergraph-driven retrieval-augmented generation. *arXiv preprint arXiv:2504.08758*, 2025.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

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

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From rag to memory: Non-parametric continual learning for large language models. In *International Conference on Machine Learning*, pp. 21497–21515. PMLR, 2025.

Haoran Luo, Haihong E, Guanting Chen, Yandan Zheng, Xiaobao Wu, Yikai Guo, Qika Lin, Yu Feng, Zemin Kuang, Meina Song, Yifan Zhu, and Anh Tuan Luu. Hypergraphrag: Retrieval-augmented generation via hypergraph-structured knowledge representation. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi,

and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 152206–152234. Curran Associates, Inc., 2025. doi: 10.52202/085713-5089. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/df55ee6e59f8ac4a625219e11fe9ddba-Paper-Conference.pdf.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 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)*, pp. 13851–13870, 2024.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. In *International Conference on Learning Representations*, volume 2025, pp. 990–1023, 2025.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations*, volume 2024, pp. 32628–32649, 2024.

Yash Saxena, Ankur Padia, Mandar S Chaudhary, Kalpa Gunaratna, Srinivasan Parthasarathy, and Manas Gaur. Ranking free rag: Replacing re-ranking with selection in rag for sensitive domains. *arXiv preprint arXiv:2505.16014*, 2025.

Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, et al. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8416–8439, 2025.

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

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-{\alpha}: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025.

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

Menglin Xia, Xuchao Zhang, Shantanu Dixit, Paramaguru Harimurugan, Rujia Wang, Victor Rühle, Robert Sim, Chetan Bansal, and Saravan Rajmohan. Memora: A harmonic memory representation balancing abstraction and specificity. In *Forty-third International Conference on Machine Learning*, 2026.

Buqiang Xu, Yijun Chen, Jizhan Fang, Ruobin Zhong, Yunzhi Yao, Yuqi Zhu, Lun Du, and Shumin Deng. Structmem: Structured memory for long-horizon behavior in llms. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 122–146, 2026.

Derong Xu, Yi Wen, Pengyue Jia, Yingyi Zhang, Yichao Wang, Huifeng Guo, Ruiming Tang, Xiangyu Zhao, Enhong Chen, Tong Xu, et al. From single to multi-granularity: Toward long-term memory association and selection of conversational agents. *arXiv preprint arXiv:2505.19549*, 2025a.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 17577–17604. Curran Associates, Inc., 2025b. doi: 10.52202/085713-0593. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/19909c36f51abc4856b4560aff3d36d6-Paper-Conference.pdf.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026.

Juwei Yue, Chuanrui Hu, Jiawei Sheng, Zuyi Zhou, Wenyuan Zhang, Tingwen Liu, Li Guo, and Yafeng Deng. Hypermem: Hypergraph memory for long-term conversations. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 35237–35254, 2026.

Dylan Zhang, Yanshan Lin, Zhengkun Wu, Yihang Sun, Bingxuan Li, Dianqi Li, and Hao Peng. Useful memories become faulty when continuously updated by LLMs. *arXiv preprint arXiv:2605.12978*, 2026a.

Jiaquan Zhang, Chaoning Zhang, Shuxu Chen, Zhenzhen Huang, Pengcheng Zheng, Zhicheng Wang, Ping Guo, Fan Mo, Sung-Ho Bae, Jie Zou, et al. Lightweight llm agent memory with small language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12914–12929, 2026b.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026c.

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

A Appendix

A.1 Memory construction without iterative refinement

NexMem constructs memory records and their indexing metadata in a single pass, without repeatedly rewriting existing memories. This design avoids additional LLM-based refinement operations. Related work by Zhang et al. (2026a) shows that repeated LLM-based consolidation can degrade memory utility in agentic tasks, highlighting that refinement does not invariably improve memory quality. Our evaluation provides complementary evidence in conversational memory: NexMem achieves the highest average accuracy on all three benchmarks while using fewer construction calls than the other profiled memory systems. These results suggest that effective memory organization and retrieval need not depend on compute-intensive construction.

A.2 Model Selection

We used Qwen3.5-27B as a practical choice under our computational and API budget constraints. Preliminary experiments with GPT-5 suggested that Qwen3.5-27B provided sufficient accuracy for our setting; these exploratory comparisons were not intended as a systematic evaluation of the two models. For LifeBench, NexMem achieved accuracy of around 81% using GPT-5 series models.

A.3 Hyperparameters

Hyperparameters used in PPR and our algorithm are listed in Table 4. In our experiments, we tested all baselines and our approach with similar retrieval content length that go into answer generation. Sometimes, it is difficult to select similar context length because of the nature of memory representation in approaches. For example, Memora retrieval context length is much larger than our method even with smaller retrieval count target (see Table 3a). The answer judge prompt is set to the same across all experiments. Also, we report LLM judge scores in evaluation tables to be fair for all the systems. One main reason is that for example for MCQ questions, answer generation sometimes generates answers together with the choices making low overlap with the ground truth, making F1 score for example artificially low.

Symbol	Description	Value
Shared		
$\tau(\textit{event-fact})$	Nexus-point type prior	1.00
$\tau(\textit{aboutness})$	Nexus-point type prior	0.90
Level 1 — Seed Memory Scoring		
α	Semantic blend weight in $s(n_m)$	0.80
η	Lexical blend weight in $s(n_m)$	0.20
θ	Temporal blend weight in $s(n_m)$	0.50
σ	Gaussian temporal kernel width	3 days
c	Anchor boost cap	0.90
w_{mem}	Node-type weight, memory-event	1.00
w_{ent}	Node-type weight, entity	0.80
w_{phr}	Node-type weight, phrase	0.70
β_1	Propagation merge weight, $\tilde{s}(v) > 0$	0.30
β_2	Propagation merge weight, $\tilde{s}(v) = 0$	0.60
—	Seed node cap before propagation	130
Level 2 — Nexus-Point Seeding and PPR Propagation		
λ	Seed- vs. anchor-overlap blend in $\text{score}_1(h)$	0.65
$w^{\text{conn}}_{\text{entity}}$	Connector-type weight, entity	0.90
$w^{\text{conn}}_{\text{phrase}}$	Connector-type weight, phrase	0.80
$\lvert H_0 \rvert$	Seed nexus points	250
γ	PPR restart probability	0.15
ϵ	PPR push threshold	1×10^{-4}
R	PPR rounds	2
δ	Round score decay	0.85
K	Local candidate neighborhood radius	2 hop
C	Candidate nexus points per round	500 (adaptive to 750)
—	Reseed count after each round	100
β_{graph}	Fusion weight on graph score $s^*(n_m)$	0.30

Table 4: NEXMEM parameter values, fixed across all benchmarks.

A.4 MEMORY EXTRACTION PROMPT

The following prompt performs single-pass memory creation and graph metadata extraction, producing memories and their associated entities, phrases, and event-facts in one LLM call per conversation segment. The reference context in the prompt is used when chunking input conversation session into smaller chunks for ease of processing or very large conversations that go beyond LLM's input context length. 'characters_json' is the label/name of the character with conversations.

```
Task: Create no-omission linked event memories and graph sidecar metadata for each memory for
     two conversation characters in one pass.

You will receive a source manifest containing raw utterances. Each source has a stable "
     source_ref", a "speaker" tag, and date/session metadata. Create event memories only from
     information grounded in those sources, and attach the relevant source_ref values to every
      memory you create. Entities, phrases, and time grounded in each memory have to be
     extracted for graph sidecar.

Your priority is coverage: do not lose substantive facts from the source manifest. A good
     output may contain more memories than a terse summary if that is needed to preserve
     details.

Input:

- Characters:
{characters_json}

- Reference Context From Earlier In The Same Session (not citeable):
Use this only to resolve pronouns, names, relationships, and missing context for the current
     source manifest. Do not create standalone memories and graph data from this reference
     context, and do not cite it.
{reference_context}
```

```
- Source Manifest JSON:
{source_manifest}

Ownership and linking rules:
1. For each raw source, use its speaker tag as the primary owner. If the source is spoken by
     speaker_a, create memories for speaker_a. If it is spoken by speaker_b or assistant,
     create memories for speaker_b.
2. Preserve facts about other people inside the memory text when relevant, but do not
     automatically file the memory under a mentioned person unless that person is the speaker
     or directly addressed by the source.
3. If a source directly addresses the other target character and states a concrete fact, event
     , plan, preference, obligation, request, recommendation, or state about that addressed
     character, also create a memory for the addressed character when it would help future
     recall.
4. For every event memory, include one or more source_refs copied exactly from the source
     manifest. These source_refs are the only required links.
5. Do not invent memory IDs, link IDs, source refs, dates, places, names, or relationship
     labels. When two character memories are derived from the same raw utterance or adjacent
     exchange, both memories may cite the same source_refs.

No-omission extraction rules:
1. Treat the Source Manifest JSON as a checklist. For every source, silently decide whether it
      contains substantive information or is only a pure greeting, acknowledgment, empty
     filler, or exact duplicate.
2. Every substantive source must be represented by at least one event memory for the
     appropriate character, unless it is fully covered by existing events.
3. A source is substantive if it contains any concrete activity, interaction, observation,
     decision, message, recommendation, request, answer, purchase, call, note, appointment,
     work task, health event, financial event, family matter, social event, travel detail,
     preference, location, deadline, plan, obligation, status, explanation, concern, or
     outcome.
4. If one source contains multiple independent facts, preserve all of them. You may either
     split them into separate event memories or write one memory that explicitly includes
     every important fact.
5. If adjacent sources form one coherent exchange, you may combine them into one memory, but
     the memory must preserve the important contribution of each cited source. Do not cite a
     source if none of its substantive details appear in the memory text.
6. Preserve exact names, relationships, dates, times, date ranges, relative time phrases,
     locations, apps/platforms/channels, amounts, counts, product names, addresses, document/
     file names, symptoms, diagnoses, medications, tasks, promises, recommendations, and next
     steps when they appear in the source.
7. Preserve negative facts and constraints when stated: cancellations, declined invitations,
     failed attempts, uncertainty, lack of time, budget limits, health limits, missing
     information, or things not yet completed.
8. If the character mentions something that happened before the current session, record it as
     a memory of that mention, for example "X mentioned that ...", while preserving the
     original timing details if available.
9. Do not compress away answer-critical details. Prefer a longer grounded memory over a
     shorter memory that omits names, dates, places, quantities, reasons, or outcomes.
10. Do not create a memory from pure greetings, acknowledgments, empty logistical filler, or
     exact duplicates already fully covered by existing events.

Memory wording rules:
1. Use third-person wording.
2. Keep each memory atomic enough for retrieval: one event, exchange, decision, or tightly
     related bundle per item.
3. Include the grounded details needed to answer who, what, when, where, why, how, and outcome
     /next step when the source supports them. Do not force a detail that is absent from the
     source.
4. Use the event item's "date" field for the source/session mention date, preferably the
     source entry's session_date in YYYY-MM-DD form. If the event happened at a different time
     , keep that event time inside the memory text.
5. Whenever possible, use exact names and concrete references instead of vague pronouns.
6. Preserve exact quotes only when they are important names, commitments, preferences,
     messages, explanations, or distinctive wording. Do not copy whole conversation sentences
     just to create a memory.
7. Maintain chronological order within each character's events as much as the source manifest
     allows.

Graph sidecar rules:
1. For each memory item, include a "graph" object with structured candidates extracted from
     the memory.
2. "graph.entities": extract important real-world entities only (people, places, organizations
     , products, artifacts, events, institutions).
3. "graph.phrases": extract concise key phrases (1-5 words) that can help retrieve this memory
     . Keep phrases grounded in the memory and avoid vague generic phrases.
4. Do not include just stop words, generic terms, vague pronouns, filler words, or meta words
     as entities or phrases (for example: mentioned, you, I, session, day, on, at, thing,
     stuff).
```

```
5. "graph.event_time": the date/time when the memory event happened, normalized to YYYY-MM-DD
     when possible. If no explicit event time is given and the event is happening in the
     current source/session context, use the source session date.
6. "graph.mentioned_times": explicit dates/times mentioned in memory text that are not the
     event happening time, normalized to YYYY-MM-DD when possible. Do not duplicate event_time
      here.
7. "graph.event_facts": precise entity-phrase pairs that are truly related in this event.
8. In each event_fact:
   - "entity": one entity from graph.entities
   - "phrase": one phrase from graph.phrases
   - optional "event_time"
   - optional "mentioned_times"
9. Do not pair unrelated entities and phrases. If a phrase has no strong entity association,
     you may omit that pair.
10. Keep all graph fields grounded in the created memories; do not invent entities, times, or
     facts unsupported by memory text.

Self-check before output:
1. Verify that every substantive source_ref appears in at least one event's source_refs.
2. Verify that every cited source_ref contributes visible information to the memory text.
3. Verify that no memory or graph field contains unsupported facts from reference memory
     context or prior knowledge.
4. Verify that every input character appears exactly once in the output "characters" array,
     even if that character has no events.
5. Verify that graph.entities and graph.phrases do not contain generic filler tokens or stop
     words and are always grounded in the memory.

General output instructions:
1. Output valid JSON only. Do not include markdown fences, prose, comments, or a separate
     coverage report.
2. Create a "characters" array with one object for each input character.
3. Each character object must contain "character" and "events".
4. Use the exact character names from the Characters input in the "character" fields. Do not
     output placeholder names such as "speaker_a_name" or "speaker_b_name".
5. Each event item must contain exactly "date", "memory", "source_refs", and "graph".
6. Do not output memories that are unsupported by cited source_refs.

Required JSON shape:
{{
  "characters": [
    {{
      "character": "speaker_a_name",
      "events": [
        {{
          "date": "YYYY-MM-DD",
          "memory": "Specific dated event involving speaker_a_name, preserving all grounded
     answer-critical details from the cited source_refs.",
          "source_refs": ["session_1|2025-01-01|example_dia_id"],
          "graph": {{
            "entities": ["Person A", "Location B"],
            "phrases": ["morning workout session", "booked concert tickets"],
            "event_time": "YYYY-MM-DD",
            "mentioned_times": ["YYYY-MM-DD"],
            "event_facts": [
              {{
                "entity": "Person A",
                "phrase": "morning workout session",
                "event_time": "YYYY-MM-DD",
                "mentioned_times": ["YYYY-MM-DD"]
              }}
            ]
          }}
        }}
      ]
    }},
    {{
      "character": "speaker_b_name",
      "events": [
        {{
          "date": "YYYY-MM-DD",
          "memory": "Specific dated event involving speaker_b_name, preserving all grounded
     answer-critical details from the cited source_refs.",
          "source_refs": ["session_1|2025-01-01|example_dia_id"],
          "graph": {{
            "entities": ["Person B"],
            "phrases": ["attended appointment"],
            "event_time": "YYYY-MM-DD",
            "mentioned_times": [],
            "event_facts": [
              {{
                "entity": "Person B",
```

```
              "phrase": "attended appointment"
            }}
          ]
        }}
      }}
    ]
  }}
 ]
}}
```

A.5 QUERY ANCHOR EXTRACTION PROMPT

```
Extract temporal constraints and query anchors.

Question:
{question}

Initial retrieved evidence (use only to disambiguate references; do not derive
dates or anchors from it):
{probe_content}

Return exactly one JSON object. No markdown or explanation.

Temporal filter:
- Use the exact schema below.
- Derive temporal patterns only from the question.
- Set is_temporal=true only when the question asks what happened on, in, or
  around a stated date/month/year.
- Set is_temporal=false for relational-time questions: before, after, later,
  earlier, since, rescheduled, postponed, moved, duration, interval, how long
  after, or first/last following an event.
- For ranges or lists (e.g., "between May 12 and May 31"), use a broad
  range-compatible month pattern, not one specific day.
- If there is no absolute event-time lookup target, use:
  is_temporal=false, resolved=false, granularity="none", patterns=[].
- Patterns match YYYY-MM-DD memory prefixes. Use "*" for an omitted year or day.

Query anchors:
- Extract only from the question.
- keywords: important lowercase content words.
- entities: explicitly named people, places, organizations, products, or objects.
- phrases: deduplicated lowercase key phrases of 1--4 words.
- Exclude generic words, pronouns, and guessed answers. Keep anchors broad.

Schema:
{{
  "temporal_filter": {{
    "is_temporal": false,
    "resolved": false,
    "granularity": "none",
    "patterns": [],
    "reason": "short reason"
  }},
  "query_anchors": {{
    "keywords": [],
    "entities": [],
    "phrases": []
  }}
}}

Each temporal pattern must be:
{{
  "year": "*",
  "month": "01",
  "day": "07",
  "glob": "*-01-07",
  "display_text": "January 7"
}}
```

A.6 ANSWER GEN PROMPT

```
    You are a knowledgeable assistant with access to character information and event records.
Answer the following question comprehensively based on the provided context.
```

```
Question: {question}

Context:
{context_text}{character_profile}

Instructions:
1. First, use <thinking>...</thinking> tags to analyze the question and available information:
   - Break down what the question is asking
   - Identify relevant information from the context
   - Consider different perspectives or interpretations
   - Note any gaps or limitations in the available information
   - Plan your approach to answering
   - **IMPORTANT**: Look for multiple answers that may be scattered across different events
   - **IMPORTANT**: Think of the time carefully, the time of an event can be relative to when
     it is mentioned by the speaker, you need to find out the real time of the event
   - **IMPORTANT**: If candidate answers conflict for the same asked fact, perform retrieval-
     time conflict resolution: use the most recent user-provided or user-attributed memory/
     update timestamp for that exact fact
   - **IMPORTANT**: Retrieval scores and retrieval order are relevance signals, not freshness
     signals. Do not prefer an older memory just because it ranks higher
   - **IMPORTANT**: Do not use outside/general knowledge to override the user's latest memory.
   - **IMPORTANT**: Use this recency rule only for direct conflicts about the exact same fact.
      If the information is compatible, cumulative, or the question asks for multiple items,
     include all relevant answers as before

2. Then, provide your final answer using <result>...</result> tags:
   - Give a complete, accurate answer based on the available information
   - **IMPORTANT**: If there are multiple answers to the question found in different events,
     include ALL of them
   - **IMPORTANT**: If conflicting answers are found for the same asked fact, return only the
     final answer after consolidating the conflicts
   - Include specific details and examples from the context when relevant
   - If information is insufficient, clearly state what is missing
   - Be factual and avoid speculation beyond what the context supports
   - Reference specific characters and events when applicable
   - Synthesize information from multiple sources when available
   - Ensure completeness - don't miss any relevant answers from the provided events

Format:
<thinking>
[Your step-by-step reasoning process here]
</thinking>

<result>
[Your final comprehensive answer here, the result should be concise and to the query]
</result>
```

A.7 JUDGE PROMPT

```
You are an expert grader that determines if answers to questions match a gold standard answer

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

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

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

Question: {question}
Gold answer: {standard_answer}
Generated answer: {generated_answer}
```

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

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