A LIGHTWEIGHT, DOMAIN-ADAPTIVE MEMORY SYSTEM FOR LLM AGENTS

Juntao Tan[*], **Liangwei Yang, Wenting Zhao, Jielin Qiu, Ming Zhu, Rithesh Murthy**
Silvio Savarese, **Huan Wang**, **Shelby Heinecke**, **Caiming Xiong**
Salesforce AI Research

ABSTRACT

Long-term memory helps LLM agents solve tasks that require reasoning over long interaction histories. Recent agentic memory systems can outperform providing the full context window or standard retrieval over text chunks, but they often rely on heavy, task-specific context engineering and complex memory pipelines, making them hard to understand, deploy, and transfer to new domains. We introduce LightMem, a lightweight, domain-adaptive memory system that keeps only the most necessary designs in three core steps: extraction, consolidation, and retrieval. LightMem removes purpose-unclear components and clearly separates what needs human input from what can be automated: users specify only memory metadata and consolidation rules, while the rest of the pipeline is general. During consolidation, LightMem automatically builds a hierarchical memory tree via non-parametric agglomerative clustering, reducing manual design and avoiding task-specific tuning. During retrieval, LightMem traverses this tree to retrieve information across clusters and multiple granularities, enabling structured access to relevant memories. We evaluate LightMem on two unrelated tasks, personalization and code repository understanding, and show that it substantially improves accuracy over vanilla RAG and prior agentic memory baselines, with latency on-par with the fastest memory baselines.

1 INTRODUCTION

LLM agents increasingly operate in settings that require reasoning over long contexts Zhang et al. (2025); Bai et al. (2025); Liu et al. (2025), including extended conversations Maharana et al. (2024); Wu et al. (2024), personalization signals accumulated over time Tan et al. (2025); Jiang et al. (2025), and large code repositories Chen et al. (2025); Qiu et al. (2025). In these scenarios, relevant information is often fragmented across lengthy inputs, scattered across many turns or documents, and mixed with irrelevant noise. As a result, agents must reliably preserve and retrieve salient information in order to answer queries, and more broadly, tacking actions.

A straightforward approach is to provide the model with the full available context window. However, this strategy is constrained by context length, often incurs substantial latency and cost, and can increase the risk that the model is distracted by irrelevant or redundant information and ultimately hurting performance Du et al. (2025); Shi et al. (2023); Liu et al. (2024). Retrieval-augmented generation (RAG) offers a scalable alternative by retrieving relevant text chunks from an external store and appending them to the prompt Lewis et al. (2020). Despite its simplicity, vanilla RAG can be unreliable for long-context reasoning: important evidence may be split across chunks, retrieved passages may be redundant or outdated, and performance can be highly sensitive to chunking strategy Gao et al. (2023); Wang et al. (2024a).

To address these limitations, recent agentic memory systems have emerged and shown promising improvements over naive context filling and flat retrieval Zhang et al. (2025); Hu et al. (2025b). These systems often introduce structured representations, multi-stage pipelines, and specialized prompting strategies Rasmussen et al. (2025b); Zhong et al. (2024). However, increasing performance frequently comes with increased complexity: many memory systems rely on heavy, task-specific

[*]Correspondence to: juntao.tan@salesforce.com

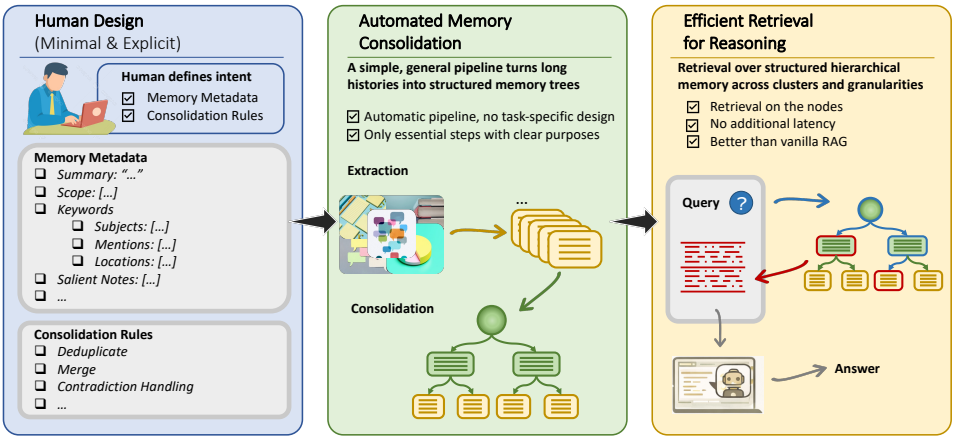

Figure 1: LightMem overview. LightMem separates minimal human design from an automated, general-purpose memory pipeline. **Left**: Users specify memory metadata and consolidation rules, which define what should be extracted as memory and provide high-level guidance for consolidation. **Middle**: LightMem extracts memory units and automatically consolidates them into a hierarchical memory tree via non-parametric agglomerative clustering, avoiding task-specific schemas. **Right**: At inference time, LightMem performs efficient tree-traversal retrieval, selecting nodes along relevant paths to access memories at different scopes and granularities while maintaining low latency.

context engineering, with multiple components whose necessity and effectiveness are not always clearly justified. This complexity makes them difficult to interpret and deploy Hu et al. (2025b), and it can hinder transfer to new domains or use cases that differ from the task assumptions baked into the memory design Hu et al. (2025a).

These trends raise a basic question: what is the minimal design, and the simplest data structures needed to support robust agent memory behavior over long contexts? Although state-of-the-art agentic memory systems can look quite complex, most can be decomposed into three essential stages: extraction, consolidation, and retrieval. We view the irreducible role of each stage as follows:

- **Extraction:** A filtering stage that removes irrelevant context and produces task-relevant memory units. This is the only stage where task-specific design should occur, ideally yielding the smallest useful granularity of memory.
- **Consolidation:** A (pre-)reasoning stage that organizes extracted units into a clean, reusable memory structure. Much of the pipeline and data-structure complexity in existing systems serves this step. At minimum, consolidation performs clustering/grouping, deduplication, and contradiction resolution. Additional heuristics may help but are not always necessary. Because this stage is heavy but largely task-agnostic, automating it can substantially reduce the engineering burden of building memory systems.
- **Retrieval:** Fetch relevant information from consolidated memory given the current query. We argue this step should be as simple and low-latency as possible for a typical memory-enabled system (e.g., a personal-assistant chatbot). This differs from open-ended discovery systems (e.g., deep-research-style agents), where the use case is less predictable and it can be beneficial to incorporate more reasoning and multi-step retrieval into the retrieval loop.

This perspective motivates a minimal memory pipeline that keeps these three stages explicit, purpose-clear, and transferable across tasks.

Motivated by this perspective, we introduce **LightMem**, a lightweight, domain-adaptive memory system that retains only the highest-leverage elements across the extraction, consolidation, and retrieval stages. As illustrated in Figure 1, LightMem enforces a clear division of labor between human input and automation: the only required task-specific design is **memory metadata** specified at the extraction stage, which defines what to store and how to filter raw context into atomic memory units. Users may optionally add a small set of consolidation rules, while the core consolidation procedure and essential operations remain general.

During consolidation, LightMem automatically constructs a hierarchical **memory tree** via **non-parametric agglomerative clustering** Sneath (1973); Müllner (2011), avoiding hand-crafted schemas and task-specific tuning. At inference time, LightMem retrieves memory nodes by traversing a small number of relevant paths in the tree, enabling access to different scopes and granularities while maintaining low latency.

We evaluate LightMem on two substantially different long-context tasks, personalization and code repository understanding, and show that it consistently improves accuracy over vanilla RAG and representative agentic memory baselines. Importantly, these gains do not come at the cost of efficiency: LightMem maintains latency on par with the fastest memory baselines by relying on embedding-based tree traversal. In summary, our contributions are threefold: (1) a minimal, purpose-clear three-stage memory pipeline with an explicit human/automation separation; (2) an automated consolidation procedure that builds a hierarchical memory tree via non-parametric clustering, reducing manual schema design and task-specific tuning; and (3) a multi-granularity retrieval mechanism that surfaces information at different scopes while achieving strong accuracy–latency trade-offs.

2 Related Work

A growing body of work equips LLM agents with external memory to overcome context-window limits and support long-context interaction Qian et al. (2025). Early systems such as MemGPT Packer et al. (2023) treat memory as an OS-like hierarchy, moving information across fast and slow tiers to preserve useful context across long documents and sessions. For persistent personalization, MemoryBank Zhong et al. (2024) stores user-centric information over time with selective retention inspired by forgetting, enabling preference-aware recall. Together, these works demonstrate the value of long-term memory while surfacing a central challenge: what to store, how to update it, and how to retrieve it reliably as histories grow.

Beyond flat chunk retrieval, recent work increasingly explores *structured* memory representations. HippoRAG Jimenez Gutierrez et al. (2024); Gutiérrez et al. (2025) and GraphRAG Edge et al. (2024) organize information using knowledge-graph-style structures and graph algorithms such as Personalized PageRank to support multi-hop retrieval and integration. A-Mem Xu et al. (2025) adopts Zettelkasten-like linked notes to retrieve related memories jointly. Zep Rasmussen et al. (2025a) proposes a temporally aware knowledge-graph memory service with structured ingestion and temporal updates, while Mem0 Chhikara et al. (2025) targets production deployments through scalable extraction, consolidation, and retrieval. Memory is also often embedded in reflection and summarization loops that store experiences and higher-level reflections for later planning Patil et al. (2025), reinforcing a common extraction–consolidation–retrieval framing.

A related emerging direction is *self-evolving* agent memory, in which agents learn storage and retrieval behaviors from optimization signals, often via reinforcement learning Yan et al. (2025); Yu et al. (2026). Although this direction is promising, it is not the focus of this work. Still, clearer decomposition of memory components may help future systems separate task-specific behaviors from more general memory mechanisms.

Despite strong results, many existing memory systems rely on richer schemas, such as note or graph structures Wang et al. (2024b); Cui et al. (2025), typed relations Ward (2025), or multi-level services Li et al. (2025), as well as multi-component pipelines Sarin et al. (2025) whose behavior can be closely tied to particular design assumptions or target use cases. In adjacent retrieval settings, tree-structured organization has also shown strong utility: RAPTOR Sarthi et al. (2024) demonstrates that hierarchical tree-based summarization and retrieval can improve navigation over large corpora, suggesting that tree structures are a plausible foundation for memory systems as well. Similarly, TreeMem Rezazadeh et al. (2024) uses a closely related tree structure, but focuses more on online memory expansion than on isolating memory functionality into minimal, general-purpose components. LightMem instead asks how far a minimal design with clear functional boundaries can go: task-specific choices are confined to lightweight *memory metadata* during extraction, with optional consolidation rules, while consolidation and retrieval remain general across domains. For broader overviews and taxonomies, see recent surveys Zhang et al. (2025); Hu et al. (2025b).

3 METHODS

3.1 PROBLEM DEFINITION

We consider an LLM agent that operates over a long context collection and must answer queries that depend on information distributed throughout that context. Let $H = \{x_1, x_2, \ldots, x_N\}$ denote a set (or ordered list) of context units, where each x_i is a unit of content such as a dialogue segment, a document passage, or a code chunk. At inference time, the agent receives a query q and produces an output y based on its intrinsic knowledge and (a subset of) H.

For an agent equipped with an external memory system, the memory system first performs **extraction** to convert context units into *memory blocks*. Given an LLM-based extractor and a memory metadata specification $\mathcal{S}$ that defines the fields of a memory block (e.g., summaries, keywords, timestamps), each context unit x_i is mapped to a set of base/leaf memory blocks:

$$\mathcal{E}_{\mathrm{LLM}}(x_i; \mathcal{S}) \to \{m_{i,1}, \ldots, m_{i,n_i}\} \tag{1}$$

Each memory block m follows the unified schema defined by $\mathcal{S}$ and includes: (i) structured memory fields (e.g., summary, keywords, salient notes, timestamps), (ii) a metadata payload (e.g., ID, number of tokens), and (iii) provenance information that tracks its source context unit, denoted by $\mathrm{src}(m) = x_i$.

The memory system then performs **consolidation** by organizing extracted blocks into a hierarchical clustering structure $\mathcal{C}$ that groups related memories at multiple levels of granularity. We define $\mathcal{C}$ abstractly: it may be instantiated by different data structures, such as a tree, a DAG, or a graph, as long as it supports hierarchical grouping and navigation. Each node $v \in \mathcal{C}$ is associated with a memory block $b(v)$ that follows the same schema $\mathcal{S}$. The lowest-level nodes correspond to the initially extracted blocks, while higher-level nodes store consolidated information (e.g., aggregated fields and evidence sources) derived from the nodes they subsume (e.g., their descendants in a tree-structured hierarchy).

At inference time, the memory system performs **retrieval** over $\mathcal{C}$ to produce a compact retrieval context $R(q)$. We write:

$$R(q) = \mathcal{R}\big(q; \mathcal{C}, \{\mathrm{src}(\cdot)\}\big) \tag{2}$$

where $\mathcal{R}$ is the retrieval procedure. Concretely, $R(q)$ can be viewed as a set of field-value pairs and/or text snippets drawn from retrieved node blocks and, when useful, their linked sources:

$$R(q) \subseteq \left(\bigcup_{v \in \mathcal{C}} \mathrm{selected\_fields}(b(v)) \right) \cup \left(\bigcup_{v \in \mathcal{C}} \mathrm{src}(b(v)) \right) \tag{3}$$

The agent then conditions on q and $R(q)$ to generate its output:

$$y = \mathrm{LLM}(q, R(q)) \tag{4}$$

We note that a classical RAG system also fits this formulation: it can be viewed as a flat memory system that directly treats each x_i (or its chunks) as a memory block, without consolidation.

The goal of an agent memory system is to retrieve a compact context that is sufficient for answering the query. This can be viewed as a trade-off between task effectiveness and retrieval size:

$$\max \ \mathbb{E}\big[\mathrm{Acc}(y, y^*)\big] \quad \text{while minimizing} \quad |R(q)| \tag{5}$$

where $y = \mathrm{LLM}(q, R(q))$ and $|R(q)|$ measures retrieval size (e.g., number of blocks or tokens).

We seek designs that keep task-specific choices localized to the memory block schema $\mathcal{S}$ during memory extraction, while allowing consolidation and retrieval to remain general and transferable.

3.2 MEMORY METADATA DESIGNING

LightMem isolates task-specific design to a single, declarative interface: the *memory metadata specification* $\mathcal{S}$. Conceptually, $\mathcal{S}$ defines the schema of a memory block, i.e., which fields a block contains and what each field is intended to represent. The LLM-based extractor uses $\mathcal{S}$ to map raw

history units into *atomic* memory blocks, and the same schema is preserved throughout consolidation: intermediate (cluster-level) memories are represented in the same block format as leaf blocks, but with aggregated fields.

We design $\mathcal{S}$ to satisfy three practical goals. First, each block should be self-contained: it must be understandable without additional surrounding context. Second, it should support selective embedding for grouping and retrieval: blocks expose high-signal fields (e.g., topics and keywords) that can be used for grouping or retrieving relevant memories. Third, it should enable retaining effective information (salient notes and grounded evidence source) for faithful answering.

Across tasks, we adopt a common set of high-level fields in $\mathcal{S}$: (i) a short `summary` and a `scope` describing what the block covers; (ii) an `applicability` field describing when the block is useful for answering questions; (iii) `topics` and `keywords` used as grouping/retrieval signals; and (iv) a list of `salient notes`, where each note is an atomic, grounded claim extracted from the input. In addition, we include provenance signals so that retrieved blocks can be traced back to supporting evidence in the original documents or interaction histories.

While the consolidation and retrieval procedures are shared across domains, the block schema $\mathcal{S}$ can be adapted with lightweight, task-appropriate fields. We provide the task-specific schema instantiations and a simplified extraction prompt template for personalization and code repository understanding in Appendix A.

A key design choice is that the same schema $\mathcal{S}$ is used for both leaf-level blocks and consolidated (cluster-level) blocks. During consolidation, higher-level blocks may contain abstracted summaries, aggregated keywords/topics, and merged salient notes (e.g., deduplicated or temporally updated), but they retain the same field structure. This ensures that retrieval can uniformly operate over blocks at different granularities, and that the returned context $R(q)$ can mix coarse and fine memories without requiring separate representations.

3.3 Memory Block Extraction

Given the memory metadata specification $\mathcal{S}$, LightMem performs extraction by applying an LLM-based extractor $\mathcal{E}_{\text{LLM}}$ to each context unit x_i in H, producing a (possibly empty) set of leaf memory blocks as defined in Eq.1.

Each memory block m follows the schema $\mathcal{S}$, including high-level descriptors fields (e.g., `summary`, `scope`, `applicability`), retrieval signals (e.g., `topics`, `keywords`), and a compact list of atomic `salient notes`. The source context to each block is also tracked as $\text{src}(m) = x_i$.

3.4 Memory Tree Construction

LightMem consolidates extracted base memory blocks into a hierarchical clustering structure $\mathcal{C}$ to enable multi-granularity access. Concretely, given the set of leaf blocks $M = \{m_{i,j}\}$, we build $\mathcal{C}$ using agglomerative clustering Sneath (1973); Müllner (2011) with a cosine-based distance between node embeddings. We initialize each leaf as an active cluster and iteratively merge the closest pair until a single root remains (or until an optional distance threshold is met).

Each node in the hierarchy (both leaves and internal nodes) is represented using the same memory block schema $\mathcal{S}$. To support clustering, we associate each node with an embedding computed from its schema fields. Specifically, for any node with memory block b, we construct an embedding input text by concatenating the text in the fields and compute an embedding vector $e(b)$. Leaf embeddings are computed from extracted blocks, and internal node embeddings are recomputed from the merged block using the same procedure.

When merging two child nodes with blocks b_a and b_b (both following $\mathcal{S}$), we use an LLM-based merge operator to produce the merged block b_p in a single generation step. The LLM is instructed to (i) preserve grounded information from both children, (ii) remove redundancy via semantic and normalization-based deduplication, and (iii) regenerate the fields (e.g., `summary`, `topics` and `keywords`) from the merged salient notes. Additional consolidation rules (e.g., conflict-handling heuristics) can be optionally provided, but are not required by the core procedure. After obtaining b_p, we compute the parent embedding as $e(b_p)$ and insert the new node back into the active set.

Stopping criterion. By default, we continue merging until a single root remains, producing a full hierarchy. Optionally, one may stop merging when the nearest-pair distance exceeds a threshold, yielding a forest of higher-level clusters. In our experiments, we use the full hierarchy.

3.5 RETRIEVAL

Given a query q, LightMem retrieves a compact set of relevant memories from the hierarchy $\mathcal{C}$. We first build a retrieval index by embedding all memory nodes (including both leaf and internal nodes) using the associated fields. For each node v with memory block b_v, we compute an embedding vector $e(b_v)$. At inference time, we embed the query to obtain $e(q)$ and score each node by cosine similarity:

$$s(v, q) \;=\; \cos\big(e(b_v), e(q)\big) \tag{6}$$

We then obtain a ranked list of candidate nodes via nearest-neighbor search over these embeddings.

A key advantage of the hierarchical structure is that nodes along the same branch naturally form a *coarse-to-fine* representation: higher-level nodes summarize broader content spanning their descendants, while nodes closer to the leaves retain more fine-grained details. This observation motivates retrieving from multiple branches to improve coverage under a fixed retrieval budget, as described below:

Branch-aware selection. A naive top-k over all nodes can return redundant memories from the same branch of the hierarchy (e.g., both a node and its ancestor), which reduces coverage and wastes retrieval budget. To encourage diversity while preserving efficiency, LightMem performs **path-based branch selection**: from the similarity-ranked candidate list, we greedily select up to k nodes such that no selected pair has an ancestor–descendant relationship. Formally, we construct

$$S(q) \subseteq \mathcal{C}, \quad |S(q)| \leq k \tag{7}$$

subject to the constraint that for any $u, v \in S(q)$, neither u is an ancestor of v nor v is an ancestor of u. In practice, we enforce this constraint using a precomputed Euler tour of the hierarchy, enabling $O(1)$ ancestor checks during selection.

Multi-granularity retrieval context. Because the candidate pool includes both leaves and internal cluster nodes, the selected set $S(q)$ naturally mixes granularities: internal nodes provide concise cluster-level summaries, while leaf nodes provide fine-grained details when they are the best non-overlapping matches. The final retrieval context $R(q)$ is constructed from the schema fields of the selected nodes and, when useful, can include linked evidence from the original sources.

Finally, this branch-aware retrieval perspective suggests an interesting direction for future work: training embedding models that are explicitly *granularity-aware* for hierarchical memory. In this paper, however, we focus on the pipeline design and use off-the-shelf embedding models.

4 EXPERIMENTS

4.1 DATASETS

To evaluate the effectiveness and adaptability of the proposed memory framework, we conduct experiments on two fundamentally different tasks, personalization and code repository understanding, both of which require LLMs to reason over long contextual histories.

Personalization The first dataset is **LoCoMo** Maharana et al. (2024), a widely used benchmark in agentic memory research and commonly adopted by state-of-the-art memory systems. LoCoMo evaluates an LLM's ability to understand and reason over extremely long-term human-to-human conversations spanning extended time periods. The model is subsequently queried about personal information discussed throughout the dialogue, requiring accurate memorization, aggregation, and reasoning over information distributed across the entire interaction history.

Code Repository Understanding The second dataset focuses on **code repository understanding**, a subtask from LongBench v2 Bai et al. (2025). This dataset consists of textual representations of code repositories paired with multiple-choice questions that require professional-level understanding of code structure, functionality, and inter-file dependencies. Successfully answering these questions demands long-context comprehension as well as precise retrieval of relevant information from large codebases.

4.2 BASELINES

4.2.1 PERSONALIZATION BASELINES

We compare our method against both classical RAG approaches and structured agentic memory systems. For RAG baselines, we include both sparse retrieval methods and dense retrieval methods using embedding models of varying sizes. For structured agentic memory baselines, we select representative prior systems including A-Mem Xu et al. (2025), LongMem [1], Zep Rasmussen et al. (2025a), and Mem0 Chhikara et al. (2025).

For agentic memory methods, we adopt the original evaluation results reported in Mem0 and use the same evaluation implementation to evaluate our method, ensuring an apples-to-apples comparison. For RAG methods, however, during our experiments, we observed that the original settings reported in prior work, which use a small number of retrieved chunks (e.g., $k = 1$ or $k = 2$), do not fully capture the performance potential of RAG on this benchmark. We therefore systematically increase the number of retrieved chunks and find that performance continues to improve until peaking at $k = 30$. We report the best-performing RAG results under this setting.

4.2.2 CODE REPOSITORY UNDERSTANDING BASELINES

Since existing agentic memory frameworks are not specifically designed for code repository understanding, we compare our method against the following baselines: (1) **closed-book inference**, where the LLM is provided only with the question and must answer using its intrinsic knowledge; (2) **full-context inference**, where the entire repository description is provided directly to the LLM; and (3) **classical RAG**, where relevant code chunks are retrieved and supplied to the model. For the RAG baseline, we refer to the empirical findings reported in the original LongBench v2 paper, which show that a chunk size of 512 tokens and $k = 256$ retrieved chunks achieve the best performance. We adopt this configuration in our evaluation.

4.3 IMPLEMENTATION DETAILS

Memory construction. During memory building, LightMem uses gpt-4.1-mini for the memory extraction phase, leveraging its strong capability to handle and generate long contexts. gpt-4o-mini is used for all other consolidation and reasoning stages. `text-embedding-3-small` is used as the embedding model.

In the extraction phase, for the LoCoMo dataset, we provide the LLM with the entire conversation session at each step to generate multiple leaf memory blocks. For the code repository understanding dataset, the context window for each QA test case can be extremely long. Therefore, we chunk the input proportionally, with each segment representing 0.5% of the total context length.

To demonstrate the robustness and adaptability of the system, we minimize prompt complexity and follow human intuition when designing both the memory metadata and the consolidation rules. In particular, we use only basic consolidation rules (merging and deduplicating salient items, and normalizing keywords). This design choice ensures that performance gains come from the proposed memory framework itself, rather than from very complex prompt design.

4.4 EVALUATION PROTOCOL AND METRICS

We set $k = 10$ for LightMem in both experiments, meaning that 10 nodes are retrieved from the memory tree. For LoCoMo, we use `text-embedding-3-small` as the dense embedding model

[1] https://langchain-ai.github.io/langmem/

for retrieval and feed the retrieved context to gpt-4o-mini to generate the final answer, matching the baseline setting. First, similar to Mem0, we do not include the original source text chunks; instead, we directly provide the salient fact notes from the retrieved memory blocks, which we find are already sufficient to produce strong answers. We also show that including both salient notes and linked raw source chunks can further boost performance, with a reasonable increase in latency.

For code repository understanding, we use the same embedding model for retrieval and evaluate answer generation with both gpt-4o-mini and gpt-4.1-mini. In this task, we find that the original code context remains helpful, so we include the linked raw source text in the retrieved context.

Metrics. For LoCoMo, we use LLM4Judge with gpt-4o-mini. The evaluation prompt and implementation strictly follow the Mem0 evaluation codebase[2]. Similarly, for code repository understanding, we use the original LongBench v2 evaluation implementation[3].

4.5 RESULTS

4.5.1 LOCOMO

Table 1 reports accuracy (LLM judge) and latency on LoCoMo. When RAG is allowed to retrieve a sufficient number of chunks (e.g., $k = 30$), strong sparse and dense retrieval baselines outperform most prior agentic memory systems. In contrast, **LightMem** achieves higher accuracy than these RAG baselines on three of the four LoCoMo categories, suggesting that hierarchical consolidation and branch-aware selection better aggregate dispersed personal facts under a compact retrieval budget. Moreover, augmenting retrieved memory notes with linked raw source chunks further improves LightMem's overall performance.

For Open Domain questions, BM25 attains the highest score, and both sparse and dense retrieval models remain highly competitive. LightMem is slightly lower in this category, indicating that open-domain queries often benefit from retrieving larger amounts of raw evidence.

Table 1 also reports both search latency (retrieval only) and total latency (retrieval plus answer generation). End-to-end latency depends not only on retrieval cost but also on the amount of text passed to the answer model. While BM25 has near-zero search time, its total latency increases substantially at larger k due to longer contexts. Dense RAG with $k = 30$ exhibits a similar pattern. By retrieving a small number of non-redundant nodes, LightMem maintains low latency, with search time comparable to dense retrieval and total latency on par with Mem0, the fastest agentic memory baseline.

4.5.2 CODE REPOSITORY UNDERSTANDING

Figure 2 summarizes results on the code repository understanding task. For both answer models, providing the full repository context reduces performance relative to closed-book inference (e.g., from 36% to 28% for gpt-4o-mini and from 50% to 46% for gpt-4.1-mini), consistent with prior observations that extremely long contexts can degrade effective reasoning Du et al. (2025). Classical RAG with a large retrieval budget ($k = 256$) also fails to improve over the no-context baseline, suggesting that retrieving many raw chunks can still be noisy and insufficient for reliable evidence aggregation. In contrast, **LightMem** achieves the best performance for both answer models (42% and 56%), indicating that structured, compact retrieval from consolidated memory provides more useful context than either full-context prompting or flat retrieval.

4.5.3 ABLATION STUDIES

Appendix B.1 (Figure 4) analyzes the effect of the retrieval budget k for both RAG and LightMem. Appendix B.2 (Table 2) compares different retrieval context types.

[2]https://github.com/mem0ai/mem0/tree/main/evaluation/src

[3]https://github.com/THUDM/LongBench/blob/main/pred.py

Table 1: Comparison of latency and accuracy across retrieval and memory methods. Results marked with * were implemented and evaluated by us using the evaluation code from mem0; all other results are directly taken from the original mem0 paper. Each experiment was run three times, and the reported values are averages.

	Latency (s)				Accuracy (LLM Judge)			
	P50		P95					
	Search	Total	Search	Total	Single Hop	Multi Hop	Open Domain	Temporal
BM25*								
$k = 1$	–	–	–	–	29.43 ±0.71	29.17 ±1.05	65.87 ±0.45	37.07 ±0.31
$k = 2$	–	–	–	–	36.88 ±0.36	36.46 ±0.00	76.58 ±0.12	50.47 ±0.62
$k = 30$	**0.001**	0.763	**0.001**	2.508	64.89 ±0.71	54.17 ±2.09	**88.94 ±0.12**	65.42 ±0.31
text-embedding-3-small*								
$k = 1$	–	–	–	–	38.30 ±0.36	34.38 ±0.52	60.64 ±0.12	34.89 ±0.16
$k = 2$	–	–	–	–	47.87 ±0.36	42.71 ±1.05	73.13 ±0.12	43.93 ±0.31
$k = 30$	0.168	0.924	0.343	2.672	74.11 ±1.34	50.59 ±2.54	87.63 ±0.62	65.73 ±1.63
Qwen3-8B*								
$k = 1$	–	–	–	–	38.65 ±1.07	39.58 ±2.09	58.03 ±0.12	31.78 ±0.31
$k = 2$	–	–	–	–	51.42 ±0.71	40.62 ±1.57	70.27 ±0.16	43.61 ±0.32
$k = 30$	0.067	0.822	0.108	2.2308	76.60 ±0.05	52.17 ±3.04	86.07 ±0.37	60.74 ±2.50
A-Mem	0.668	1.410	1.485	4.374	39.79 ±0.38	18.85 ±0.31	54.05 ±0.22	49.91 ±0.31
LangMem	17.99	18.53	59.82	60.40	62.23 ±0.75	47.92 ±0.47	71.12 ±0.20	23.43 ±0.39
Zep	0.513	1.292	0.778	2.926	61.70 ±0.32	41.35 ±0.48	76.60 ±0.13	49.31 ±0.50
Mem0	0.148	**0.708**	0.200	1.440	67.13 ±0.65	51.15 ±0.31	72.93 ±0.11	55.51 ±0.34
Mem0g	0.476	1.091	0.657	2.590	65.71 ±0.45	47.19 ±0.67	75.71 ±0.21	58.13 ±0.44
LightMem*	0.171	0.757	0.367	**1.087**	**78.01 ±0.36**	56.60 ±2.43	84.86 ±0.63	66.46 ±2.08
LightMem w/ source*	0.169	0.871	0.314	1.689	76.71 ±0.59	**63.89 ±0.69**	88.55 ±0.20	**76.85 ±0.53**

Note: Latency is in seconds (lower is better). Accuracy is fraction correct (higher is better).

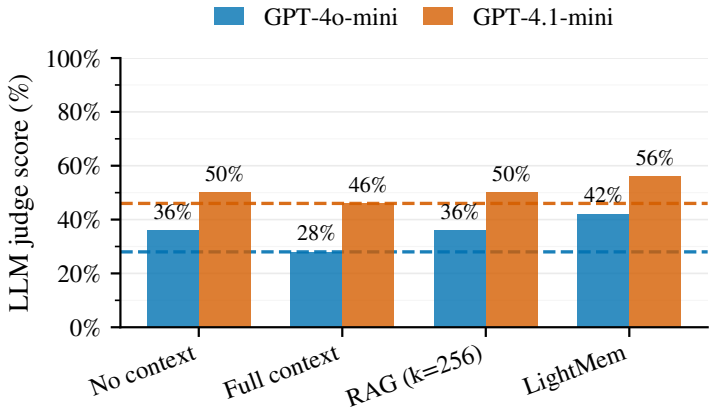

Figure 2: **Code repository understanding results.** LLM-judge accuracy on the code repository understanding task under four context settings: closed-book (no context), full-context prompting, classical RAG ($k = 256$), and LightMem. Results are reported for two answer models (gpt-4o-mini and gpt-4.1-mini).

5 Conclusion

We introduced LightMem, a lightweight, domain-adaptive memory framework for LLM agents. By clearly separating human-guided specification from automated memory operations, LightMem simplifies memory construction while remaining effective across tasks. On personalization and code understanding benchmarks, LightMem improves accuracy over vanilla RAG and representative agentic memory baselines while maintaining competitive latency.

Beyond these results, LightMem's minimal interface enables rapid task-specific customization with limited context engineering. The explicit split between human inputs and automated consolidation also provides a clean foundation for future self-evolving agents, where memory can adapt over time with minimal manual effort. Overall, LightMem takes a step toward more transparent, transferable, and scalable long-term memory for LLM agents.

REFERENCES

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.183.

Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. Locagent: Graph-guided llm agents for code localization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8697–8727, 2025.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

Justin Cui, Kevin Pu, and Tovi Grossman. Loom: Personalized learning informed by daily llm conversations toward long-term mastery via a dynamic learner memory graph. *arXiv preprint arXiv:2511.21037*, 2025.

Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. Context length alone hurts llm performance despite perfect retrieval. *arXiv preprint arXiv:2510.05381*, 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.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

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 Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 21497–21515. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/gutierrez25a.html.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions. *arXiv preprint arXiv:2507.05257*, 2025a.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025b.

Bowen Jiang, Zhuoqun Hao, Young-Min Cho, Bryan Li, Yuan Yuan, Sihao Chen, Lyle Ungar, Camillo J Taylor, and Dan Roth. Know me, respond to me: Benchmarking llms for dynamic user profiling and personalized responses at scale. *arXiv preprint arXiv:2504.14225*, 2025.

Bernal Jimenez Gutierrez, 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.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Rui Li, Zeyu Zhang, Xiaohe Bo, Zihang Tian, Xu Chen, Quanyu Dai, Zhenhua Dong, and Ruiming Tang. Cam: A constructivist view of agentic memory for llm-based reading comprehension. *arXiv preprint arXiv:2510.05520*, 2025.

Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, et al. A comprehensive survey on long context language modeling. *arXiv preprint arXiv:2503.17407*, 2025.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

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, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.747.

Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

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

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Hongjin Qian, Zheng Liu, Peitian Zhang, Kelong Mao, Defu Lian, Zhicheng Dou, and Tiejun Huang. Memorag: Boosting long context processing with global memory-enhanced retrieval augmentation. In *Proceedings of the ACM on Web Conference 2025*, pp. 2366–2377, 2025.

Jielin Qiu, Zuxin Liu, Zhiwei Liu, Rithesh Murthy, Jianguo Zhang, Haolin Chen, Shiyu Wang, Ming Zhu, Liangwei Yang, Juntao Tan, et al. Locobench-agent: An interactive benchmark for llm agents in long-context software engineering. *arXiv preprint arXiv:2511.13998*, 2025.

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*, 2025a.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory, 2025b. URL `https://arxiv.org/abs/2501.13956`.

Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. In *The Thirteenth International Conference on Learning Representations*, 2024.

Samarth Sarin, Lovepreet Singh, Bhaskarjit Sarmah, and Dhagash Mehta. Memoria: A scalable agentic memory framework for personalized conversational ai. In *2025 5th International Conference on AI-ML-Systems (AIMLSystems)*, pp. 32–39. IEEE, 2025.

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

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pp. 31210–31227. PMLR, 2023.

PHA Sneath. The principles and practice of numerical classification. *Numerical Taxonomy*, 573, 1973.

Juntao Tan, Liangwei Yang, Zuxin Liu, Zhiwei Liu, Rithesh RN, Tulika Manoj Awalgaonkar, Jianguo Zhang, Weiran Yao, Ming Zhu, Shirley Kokane, et al. Personabench: Evaluating ai models on understanding personal information through accessing (synthetic) private user data. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 878–893, 2025.

Xiaohua Wang, Zhenghua Wang, Xuan Gao, Feiran Zhang, Yixin Wu, Zhibo Xu, Tianyuan Shi, Zhengyuan Wang, Shizheng Li, Qi Qian, et al. Searching for best practices in retrieval-augmented generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 17716–17736, 2024a.

Zheng Wang, Zhongyang Li, Zeren Jiang, Dandan Tu, and Wei Shi. Crafting personalized agents through retrieval-augmented generation on editable memory graphs. *arXiv preprint arXiv:2409.19401*, 2024b.

Joel Ward. Memoriesdb: A temporal-semantic-relational database for long-term agent memory/-modeling experience as a graph of temporal-semantic surfaces. *arXiv preprint arXiv:2511.06179*, 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.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

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

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

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

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 MEMORY METADATA SCHEMAS AND ABBREVIATED EXTRACTION PROMPT TEMPLATE

This appendix reports the task-specific memory block schemas used in our experiments, followed by a single abbreviated extraction prompt template. The same template is instantiated by substituting the appropriate schema (personalization or repository understanding) into the `<<SCHEMA>>` placeholder.

A.1 MEMORY BLOCK SCHEMAS

Personalization schema ($\mathcal{S}_{\text{PERS}}$).

```
{
  "summary": "string",
  "subjects": ["string"],
  "scope": "string",
  "applicability": "string",
  "topics": ["string"],
  "keywords": {
    "mentions": ["string"],
    "predicate": ["string"],
    "locations": ["string"]
  },
  "salient_notes": [
    {
      "text": "string",
      "event_time": "YYYY | YYYY-MM | YYYY-MM-DD | null",
      "note_time": "YYYY-MM-DD HH:MM | null"
    }
  ],
  "time_description": "string"
  "example_questions": ["string"]
}
```

Repository schema ($\mathcal{S}_{\text{REPO}}$).

```
{
  "summary": "string",
  "scope": "string",
  "applicability": "string",
  "topics": ["string"],
  "keywords": {
    "symbols": ["string"],
    "endpoints": ["string"],
    "cli": ["string"],
    "configs": ["string"],
    "files": ["string"],
    "libraries": ["string"],
    "concepts": ["string"]
  },
  "salient_notes": [
    { "text": "string" }
  ],
  "example_questions": ["string"]
}
```

A.2 ABBREVIATED EXTRACTION PROMPT TEMPLATE

Figure 3: Simplified representation of the LLM extraction prompt. The template is instantiated by inserting a task-specific memory schema $\mathcal{S}$ (Appendix A.1). For clarity, we show the high-level structure and intent while omitting detailed instructions.

```
{
  "System": "You are an expert information extractor that converts an
      input document unit into memory blocks.
  Output MUST be a STRICT JSON ARRAY of 0-5 blocks (no wrapper keys, no
      extra text).
  Do not invent facts.
  /* Detailed formatting constraints and safety checks */",

  "Input": {
    "segment id": "{segment_id}",
    "timestamp": "{optional}",
    "content": "{raw text}"
  },

  "Goal": "Split into 0-5 topical memory blocks.
  Each block should be self-contained and cover a coherent topic/scope.
  Return [] if no eligible information exists.",

  "BlockSeparation": "Create separate blocks when topic/scope/timeframe
      differ materially.
  Avoid mixing unrelated facts/features in one block.
  Prefer fewer, cleaner blocks over over-splitting.",

  "Eligibility": "Extract only task-relevant, grounded facts from the
      input.
  Ignore fluff, meta-chat, or unsupported claims.
  /* Task-specific eligibility details */",

  "Schema": "<<memory metadata schema S>>"
}
```

B Ablation Studies

B.1 Effect of the retrieved k

Figure 4 shows LoCoMo accuracy as a function of the retrieval budget k for classical RAG and LightMem under the notes-only setting. LightMem consistently outperforms RAG at the same k, indicating that hierarchical consolidation and branch-aware retrieval select more informative and less redundant evidence than flat chunk retrieval. As k increases, both methods improve, but the gap remains, suggesting that LightMem is not only more retrieval-efficient but also provides higher-quality retrieved context even when restricted to compact memory notes.

B.2 Notes-only vs. source-only vs. combined retrieval

Table 2 compares three retrieval settings: retrieving only salient notes, only raw source chunks, or both. Retrieving **both** consistently performs best on LoCoMo and code repository understanding, suggesting that notes and sources provide complementary signals. On LoCoMo, notes-only is already strong, indicating that consolidated notes preserve most personal facts, while adding sources recovers additional details. On code repositories, source-only outperforms notes-only by a larger margin, reflecting that details are often best captured in raw context for code understanding.

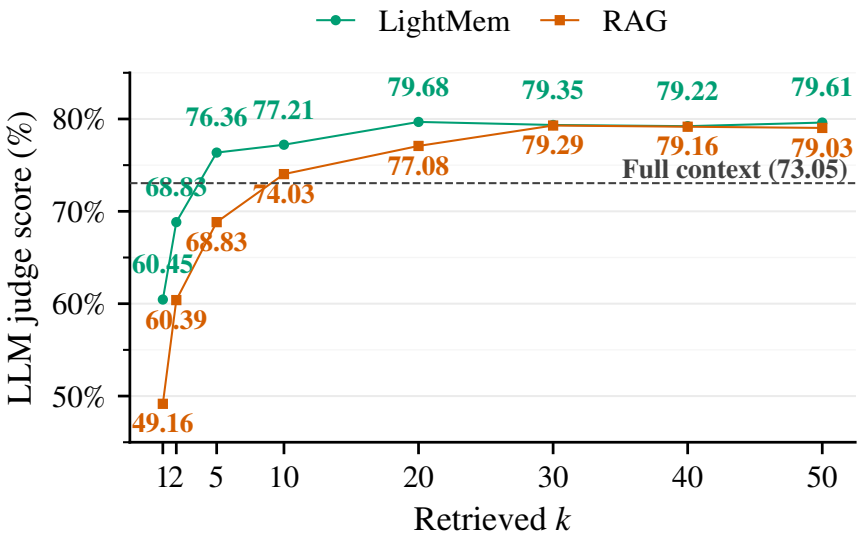

Figure 4: **Effect of retrieval k.** LoCoMo accuracy versus the number of retrieved units for RAG and LightMem (notes only).

Table 2: Ablation study of overall LLM-as-a-judge scores for different retrieval settings across datasets. Scores are computed using GPT-4o-mini as the base LLM.

Retrieval Setting	**LoCoMo**	**Code Repo**
Notes only	77.21	34.00
Source only (raw chunks)	79.81	42.00
Notes + Source	**82.21**	**44.00**