SOLM: SELF-ORGANIZING LATTICE MEMORY FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-term memory for LLM agents requires imposing structure on an ever-growing history, but several recent memory architectures pay for that structure with real, per-message LLM calls: A-Mem generates a structured note and re-evaluates evolution for a set of neighboring memories on every insertion, MemTree re-aggregates content across an average of over three ancestor nodes per insertion, and Zep extracts entities, relations, and deduplicates edges via multiple LLM calls per episode. This construction cost scales with memory volume, tying a deployment's throughput directly to model inference capacity.

SOLM's structure-building (bottom-up placement, quantization-error-driven splitting, cross-branch *weak links*) needs zero LLM calls, computed from embeddings alone. Its only LLM usage: one fixed call per query for generation, and one optional, cheap small-model call per message for importance scoring; this is the sole cost scaling with volume, far lighter than A-Mem, MemTree, or Zep's per-insertion costs. Weak links directly address what we call *aggregation dilution*: a cluster's centroid can fail to resemble a query even when a member buried inside it would match well, causing that whole branch to be wrongly discarded. (A non-generative reranking step narrows and reorders an already-retrieved pool at query time and is not counted here; see Section 3.5.)

On LongMemEval-S, a controlled ablation (mean over 3 independent runs) shows weak links yield a relative accuracy improvement of 7.7% (68.6% to 74.0%) at matched settings; at nearly identical retrieved-context size, SOLM outperforms a flat similarity-retrieval baseline by a relative 13.8% (mean over 3 runs: 76.0% vs. 66.7%); and accuracy scales predictably with the retrieved-context budget: 76.0%, 83.0%, and 86.0% when the top-5, top-10, and top-15 retrieved memories, respectively, are used as answer context. These results show that competitive long-term memory accuracy does not require scaling LLM usage in proportion to memory volume.

1 INTRODUCTION

LLM agents deployed in long-running, multi-session settings cannot rely on the context window alone to remember what has happened, and re-processing the full history at every turn does not scale as conversation history grows. A common alternative is to impose explicit structure on the accumulated history (clustering, summarization, or graph construction) so that only a small, relevant slice needs to be retrieved at query time.

Existing memory systems typically pay for this structure with real LLM calls made during construction, and this cost is often unbounded: A-Mem generates a structured note and re-evaluates a set of existing neighboring memories on every insertion; MemTree re-aggregates content across an average of more than three ancestor nodes per insertion; and Zep extracts entities and relations and deduplicates edges via several LLM calls per ingested episode. Because this cost scales with memory volume, a deployment's throughput becomes tied directly to model inference capacity as users, messages, and concurrent sessions grow.

Separately, hierarchical, centroid-based memory structures introduce a structural risk we call *aggregation dilution*: a cluster's centroid is a lossy aggregate of its members, and top-down retrieval that compares a query only against centroids can prune an entire branch even when a specific member

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.

buried inside it would have matched well. This is not a bug particular to any one system: it is an inherent cost of aggregating many things into one vector, one a purely hierarchical design has no mechanism to recover from once a branch is pruned. Appendix A.6 derives this precisely.

We take the position that a production memory system should neither rely extensively on LLM calls for memory management, nor give up recoverability once a branch has been pruned by centroid similarity alone. SOLM addresses aggregation dilution from both directions, via embedding-only computation: a cluster whose members have grown disproportionately heterogeneous proactively splits into two more internally coherent siblings (Section 3.3), keeping each centroid a faithful aggregate of a narrower group and reducing how often a poorly representative centroid arises in the first place; and *weak links* (cross-branch, same-layer associative connections) give a pruned branch a second, associative path back into contention when a mismatch still occurs despite that.

This paper contributes:

- A self-organizing hierarchical memory lattice combining bottom-up placement, quantization-error-driven growth and splitting, and asymmetric depth-scaled top-down beam search, with minimal LLM usage: one fixed call per query plus an optional per-message small-model call, far cheaper than prior systems' per-insertion calls.
- A characterization of aggregation dilution as a structural risk in centroid-based hierarchical retrieval, and weak links as a zero-LLM-call mitigation.
- A controlled ablation on LongMemEval-S, run 3 times independently per condition: isolating weak links' effect (7.7% relative mean accuracy improvement, non-overlapping ranges), a matched-context comparison against flat similarity retrieval (13.8% relative improvement, likewise non-overlapping), and how accuracy scales with retrieved-context budget.
- As a secondary, less controlled comparison (six of seven published systems share a GPT-4o-mini backbone; SOLM uses Gemini Flash 3.5, without thinking or reasoning) against seven published memory systems on the same split, plus a per-category analysis: SOLM's accuracy matches the strongest of them despite far fewer LLM calls during memory construction, with single-session-preference recall its main weakness.
- A preliminary accuracy result on a 120-question LongMemEval-M subset, to our knowledge the first reported among the systems compared here (Section 5.5).

2 Related Work

Recent agentic memory systems consistently pay their LLM cost at construction time, and that cost is often neither fixed nor bounded. A-Mem generates a full structured note (context, keywords, tags) for every memory and separately re-evaluates a pool of nearest-neighbor memories for possible evolution on every insertion, incurring multiple additional LLM calls beyond the note-generation step itself, scaling with the configured neighbor-pool size [5]. MemTree updates its memory structure along the root-to-leaf traversal path identified for each new piece of information, requiring repeated LLM calls to re-aggregate content at each ancestor node visited, a cost that scales with tree depth [6]. Zep builds a temporal knowledge graph via several distinct LLM-based steps per ingested episode (entity extraction with a reflection step, entity resolution, fact extraction, and fact/edge deduplication), though the exact number of calls per episode is not reported [7]. MemOS treats memory as a managed system resource with scheduling, layering, and lifecycle governance for its MemCube abstraction, but likewise does not quantify the LLM-call cost of constructing or consolidating a MemCube [8]. None of these four systems reports its construction-time LLM-call cost as a fixed, bounded quantity independent of memory volume.

LongMemEval [4] is the benchmark against which we evaluate SOLM throughout this paper: it tests five core long-term memory abilities (information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention) and defines two standard settings, LongMemEval-S (roughly 115K tokens per haystack) and LongMemEval-M (roughly 500 sessions and 1.5M tokens per haystack); we evaluate on the former. Of the four systems above, Zep and MemOS both evaluate on LongMemEval specifically; we report their published numbers, alongside several further systems reported in MemOS's own comparison table, in Table 4 (Section 5).

3 METHOD

SOLM represents every unit of memory, a raw ingested message and every cluster above it, as a single node type spanning layers 0 through n. A layer-0 node holds one raw memory: its text content and embedding centroid are fixed forever at creation, and it is never merged, split, or otherwise touched again. A layer-≥ 1 node is a cluster: its centroid is the weighted mean of its children's centroids, and its child list points at layer-0 memories, other clusters, or a mix of both. The depth cap n is a configuration parameter fixed in advance by the operator, not learned or adjusted by the algorithm itself; in practice, every branch grows to reach exactly this ceiling regardless of how few memories it ends up containing. Section 4.3 discusses a simple heuristic motivating our choice of n.

3.1 IMPORTANCE SCORING AND THE LAYER-n REFERENCE

At insertion, a single small-language-model call scores the new memory's importance on a fixed integer scale from its content alone. If that score meets or exceeds a configured threshold, a second, independent node is created directly at the top layer n, with the new memory as its only child and its centroid copied exactly from that memory's own embedding rather than produced by any averaging process. This reference node competes normally in later insertions and in retrieval, like any other top-level node: importance alone does not force a memory into retrieved context, since relevance to the query still governs what is ultimately retrieved. The memory's own parent pointer is left pointing at its natural cluster parent, unchanged, so a sufficiently important memory becomes reachable via two independent paths, its ordinary cluster placement and this direct, layer-skipping shortcut. This is the only point in the pipeline besides final answer generation where a model call occurs.

This scoring call is optional: disabling it (fixing every message to a constant score, or setting the threshold above the scale's maximum) removes the layer-n shortcut entirely, leaving ordinary cluster placement as the only path to retrieval. We treat this as an architectural option rather than a fixed requirement primarily for scalability: Section 5.5 shows real per-message scoring becoming intractable as haystack size grows, since its cost scales with message count while the rest of the pipeline (Section 3.5) does not. A secondary, more limited observation from Table 7 (Appendix A.4) is consistent with this option being viable on LongMemEval-S: disabling the mechanism entirely lands on the same Overall accuracy as our reported default, though the two conditions disagree on 28 of 470 individual questions underneath that tie, so this is not evidence that the mechanism has no effect, only that removing it is not catastrophic at this particular operating point. We discuss this tradeoff further in Section 6.

3.2 BOTTOM-UP PLACEMENT AND CENTROID PROPAGATION

A new memory is always created as a fresh layer-0 node. Placement then proceeds recursively one layer up, starting from layer 1: at each layer, the new node's centroid is compared by cosine similarity against existing clusters at that layer, and it joins the most similar one if that similarity clears a join threshold, or becomes a brand-new standalone cluster node otherwise.

If it joins an existing cluster, that cluster's centroid is recomputed as a count-weighted average incorporating the new member: $c \leftarrow \frac{n_c \cdot c + v_{\text{new}}}{n_c + 1}$, where n_c is the cluster's member count before the update. Because the parent's own centroid has now shifted, the same join-or-create decision repeats one layer further up using this updated centroid. This propagation continues layer by layer until it either creates a new top-level cluster or reaches layer n itself, so a single insertion can touch every layer of its branch in one pass. This entire cascade is pure vector arithmetic and involves no LLM call at any point. Figure 1 illustrates the resulting lattice, including the weak links and layer-n reference introduced in the following subsections.

3.3 CLUSTER SPLITTING

Each cluster maintains a decayed quantization-error accumulator qe, updated on every new member as $qe \leftarrow \lambda \cdot qe + (v_{\text{new}} - c_{\text{old}})^2$, where c_{old} is the cluster's centroid before the new member is folded in and λ is a decay rate. A cluster is only checked for splitting once it has accumulated at least a minimum number of members μ; below that size, a small sample's qe is not yet a meaningful signal of heterogeneity, and the check is skipped regardless of the cluster's current qe. Once a cluster has

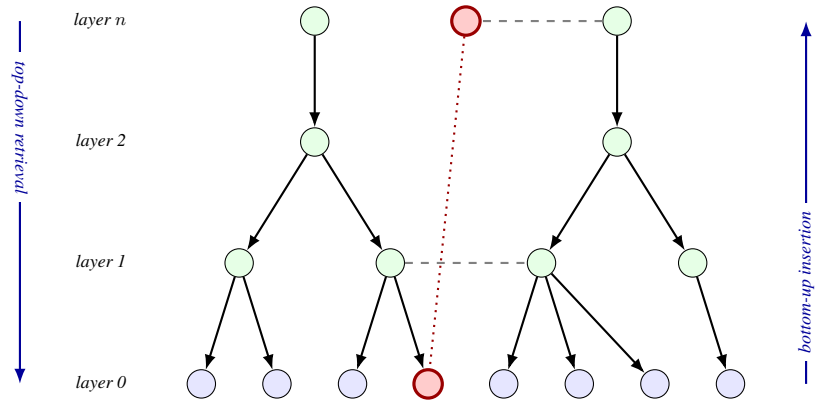


Figure 1: SOLM lattice structure. Solid arrows are strong parent-child links (vertical, built bottom-up); dashed gray edges are weak links (horizontal, cross-branch, same layer); the dotted red edge is a layer-n reference, created only when a memory's importance score clears the threshold, skipping directly from the top layer to that layer-0 memory. Every edge shown is constructed by vector arithmetic alone, except the layer-n reference edge, which depends on the importance-scoring model call (Section 3.1).

reached μ members, it splits once $qe_{\text{node}} > \tau \cdot qe_{\text{parent}}$, i.e., once it has become disproportionately more heterogeneous than its own parent. A split partitions the cluster's members into two count-weighted-coherent groups via a distance-seeded 2-means procedure; the original node keeps one group and its identity, and a new sibling node, at the same layer under the same parent, takes the other. Top-level clusters, having no parent centroid to compare against, are never split.

3.4 Weak Links

Weak links are formed in an explicit batch pass rather than inline during insertion, since forming links against a still-changing lattice couples linking decisions to structure that has not yet stabilized. In a continuously-running production deployment, this same batch pass would be invoked on a periodic or as-needed basis, for example as a maintenance-window job triggered against whatever has accumulated so far. In our evaluation setting, by contrast, each haystack is a bounded batch of messages known in advance, so we invoke this pass exactly once, after every message in that haystack has been ingested, rather than on a recurring schedule. Within this pass, nodes at each layer are processed in descending order of their own qe: this prioritizes the clusters most likely to be hiding a buried member that centroid-based descent would prune past, letting the nodes with the most to gain from a weak link claim their candidates first, before the scarce per-layer slots run out.

For each node, candidates are drawn from the same-layer nearest neighbors of its centroid, filtered to cross-branch pairs (confirmed by walking parent pointers to the root) exceeding a threshold τ_{weak}, intentionally looser than the join threshold since a weak link is an associative hint rather than a placement decision. Among qualifying candidates, order is refined via Reciprocal Rank Fusion (RRF) across two axes: similarity rank and a heterogeneity rank given by the candidate's own qe, descending. For a candidate c,

$$\text{RRF}(c) = \frac{1}{k_{\text{rrf}} + \text{rank}_{\text{sim}}(c)} + \frac{1}{k_{\text{rrf}} + \text{rank}_{qe}(c)},$$

where $\text{rank}_{\text{sim}}(c)$ is c's rank among the qualifying candidates by cosine similarity to the node's centroid (rank 1 being most similar), $\text{rank}_{qe}(c)$ is c's rank by its own qe (rank 1 being highest), and k_{rrf} is a damping constant. Candidates are then ordered by descending $\text{RRF}(c)$. τ_{weak} remains a hard gate throughout this process, and fusion only reorders already-qualifying candidates. Each node's link count is capped at p_{weak} per layer. Weak-link formation costs zero LLM calls: every signal used, embeddings and the qe accumulator, is already available without a model call. Weak links are a mandatory part of the architecture we propose for deployment, not a tunable design choice the way importance scoring is (Section 3.1); Section 5 disables them only for a controlled ablation measuring their contribution, not to suggest they are optional in practice.

3.5 Retrieval and Answer Generation

Retrieval descends the lattice top-down, re-shortlisting at each level by cosine similarity to the query. A beam-width parameter k sets the shortlist size at the top layer; the shortlist widens quadratically toward layer 0, with width $(n - \ell + 1)^2 \cdot k$ at layer ℓ (k at the top layer, $\ell = n$; largest just above the raw memories, $\ell = 1$), since pruning mistakes near the top, where a whole branch can be discarded on one centroid comparison, are costlier than mistakes near the bottom, where each node covers far fewer memories. At each layer, surviving nodes pull in weak-linked neighbors via bounded breadth-first traversal, controlled by a hop limit h ($h = 1$ adds only directly weak-linked neighbors; larger h chains further before children are collected), bounded because weak links are an escape hatch, not a graph to explore exhaustively.

Child expansion checks each child's actual layer rather than assuming it sits exactly one layer below its parent: a layer-n reference node's (Section 3.1) single child is a layer-0 memory several layers down, and checking its real layer lets that memory be collected immediately, skipping every intermediate layer, via the same expansion step used for ordinary descent.

Layer-0 memories collected across the descent are ranked by cosine similarity to the query, and only the top $r \cdot m$ of them (rerank-pool multiplier r) are passed to a cross-encoder reranker, with the rest discarded. The reranker scores the query and each candidate jointly, and this score replaces cosine order for selecting the final top-m within that small, fixed-size pool, bounding cost regardless of how many memories survived the descent. A candidate too long for the reranker's token budget is split into chunks overlapping by o tokens, each scored independently, with the highest-scoring chunk standing in for the whole candidate so nothing is silently truncated; the overlap keeps a boundary-straddling sentence from being cut from every chunk. This reranker is a small, non-generative scoring model, not an autoregressive LLM call, so its cost (a deterministic forward pass per candidate, sub-second on CPU for the pool sizes used here) is not counted toward the two-call model described below. Like weak links, reranking is a mandatory part of the architecture we propose for deployment, not a tunable design choice the way importance scoring is (Section 3.1).

The top-m memories selected this way, each paired with its original timestamp, are assembled into a single prompt alongside the query and handed to one LLM call to produce the final answer. Across the whole pipeline, at most two points make a model call: the small-model importance-scoring call described above, and this one call per query to generate the final answer. Every other operation described in this section is deterministic vector arithmetic, with a cost that depends only on embedding dimensionality and the number of nodes touched, not on model inference capacity. Since importance scoring is optional (Section 3.1), this is a two-call model when enabled and a one-call model (generation only) otherwise; the scoring call is also the only one of the two whose per-run cost scales with haystack size, since it runs once per ingested message rather than once per query.

4 Experimental Setup

4.1 Dataset, Pipeline, and Models

We evaluate on LongMemEval-S [4] (Section 2): 500 questions spanning six types, 30 of which are abstention variants requiring the system to recognize insufficient information; we report the 470 regular and 30 abstention questions' accuracy separately. Each question has its own independent haystack, so we build a fresh SOLM instance per question: every message is ingested, weak links are built once ingestion completes (Section 3), the question is retrieved against, an answer generated, and judged against ground truth via LongMemEval's official judge prompt (a separate branch handles the 30 abstention questions).

Five models fill five non-overlapping roles: Qwen2.5-7B (local, via Ollama) scores each ingested message's importance (Section 3.1); Gemini Flash 3.5 (without thinking / reasoning), run at temperature 0, generates the answer; a BAAI/bge-reranker-base cross-encoder performs reranking (Section 3); a Claude Sonnet 4.5 model, likewise run at temperature 0 via an internal inference gateway, judges generated answers against ground truth using LongMemEval's official prompt verbatim; and a sentence-transformer model embeds messages and queries for both SOLM and the vanilla-RAG baseline.

Although the generation and judge models are both run at temperature 0, hosted LLM APIs do not guarantee bitwise-deterministic output at this setting, due to non-deterministic GPU kernel execution order and provider-side batching; this is the source of the run-to-run variance reported in Sections 5.1 and 5.3, not any non-determinism in lattice construction or retrieval, which is deterministic vector arithmetic (Section 3).

4.2 Baseline

We compare against a vanilla-RAG baseline sharing every SOLM component except the memory structure itself: the same embedding model embeds every haystack message once, the top-m by flat cosine similarity to the query are retrieved (no lattice, clusters, weak links, importance scoring, or reranking), and the same generation model produces the answer. Holding embeddings and generation fixed isolates the retrieval mechanism, hierarchical lattice versus flat similarity search, as the only variable tested.

4.3 Hyperparameter Selection

Hyperparameter values in Table 6 were primarily set through domain intuition and informal estimation prior to systematic evaluation, rather than through iterative tuning or grid search against this benchmark. We did not conduct a systematic search process against the evaluation set, though we cannot rule out minor adjustments to individual values during early development. We report this plainly rather than implying a fully controlled selection process; these values should be treated as unvalidated defaults, and a practitioner deploying SOLM on a different workload should expect to retune them against representative data.

Every Table 6 value besides the depth cap n (set via this section's heuristic per dataset scale) was fixed using only LongMemEval-S and applied unchanged to LongMemEval-M (Section 5.5); importance scoring itself is additionally disabled for that run for the infrastructural reason given there. Accuracy holding up despite the $\sim$10x larger haystack and this second change is a preliminary check against overfitting to S, not formal validation.

One exception to this manual selection process is the depth cap n: for a balanced hierarchical structure with branching factor b, the expected number of layers needed to hold N items follows $n \approx \lceil \log_b(N) \rceil$. Using $\mu = 10$ (Table 6) as a proxy for a typical branching factor and $N \approx 500$ messages per LongMemEval-S haystack, this gives $\lceil \log_{10}(500) \rceil = 3$ layers, corresponding to a top-layer index of $n = 2$ under our 0-indexed convention (Section 3), matching our configured value. We offer this as a rough consistency check rather than a rigorous derivation, since it assumes a perfectly balanced lattice that our join-threshold and splitting dynamics do not guarantee in practice.

Hyperparameter-selection methodology is also inconsistently reported among the systems in Table 4: Zep [7], MIRIX [1], Mem0 [2], and MemTree [6] state only final values with no selection process disclosed; A-Mem [5] instead reports a sensitivity sweep against the same benchmark used for its final results, not a held-out split, and uses that sweep to select its deployed value. Our reported values, including $\theta_{\text{imp}} = 8$, were all set the same informal way described above, consistent with the former pattern; the sensitivity sweep in Table 7 was added afterward as a check on that already-fixed value, not as the mechanism that chose it, and so does not make our process resemble A-Mem's. We flag this as a limitation shared with, not unique to, our work.

5 Results

5.1 Effect of Weak Links

Table 1 isolates the effect of weak links with every other setting held fixed ($k = 5, m = 5$), reporting the Overall column as the mean (and range) across 3 independent runs to characterize run-to-run variance. Removing weak links from retrieval costs a mean 5.3 accuracy points on regular questions (68.6% vs. 74.0%), a relative improvement of 7.7% when weak links are present; critically, the two configurations' ranges across all 3 runs (68.3–68.9% off vs. 73.8–74.3% on) never overlap, so this gap is not an artifact of a single lucky run.(variance arises from generation/judge non-determinism

at temperature 0; see Section 4.1) As a complementary paired test, taking a majority vote across the 3 runs per question (denoising each condition to one correctness label per question) and running McNemar's test on the resulting agreement/disagreement pattern rejects the null of no difference ($\chi^2 = 8.80$, $p = 0.0030$, $n = 470$). Appendix A.5 breaks this down further at the retrieval level with session-level Recall@k.

Configuration	SS-User	SS-Asst	SS-Pref	Multi-S	Know-Upd	Temp-Reas	Overall	Abst.
Weak links off	88.5%	94.6%	52.2%	49.0%	89.4%	58.0%	68.6% (68.3–68.9)	83.3%
Weak links on	92.2%	97.6%	51.1%	57.0%	93.1%	65.1%	**74.0% (73.8–74.3)**	83.3%

Table 1: Effect of weak links on LongMemEval-S at $k = 5$, $m = 5$, every other setting held fixed. Every column reports the mean across 3 independent runs; Overall additionally shows the range. Abst. = abstention accuracy; other column abbreviations as in Table 4.

5.2 Effect of Retrieved-Context Budget

Table 2 varies m at $k = 20$ with weak links on, alongside the average number of context tokens each configuration actually assembles for answer generation (measured directly from the retrieved memories returned per question, not estimated). Accuracy increases monotonically with m, but with diminishing returns: moving from $m = 5$ to $m = 10$ costs roughly 1,570 additional tokens per query for a 7-point gain, while moving from $m = 10$ to $m = 15$ costs a similar additional token budget (roughly 1,650 tokens) for only a 3-point gain.

m	Ctx(tok)	SS-User	SS-Asst	SS-Pref	Multi-S	Know-Upd	Temp-Reas	Overall	Abst.
5	1,381	94.3%	94.6%	46.7%	63.2%	89.7%	69.9%	76.0%	83.3%
10	2,948	91.4%	96.4%	46.7%	77.4%	92.3%	77.4%	**83.0%**	66.7%
15	4,594	95.7%	98.2%	53.3%	82.0%	92.3%	82.0%	86.0%	80.0%

Table 2: Accuracy versus retrieved-context budget on LongMemEval-S, $k = 20$, weak links on. The $m = 5$ row is the same configuration as Table 3's SOLM row, reported here as its mean over 3 independent runs for consistency with that table; the $m = 10$/$m = 15$ rows are single runs. Ctx(tok) = average retrieved-context size in tokens; Abst. = abstention accuracy; other column abbreviations as in Table 4. We treat $m = 10$ as the more representative operating point given this tradeoff, rather than the single highest-accuracy configuration ($m = 15$).

5.3 Comparison to Flat Retrieval

Table 3 compares SOLM against vanilla RAG (Section 4) at $m = 5$ for both, with nearly identical context sizes (1,381 vs. 1,168 tokens): our most controlled comparison, since the gap is then attributable to retrieval itself, not a larger generation context. As with Table 1, the Overall column's 3-run ranges never overlap (SOLM 75.5–76.4%, vanilla RAG 66.4–67.0%; Section 4.1 gives the source of this variance), and the same McNemar's test (Section 5.1) rejects the null here too ($\chi^2 = 15.86$, $p < 0.0001$, $n = 470$). The win is not uniform, however: single-session-preference ties (52.2% each), while SOLM leads every other category.

Method	Ctx(tok)	SS-User	SS-Asst	SS-Pref	Multi-S	Know-Upd	Temp-Reas	Overall	Abst.
Vanilla RAG ($m = 5$)	1,168	89.6%	94.0%	52.2%	46.8%	78.7%	58.8%	66.7% (66.4–67.0)	85.6%
SOLM ($k = 20, m = 5$)	1,381	93.8%	95.8%	52.2%	62.0%	93.5%	67.2%	**76.0% (75.5–76.4)**	83.3%

Table 3: SOLM versus flat similarity retrieval at nearly matched context size, weak links on. Every column reports the mean across 3 independent runs; Overall additionally shows the range. Relative improvement (mean): 13.8%. Ctx(tok) = average retrieved-context size in tokens; Abst. = abstention accuracy; other column abbreviations as in Table 4.

5.4 Comparison to Published Systems

Table 4 places SOLM's own LongMemEval-S per-category results, generated using Gemini Flash 3.5 (without thinking / reasoning), alongside seven other systems' published numbers on the same benchmark split [1, 7, 2, 8, 3]. MIRIX, Zep, Mem0, Memobase, MemU, and MemOS were all run on a GPT-4o-mini backbone [8]. Despite far fewer LLM calls during memory construction than these six systems (Section 2), SOLM's overall accuracy (83.0%) matches the strongest of these seven systems (tied with EverMemOS); this is not a controlled comparison (Limitations, Section 6), and Table 3's matched-backbone comparison isolates the retrieval mechanism instead. Accuracy is uneven across categories: single-session-preference is SOLM's weak point (46.7%, lowest of all eight rows), versus near-top results on single-session-assistant (96.4%) and knowledge-update (92.3%).

Method	SS-User	SS-Asst	SS-Pref	Multi-S	Know-Upd	Temp-Reas	Overall
MIRIX [1, 8]	72.9%	63.6%	53.3%	30.1%	52.6%	25.6%	43.5%
Zep [7, 8]	92.9%	75.0%	53.3%	47.4%	74.4%	54.1%	63.8%
Mem0 [2, 8]	82.9%	26.8%	90.0%	63.2%	66.7%	72.2%	66.4%
Memobase [8]	92.9%	23.2%	80.0%	66.9%	89.7%	75.9%	72.4%
MemU [8]	67.1%	19.6%	76.7%	42.1%	41.0%	17.3%	38.4%
MemOS [8]	95.7%	67.9%	96.7%	70.7%	74.3%	77.4%	77.8%
EverMemOS [3]	97.1%	85.7%	93.3%	73.7%	89.7%	77.4%	83.0%
SOLM ($k = 20, m = 10$)	91.4%	96.4%	46.7%	77.4%	92.3%	77.4%	**83.0%**

Table 4: LongMemEval-S per-category accuracy. SOLM's results use Gemini Flash 3.5 (without thinking / reasoning) for answer generation. SS-User/SS-Asst/SS-Pref = single-session-user/assistant/preference; Multi-S = multi-session; Know-Upd = knowledge-update; Temp-Reas = temporal-reasoning.

5.5 Preliminary Results on LongMemEval-M

We use LongMemEval-M as an additional dataset, beyond LongMemEval-S, for a preliminary scalability check. We evaluate SOLM on a 120-question subset of LongMemEval-M (out of 500 total), stratified across all six question categories (20 questions each: SS-User, SS-Asst, SS-Pref, Multi-S, Know-Upd, Temp-Reas; abbreviations as in Table 4), with abstention questions excluded. None of the seven systems in Table 4 report results on LongMemEval-M. This evaluation is therefore a standalone scalability demonstration, not a competitive comparison. LongMemEval-M haystacks average roughly 4,894 messages, an order of magnitude larger than LongMemEval-S's $N \approx 500$; applying the depth-cap heuristic from Section 4.3 ($n \approx \lceil \log_b(N) \rceil$, $b = \mu = 10$) to this larger N gives $\lceil \log_{10}(4894) \rceil = 4$ layers, i.e. a top-layer index of $n = 3$ under our 0-indexed convention.

We exclude importance scoring (Section 3.1) here for a concrete infrastructural reason: scoring every message across LongMemEval-M's full 500-haystack set ($\sim$4,894 messages/haystack) would require $\sim$2.45 million calls to our local Qwen2.5-7B model, intractable on our single consumer laptop even restricted to this 120-question subset (which still spans full-length haystacks). We instead fix every message to a constant importance score; all other hyperparameters match Table 6. This makes ingestion and retrieval fully deterministic, so we obtain 3 independent runs by reusing the same retrieved memories and re-invoking only answer generation (Section 3.5), isolating the one remaining source of variance without repeating ingestion. *We plan to extend this evaluation to the full 500-question LongMemEval-M set (still without importance scoring, for the same infrastructural reason) and report results during the author response period.* Table 5 reports these results.

6 Limitations

- **Backbone-confounded published-system comparison.** Six of the seven systems in Table 4 share a GPT-4o-mini backbone [8], EverMemOS uses another, and SOLM uses Gemini Flash 3.5 (without thinking / reasoning), conflating architecture with generation model. Only Table 3's matched-backbone comparison isolates the retrieval mechanism.
- **Single-session-preference weakness is a generation, not a retrieval, failure.** This is the one category where SOLM trails every other system in Table 4; at matched context size (Table 3), it ties

Configuration	SS-User	SS-Asst	SS-Pref	Multi-S	Know-Upd	Temp-Reas	Overall
SOLM ($k = 20$, $m = 10$, $n = 3$, no importance)	100.0%	95.0%	36.7%	80.0%	95.0%	95.0%	83.6% (83.3–84.2)

Table 5: SOLM accuracy on a 120-question stratified subset of LongMemEval-M (20 per category, abstention excluded), no importance scoring, $n = 3$. Every column reports the mean across 3 independent runs; Overall additionally shows the range. This configuration omits the layer-n importance-based shortcut (Section 3.1); results characterize SOLM without importance scoring and the layer-n reference at scale, not the full two-mechanism architecture.

vanilla RAG (52.2% each). Session-level Recall@k for this category, i.e. considering only the top-k of the $m = 20$ memories retrieved per question, is 100% at $k = 5, 10, 20$, at or above the other single-session categories (Table 9, Appendix A.5). That measurement used retrieval beam width 10, not the beam width 20 used for the accuracy numbers above; since a wider beam only enlarges the candidate pool considered at each layer (Section 3.5), a recall already at this 100% ceiling cannot be lower at beam width 20. So the correct content always reaches generation, yet accuracy is roughly half theirs. Unlike those categories' short, verbatim facts, preference answers require synthesizing an inferred judgment across scattered details, including what the user would *not* want, which the ground truth always states explicitly. We attribute this to harder generation-side synthesis rather than retrieval; isolating which part fails (positive inference, negative inference, or rubric matching) is left to future work. We used the same generic answer-generation prompt across all six categories and did not attempt category-specific prompting or fine-tuning for this one.

- **Importance scoring's optionality and unproven benefit.** Disabling the importance-driven layer-n reference (Section 3.1) is motivated by scalability (Section 5.5), not a controlled accuracy comparison: Table 7's "None" column ties our default's Overall accuracy exactly, yet 28 of 470 questions disagree underneath, a coincidental cancellation rather than proof of inactivity, and weaker evidence than Table 1's 3-run ablation. We intuit the mechanism should help accuracy but have not proven this, possibly because our scoring rubric is generic rather than tuned to LongMemEval's taxonomy. A controlled on-vs-off ablation is left to future work.
- **No formal test for cluster multimodality.** Cluster splitting is triggered purely by a quantization-error threshold ($qe_{\text{node}} > \tau \cdot qe_{\text{parent}}$), without a formal statistical test confirming a cluster's members are actually multimodal rather than just gradually more diffuse. A confirmatory multimodality test before committing to a split is a natural refinement left to future work.
- **No splitting ablation reported.** Unlike weak links (Table 1), we report no controlled on-vs-off splitting ablation: the dilution argument in Appendix A.6 needs a cluster's member count n to grow large, and LongMemEval-S's haystacks average only $N \approx 500$ messages (Section 4.3), too few to accumulate the heterogeneity splitting addresses under our default hyperparameters.

7 Conclusion

We presented SOLM, a hierarchical memory architecture built entirely through embedding-only computation (bottom-up cluster placement, quantization-error-driven splitting, cross-branch weak links), with one fixed call per query and one optional, lightweight per-message small-model call, cheap and removable (Section 5.5) at scale. Weak links address aggregation dilution, a structural risk we identified in centroid-based hierarchical retrieval: a cluster's centroid can fail to represent a member that would otherwise match a query well, pruning it out of reach. A controlled ablation on LongMemEval-S (3 independent runs) showed weak links recover a substantial share of this loss (7.7% relative mean improvement, non-overlapping ranges), and at matched context size SOLM outperformed flat similarity retrieval by 13.8% (relative mean improvement, likewise non-overlapping across 3 runs). As a secondary, less controlled comparison (six of seven other published systems share a GPT-4o-mini backbone; SOLM uses Gemini Flash 3.5, without thinking or reasoning), SOLM's accuracy was comparable to the strongest of them despite far fewer LLM calls during construction, with single-session-preference recall its main remaining weakness. These results suggest competitive long-term memory accuracy does not require scaling LLM usage in proportion to memory volume.

REFERENCES

[1] Yu Wang and Xi Chen. MIRIX: Multi-Agent Memory System for LLM-Based Agents. arXiv:2507.07957, 2025.

[2] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. arXiv:2504.19413, 2025.

[3] Chuanrui Hu, Xingze Gao, Zuyi Zhou, Dannong Xu, Yi Bai, Xintong Li, Hui Zhang, Tong Li, Chong Zhang, Lidong Bing, and Yafeng Deng. EverMemOS: A Self-Organizing Memory Operating System for Structured Long-Horizon Reasoning. arXiv:2601.02163, 2026.

[4] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. *ICLR*, 2025. arXiv:2410.10813.

[5] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic Memory for LLM Agents. arXiv:2502.12110, 2025.

[6] Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. From Isolated Conversations to Hierarchical Schemas: Dynamic Tree Memory Representation for LLMs. arXiv:2410.14052, 2024.

[7] Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A Temporal Knowledge Graph Architecture for Agent Memory. arXiv:2501.13956, 2025.

[8] Zhiyu Li, Chenyang Xi, Chunyu Li, et al. MemOS: A Memory OS for AI System. arXiv:2507.03724, 2025.

A Appendix

A.1 Importance Scoring Prompt

The following prompt is rendered once per ingested message and sent to Qwen2.5-7B (Section 3.1); only the `content` placeholder is filled in.

```
You are evaluating a memory an AI agent is about to store in its
↪  long-term memory system.

Rate how important this memory is on a scale of 0 to 9, where:
- 0-2: routine, forgettable content (small talk, one-off logistics,
↪  trivial remarks)
- 3-5: moderately useful context (project details, preferences, plans,
↪  casual facts worth
  some retention)
- 6-8: significant facts worth remembering long-term (deadlines,
↪  commitments,
  relationships, decisions)
- 9: critical information that must never be forgotten (medical
↪  conditions, safety-critical
  facts, explicit requests to remember something forever)

Memory content:
"""
{{ content }}
"""

Respond with ONLY a JSON object in this exact format, nothing else, no
↪  explanation:
{"importance": <integer from 0 to 9>}
```

A.2 Answer Generation Prompt

The following prompt is rendered once per query and sent to Gemini Flash 3.5, without thinking or reasoning (Section 3.5); `query` and `question_date` come from the benchmark question, and `memories` is the ranked list of retrieved layer-0 memories, each with its own timestamp where available.

```
You are answering a question using memories retrieved from a long-term
↪  memory system.
{% if question_date %}
The question is being asked on: {{ question_date }}
{% endif %}
Question:
{{ query }}

Retrieved Context, ordered from most to least relevant:
{% for memory in memories %}
{{ loop.index }}. {% if memory.date %}<date>[{{ memory.date }}]<date> {%
↪  endif %}
  <content>{{ memory.content }}</content>
{% endfor %}

Using only the information <content> in the context above, answer the
↪  question.
Each piece of <content> in context is timestamped with when it was said,
↪  where
available -- use these timestamps <date> to reason about ordering,
↪  duration,
and recency (e.g. which fact is more recent if two entries conflict).
If the context don't contain enough information to answer confidently,
say so clearly rather than guessing.

<important>
If you are asked about a specific amount / time. Always calculate with
↪  the
information you have provided with in the Context. Never answer that you
↪  cannot
calculate
</important>

Respond with ONLY a JSON object in this exact format, nothing else, no
explanation outside the JSON: {"answer": "<your answer as a string>"}
```

A.3 Configuration Used for Reported Results

Table 6 lists every lattice and retrieval hyperparameter introduced in Section 3, alongside the value used to produce every result in Section 5, except k and m, which vary by experiment and are already given in Tables 1, 2, and 3 themselves. As discussed in Section 4, these values were selected by manual exploration against the evaluation set itself, not a held-out split; we expect a practitioner deploying SOLM on a different workload to retune them against their own data rather than reuse the values reported here.

Symbol	Description	Value
n	depth cap (`max_layer`)	2
τ	split sensitivity: splits if $qe_{\text{node}} > \tau \cdot qe_{\text{parent}}$	0.9
λ	qe accumulator decay rate	0.5
μ	minimum members before a split is checked	10
θ_{join}	join threshold (similarity for placement)	0.85
θ_{imp}	importance threshold for the layer-n reference	8
τ_{weak}	weak-link creation threshold	0.6
p_{weak}	weak-link cap per layer	2
k_{rrf}	RRF damping constant (Section 3)	15.0
h	weak-link traversal hop limit	3
r	rerank-pool multiplier	5
o	reranker chunk overlap (tokens)	50
—	importance-scoring model	Qwen2.5-7B

Table 6: Full configuration for every reported result, except k and m (see Tables 1–3 for the values used in each experiment).

A.4 Sensitivity Sweep

Table 7 sweeps the importance threshold θ_{imp} (Section 3.1) across its 0–9 range at $k = 5$, $m = 5$, weak links on, every other setting held fixed at the values in Table 6. As discussed in Section 4.3, this sweep was run after $\theta_{\text{imp}} = 8$ was already fixed, as a check on that value rather than the process that selected it; each column is a single run, not averaged over multiple runs like Tables 1 and 3. Overall accuracy rises roughly monotonically from $\theta_{\text{imp}} = 2$ to our reported default of 8, where it peaks, then dips slightly at 9. Overall accuracy is sensitive to this threshold, with our reported default sitting at or near the best value tested; per-category columns are single runs and not independently validated. The "None" column disables the layer-n reference entirely (still scoring every message for real, just with an unreachable threshold) and lands on the exact same Overall accuracy as $\theta_{\text{imp}} = 8$ (347/470 both), but not because the mechanism has no effect: 28 of 470 individual question verdicts flip between the two conditions, with per-category accuracy moving in both directions (e.g. Temp-Reas 66.9%→63.0%, Multi-S 56.2%→58.7%); the aggregate tie is a coincidental cancellation, not evidence of inactivity. We discuss this alongside the mechanism's cost (Section 3.5) in Section 6.

θ_{imp}	2	4	6	7	8	9	None
SS-User	93.8%	90.6%	92.2%	93.8%	92.2%	90.6%	92.2%
SS-Asst	96.4%	98.2%	96.4%	94.6%	96.4%	96.4%	96.4%
SS-Pref	53.3%	60.0%	56.7%	56.7%	50.0%	53.3%	53.3%
Multi-S	50.4%	52.9%	55.4%	55.4%	56.2%	57.9%	58.7%
Know-Upd	86.1%	90.3%	91.7%	91.7%	91.7%	93.1%	93.1%
Temp-Reas	59.8%	59.8%	59.8%	63.0%	66.9%	63.8%	63.0%
Overall	70.0%	71.5%	72.1%	73.0%	**73.8%**	73.6%	73.8%

Table 7: Sensitivity of accuracy on LongMemEval-S to the importance threshold θ_{imp}, at $k = 5$, $m = 5$, weak links on. $\theta_{\text{imp}} = 8$ is the value reported throughout the paper. "None" disables the importance-driven layer-n reference entirely ($\theta_{\text{imp}} = 10$, unreachable on the 0–9 scale) while still running real per-message scoring, isolating the mechanism's effect without also changing anything else. Column abbreviations as in Table 4.

A.5 Retrieval Recall

As a retrieval-only diagnostic, separate from the accuracy-level ablation in Table 1, we measure session-level Recall@k: whether a retrieved memory's source session matches one of a question's gold `answer_session_ids`, at $k \in \{5, 10, 20\}$, with no generation or judging involved. Importance scoring is on at its default threshold ($\theta_{\text{imp}} = 8$, Table 6) throughout this section. Each row retrieves once at $m = 20$ per question on LongMemEval-S; Recall@5/10 are read as prefixes of that same ranked list, so all three columns come from one retrieval call, not three. We report this at beam widths 5 and 10, weak links on and off; beam width 20 is left to future work.

Configuration	Recall@5	Recall@10	Recall@20
Beam width 5, weak links on	91.3%	95.3%	96.6%
Beam width 5, weak links off	88.4%	92.1%	92.8%
Beam width 10, weak links on	92.4%	96.0%	98.0%
Beam width 10, weak links off	90.8%	94.2%	96.3%

Table 8: Session-level Recall@k on LongMemEval-S ($n = 500$), $m = 20$. Beam width 5/weak-off is truncated at higher k: 161/500 questions returned fewer than 10 candidates and 438/500 fewer than 20, so its Recall@10/@20 reflect a much smaller retrieved set for most questions and are not directly comparable to the other rows at those columns. Beam width 5/weak-on is truncated similarly but far less (35/500 below 20); beam width 10/weak-on is nearly unaffected (2/500 below 20); beam width 10/weak-off falls in between (4/500 below 10, 168/500 below 20).

Two things stand out, with the second more informative than the first. First, Recall@5, the only column unaffected by beam width 5/weak-off's truncation, is higher with weak links (91.3% vs. 88.4%), corroborating Table 1's accuracy-level finding at the retrieval stage itself. Second, and more tellingly, the truncation asymmetry: without weak links, the beam at width 5 fails to even surface 10 candidates for 161/500 questions, versus just 4/500 with them. This suggests weak links' main contribution at this beam width is enlarging the pool of candidates that reach the reranker at all, not merely reordering an already-adequate one. These readings are preliminary given the table is partial.

As a separate check, Table 9 breaks session-level Recall@k down by single-session category, on the full 500-question set ($k = 10$ beam width, weak links on).

Category	Recall@5	Recall@10	Recall@20
Single-session-user	98.6%	98.6%	98.6%
Single-session-assistant	100.0%	100.0%	100.0%
Single-session-preference	100.0%	100.0%	100.0%

Table 9: Session-level Recall@k by single-session category, full 500-question LongMemEval-S set, $k = 10$ beam width, weak links on, $m = 20$. Single-session-preference's recall is not lower than the other single-session categories (it is, if anything, marginally higher), which is the basis for attributing that category's accuracy gap (Section 6) to generation rather than retrieval.

A.6 A Formal Argument for Aggregation Dilution

Aggregation dilution (Section 1) is stated informally above as a structural risk. Here we derive it exactly.

Setup. Assume unit-normalized embeddings, so $\langle a, b\rangle = \cos(a, b)$. A cluster has members $v_1, \ldots, v_n$ with centroid $c_n = \frac{1}{n}\sum_i v_i$, normalized to $\hat{c}_n = c_n/\|c_n\|$. Let q be a query, and suppose v_1 is a true match with $\cos(q, v_1) = s$ close to 1, while the remaining $n - 1$ members are background content.

Fact 1 (raw similarity is an exact average). By linearity,

$$\langle q, c_n\rangle = \Big\langle q,\ \frac{1}{n}\sum_i v_i \Big\rangle = \frac{1}{n}\sum_i \langle q, v_i\rangle = \frac{1}{n}\sum_i \cos(q, v_i).$$

Fact 2 (centroid norm is governed by pairwise coherence).

$$\|c_n\|^2 = \langle c_n, c_n \rangle = \frac{1}{n^2} \sum_{i,j} \langle v_i, v_j \rangle = \frac{1}{n^2} \Big[\underbrace{\textstyle\sum_i \langle v_i, v_i \rangle}_{=n} + \underbrace{\textstyle\sum_{i \neq j} \langle v_i, v_j \rangle}_{=n(n-1)\bar{\rho}_n} \Big] = \frac{1}{n} + \frac{n-1}{n} \bar{\rho}_n,$$

where $\bar{\rho}_n$ is the average pairwise similarity between distinct members. As $n \to \infty$, if $\bar{\rho}_n \to \bar{\rho}$, then $\|c_n\| \to \sqrt{\bar{\rho}}$.

Theorem (dilution). Assume the background members' average similarity to q converges, $\frac{1}{n-1} \sum_{i \geq 2} \cos(q, v_i) \to \bar{b}$ as $n \to \infty$, with $\bar{b} \neq s$. Splitting the sum in Fact 1,

$$\frac{1}{n} \sum_i \cos(q, v_i) = \frac{1}{n} s + \frac{n-1}{n} \cdot \frac{1}{n-1} \sum_{i \geq 2} \cos(q, v_i) \;\longrightarrow\; 0 \cdot s + 1 \cdot \bar{b} = \bar{b}.$$

The true match's own term is scaled by $1/n \to 0$: its similarity s never changes, but its *share* of the average vanishes. Combined with Fact 2,

$$\cos(q, \hat{c}_n) = \frac{\langle q, c_n \rangle}{\|c_n\|} \;\longrightarrow\; \frac{\bar{b}}{\sqrt{\bar{\rho}}},$$

independent of s.

Corollary (pruning failure). If retrieval prunes whenever $\cos(q, \hat{c}) < \tau_{\text{prune}}$, and $\bar{b}/\sqrt{\bar{\rho}} < \tau_{\text{prune}} \leq s$, then since $\cos(q, \hat{c}_n) \to \bar{b}/\sqrt{\bar{\rho}}$, there is a finite n^* such that for all $n > n^*$, the cluster is pruned despite containing a perfect match.

SOLM's centroid update (Section 3) is count-weighted: $c \leftarrow \frac{n_c \cdot c + v_{\text{new}}}{n_c + 1}$. This is exactly the weighted-mean-of-sub-means identity, so a node at any layer is *exactly* the flat average over all its leaf descendants, not merely an approximation of it; the theorem above applies directly to any node, with n equal to that node's total leaf count. Splitting (Section 3.3) bounds n per node by capping how large a cluster grows before dividing, which directly limits the dilution derived above.

A.7 AI Usage Disclosure

The author used an AI coding/writing assistant (Claude, Anthropic) during preparation of this paper and its accompanying code. Its use included: polishing and revising prose across paper sections under author direction and review; verifying citation accuracy and reported numbers against primary sources for Table 4; checking internal consistency of cross-references and terminology across drafts; and coding assistant is used in the development of the codebase used to produce all reported results. The SOLM architecture, its core implementation, and all reported experiments are the author's own work. The author reviewed all AI-assisted contributions and takes full responsibility for the paper's content.