DuCoMem: Efficient Long-Term Memory for LLMs through Dual Compression and Local Recovery

Anonymous authors
Paper under double-blind review

Abstract

External memory enables large language models (LLMs) to use long-term interaction histories, but its construction and retrieval costs grow rapidly as histories accumulate. Existing lightweight memory methods, however, compress the memory content to preserve its semantics, overlooking that the details discarded by compression, *e.g.,* names, locations, or temporal cues, are often essential for locating the memory in response to future queries. Meanwhile, graph-based memories rely on the LLMs to construct semantic relations before any query arrives, incurring additional costs. In this paper, we propose *DuCoMem*, a dual compression memory framework that jointly compresses the memory content and graph structure while preserving on-demand access to the original records. Specifically, we first reveal that the content retention and content retrievability diverge under the compression. The residual addresses are then introduced to store the key details discarded by the compressor, such that the compact memories remain locatable. For graph compression, we first simplify a low-cost candidate graph by removing the redundant nodes and edges, and then use an LLM to check relations only on the smaller graph. During retrieval, DuCoMem starts from a few representative nodes on the compressed graph, reads the nearby original records, and searches further only when the evidence is not enough to answer the question. This design reduces the structure processed during construction and the search space explored during retrieval. Experimental results demonstrate that DuCoMem outperforms the strongest lightweight-memory baseline across model scales, with an average relative improvement of 23.7% in macro F1 across seven answering models on LoCoMo while maintaining a comparable token cost.

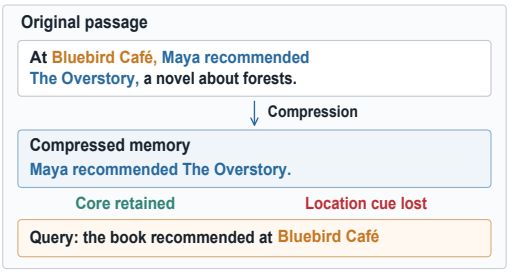
1 Introduction

Large language models (LLMs) increasingly serve as agents that interact with users over many sessions (Packer et al., 2023). Such agents rely on the external memory to store interaction histories and retrieve relevant information when needed (Park et al., 2023; Zhong et al., 2024). In some long conversation benchmarks, such as LoCoMo and LongMemEval, the agents are further required to locate and combine evidence distributed across conversation contents (Maharana et al., 2024; Wu et al., 2025). As histories grow, however, constructing and searching the memory become costly in terms of both computations and tokens (Chhikara et al., 2025; Kang et al., 2025; Fang et al., 2026).

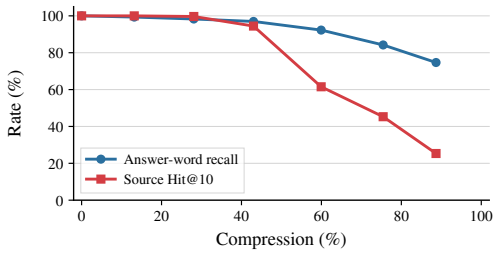
Graph-based memories explicitly link the related records, thus improving access to the cross-record and multi-hop evidence (Edge et al., 2024; Xu et al., 2025; Rasmussen et al., 2025). These benefits often come with an additional construction and retrieval overhead. Existing lightweight memory methods mainly reduce cost by filtering the redundant text and constructing compact memory representations (Pan et al., 2025; Fang et al., 2026; Liu et al., 2026). However, future queries are unknown during the memory construction, making it difficult to decide *what to keep* and *what to link* in advance. This leads to the following two challenges.

i) Preserving the content of a memory does not ensure that it can still be retrieved. A detail that is unnecessary for describing a fact may be exactly what a query uses to find it. For example, as shown in Fig. 1(a), compression retains Maya's recommendation of *The Overstory* but removes Bluebird

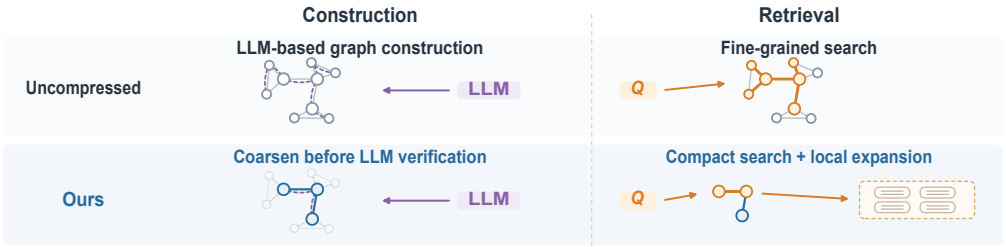
Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping 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.



(a) Compression can remove retrieval cues



(b) Content retention vs. source retrieval



(c) Graph compression and on-demand recovery

Figure 1: **Motivations for dual compression.** (a) Compression can remove retrieval cues while retaining answer content. (b) Answer-word recall remains high as source Hit@10 declines under controlled compression on selected LoCoMo pairs. Here, answer-word recall is the fraction of words in the reference answer that remain in the compressed memory. Source Hit@10 is the fraction of queries whose correct source appears among the top ten retrieved memories. (c) Graph coarsening reduces subsequent LLM verification, while local recovery supports on-demand source access.

Café, the term a later question uses to identify it. To measure this gap, we compress the LoCoMo memories with LLMLingua-2 (Pan et al., 2024) while keeping the queries unchanged. The results in Fig. 1(b) suggest that when approximately 60% of the tokens are removed, 92.28% of answer words are retained, but the source Hit@10 falls to 61.52%. Memory compression should therefore preserve not only what a memory brings, but also how it can be found.

ii) The cost of graph memory is paid at construction time. Existing graph memories use LLMs to extract and verify relations among records before any query arrives (Edge et al., 2024; Xu et al., 2025; Rasmussen et al., 2025). However, only a small subset of these relations may be relevant to a future query. Constructing a fine-grained semantic graph in advance incurs unnecessary cost. The memory should instead maintain a compact graph during construction and defer fine-grained expansion to query time, when only the relevant local region and original records need to be recovered.

To address these challenges, we propose DuCoMem, a dual compression memory framework with on-demand local recovery. For content compression, DuCoMem compresses each memory capsule and stores the key terms removed by the compressor as residual addresses. A query can then find a compressed capsule through these terms. For graph compression, DuCoMem builds a low-cost candidate graph from embedding similarity and shared entities, as illustrated in Fig. 1(c). It removes redundant nodes and edges, and then uses an LLM only to verify selected relations on the smaller graph, rather than to extract and process relations across the full graph. During retrieval, DuCoMem starts from a few relevant nodes, locally recovers nearby original records, and searches further only when the evidence is insufficient. Across seven answering models on LoCoMo, DuCoMem yields a 23.7% average relative improvement in macro F1 over the strongest lightweight-memory baseline, while retaining a comparable token cost. Our main contributions can be summarized as follows.

- We show that content retention and retrievability diverge under the memory compression, and propose the residual addresses to keep compressed memories retrievable.
- We propose an efficient graph compression scheme that constructs a low-cost candidate graph before the LLM relation verification, with recovery mappings that keep all of the original records reachable.

- We design a local retrieval process that searches around a few entry nodes and performs a focused search only when the evidence is insufficient.

2 Related Work

Efficient Memory for LLM Agents. Generative Agents, MemoryBank, and MemGPT establish different ways to store and revisit past interactions (Park et al., 2023; Zhong et al., 2024; Packer et al., 2023). More recent systems reduce the work involved in maintaining those memories. SeCom groups conversation turns into coherent segments (Pan et al., 2025), while Mem0 extracts salient facts and updates them as new interactions arrive (Chhikara et al., 2025). LightMem uses LLMLingua-2 to filter incoming text, groups related content by topic, and moves long-term updates away from the immediate response path (Fang et al., 2026; Pan et al., 2024). SimpleMem instead builds compact atomic memories and retrieves them through semantic, lexical, and symbolic indexes (Liu et al., 2026). LLMLingua and LLMLingua-2 show how token selection can reduce the text processed by an LLM (Jiang et al., 2023a; Pan et al., 2024). These approaches save computation, although a compact memory may omit an identifying detail needed by a later query.

Structured Memory RAPTOR uses recursive summaries to organize documents at multiple levels (Sarthi et al., 2024), whereas StructMem keeps event bindings when consolidating memories (Xu et al., 2026). Graph-based approaches make cross-record associations more explicit. GraphRAG extracts relations and summarizes graph communities (Edge et al., 2024), while HippoRAG and HippoRAG 2 connect evidence across passages for multi-step retrieval (Gutiérrez et al., 2024; Gutiérrez et al., 2025). A-MEM links contextualized notes, and Zep maintains temporal relations (Xu et al., 2025; Rasmussen et al., 2025). These structures improve access to related information, but constructing semantic links and verifying their source support can be expensive as memory grows.

Retrieval and Adaptive Evidence Acquisition. Long-context evaluations show that relevant evidence is not always used reliably when supplied in a lengthy input (Bai et al., 2024; Liu et al., 2024). Dense passage retrieval locates relevant text before generation (Karpukhin et al., 2020; Lewis et al., 2020), while SimpleMem combines multiple memory indexes for different query cues (Liu et al., 2026). Other approaches obtain additional evidence as reasoning proceeds. IRCoT alternates retrieval and reasoning, FLARE issues further searches from generated text, and Self-RAG assesses when retrieval is useful (Trivedi et al., 2023; Jiang et al., 2023b; Asai et al., 2024). Adaptive-RAG chooses a retrieval strategy based on question complexity (Jeong et al., 2024). A single retrieval pass can still miss evidence for a relation, while repeated broad searches increase the work needed to resolve it.

3 DuCoMem

3.1 Overview

Given a conversation history across multiple sessions, DuCoMem constructs an external memory for subsequent retrieval and question answering. Let $X = \{x_j\}_{j=1}^{N}$ denote the original conversation records, each with a persistent identifier and available session metadata. Each memory capsule p_i, for $i = 1, \ldots, n$, summarizes the records indexed by $\mathcal{I}_i \subseteq \{1, \ldots, N\}$. We retain the original records in a separate source archive and represent the retrieval state as

$$M = (C, R, S, \Phi, W),$$

where $C = \{c_i\}_{i=1}^{n}$ contains compact capsule representations, $R = \{r_i\}_{i=1}^{n}$ contains residual retrieval cues, S is the coarsened graph, and Φ stores representative mappings and recovery paths. The source references $\mathcal{I}_i$ locate records in the archive. The auxiliary information W includes source indexes, embeddings, metadata, a motif index, and verified semantic links. Fig. 2 shows the construction and retrieval processes.

3.2 Content Compression with Residual Addressing

Compact Memory Representations. We group related conversation records into a memory capsule p_i and retain their source identifiers $\mathcal{I}_i$. Compact capsules reduce the text stored in the retrieval

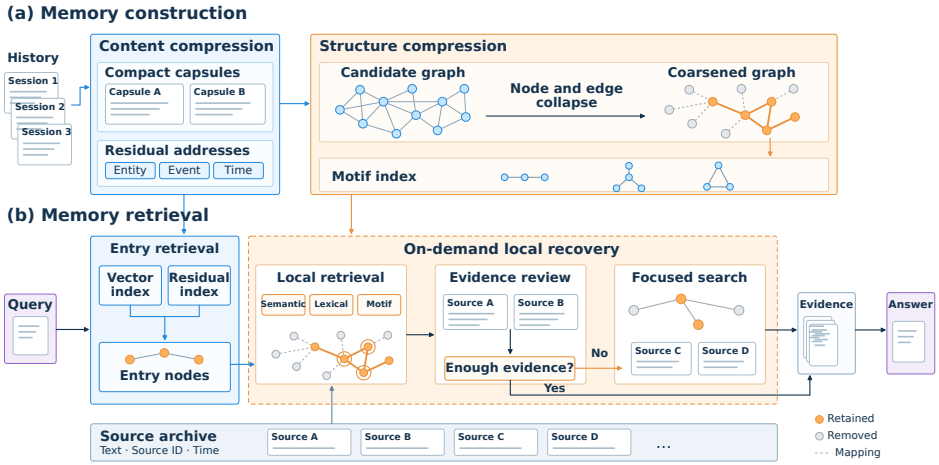


Figure 2: **Overview of DuCoMem.** Construction compresses memory content and graph structure while retaining retrieval cues and source mappings. Retrieval locates relevant regions, recovers local evidence, and performs focused search when needed.

index and the content processed when memories are searched. A query-independent compressor produces $c_i = \mathcal{C}_{\text{cont}}(p_i; k_c)$, where k_c sets the content budget. By default, we use LLMLingua-2 (Pan et al., 2024) for query-independent token selection. We embed c_i for semantic retrieval and keep the complete records in a separate source archive. This interface allows retrieval to operate on short representations without assuming a particular compression algorithm. Because future questions are unknown when capsules are compressed, we next preserve identifying cues that may be missing from c_i.

Residual Retrieval Cues. The terms that help summarize a capsule are not always the terms that help to retrieve it. We therefore retain a small set of omitted names, places, times, and other identifying terms as residual retrieval cues. Let $\mathcal{T}_i$ contain normalized terms from the capsule's source text and metadata, and let $\mathcal{V}(c_i)$ contain the terms in its compact representation. Among the omitted terms $\mathcal{A}_i = \mathcal{T}_i \setminus \mathcal{V}(c_i)$, we favor terms that occur in this capsule but are uncommon across capsules. We use a sublinear weight TF–IDF following the classical weighting of the term (Salton & Buckley, 1988). Recent retrieval systems also use TF–IDF to prioritize informative terms (Kim et al., 2025). The weight is

$$w_i(t) = (1 + \log \mathrm{tf}_i(t)) \log \frac{n}{\mathrm{df}(t)}, \tag{1}$$

where $\mathrm{tf}_i(t)$ counts occurrences of t in capsule i's source text and metadata, and $\mathrm{df}(t)$ counts the capsules containing it. The weight favors a cue that is common within its capsule but rare across the memory collection. Each candidate occurs in at least one capsule, so both counts are positive. We select $\mathcal{K}_i = \mathrm{Top}_{k_r}(\mathcal{A}_i; w_i)$, where Top_{k_r} returns at most k_r terms. Each selected term is stored once in residual record r_i with its capsule identifier and source references $\mathcal{I}_i$. An inverted index maps these terms to capsule identifiers, so a query can find a capsule even when its identifying cue was removed from c_i. Since $\mathcal{A}_i$ depends on the output c_i, residual addressing works with any compressor using this interface. In Fig. 1(a), the residual cues retain "Bluebird Café", allowing the café name to locate the recommendation.

3.3 Graph Coarsening and Source Recovery

Candidate Graph and Collapse. We form an undirected candidate graph $G = (V, E)$ with one node per capsule. We connect capsules whose compact-content embeddings are similar or whose entities or source metadata overlap. The resulting edges offer possible retrieval paths without asserting factual relations. At step t, let $G_t = (V_t, E_t)$ and let $N_t[u] = \{u\} \cup \{v : (u, v) \in E_t\}$ contain node u and its neighbors. Motivated by the graph strong collapse and graph edge collapse criteria of STPGC (Wu et al., 2026), we remove a node u in favor of a neighbor v when their closed

neighborhoods satisfy

$$N_t[u] \subseteq N_t[v]. \tag{2}$$

This operation deletes u and its incident edges. An existing edge (u, v) can also be deleted when a node $w \notin \{u, v\}$ covers the endpoints' shared neighborhood, giving

$$N_t[u] \cap N_t[v] \subseteq N_t[w]. \tag{3}$$

We update affected neighborhoods after each deletion. Both operations remove nodes or edges already in G and add none. The resulting coarsened graph $S = (V_S, E_S)$ is therefore a subgraph of G. It guides retrieval, while the mappings below preserve access to the original capsules and sources.

Recovery Mappings. Deleting a node from G_t must not make its capsule or source records inaccessible. Each original node i is therefore assigned to a surviving representative $\phi_t(i) \in V_t$, with an attachment path P_i^t in the original graph G. The original nodes assigned to s form its recovery group $\mathcal{G}_t(s) = \{i : \phi_t(i) = s\}$. When u is removed in favor of v, their groups combine as

$$\mathcal{G}_{t+1}(v) = \mathcal{G}_t(v) \cup \mathcal{G}_t(u).$$

The representative mappings and paths form Φ_t. We accept a node collapse only if every updated attachment path has at most h edges in the original graph and each resulting recovery group contains at most $g_{\max}$ original nodes. These bounds keep local recovery from expanding without limit. We also require the serialized graph and its recovery metadata to use less storage after each accepted collapse. Edge deletion leaves the node mappings unchanged, and at retrieval a surviving node exposes the capsules in its recovery group through their source references $\mathcal{I}_i$.

After an initial round of collapse, we select nearby pairs of surviving nodes for semantic verification. For each pair, an LLM checks a bounded set of source records from their recovery groups to see whether the records support a proposed relation and its direction. Node collapse leaves fewer representatives and edge deletion leaves fewer neighboring pairs, so these selective checks avoid examining every candidate relation in a full semantic graph with the LLM. We protect nodes needed by verified relations and repeat the admissible collapses to obtain the final S. Each verified relation and its supporting source identifiers are stored in W as a semantic link for later retrieval, separately from the edges of S. Once S is fixed, we index short paths and stars as structural motifs on this graph (Benson et al., 2016; Huang et al., 2025). Their role during retrieval is to prioritize nearby nodes whose source records may supply complementary evidence.

3.4 Local Retrieval and Evidence Recovery

Entry Retrieval and Local Candidates. Given a query Q, we retrieve entry capsules through two indexes. Vector similarity between the query and compact capsule embeddings provides semantic matches. An inverted index maps query terms, such as names and places, to capsules whose residual addresses contain them. We merge the top candidates from both indexes, remove duplicates, and map each capsule i to its surviving representative $\phi(i)$ to obtain the entry set V_Q^{entry}. These index lookups locate a few memory regions without reading source records or making an LLM call.

Within bounded neighborhoods around the entry nodes, three local retrieval routes rank candidates for source access. The semantic route compares the query with compact capsule embeddings in nearby recovery groups, reusing entry-search scores when available. The lexical route matches query terms against residual addresses in those groups, so omitted cues remain useful after entry selection. The structural route favors graph nodes that share an indexed motif with an entry or visited node. For a connected motif template m, such as a short path or star, its matching subgraphs are $\mathcal{O}_m(S) = \{H \subseteq S : H \cong m\}$, where H has the same node and edge pattern as m. Nodes that complete such a pattern are worth inspecting for complementary evidence. A motif match helps rank candidates but does not establish a factual relation, which must be checked against source records.

Let V_Q^{sem}, V_Q^{lex}, and V_Q^{motif} denote the surviving nodes proposed by the three routes. We merge their candidates and remove duplicate nodes. Motif matches increase priority without excluding semantic or lexical candidates. The local candidate set is

$$V_Q^{\text{cand}} = V_Q^{\text{sem}} \cup V_Q^{\text{lex}} \cup V_Q^{\text{motif}}.$$

Evidence Assessment and Focused Search. We visit a subset $V_Q \subseteq V_Q^{\text{cand}}$ within the graph-search budget. Each visited node s gives access to its recovery group $\mathcal{G}(s)$, forming the pool of original capsule nodes

$$\mathcal{U}_Q^{\text{owner}} = \bigcup_{s \in V_Q} \mathcal{G}(s).$$

The source references $\mathcal{I}_i$ of these capsules locate the corresponding records. We read a budgeted subset to obtain initial evidence $\mathcal{E}_0$, then ask an LLM whether it is sufficient to answer Q. If it is sufficient, we select the answer context. Otherwise, the assessment specifies what is missing, such as an entity, an event, a relation, or a time constraint.

If evidence is insufficient, we use the identified gap to formulate a focused query with the missing entity, event, relation, or time constraint. Within the recovered region, we match its terms against residual addresses and source-text postings, then rank matching records by relevance to the missing information. A residual match identifies the capsule, whose stored source identifiers lead to the corresponding full records. Original-graph connections and previously verified links provide access to nearby sources. This stage searches for specific missing information without repeating global entry retrieval or motif matching.

For example, a query asks whether Alex bought the book Maya recommended at Bluebird Café. Initial retrieval finds the recommendation of The Overstory, but no purchase evidence. The focused query then searches the same recovered region for Alex, The Overstory, and a later purchase event, opening relevant source records rather than searching every memory again. Let $\mathcal{X}_Q$ be the eligible local source records and $\mathcal{X}_Q^{\text{read}} \subseteq \mathcal{X}_Q$ those actually read within the source-access budget. An LLM checks the proposed recommendation-to-purchase relation against these records. We retain a link only when the text supports it, together with the identifiers of its supporting records. Such links connect previously unlinked memories and guide later searches when reuse is enabled.

The evidence selector orders the candidate source records as $e_1, \ldots, e_K$. After resolving unambiguous timestamps through source identifiers, we retain the longest complete-record prefix within the evidence budget B_E according to

$$k^* = \max\{0 \le k \le K : \text{cost}(e_1 \| \cdots \| e_k) \le B_E\}. \tag{4}$$

The selected records form the answer context $E_B = e_1 \| \cdots \| e_{k^*}$, where the empty prefix has zero cost. The answering model then generates $\hat{Y} \sim p_\theta(\cdot \mid Q, E_B)$ using a fixed prompt and output parser for each evaluated model configuration.

4 Experiments

4.1 Experimental Setup

Datasets and Metrics. Following SimpleMem (Liu et al., 2026), we evaluate DuCoMem on LoCoMo and LongMemEval-S across multiple LLM backends. LoCoMo evaluates recall and reasoning over long conversations (Maharana et al., 2024). Its four-category evaluation contains 1,540 questions covering MultiHop, Temporal, OpenDomain, and SingleHop, with 446 adversarial questions excluded. LongMemEval-S covers six types of long-term memory questions, including temporal reasoning and knowledge updates (Wu et al., 2025). Tables 1–3 retain the published baseline metrics. We report F1 and BLEU-1 for each LoCoMo category and their unweighted four-category means. Category 5 is excluded from these means. For LongMemEval-S, we report per-category accuracy and the unweighted mean across six categories. Scoring details appear in Appendix A.1.

Baselines. The comparison includes full-context input and ReadAgent (Lee et al., 2024), MemGPT (Packer et al., 2023), A-Mem (Xu et al., 2025), LightMem (Fang et al., 2026), Mem0 (Chhikara et al., 2025), and SimpleMem (Liu et al., 2026) where results are available. We retain their published scores as reported in the SimpleMem comparison (Liu et al., 2026).

Evaluation Protocol. For LoCoMo, we report F1, BLEU-1, and the query-time Token Cost metric used by SimpleMem (Liu et al., 2026). We refer to this metric as retrieval cost in Fig. 3 to distinguish it from memory-construction cost, which is reported separately. For LongMemEval-S, we report

Table 1: LoCoMo results with high-capability models (GPT-4.1-mini, GPT-4o, and Qwen3-Plus). Average is the unweighted mean across four categories.

Model	Method	MultiHop		Temporal		OpenDomain		SingleHop		Average		Token Cost
		F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	
GPT-4.1-mini	LoCoMo	25.02	21.62	12.04	10.63	19.05	17.07	18.68	15.87	18.70	16.30	16,910
	ReadAgent	6.48	5.60	5.31	4.23	7.66	6.62	9.18	7.91	7.16	6.09	643
	MemGPT	17.72	16.02	19.44	16.54	11.29	10.18	25.59	24.25	18.51	16.75	16,977
	A-Mem	25.06	17.32	51.01	44.75	13.22	14.75	41.02	36.99	32.58	28.45	2,520
	LightMem	24.96	21.66	20.55	18.39	19.21	17.68	33.79	29.66	24.63	21.85	612
	Mem0	30.14	27.62	48.91	44.82	16.43	14.94	41.30	36.17	34.20	30.89	973
	SimpleMem	**43.46**	**38.82**	58.62	50.10	19.76	18.04	51.12	43.53	43.24	37.62	531
	DuCoMem	40.28	30.46	**60.84**	**53.63**	**35.63**	**30.25**	**62.11**	**55.72**	**49.72**	**42.52**	**502**
GPT-4o	LoCoMo	28.00	18.47	9.09	5.78	16.47	14.80	**61.56**	**54.19**	28.78	23.31	16,910
	ReadAgent	14.61	9.95	4.16	3.19	8.84	8.37	12.46	10.29	10.02	7.95	805
	MemGPT	30.36	22.83	17.29	13.18	12.24	11.87	40.16	36.35	25.01	21.06	16,987
	A-Mem	32.86	23.76	39.41	31.23	17.10	15.84	44.43	38.97	33.45	27.45	1,216
	LightMem	28.15	21.83	36.53	29.12	13.38	11.54	33.76	28.02	27.96	22.63	645
	Mem0	35.13	27.56	52.38	44.15	17.73	15.92	39.12	35.43	36.09	30.77	985
	SimpleMem	35.89	**32.83**	**56.71**	20.57	18.23	16.34	45.41	39.25	39.06	27.25	550
	DuCoMem	**41.85**	32.65	52.94	**47.11**	**23.26**	**19.84**	58.92	52.20	**44.24**	**37.95**	**523**
Qwen3-Plus	LoCoMo	24.15	18.94	16.57	13.28	11.81	10.58	38.58	28.16	22.78	17.74	16,910
	ReadAgent	9.52	6.83	11.22	8.15	5.41	5.23	9.85	7.96	9.00	7.04	742
	MemGPT	25.80	17.50	24.10	18.50	9.50	7.80	40.20	42.10	24.90	21.48	16,958
	A-Mem	26.50	19.80	46.10	35.10	11.90	11.50	43.80	36.50	32.08	25.73	1,427
	LightMem	28.95	24.13	42.58	38.52	16.54	13.23	40.78	36.52	32.21	28.10	606
	Mem0	32.42	21.24	47.53	39.82	17.18	14.53	46.25	37.52	35.85	28.28	1,020
	SimpleMem	33.74	29.04	50.87	43.31	18.41	16.24	46.94	38.16	37.49	31.69	**583**
	DuCoMem	**38.69**	**31.91**	**54.61**	**49.38**	**25.36**	**23.17**	**59.20**	**54.21**	**44.46**	**39.67**	597

accuracy by question type. Each model uses a fixed prompt, output format, and parser. The ablation variants use the same model, question set, and evidence budgets as full DuCoMem. Scoring and implementation details appear in Appendices A.1 and B.1.

4.2 Results and Analysis

LoCoMo. Tables 1 and 3 report LoCoMo results across seven LLM backends spanning different model scales. DuCoMem consistently improves macro F1 over SimpleMem across all seven backends. For GPT-4.1-mini, GPT-4o, and Qwen3-Plus in Table 1, the gains are 6.48, 5.18, and 6.97 points, respectively. GPT-4.1-mini reaches 49.72 F1 and 42.52 BLEU-1, compared with 34.20 F1 for Mem0 and 43.24 for SimpleMem. GPT-4o achieves 44.24 F1, versus 36.09 for Mem0 and 39.06 for SimpleMem. These gains vary across question types and GPT-4o remains below SimpleMem on Temporal F1 despite its higher four-category average.

Table 3 further shows that the improvements persist with smaller models. Across Qwen2.5-1.5B, Qwen2.5-3B, Qwen3-1.7B, and Qwen3-8B, DuCoMem improves macro F1 over SimpleMem by 5.72, 11.46, 6.16, and 2.53 points, respectively, while BLEU-1 also improves for all four models. With Qwen2.5-1.5B, DuCoMem reaches 30.95 F1, compared with 23.77 for Mem0 and 25.23 for SimpleMem. The largest gain appears with Qwen2.5-3B, driven mainly by improvements on Temporal and SingleHop questions, although its OpenDomain F1 remains slightly below SimpleMem.

LongMemEval-S. Table 2 reports LongMemEval-S results across six question types for GPT-4.1-mini and GPT-4.1. DuCoMem achieves macro accuracies of 81.29% and 87.40%, improving over SimpleMem by 4.42 and 3.43 points, respectively. The corresponding gains over Mem0 are 21.48 and 28.89 points. Performance varies across question types: with GPT-4.1-mini, DuCoMem gains most on Single-Session-Assistant questions but remains below SimpleMem on Temporal and Knowledge-Update questions. With GPT-4.1, it improves on four of the six categories, matches SimpleMem on Single-Session-User questions, and remains lower on Preference questions.

4.3 Compression and Cost Analysis

Fig. 3(a) examines whether compressed memories remain retrievable as the content budget decreases. At matched compression rates, our residual addressing consistently preserves source access

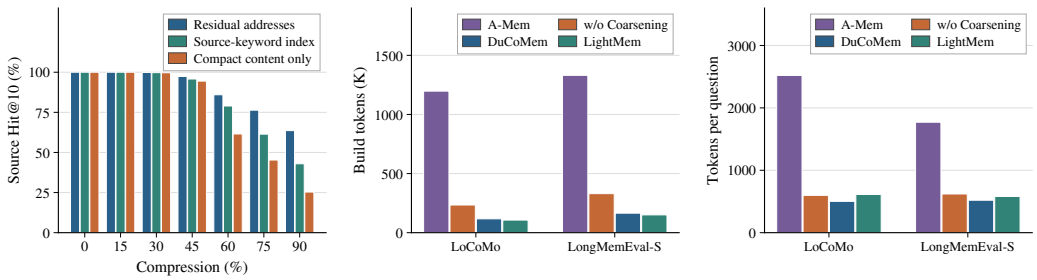
Table 2: LongMemEval-S accuracy in percent. Average is the unweighted mean across six categories.

Model	Method	Temporal	Multi-Session	Knowledge Update	Single-Session User	Single-Session Assistant	Single-Session Preference	Average
GPT-4.1-mini	Full-context	27.06	30.08	41.03	47.14	32.14	60.00	39.57
	Mem0	40.60	50.37	69.23	87.14	48.21	63.33	59.81
	LightMem	**85.71**	47.37	**92.30**	88.57	21.43	76.67	68.67
	SimpleMem	83.46	60.92	79.48	85.71	75.00	76.67	76.87
	DuCoMem	71.43	**63.91**	71.79	**95.71**	**98.21**	**86.67**	**81.29**
GPT-4.1	Full-context	51.88	39.10	70.51	65.71	**96.43**	16.67	56.72
	Mem0	43.61	54.89	75.64	54.29	39.29	**83.33**	58.51
	LightMem	84.96	57.89	**89.74**	87.14	71.43	70.00	76.86
	SimpleMem	86.47	81.20	80.76	**98.57**	76.79	80.00	83.97
	DuCoMem	**87.22**	**82.71**	84.62	**98.57**	94.64	76.67	**87.40**

Table 3: LoCoMo results with smaller models (Qwen2.5-1.5B/3B and Qwen3-1.7B/8B). Average is the unweighted mean across four categories.

Model	Method	MultiHop		Temporal		OpenDomain		SingleHop		Average		Token Cost
		F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	
Qwen2.5-1.5B	LoCoMo	9.05	6.55	4.25	4.04	9.91	8.50	11.15	8.67	8.59	6.94	16,910
	ReadAgent	6.61	4.93	2.55	2.51	5.31	12.24	10.13	7.54	6.15	6.81	752
	MemGPT	10.44	7.61	4.21	3.89	13.42	11.64	9.56	7.34	9.41	7.62	16,953
	A-Mem	18.23	11.94	24.32	19.74	16.48	14.31	23.63	19.23	20.67	16.31	1,300
	LightMem	16.43	11.39	22.92	18.56	15.06	11.23	23.28	19.24	19.42	15.11	**605**
	Mem0	20.18	14.53	27.42	22.14	19.83	15.68	27.63	23.42	23.77	18.94	942
	SimpleMem	21.85	16.10	29.12	23.50	**21.05**	**16.80**	28.90	24.50	25.23	20.23	678
	DuCoMem	**27.02**	**18.14**	**35.49**	**26.73**	19.44	15.66	**41.85**	**35.01**	**30.95**	**23.88**	662
Qwen2.5-3B	LoCoMo	4.61	4.29	3.11	2.71	4.55	5.97	7.03	5.69	4.83	4.67	16,910
	ReadAgent	2.47	1.78	3.01	3.01	5.57	5.22	3.25	2.51	3.58	3.13	776
	MemGPT	5.07	4.31	2.94	2.95	7.04	7.10	7.26	5.52	5.58	4.97	16,961
	A-Mem	12.57	9.01	27.59	25.07	7.12	7.28	17.23	13.12	16.13	13.62	1,137
	LightMem	16.43	11.39	6.92	4.56	8.06	7.23	18.28	15.24	12.42	9.61	605
	Mem0	16.89	11.54	8.52	6.23	10.24	8.82	16.47	12.43	13.03	9.76	965
	SimpleMem	17.03	11.87	21.47	19.50	**12.52**	**10.19**	20.90	18.01	17.98	14.89	**572**
	DuCoMem	**25.80**	**18.56**	**34.29**	**26.71**	12.30	9.31	**45.36**	**37.94**	**29.44**	**23.13**	598
Qwen3-1.7B	LoCoMo	10.28	8.82	6.45	5.78	10.42	9.02	11.16	10.35	9.58	8.49	16,910
	ReadAgent	7.50	5.60	3.15	2.95	6.10	12.45	10.80	8.15	6.89	7.29	784
	MemGPT	11.50	8.20	4.65	4.10	13.85	11.90	10.25	7.85	10.06	8.01	16,954
	A-Mem	18.45	11.80	25.82	18.45	10.90	9.95	21.58	16.72	19.19	14.23	1,258
	LightMem	14.84	11.56	9.35	7.85	13.76	10.59	28.14	22.89	16.52	13.22	**679**
	Mem0	18.23	13.44	18.54	14.22	16.82	13.54	31.15	26.42	21.19	16.91	988
	SimpleMem	20.85	15.42	26.75	18.63	**17.92**	**14.15**	32.85	26.46	24.59	18.67	730
	DuCoMem	**25.43**	**17.64**	**40.22**	**32.42**	15.96	11.71	**41.38**	**34.53**	**30.75**	**24.08**	755
Qwen3-8B	LoCoMo	13.50	9.20	6.80	5.50	10.10	8.80	14.50	11.20	11.23	8.68	16,910
	ReadAgent	7.20	5.10	3.50	3.10	5.50	5.40	8.10	6.20	6.08	4.95	721
	MemGPT	14.20	9.80	5.50	4.20	12.50	10.80	11.50	9.10	10.93	8.48	16,943
	A-Mem	20.50	13.80	22.50	18.20	13.20	10.50	26.80	21.50	20.75	16.00	1,087
	LightMem	18.53	14.23	26.78	21.52	14.12	11.24	29.48	23.83	22.23	17.71	744
	Mem0	22.42	16.83	32.48	26.13	15.23	12.54	33.05	27.24	25.80	20.69	1,015
	SimpleMem	28.97	24.93	**42.85**	**36.49**	15.35	13.90	46.62	40.69	33.45	29.00	621
	DuCoMem	**33.27**	**25.96**	40.20	32.27	**17.74**	**14.28**	**52.71**	**45.71**	**35.98**	**29.55**	**603**

better than retrieval from compact content alone or a source-keyword index. The latter performs lexical retrieval over keywords indexed from the original source records, without explicitly targeting cues removed by compression. Note that at 90% compression, our residual addressing achieves 63.60% source Hit@10, compared with 42.94% for the source-keyword index and 25.32% for compact content alone. This result isolates the role of the proposed residual cues that information omitted from the compact representation can still provide useful addresses for locating the corresponding source. Fig. 3(b) and (c) compare DuCoMem with A-Mem and LightMem as representative structured and lightweight memory methods. A-Mem incurs substantially higher cost, whereas Light-



(a) Source retrieval (b) Construction cost (c) Retrieval cost

Figure 3: Compression and cost diagnostics. (a) Source Hit@10 after content compression. (b) Construction tokens per conversation. (c) Retrieval Token Cost per question.

Mem provides a low-cost reference. DuCoMem remains closer to the lightweight regime because it constructs a low-cost candidate graph and applies LLM verification only to selected relations, rather than relying on exhaustive LLM-based graph construction. Coarsening further reduces the relations exposed to LLM checks, while bounded local recovery controls query-time cost. Thus, even w/o Coarsening remains substantially cheaper than A-Mem, and the full method further approaches lightweight-memory efficiency. Consistent with this retrieval-side effect, Tables 1 and 3 show that DuCoMem remains within the token-cost range of compact-memory baselines across model scales. For GPT-4.1-mini and GPT-4o, DuCoMem has retrieval costs of 502 and 523 tokens, respectively, compared with 531 and 550 tokens for SimpleMem. Across the seven models, DuCoMem achieves the reported accuracy gains with comparable token cost, using fewer tokens in four configurations and only slightly more in the remaining three.

4.4 Component Ablation

Table 5 in Appendix A.2 reports ablations on LoCoMo using GPT-4.1-mini. *w/o Information Compression* replaces compact capsules and residual addresses with full summaries and ordinary indexes. *w/o Graph Coarsening* uses the candidate graph without node or edge collapse. *w/o Focused Search* answers from initial evidence without follow-up source search. These variants lower macro F1 by 4.61, 0.94, and 4.20 points, respectively, with Token Costs of 557, 598, and 483 versus 502 for the full system. The results support both compression components, while the focused-search ablation shows that a small token saving can incur a substantial accuracy loss.

5 Limitations

Compared with the index-based compact memories, such as SimpleMem (Liu et al., 2026), DuCoMem maintains additional graph and recovery metadata and performs selective relation verification and navigation. The low-cost candidate construction, pre-verification coarsening, and bounded local recovery can limit this overhead, but the system remains more complex than index-only memory. In addition, our evaluation is limited to two conversational-memory benchmarks and does not yet characterize continuously growing memory in long-running deployments.

6 Conclusion

In this paper, we proposed DuCoMem, an efficient dual compression memory framework that compressed the long-term memory content and the memory graph for LLM agents. Residual addresses stored the key terms discarded by the content compressor, keeping the compressed memory retrievable. A graph compression method was also proposed to coarsen a low-cost candidate graph before LLM relation checking, and keep mappings from the remaining nodes to the original records. During retrieval, DuCoMem started from a few relevant nodes on the compressed graph, read nearby original records, and searched further only when the evidence is insufficient. Experiments on LoCoMo and LongMemEval-S have shown that DuCoMem outperformed all the baselines across seven models with a similar token cost.

AI USE STATEMENT

Generative AI tools were used only to assist with language editing and improving the readability of the manuscript. Generative AI tools were not used for research tasks requiring mandatory disclosure under the policy. All AI-assisted edits were reviewed by the authors, who take responsibility for the final content of this work.

ETHICS STATEMENT

The proposed method is intended for research on memory construction and retrieval. This work studies efficient long-term memory for LLM agents using established public benchmarks and does not involve the collection of new human-subject data.

REPRODUCIBILITY STATEMENT

We provide detailed experimental and implementation information to support reproducibility. Appendix A describes the evaluation protocol, benchmark organization, baseline provenance, scoring procedures, component ablations, and cost accounting. Appendix B specifies the memory construction and retrieval procedure, implementation configuration, search and recovery budgets, and evidence packing.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Austin R Benson, David F Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

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, Huajun Chen, and Ningyu Zhang. Lightmem: Lightweight and efficient memory-augmented generation. In *The Fourteenth International Conference on Learning Representations*, 2026.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 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 *Forty-second International Conference on Machine Learning*, 2025.

Xingyue Huang, Pablo Barcelo, Michael M. Bronstein, Ismail Ilkan Ceylan, Mikhail Galkin, Juan L Reutter, and Miguel Romero Orth. How expressive are knowledge graph foundation models? In *Forty-second International Conference on Machine Learning*, 2025.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 conference of the north american chapter of the association for computational linguistics: Human language technologies (volume 1: Long papers)*, pp. 7036–7050, 2024.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 13358–13376, 2023a.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 7969–7992, 2023b.

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

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 6769–6781, 2020.

Jihee Kim, Subeen Park, Hakyung Lee, YongTaek Lim, Hyo-won Suh, and Kyungwoo Song. Tides: Technical information discovery and extraction system. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21756–21772, 2025.

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. A human-inspired reading agent with gist memory of very long contexts. In *Forty-first International Conference on Machine Learning*, 2024.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for LLM agents. In *Forty-third International Conference on Machine Learning*, 2026.

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, 2024.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981, 2024.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Xufang Luo, Hao Cheng, Dongsheng Li, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Jianfeng Gao. Secom: On memory construction and retrieval for personalized conversational agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

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

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.

Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

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*, 2024.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 10014–10037, 2023.

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

Xiang Wu, Rong-Hua Li, Xunkai Li, Kangfei Zhao, Hongchao Qin, and Guoren Wang. Scalable topology-preserving graph coarsening: Concepts and algorithms. 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.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 17577–17604, 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*, pp. 19724–19731, 2024.

APPENDIX

This appendix is organized into three main sections. Section A provides the experimental protocol and additional analyses, including component ablations and compression and cost diagnostics. Section B describes the implementation and retrieval procedure, including the default configuration, graph construction, search budgets, and recovery constraints. Section C analyzes recoverability under compression and retrieval budgets.

A EXPERIMENTAL DETAILS AND ADDITIONAL ANALYSIS

A.1 EXPERIMENTAL PROTOCOL AND EVALUATION

A.1.1 SHARED EXPERIMENTAL SETTINGS

We follow the SimpleMem experimental setup for the common conversation-processing and evaluation components (Liu et al., 2026). Table 4 makes these shared settings explicit. The reference is the version of SimpleMem reporting both LoCoMo and LongMemEval-S (arXiv:2601.02553v3, Table 7 and Appendix A.4). DuCoMem replaces its memory representation and retrieval procedure with those in Section 3; consequently, SimpleMem's method-specific retrieval depths are not imposed on DuCoMem's graph visits or recovered source records.

Table 4: Shared experimental settings following SimpleMem. Window stride is the number of turns advanced between consecutive windows.

Setting	Configuration
Window size / sliding stride	20 turns / 5 turns
Structured intermediate output	JSON
Embedding model	Qwen3-Embedding-0.6B, 1,024 dimensions
LoCoMo LLM backends	GPT-4.1-mini, GPT-4o, Qwen3-Plus, Qwen2.5-1.5B, Qwen2.5-3B, Qwen3-1.7B, Qwen3-8B
LongMemEval-S LLM backends	GPT-4.1-mini and GPT-4.1
LongMemEval-S judge	GPT-4.1-mini

Following SimpleMem, we evaluate multiple LLM backends across capability scales. The evaluated models are listed in Table 4. Common experimental settings follow SimpleMem, while DuCoMem uses the memory construction and retrieval procedure described in Section 3.

A.1.2 BENCHMARK ORGANIZATION AND BASELINE PROVENANCE

LoCoMo evaluation uses the four non-adversarial categories specified in Section 4, totaling 1,540 questions; the remaining 446 adversarial questions are excluded from the four-category averages. Questions from different conversations do not share a source archive. Session order and available timestamps accompany the original records, allowing temporal evidence to be interpreted relative to its source context.

LongMemEval-S results are grouped into Temporal, Multi-Session, Knowledge-Update, Single-Session-User, Single-Session-Assistant, and Preference questions. Its evaluation concerns answer correctness, whereas LoCoMo additionally reports lexical overlap. The six-category average used here gives each question type equal weight and is not the same statistic as an example-weighted accuracy when category sizes differ.

The baseline entries in Tables 1–3 and Table 2 retain the published comparison results used in the main text. They are not described as new, jointly executed baseline runs. The controlled component comparisons in Table 5, in contrast, use the same DuCoMem evaluation configuration. This distinction matters when interpreting small score differences or attributing cost changes to a single component.

For LoCoMo, we follow the SimpleMem scoring protocol and report F1 and BLEU-1, averaged equally across the four non-adversarial categories.

A.1.3 LONGMEMEVAL-S JUDGING

Following SimpleMem, GPT-4.1-mini judges a generated answer against the question and reference answer. Semantic agreement and equivalent time expressions determine correctness; literal string identity is not required. The judge returns a binary correctness label, from which category accuracy is computed. The reference answer is supplied only to this evaluation stage, not to memory construction, retrieval, or answer generation.

For category $\mathcal{L}_c$, let $z_i \in \{0, 1\}$ denote the parsed correctness label. We report

$$\mathrm{Acc}_c = \frac{100}{|\mathcal{L}_c|} \sum_{i \in \mathcal{L}_c} z_i. \tag{A.1}$$

The unweighted six-category mean is

$$\mathrm{MacroAcc} = \frac{1}{6} \sum_{c=1}^{6} \mathrm{Acc}_c. \tag{A.2}$$

A judgment compares the content of the response, not its length or wording alone. In particular, a paraphrase that identifies the same date can be correct, whereas an unsupported update to a previously stated fact can be wrong despite high lexical overlap.

A.2 COMPONENT ABLATION AND INTERPRETATION

A.2.1 CONTROLLED VARIANTS

All variants use GPT-4.1-mini as the answer model, the same question set, the same precomputed capsule summaries, and the same evidence-budget rule. Each variant changes the named mechanism and retains the remaining processing stages.

Without Information Compression. This variant replaces compact descriptions and their residual addresses with full capsule summaries and ordinary indexes. It tests the content-compression-and-addressing component jointly. Because both the description length and the addressing mechanism change, this row is not an isolated ablation of residual cues. The matched-compression diagnostic in Appendix A.3 addresses source retrieval more directly.

Without Graph Coarsening. This variant retains the original candidate graph without node or edge collapse. Each capsule acts as its own representative, and its recovery group is a singleton. Source access, local ranking, focused search, and answer generation remain available. Semantic links are still checked against source text. The variant tests the role of a reduced navigation structure under the same downstream budgets.

Without Focused Search. This variant selects evidence and generates an answer from the initial retrieval result. It does not perform follow-up source search to resolve missing information. Compression, residual addressing, graph coarsening, and the initial semantic, lexical, and motif proposals remain enabled. It therefore tests the added value of targeted evidence acquisition after initial retrieval.

Table 5: Component ablation on LoCoMo with GPT-4.1-mini. Average gives equal weight to the four question categories.

Configuration	MultiHop		Temporal		OpenDomain		SingleHop		Average		Token Cost
	F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	
Full DuCoMem	40.28	30.46	60.84	53.63	35.63	30.25	62.11	55.72	49.72	42.52	502
w/o Information Compression	32.83	24.18	51.49	44.98	34.59	29.22	61.52	55.15	45.11	38.38	557
w/o Graph Coarsening	37.80	28.37	60.17	53.05	35.27	29.81	61.88	55.49	48.78	41.68	598
w/o Focused Search	38.50	28.96	57.73	50.75	27.30	22.01	58.55	52.31	45.52	38.51	483

A.2.2 Category-Level Effects

Table 5 reports the complete category results. Relative to full DuCoMem, removing information compression reduces macro F1 by 4.61 points. The largest differences occur in Temporal (9.35 points) and MultiHop (7.45 points), while SingleHop changes by only 0.59 points. This pattern is consistent with compact representations and residual cues helping to locate evidence across records under a bounded search budget. It does not imply that compression increases the information contained in the original history.

Removing graph coarsening reduces macro F1 by 0.94 points and increases the reported Token Cost from 502 to 598, a 19.12% increase relative to the full system. The largest degradation occurs on MultiHop, which drops by 2.48 points, while Temporal, OpenDomain, and SingleHop decrease by 0.67, 0.36, and 0.23 points, respectively. This pattern is consistent with graph coarsening primarily reducing navigation overhead while providing a modest accuracy benefit for questions that require traversing multiple related memories. The source archive and recovery mechanism remain important: graph reduction alone does not explain source availability.

Removing focused search reduces macro F1 by 4.20 points while decreasing Token Cost by 19, or 3.78%. The largest category loss is in OpenDomain, at 8.33 points; Temporal and SingleHop lose 3.11 and 3.56 points, respectively. This comparison is consistent with some questions needing evidence beyond the first retrieval pass. It does not show how often the sufficiency check triggers or whether its decisions are calibrated; those quantities require query-level traces.

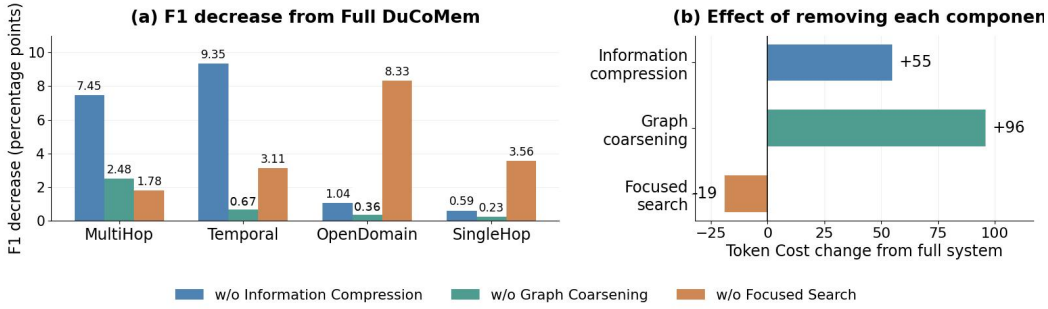


Figure 4: Ablation effects computed from Table 5. Left: decrease in category-level F1 after removing each component, relative to Full DuCoMem. A larger value indicates a larger performance degradation; 0 indicates no change at the reported precision. Right: the change in the reported Token Cost relative to 502. Token Cost follows the retrieval-stage metric used in the main tables.

A.2.3 Scope of the Ablation Evidence

The three rows evaluate component-level interventions, not a factorial decomposition of all internal mechanisms. In particular, they do not isolate motif prioritization, semantic-link reuse, residual capacity, or the evidence-sufficiency prompt. The gains cannot be added to predict the effect of disabling several components simultaneously. The full-method comparison also does not establish an optimal compression ratio or recovery depth.

A.2.4 Residual Address Capacity

Residual capacity k_r controls the number of omitted identifying terms retained for each capsule. A small capacity restricts lexical access to compressed memories, while a larger capacity increases the number of indexed cues. A focused comparison uses $k_r \in \{10, 20, 30\}$ around the default value of 20, with GPT-4.1-mini, the same LoCoMo question set, and fixed compression, graph, and evidence-budget settings. Residual addresses and their indexes must be rebuilt for each capacity. Macro F1 assesses answer quality, while Source Hit@10 assesses source accessibility.

The comparison separates the role of residual capacity from the joint information-compression ablation. Performance and source accessibility should be examined together: better source retrieval need

Table 6: Sensitivity to residual-address capacity on LoCoMo with GPT-4.1-mini. Macro F1 and Source Hit@10 are reported under fixed compression, graph, and evidence-budget settings.

Residual capacity k_r	Macro F1 ↑	Source Hit@10 (%) ↑
10	48.86	82.40
20 (default)	49.72	85.10
30	49.81	85.60

not yield a proportional change in answer quality, because evidence selection and answer generation intervene downstream. This comparison alone cannot establish an optimal capacity across models or datasets.

A.3 Compression Diagnostics and Cost Accounting

A.3.1 Content Retention Versus Source Retrieval

The diagnostic in Fig. 1(b) keeps the original questions unchanged and varies the compact-content budget on a fixed set of question–source pairs. It examines two different events: whether answer words survive compression, and whether the compressed item remains retrievable using the question. Neither diagnostic requires generating an answer.

For pair i, let a_i be the reference answer, $c_i^{(r)}$ the compressed representation at compression setting r, and s_i the associated source identifier. With a fixed word normalization $\mathcal{V}$, answer-word recall is

$$\mathrm{AWR}(r) = \frac{100}{n_D} \sum_{i=1}^{n_D} \frac{|\mathcal{V}(a_i) \cap \mathcal{V}(c_i^{(r)})|}{|\mathcal{V}(a_i)|}, \tag{A.3}$$

where the diagnostic includes only pairs with nonempty normalized answers. Let $\mathcal{R}_{10}(q_i; r)$ be the ten highest-ranked memory items and $\mathrm{src}(m)$ their stored source identifiers. Source retrieval is measured by

$$\mathrm{Hit@10}(r) = \frac{100}{n_D} \sum_{i=1}^{n_D} \mathbf{1}\left[s_i \in \bigcup_{m \in \mathcal{R}_{10}(q_i;r)} \mathrm{src}(m) \right]. \tag{A.4}$$

If a memory item references several records, success is determined by the associated source identifier, not by matching an answer string. The diagnostic unit is a question–source pair, so its denominator must be held fixed across compression settings. The candidate corpus, retrieval procedure, and tie-breaking must likewise remain fixed for the comparison to isolate compression.

The horizontal axis denotes removed content, rather than retained content. Under a fixed length measure ℓ, the achieved corpus-level removal fraction is

$$r_{\mathrm{actual}} = 1 - \frac{\sum_i \ell(c_i)}{\sum_i \ell(p_i)}. \tag{A.5}$$

The unit of ℓ must be consistent before and after compression. A target compressor rate and the achieved removal fraction can differ; the diagnostic should be interpreted using the achieved fraction. Reference answers are used to compute Equation A.3 only after compression.

The source-access comparison in Fig. 3(a) distinguishes compact-content retrieval, a source-keyword index, and residual addressing. The residual index contains only selected cues missing from the compact representation, whereas a source-keyword index can include terms already represented there. This comparison tests whether preserving omitted identifying information supports source access; it does not establish that every recovered source will yield a correct answer.

A.3.2 Construction, Retrieval, and Storage

Following SimpleMem (Liu et al., 2026), we report Token Cost as a retrieval-stage efficiency metric, averaged per question. Memory-construction cost is reported separately, per conversation, as shown in Fig. 3(b).

Serving-memory storage includes compact descriptions, residual addresses, graph structure, recovery mappings, vectors, and auxiliary indexes. Graph compression is evaluated using S together with its recovery metadata Φ, rather than the surviving node count alone. The original source archive is accounted for separately. The storage quantity in Appendix C.1 refers to the serving memory.

A.3.3 Computational Scope

Let n be the capsule count, d the embedding dimension, and Δ a candidate-graph degree bound. A sparse candidate graph requires $O(n\Delta)$ edge storage, while explicit attachment paths of length at most h require $O(nh)$ storage. These terms exclude text and vector indexes. An exact all-pairs embedding comparison would require $O(n^2 d)$ work; graph sparsity alone does not make that construction step linear.

If query navigation visits at most v representatives, at most $vg_{\max}$ original capsules are exposed through their recovery groups before deduplication. This bound concerns capsule ownership rather than source tokens: one capsule may reference multiple records. A separate record or byte budget is therefore needed to bound source access. Only selected representative pairs undergo LLM relation checking. The number and length of these checks, rather than an unqualified graph-size bound, determine generative construction cost.

B Implementation and Retrieval Details

B.1 Implementation Details

B.1.1 Memory Objects and Construction Order

The implementation distinguishes the source archive from the memory used to locate evidence. An archive record contains its original text, stable identifier, speaker, and available session time. A capsule stores a compact description and a list of source identifiers; several capsules may refer to the same record. Recovery mappings refer to capsule identifiers, whereas archive reads and evidence deduplication use source identifiers. Keeping these identifiers separate prevents repeated source text from being counted as distinct evidence after graph expansion.

Table 7 maps the notation in Section 3 to stored objects. Construction proceeds from capsule compression to candidate-graph formation, initial collapse, selective relation checking, and final collapse. Motifs are indexed only after the final graph is fixed. Candidate edges and motif matches provide navigation structure; a factual relation requires separate support from source records.

Table 7: Stored objects and their roles in DuCoMem. The source archive is accounted for separately from the serving memory.

Object	Stored information	Role
X	Original records and session metadata	Recover complete evidence
C	Compact capsule descriptions	Semantic entry retrieval and local ranking
R	Omitted identifying cues with capsule IDs	Lexical access to compressed memories
S	Surviving nodes and navigation edges	Bound the region explored per query
Φ	Representatives, groups, attachment paths	Recover original capsule ownership
W	Indexes, vectors, motifs, verified links	Locate candidates and reuse source-supported relations

B.1.2 End-to-End Procedure

Algorithm 1 summarizes the two stages. Memory construction is query-independent: benchmark questions, reference answers, and annotated evidence are not inputs to capsule formation, residual addressing, or graph coarsening. The reference answer is used only for evaluation. During a query, source identifiers provide the connection between a retrieved capsule and the original evidence.

Algorithm 1 DuCoMem memory construction and retrieval

Construction conversation records X, memory capsules, configuration

1. Retain original source records and their stable identifiers.
2. Apply $\mathcal{C}_{\text{cont}}$ to construct compact representations C, then construct residual addresses R from omitted retrieval cues as in Section 3.2.
3. Build capsule embeddings, source indexes, and the candidate graph G.
4. Apply graph strong collapse and graph edge collapse while retaining bounded attachment paths to obtain an intermediate coarsened graph.
5. Check proposed relations among selected pairs of nearby surviving nodes using their source records. Protect nodes needed by verified links and repeat admissible collapses.
6. Store the final graph S, recovery mappings Φ, and verified semantic links separately.
7. Index motifs on S, complete the auxiliary indexes W, and obtain $M = (C, R, S, \Phi, W)$.

Retrieval query Q, memory M, source archive, budget B

8. Retrieve entry capsules using compact-content vectors and residual-term postings. Merge matches and map them to surviving representatives.
9. Rank candidates in bounded entry neighborhoods using compact-content similarity, residual terms, and motif support. Recover original capsules and read initial source evidence.
10. Assess evidence sufficiency with an LLM. If it is insufficient, identify the missing information and search residual and source-text postings in the local region.
11. Check proposed relations against the records actually read. Store supported connections with their source identifiers when reuse is enabled.
12. Select source evidence under the access and context budgets.
13. Construct the answer context E_B, using Equation 4, with source references and time metadata.
14. Generate an answer from the selected context using the fixed model configuration.

B.1.3 Content Compression and Minimal Method Configuration

The content compressor is LLMLingua-2 (Pan et al., 2024), applied to precomputed capsule summaries. Its output defines the compact representation c_i used by the embedding index. Residual addressing then selects identifying terms absent from c_i, using Equation 1. Changing the compression budget therefore requires rebuilding residual addresses and the associated indexes; keeping residuals from a different compressed representation would change the method being evaluated.

Unless otherwise stated, the values below specify the default execution path of the submitted implementation when no external configuration is supplied. Table 8 records limits for capsule descriptions, residual terms, attachment paths, merged recovery groups, and final evidence construction. These controls apply to different objects and should not be interpreted as interchangeable token or source-access budgets. The graph and search settings described below are shared across the full method and its controlled ablations.

Table 8: Default configuration.

Control	Value	Scope
Compact capsule length	At most 64 whitespace-delimited words	Description after compression
Residual address size k_r	At most 20 distinct terms	Omitted cues per capsule
Attachment bound h	At most 2 hops	Path in the original candidate graph
Recovery group size	At most 64 capsules	Capsules in a merged recovery group
Recovery-group source budget	8,192 bytes	Serialized source budget for a merged recovery group
Final evidence capsule cap	At most 32 records	Capsule-level evidence records before formatting
Formatted evidence budget	At most 896 approximate tokens	Formatting-budget estimate, not a measured tokenizer count

The 8,192-byte constraint applies when forming a merged recovery group rather than to per-query source reading. An individual capsule that exceeds this merge budget can remain as an unmerged recovery group rather than being discarded or forced into a larger merge.

Candidate-graph construction. For each capsule, the embedding-neighbor candidate list uses the default kNN setting $k = 8$. A vector-similarity edge requires cosine similarity of at least 0.30. This threshold applies only to embedding-based similarity edges; shared source references, entity interfaces, and adjacent records within the same session can also induce candidate edges without this threshold. Candidate navigation edges are admitted in descending edge-score order, subject to a degree cap of 16 per node. Separately stored verified semantic links are not counted toward this navigation-graph degree cap. For an ordered list associated with a shared interface, we inspect up to the three preceding and three following entries around the current position.

Relation verification. Node pairs are not sampled at random. On the initially coarsened graph, we first prioritize edges by the sum of their endpoints' degrees, then check reverse-collapse relations and uncovered singleton nodes. Each verification packet contains at most eight capsules, and a capsule may reference multiple original source records. Per conversation history, we allow at most eight verification calls. For each call, the serialized input reserves at most $3{,}072$ bytes, including UTF-8 encoded content and message-formatting overhead, while generation uses `max_tokens=512`. The 3,072 value is therefore an input-serialization reservation rather than a measured model input-token count.

Graph traversal and source access. Initial inspection uses at most two representatives, three compact records, and 32 edge accesses. The shared per-question ceilings are 16 representatives, six hops, 384 edge accesses, and 160 source-bearing capsule reads. Within those global limits, the initial evidence probe uses at most four representatives, 96 edge accesses, and eight source-bearing capsule reads; if the initial evidence is insufficient, focused local search uses the remaining query budget. These quantities are upper bounds rather than fixed per-query usage. Final evidence is additionally limited to at most 32 capsule-level records and approximately 896 formatting-budget tokens before answer generation.

B.1.4 Recovery Invariants and Evidence Packing

Every original capsule retains a valid representative. A proposed collapse is accepted only if the updated attachment paths obey h, recovery groups obey $g_{\max}$, and the serialized graph-plus-mapping representation becomes smaller. The storage comparison includes the new mappings, rather than counting removed edges alone. Edge collapse does not change capsule ownership. Attachment paths are interpreted in the original graph, which is the graph used in Proposition C.2.

At query time, semantic, lexical, and motif proposals are merged before source selection. A motif match can increase priority but does not exclude candidates proposed by the other routes. Source records reached through multiple capsules are deduplicated by identifier. Candidate capsule-level evidence is first constrained by the query-access limits above and the 32-record cap, after which formatting applies the approximately 896-token evidence budget. Evidence packing follows Equation 4: the system takes the longest complete-record prefix that fits the formatting budget. This rule preserves record boundaries and their metadata. If the first record alone exceeds the budget, the feasible prefix is empty; the rule does not silently skip it to choose a different subset.

Recovery-group byte reservations, query-access counts, evidence-budget estimates, and tokenizer counts measure different quantities. In particular, the 8,192-byte merge budget is not a per-query source-reading budget, and the approximately 896-token formatting budget is not a measured tokenizer count. A record cap therefore does not imply a fixed token count. Query-time token usage and construction token usage are recorded separately from these search controls; their interpretation is specified in Appendix A.3.2.

C Recoverability Analysis

The results in this section are general information identities and structural invariants under the stated assumptions. They organize where task information can remain unavailable to the final answer context; by themselves, they do not establish an accuracy or efficiency advantage for DuCoMem over another retrieval system.

C.1 Task Information Under a Retrieval Budget

Let Y denote the target answer, T_B the internal transcript of budgeted access and processing, and E_B the final evidence, including its order and displayed metadata. We analyze a fixed query distribution and fixed model background, with finite conditional entropies. The construction stage samples $M \sim p(m \mid X)$. Budgeted access produces $T_B \sim p_\pi(t \mid M, X, Q)$, and evidence selection produces $E_B \sim p(e \mid M, T_B, Q)$. Access and construction do not consult the target answer. Accordingly, $Y \perp (M, T_B, E_B) \mid (X, Q)$ and $Y \perp E_B \mid (M, T_B, Q)$. The retrieval transcript depends directly on X because DuCoMem can access the original source records. This access is not represented by the simpler chain $X \to M \to E_B$.

Define the task-information gap relative to the complete conversation history as

$$\Delta_B = I(Y; X \mid Q) - I(Y; E_B \mid Q). \tag{C.1}$$

Under the source conditions above, $\Delta_B = I(Y; X \mid E_B, Q)$, so the gap is nonnegative. Equivalently, $\Delta_B = H(Y \mid E_B, Q) - H(Y \mid X, Q)$. The gap accounts for information lost during memory compression, budgeted retrieval, and context construction. To couple recoverability with storage, let $L_{\text{serve}}(M) = L(C, R) + L(S, \Phi) + L(W)$ denote the serialized size of the memory used for retrieval. We define

$$\mathcal{J}_B = \mathbb{E}[L_{\text{serve}}(M)] + \lambda \Delta_B. \tag{C.2}$$

Equation C.2 characterizes the trade-off between memory storage and task-relevant information available after retrieval. We use $\mathcal{J}_B$ to analyze this trade-off without optimizing it during memory construction.

C.2 Preservation, Source Recovery, and Context Exposure

Proposition C.1 (Information-gap decomposition)**.** Under the source conditions stated above,

$$\boxed{\Delta_B = \underbrace{I(Y; X \mid M, Q)}_{\mathcal{D}_{\text{repr}}} - \underbrace{I(Y; T_B \mid M, Q)}_{\mathcal{G}_{\text{read}}} + \underbrace{I(Y; M, T_B \mid E_B, Q)}_{D_{\text{unexposed}}}.} \tag{C.3}$$

Moreover, $0 \le \mathcal{G}_{\text{read}} \le \mathcal{D}_{\text{repr}}$ and $\mathcal{D}_{\text{repr}} - \mathcal{G}_{\text{read}} = I(Y; X \mid M, T_B, Q)$.

The identity follows by expanding $I(Y; M, T_B \mid Q)$ before and after conditioning on the final evidence. A complete proof appears in Appendix C.4.

The first term measures task-relevant information absent from the entire serving memory M, including its auxiliary indexes, rather than from compact descriptions alone. The second measures additional information obtained from source access beyond M. The third measures information present in M or the internal transcript but absent from the final answer context. This unexposed information can arise when retrieval never accesses useful content already stored in M, or when evidence selection, formatting, or truncation omits information that was accessed. The three terms therefore do not uniquely attribute the gap to three implementation stages.

The decomposition also explains why content and structure should be analyzed jointly. The compact memory representations and the coarsened graph together affect which source observations enter T_B and which are retained in E_B. Even when S is derived deterministically from the candidate graph, it changes which sources can be reached under a fixed access budget. Source recovery therefore connects the compressed memory representation to the evidence available for answer generation.

For an adaptive retrieval policy, let A_t be the action and O_t its source observation. If the action uses only available state and independent randomness, then

$$\mathcal{G}_{\text{read}} = \sum_t I(Y; O_t \mid M, Q, T_{<t}, A_t). \tag{C.4}$$

Equation C.4 measures the information gained from each source given the preceding retrieval steps. Two sources may provide useful information only in combination. Relations inferred from them can guide further retrieval and reasoning, while additional task information comes from newly accessed sources.

C.3 Recoverable Paths and Evidence Utilization

Proposition C.2 (Path recovery)**.** Let every original memory node have an attachment path of length at most h to its representative in S. Since S is a subgraph of the original graph, a path of k edges in S between the representatives of original nodes x, y yields a path from x to y in the original graph of length at most $2h + k$.

Concatenating the two attachments and the k edge paths proves the bound, and removing loops can only shorten the resulting route. Proposition C.2 is a conditional structural invariant of the stored attachments and retained navigation edges. Its assumptions do not bound k by the query hop budget or ensure that a retrieval policy discovers the path. Source selection and verification against source text remain separate stages of retrieval.

Finally, preservation of task information must be distinguished from a fixed answering model's ability to use it. For a conditional answer distribution $p_\theta(Y \mid E_B, Q)$, define the expected log loss as $\mathcal{R}_\theta = \mathbb{E}[-\log p_\theta(Y \mid E_B, Q)]$. It decomposes as

$$\mathcal{R}_\theta = H(Y \mid X, Q) + \Delta_B + \mathbb{E} D_{\mathrm{KL}}(p(Y \mid E_B, Q) \,\|\, p_\theta(Y \mid E_B, Q)) \,. \tag{C.5}$$

Here θ denotes the answering model together with its answer configuration. With E_B fixed, changing the answering model affects only the final mismatch term, while Δ_B remains unchanged. This separates evidence recoverability from evidence utilization. The same retrieved information can lead to different answer quality under different context formats or answer instructions. Equation C.5 characterizes this distinction under log loss.

C.4 Proofs and Interpretation

C.4.1 Source Conditions and the Task-Information Gap

All information quantities refer to a population distribution over conversation histories, queries, and answers, conditional on the fixed model background. These quantities are not empirically estimated from the benchmark. Randomness used by construction and retrieval has no access to Y beyond X, Q. The final evidence depends only on the declared internal state and query. If an external information source is introduced, its observations must be included in that state.

The chain rule and $I(Y; M, T_B, E_B \mid X, Q) = 0$ give

$$I(Y; X, E_B \mid Q) = I(Y; X \mid Q),$$
$$I(Y; X, E_B \mid Q) = I(Y; E_B \mid Q) + I(Y; X \mid E_B, Q).$$

Subtracting proves Equation C.1. Moreover, the same conditional independence yields

$$\Delta_B = \mathbb{E}_{X, E_B, Q} D_{\mathrm{KL}}(p(Y \mid X, Q) \,\|\, p(Y \mid E_B, Q)) \,.$$

Thus Δ_B compares the ideal answer distributions available from full conversation history and final evidence. It does not measure how much natural-language text can be reconstructed, nor does it equate lexical F1 with semantic information.

C.4.2 Proof of Proposition C.1

Recall that $\mathcal{D}_{\mathrm{repr}} = I(Y; X \mid M, Q)$. From $I(Y; M \mid X, Q) = 0$,

$$I(Y; X \mid Q) - I(Y; M \mid Q) = \mathcal{D}_{\mathrm{repr}}.$$

Two expansions of the information in the internal state give

$$I(Y; M, T_B \mid Q) = I(Y; M \mid Q) + I(Y; T_B \mid M, Q),$$
$$I(Y; M, T_B \mid Q) = I(Y; E_B \mid Q) + I(Y; M, T_B \mid E_B, Q).$$

The second equality uses $I(Y; E_B \mid M, T_B, Q) = 0$. Substitution establishes $\Delta_B = \mathcal{D}_{\mathrm{repr}} - \mathcal{G}_{\mathrm{read}} + D_{\mathrm{unexposed}}$. Applying the same argument before final context construction shows

$$\mathcal{D}_{\mathrm{repr}} - \mathcal{G}_{\mathrm{read}} = I(Y; X \mid M, T_B, Q).$$

Nonnegativity of the right-hand side proves $\mathcal{G}_{\mathrm{read}} \le \mathcal{D}_{\mathrm{repr}}$. Nonnegativity of $\mathcal{G}_{\mathrm{read}}$ follows from that of conditional mutual information. The result follows from the chain rule under the stated source assumptions. To see why $D_{\mathrm{unexposed}}$ is broader than evidence-packing loss, consider the allowed case $M = X$ with constant T_B and E_B. Then $\mathcal{D}_{\mathrm{repr}} = \mathcal{G}_{\mathrm{read}} = 0$ and $D_{\mathrm{unexposed}} = I(Y; X \mid Q)$: information stored in memory is missed before it can be presented.

C.4.3 Adaptive Source Recovery and Complementarity

Write a complete access transcript as a sequence of actions and observations. For an action chosen from available state with independent randomness,

$$I(Y; A_t \mid M, Q, T_{<t}) = 0.$$

Expanding the transcript by the chain rule leaves the sum of conditional observation gains in Equation C.4. Local transformations of existing observations contribute no additional Shannon information under the same source condition. Variable-length transcripts can be padded after stopping with constant null actions and observations.

The information gained from a source need not decrease as more sources are observed. Let U, V be independent fair bits and $Y = U \oplus V$. Then $I(Y; U) = I(Y; V) = 0$, whereas $I(Y; U, V) = \log 2$. A source that is individually uninformative can become decisive once another source is available. This example illustrates why local recovery seeks complementary evidence.

A controlled evidence replacement has a related identity. If two experimental conditions preserve a common evidence component K and respectively include O or N, then

$$I(Y; K, N \mid Q) - I(Y; K, O \mid Q) = I(Y; N \mid K, Q) - I(Y; O \mid K, Q).$$

The common component must be defined as part of the experiment. A text overlap identified after retrieval need not provide an exact decomposition of the two evidence contexts.

C.4.4 Proof and Scope of Proposition C.2

Let a path in S be $s_0, \ldots, s_k$, where $s_0 = \phi(x)$ and $s_k = \phi(y)$. Concatenate the attachment from x to s_0, the k retained original-graph edges, and the reverse attachment from s_k to y. This is a walk of length at most $h + k + h$. Erasing loops produces a path no longer than the walk.

The bound applies only to valid attachment paths and retained navigation edges. A separately stored semantic relation is not a navigation edge covered by this argument. The assumptions do not relate k to the shortest-path distance in the original graph or to the query hop budget. Even when the path exists, a query policy need not find or follow it, its records may exceed the source budget, and its evidence may not answer the query. This is a conditional reachability statement, not a guarantee of answer accuracy or efficiency.

C.4.5 Answer Generation Loss and Evidence Trimming

For any conditional answer distribution with finite expected log loss,

$$\mathbb{E}[-\log p_\theta(Y \mid E_B, Q)] = H(Y \mid E_B, Q) + \mathbb{E} D_{\mathrm{KL}}(p(Y \mid E_B, Q) \,\|\, p_\theta(Y \mid E_B, Q)) \,.$$

Substituting Equation C.1 proves Equation C.5. If $E' = f(E, Q)$ is a deterministic trimming with no new information, the data-processing inequality gives $I(Y; E' \mid Q) \leq I(Y; E \mid Q)$. This inequality concerns an ideal information measure, not the accuracy of a particular answering model. Distracting context can reduce answer accuracy, so removing it may improve empirical F1. No upper or lower bound on F1 follows from the displayed log-loss decomposition without additional assumptions.