TOWARD ADAPTIVE MEMORY INJECTION IN AGENT SYSTEMS: A STRUCTURE-DRIVEN APPROACH

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory-augmented large language model (LLM) agents have largely focused on storing and retrieving memories. Yet retrieved candidates may contain redundant or task-irrelevant information, while answering a question may require complementary information distributed across multiple memories. Injecting all candidates increases context cost, whereas selecting them solely by individual relevance can omit information needed to cover the task. We introduce Learnable Memory Injection (LMI), which selects from already-retrieved candidates based on the task-relevant information missing from the current selection. It identifies task requirements and links them to source-bound facts in a task–memory relation graph. Using this graph, LMI assesses what information each candidate adds beyond the selected memories and prioritizes memories that supply missing task-relevant information under a memory-token budget. It then organizes the selected memories into the model context. LMI also supports learning context-dependent memory utility from downstream feedback to guide subsequent selections. Experiments across three LLM backbones show that LMI maintains or improves answer accuracy. The largest relative gain is 20.43%: with Llama-3.1-8B and Top-200 candidates, accuracy increases from 64.60% to 77.80%, while injected memory tokens decrease by 83.90% compared with directly injecting all candidates.

1 INTRODUCTION

Memory-augmented large language model (LLM) agents can accumulate experience, store reflections, maintain long-term memories, and retrieve information for a new task. Representative systems include Generative Agents (Park et al., 2023), Reflexion (Shinn et al., 2023), MemGPT (Packer et al., 2023), and A-MEM (Xu et al., 2025). Retrieval returns candidate memories but does not determine which should enter the context for the current decision.

Relevance ranking alone does not determine whether selected memories contain the information required by a task. For example, a question about how a team's size changed requires information about both its initial and current sizes. Several highly relevant memories may describe only the current size; removing duplicates among them still leaves the initial size missing. Conversely, similar memories may both be needed when they describe different times or states. MMR (Carbonell & Goldstein, 1998) and coverage-based summarization methods (Lin & Bilmes, 2011) already make selection set-dependent, but their effectiveness depends on how redundancy and coverage are defined. The challenge is therefore not only to select a set, but to determine which content satisfies task conditions and what information each memory adds beyond the current selection.

Figure 1 illustrates this decision on a LongMemEval question about the chronological order of three sports events. Full injection and Top-24 truncation yield different omissions, whereas LMI selects 24 memories from the same candidates and produces a complete answer.

We study this decision as **post-retrieval memory injection**. Given a candidate set already returned by a host memory system, the objective is to identify the information required by the task, select a subset that jointly supports the task within a memory-text selection budget, and organize the selected memories as context that the model can directly use. We use the same retrieved candidates across injection policies to compare how they select and organize memory.

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.

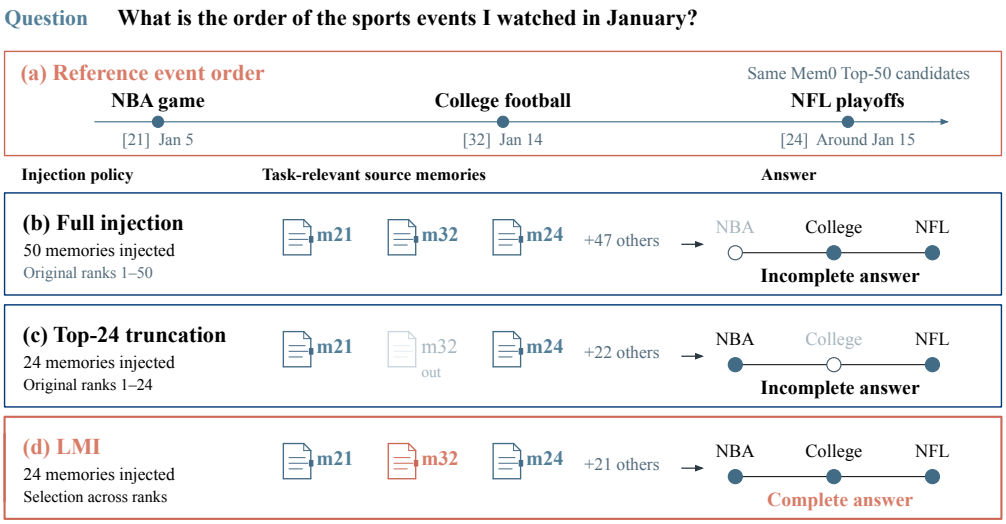

Figure 1: Post-retrieval memory injection on LongMemEval. (a) Reference events from 50 shared candidates. (b) Full injection yields an incomplete answer; (c) Top-24 excludes rank 32; (d) LMI selects and organizes 24 memories and answers completely. Answer events align to the reference order. Brackets indicate retrieval ranks; see Appendix A.

We introduce **Learnable Memory Injection (LMI)**, a layer between host retrieval and agent execution. LMI identifies task requirements and links them to supporting facts and their source memories in a *task–memory relation graph*. These links let the planner determine which requirements the current selection already supports and which candidate memories can fill the remaining gaps. For the team-size question, another current-size-only memory cannot substitute for a memory providing the missing initial size. Within a memory-token budget, LMI prioritizes whole memories that supply missing task-relevant information, then organizes the selected memories as context for agent execution without rewriting their original content.

LMI also learns from memory use. It trains a context-dependent memory scorer on historical execution feedback to guide candidate priorities, while structural constraints specify which task-relevant information must be retained. Only the scorer is trained; the pipeline is not trained end to end.

Recent public preprints on adaptive memory use include RCR-Router (Liu et al., 2025), RSCB-MC (Iscan, 2026), MemCon (Jiang et al., 2026), and NapMem (Xu et al., 2026). LMI addresses a distinct decision within this direction: budgeted joint selection from fixed retrieval candidates, guided by task coverage and source-linked requirement–fact relations. See Section 2 for details.

Our contributions are summarized as follows:

1. **A formulation of budgeted post-retrieval memory injection.** We formulate memory injection as budgeted whole-memory selection followed by context organization, enabling controlled comparisons between injection policies on fixed candidates.

2. **Structure-driven memory injection with feedback learning.** LMI uses task–memory relations to guide budgeted joint selection based on the task-relevant information each candidate adds to the current set. It learns a context-dependent memory-scoring function from historical execution feedback and uses the learned scores to guide candidate priorities within this structured selection process.

3. **Evaluation across benchmarks and model backbones.** We evaluate LMI with three LLMs on LongMemEval and LoCoMo. On LongMemEval, relative accuracy gains over direct injection reach 20.43%, with 83.90% fewer injected-memory tokens at Top-200. Ablations examine set-dependent selection, structured rendering, and learned utility.

2 Related Work

LMI relates to agent memory management, adaptive memory use, and budgeted context selection.

Agent memory lifecycles. Prior systems study how memories are formed, maintained, and retrieved. Generative Agents uses a memory stream and reflection mechanism (Park et al., 2023); Reflexion converts failures into verbal feedback (Shinn et al., 2023); MemGPT uses hierarchical storage (Packer et al., 2023); and A-MEM organizes linked, evolving memories (Xu et al., 2025). These systems determine which memories are available to downstream components. LMI begins after a host returns candidates, selecting a source-traceable subset for the current task.

Adaptive memory use. Recent work studies how agents decide when and how to use memory. RCR-Router routes memory subsets by role, task stage, and token budget using heuristic scores and greedy selection (Liu et al., 2025). RSCB-MC uses a risk-sensitive bandit for memory-reading decisions (Iscan, 2026), while MemCon uses online UCB to select memory lifecycle operations (Jiang et al., 2026). NapMem learns to navigate multi-granular memories through reinforcement learning (Xu et al., 2026). BudgetMem learns query-aware budget-tier routing across memory-processing modules to trade extraction cost against answer quality (Zhang et al., 2026).

In contrast, LMI fixes the retrieved candidates and uses downstream feedback to update utility for later joint selections, rather than allocating computation across extraction modules or changing memory lifecycle operations.

Context selection and compression. Self-RAG (Asai et al., 2024) and Adaptive-RAG (Jeong et al., 2024) adapt retrieval, whereas FILCO (Wang et al., 2023) learns to filter retrieved context and LLMLingua (Jiang et al., 2023) compresses prompts. MemGuide combines intent-aligned retrieval with missing-slot-guided memory filtering for task-oriented dialogue (Du et al., 2026). Selection need not rely on independent scores: MMR (Carbonell & Goldstein, 1998) balances relevance against similarity to already selected content, and submodular summarization (Lin & Bilmes, 2011) combines coverage and diversity. PACMS applies query-weighted facility-location coverage, based on embedding similarity, to select whole units from pooled agent context under a token budget (Ghulyani et al., 2026). LMI shares this budgeted whole-unit selection setting, but defines coverage through source-linked requirement–fact checks, counting each recognized support group once. On fixed retrieved candidates, it combines these checks with temporal and state retention constraints during replacement and utility learning from paired execution feedback. Information position also affects long-context use (Liu et al., 2024), motivating a distinction between selection and presentation within context engineering (Mei et al., 2025).

3 Methodology

LMI implements learnable memory injection through structured joint selection. A semantic frontend identifies task requirements and source-bound facts; a task–memory relation graph links them to assess what each memory adds to the current selection. Within fixed budgets, the graph guides which information to retain jointly. A memory-scoring model trained on prior execution feedback orders candidates during initial selection. When candidate memories provide the same coverage gain, the planner gives priority to those with higher learned scores. Learning updates only this scoring model; the semantic frontend, relation-labeling rules, and replacement checks remain fixed. The scoring parameters are frozen during validation and testing; the selection pipeline is not trained end to end. A compiler organizes selected memories for host inference (Figure 2).

3.1 Problem Setup

At decision t, the host has frozen an ordered candidate list $\mathcal{M}_t = (m_1, \ldots, m_n)$. LMI receives the current query and observable task state q_t, selection budgets $\mathcal{B}_t = (K_t, T_t)$, and a fixed model, prompt, tokenizer, and context-construction configuration Θ_t.

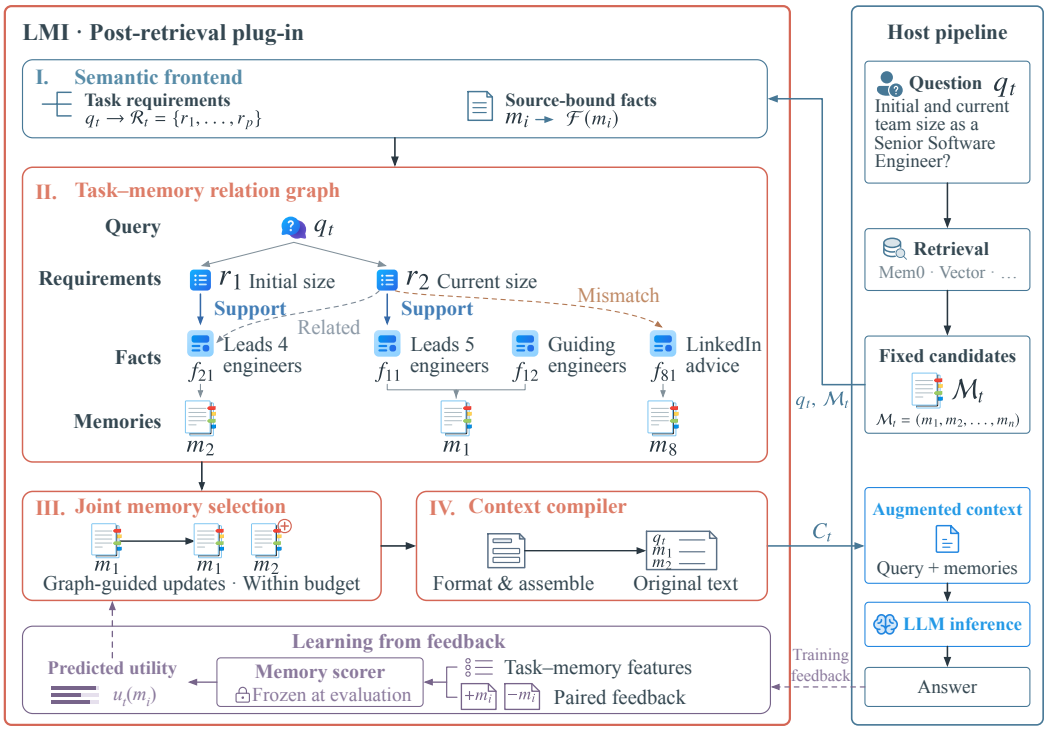

Figure 2: Overview of LMI as a post-retrieval plug-in. Given a query and fixed retrieved candidates, LMI uses task–memory relations to jointly select whole memories and assemble them as context for host inference. The example uses real memory content; its relations and selection illustrate the mechanism rather than an execution trace.

LMI selects a set $S_t^\star$ of whole MemoryItems and compiles it into the injection context C_t:

$$X_t = (q_t, \mathcal{M}_t, \mathcal{B}_t, \Theta_t), \quad S_t^\star \subseteq \mathcal{M}_t, \quad C_t = \mathrm{Organize}(S_t^\star; \Theta_t),$$
$$|S_t^\star| \le K_t, \qquad \sum_{m_i \in S_t^\star} c_t(m_i) \le T_t. \tag{1}$$

Here K_t and T_t are selection-stage limits on memory count and text-token cost. The cost $c_t(m_i)$ counts each original memory text and its bullet prefix with a frozen planning tokenizer. The compiled memory-context count T_t^{ctx} additionally includes guidance, labels, dates, and formatting; it is measured separately and is not capped by T_t. LMI operates only on $\mathcal{M}_t$, selecting and organizing whole memories without additional retrieval or rewriting their text. Neither graph construction nor utility learning can recover facts absent from these candidates. All graph and utility features use only information available before selection; model outputs, evaluation labels, and future traces cannot affect the current graph or decision.

3.2 Semantic Frontend

LMI's semantic frontend uses a frozen spaCy parser (Honnibal et al., 2020) with hand-specified dependency and lexical rules to extract requirements $\mathcal{R}_t = \{r_1, \ldots, r_p\}$ and source-linked facts:

$$(\mathcal{R}_t, \{\mathcal{F}(m_i)\}_{i=1}^n) = \mathrm{Parse}(q_t, \mathcal{M}_t). \tag{2}$$

Predefined patterns identify subjects, actions or properties, targets, quantities, and time/state constraints for single-answer, counting, temporal-endpoint, and multi-requirement queries. Extraction around event predicates and attributes preserves $\mathrm{src}(f) = m_i$ and text spans. Partial parses retain the available fields; failed parses are recorded as abstentions. The original memories remain eligible for initial selection and budget filling. Extraction status alone does not determine requirement–fact labels. Only recognized patterns are supported, without LLM fallback (Appendix B.1). Section 3.3 describes the temporal and state matching checks.

3.3 TASK–MEMORY RELATION GRAPH

LMI links the query, requirements, facts, and source memories in a graph. Requirement–fact relations identify supporting facts for each requirement; fact–fact links form source groups counted once; and fact–memory links identify the whole memories providing each group. The planner uses these relations to assess what each candidate adds beyond the current selection. Formally:

$$\begin{aligned} G_t &= (V_t, E_t), \quad V_t = \{q_t\} \cup \mathcal{R}_t \cup \mathcal{F}_t \cup \mathcal{M}_t, \quad \mathcal{F}_t = \bigcup_i \mathcal{F}(m_i), \\ E_t &= E_t^{qr} \cup E_t^{rf} \cup E_t^{fm} \cup E_t^{ff}, \quad E_t^{rf} = \{(r_j, f, \rho_t(r_j, f)) : r_j \in \mathcal{R}_t,\ f \in \mathcal{F}_t\}. \end{aligned} \tag{3}$$

Query–requirement edges $E_t^{qr} = \{(q_t, r_j) : r_j \in \mathcal{R}_t\}$ organize per-requirement assessment; fact–memory edges $E_t^{fm} = \{(f, \mathrm{src}(f)) : f \in \mathcal{F}_t\}$ identify each fact's whole source memory. Requirement–fact edges in E_t^{rf} are labeled by $\rho_t : \mathcal{R}_t \times \mathcal{F}_t \to \mathcal{L}$, where $\mathcal{L} = \{\text{SUPPORT}, \text{RELATED}, \text{MISMATCH}, \text{UNRESOLVED}\}$. SUPPORT indicates that a fact is judged to contribute information toward a requirement and passes all prescribed matching checks. RELATED indicates that a fact matches the requirement in subject, target, and content role but does not meet all SUPPORT checks. Other pairs are MISMATCH for rule-established mismatches and UNRESOLVED otherwise. Time checks use source-linked lexical matches and date-interval comparisons for supported relative windows; state checks use tense, modality, and lexical cues for planned, completed, and current states. These provide neither general temporal inference nor cross-memory state reconstruction. Source-qualified context and unresolved information constrain retention; relation statistics supply utility features without relabeling (Sections 3.4–3.5; Appendix B.2).

LMI derives requirement-scoped fact links E_t^{ff} to control repeated counting based on sources. Let $\mathcal{F}_{t,j}$ denote the facts eligible for grouping under r_j: SUPPORT facts for answer-support requirements, or source-qualified RELATED facts for the distinct personal-history context branch (Appendix B.3). Within this set, the slice $E_{t,j}^{ff}$ tagged with r_j connects facts with identical source text; within one memory, it also links composed facts to recorded components. Its connected components, including isolated facts, define g_j source groups; E_t^{fm} maps each group to its providers:

$$\begin{aligned} \{H_{j1}, \ldots, H_{jg_j}\} &= \mathrm{CC}(\mathcal{F}_{t,j}, E_{t,j}^{ff}), \\ P_{jk} &= \{\mathrm{src}(f) : f \in H_{jk}\} \subseteq \mathcal{M}_t, \quad k = 1, \ldots, g_j. \end{aligned} \tag{4}$$

Here CC denotes connected components. A memory may provide multiple groups; each group counts once if any provider is selected. This is source-aware counting, not semantic deduplication: paraphrases may count separately, whereas distinct facts sharing a source span or composition lineage may count together. Grouping changes counts, not facts or memory text. Distinct groups may provide complementary information, but grouping alone does not establish this (Appendix B.3).

3.4 MEMORY SELECTION AND INJECTION

LMI uses G_t to assess what a memory adds to the current selection (Section 3.3). Query–requirement edges identify the requirements evaluated separately, such as the initial and current team sizes in Figure 2. For each requirement, requirement–fact judgments determine which facts enter grouping, fact–fact links form source groups, and fact–memory edges identify the memories providing each group. Two memories reporting only the current team size cannot substitute for missing initial-size information, even if both are relevant to the question. These mappings allow the planner to recognize that a memory reporting the initial size adds information that another current-size-only memory does not. LMI therefore computes coverage per requirement and uses source records to reject replacements that lose required information despite unchanged group counts.

Here, r_j denotes a separately evaluated requirement unit. When a requirement is decomposed into parts, only those parts contribute to the coverage sum. For selection S, group H_{jk} counts once if $S \cap P_{jk} \neq \varnothing$. With original target τ_j, g_j recognized groups in the fixed candidates, and selection cap $\kappa_j = \min(\tau_j, g_j)$, we define coverage and marginal gain using indicator $\mathbb{I}[\cdot]$:

$$\begin{aligned} \mathrm{cov}_j(S) &= \min\Big(\kappa_j, \textstyle\sum_{k=1}^{g_j} \mathbb{I}[P_{jk} \cap S \neq \varnothing]\Big), \\ \Delta_{\mathrm{cov}}(m_i \mid S) &= \textstyle\sum_{j=1}^{p} \left[\mathrm{cov}_j(S \cup \{m_i\}) - \mathrm{cov}_j(S)\right]. \end{aligned} \tag{5}$$

We use the following coverage objective under the selection-stage limits:

$$\max_{S \subseteq \mathcal{M}_t} \sum_{j=1}^{p} \mathrm{cov}_j(S) \quad \text{s.t.} \quad |S| \le K_t, \qquad \sum_{m_i \in S} c_t(m_i) \le T_t. \tag{6}$$

The objective favors memories that add groups to unmet units. A composite requirement is considered covered only when every required unit meets its original target and none remains unresolved. The cap κ_j bounds selection credit, not the original target τ_j: all available groups may be selected while a required part remains missing (Appendix B.3).

LMI uses a constrained heuristic. It first retains a few top-ranked host memories independently of the graph, then adds other candidates in learned-utility order within both budgets. Without learned utility, it uses static ranking.

LMI next considers unselected memories with positive coverage gain or with RELATED facts whose source records justify their use as context. These RELATED facts add no groups outside the personal-history branch. Candidates are processed by decreasing gain, with higher utility $u_t(m_i)$ preferred at equal gain (Section 3.5). LMI tries adding each candidate, then replacing one selected memory if needed. Updates must satisfy both budgets and retain the top-ranked host memories, all represented groups, and required temporal, state, and unresolved source information, even when group counts remain unchanged. After each accepted update, LMI recomputes gains with G_t fixed; it stops when no addition or replacement passes these checks. Global optimality is not guaranteed (Appendix B.4).

A fixed-template compiler organizes $S_t^\star$ with guidance, role labels, record dates, and fixed ordering, without reselection or rewriting (Appendix B.4). LMI records submitted memories for feedback.

3.5 LEARNING FROM FEEDBACK

LMI trains a memory-scoring model from historical execution feedback, using structured task–memory features available before selection. It predicts the score difference with and without a memory in the submitted set. Let $S_t \subseteq S_t^\star$ denote the memories in the actual model request and $R_t(S_t)$ the resulting task score. For $m_i \in S_t$, paired runs give the score difference

$$\delta_{t,i} = R_t(S_t) - R_t(S_t \setminus \{m_i\}). \tag{7}$$

Paired runs keep the question, candidates, answer model, prompt template, and decoding settings fixed, changing only the target memory's presence. The difference $\delta_{t,i}$ estimates contribution within the submitted set, not intrinsic memory value. Two interchangeable memories can each have zero difference despite providing necessary shared information; repeated runs reduce execution noise but cannot separate their contributions. Repeated differences are aggregated into a training target y_h with consistency weight $w_h \in [0, 1]$; conflicting or invalid outcomes yield no sample (Appendix B.5).

Each candidate's feature vector $\phi_t(m_i) \in \mathbb{R}^d$ combines retrieval and static-ranking signals with features of task requirements and source-linked content. Requirement–memory features include relation-label proportions and coverage state. Features exclude model outputs and task outcomes. From valid historical samples $\{(\phi_h, y_h, w_h)\}_{h<t}$, weighted ridge regression learns $\widehat{\theta}_t$ with regularization $\lambda > 0$. A LinUCB-style term adds training-time exploration (Li et al., 2010):

$$A_t = \lambda I + \sum\nolimits_{h<t} w_h \phi_h \phi_h^\top, \qquad b_t = \sum\nolimits_{h<t} w_h y_h \phi_h, \qquad \widehat{\theta}_t = A_t^{-1} b_t, \tag{8}$$

$$\begin{aligned} \mu_t(m_i) &= \phi_t(m_i)^\top \widehat{\theta}_t, \qquad \sigma_t(m_i) = \sqrt{\phi_t(m_i)^\top A_t^{-1} \phi_t(m_i)}, \\ u_t(m_i) &= \mu_t(m_i) + \beta \sigma_t(m_i). \end{aligned} \tag{9}$$

Here, μ_t predicts the score difference, σ_t measures uncertainty, and $\beta \ge 0$ controls training-time exploration. Valid feedback updates the model only after an episode. Learned scores guide selection as described in Section 3.4. The graph, selection procedure, coverage targets, retention constraints, and budgets remain fixed.

In our reference-supervised training, answer correctness gives $R_t \in \{0, 1\}$. References provide supervision, not selection features or answer-model inputs. Validation and test retain learned parameters with $\beta = 0$ and no updates. Frozen deployment needs no reference answers. Learning from user ratings or task outcomes requires a separate protocol: a single observed score does not reveal the score without the memory.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Benchmarks and models. We use LongMemEval (Wu et al., 2025) and LoCoMo (Maharana et al., 2024) with frozen Top-50/Top-200 candidates from local Mem0 OSS (Chhikara et al., 2025). Answer models are Qwen2.5-7B-Instruct (Yang et al., 2024), Llama-3.1-8B-Instruct (Grattafiori et al., 2024), and GPT-5 (OpenAI, 2025).

Compared methods. *Direct injection* uses all candidates; *Budgeted MMR* balances query relevance and redundancy (Carbonell & Goldstein, 1998); *Budgeted Semantic Coverage (SC)* combines query-weighted coverage and query relevance, adapting the MixModSub objective (Kumari & Bilmes, 2021). MMR, SC, and LMI select whole memories under shared caps of 24 items and 1,280 selection tokens. Reported memory tokens also include compiler-added guidance, labels, dates, and formatting, and can therefore exceed selection caps.

Utility training. Reference-scored paired removal runs train LMI's memory scorer (Appendix C.9). Evaluation freezes the learned parameters with $\beta = 0$; references serve grading, not selection.

Evaluation protocol. LongMemEval methods share questions, frozen candidates, accuracy denominators, and GPT-5 judging. The LoCoMo comparisons at both retrieval depths use the same 1,976 scored questions across methods; failed answers receive no correctness credit (Appendix C.4). We report accuracy, memory and full-input tokens, and post-retrieval latency.

4.2 OVERALL PERFORMANCE

Table 1: Answer accuracy (%) on LongMemEval and LoCoMo.

LLM Backbone	Top-K	**LongMemEval**					**LoCoMo**				
		Direct	MMR	SC	LMI	Gain	Direct	MMR	SC	LMI	Gain
Qwen2.5-7B	50	68.40	70.60	74.00	**75.20**	+9.94	70.50	72.17	69.59	**76.57**	+8.61
	200	64.20	61.00	67.80	**75.00**	+16.82	65.54	66.40	68.47	**78.49**	+19.77
Llama-3.1-8B	50	69.40	52.20	68.40	**76.20**	+9.80	74.39	73.28	66.55	**75.46**	+1.43
	200	64.60	42.40	66.20	**77.80**	+20.43	63.36	69.43	63.51	**72.17**	+13.90
GPT-5	50	**90.20**	85.60	87.00	**90.20**	0.00	89.27	89.83	90.13	**90.89**	+1.81
	200	90.80	73.80	85.40	**91.40**	+0.66	90.64	88.97	89.78	**91.75**	+1.23

Gain: LMI's relative improvement (%) over Direct.

Overall results. On LongMemEval (Table 1), LMI's relative gains over Direct reach 16.82% for Qwen and 20.43% for Llama at Top-200. For GPT-5, it matches Direct at Top-50 and improves accuracy by 0.60 percentage points at Top-200. LMI exceeds MMR and SC in all six settings. SC also exceeds MMR in all six LongMemEval settings. The larger Top-200 gains for Qwen and Llama are consistent with reduced distraction from irrelevant memory. GPT-5's higher Direct accuracy leaves less room for further improvement.

On LoCoMo, LMI exceeds Direct and MMR in all six settings. All GPT-5 outputs are evaluable; Llama's Top-200 gain also reflects output evaluability, not memory selection alone (Appendix C).

Table 2: LongMemEval accuracy (%) by question type at Top-200.

Question type	Qwen2.5-7B				Llama-3.1-8B				GPT-5			
	Direct	MMR	SC	LMI	Direct	MMR	SC	LMI	Direct	MMR	SC	LMI
Single-session user	91.43	**92.86**	91.43	91.43	78.57	57.14	81.43	**94.29**	**95.71**	94.29	91.43	**95.71**
Single-session assistant	**80.36**	71.43	66.07	69.64	**76.79**	44.64	73.21	**76.79**	83.93	80.36	83.93	**85.71**
Single-session preference	50.00	53.33	43.33	**73.33**	56.67	40.00	46.67	**66.67**	90.00	86.67	93.33	**96.67**
Multi-session	53.38	45.11	59.40	**73.68**	47.37	36.09	59.40	**75.19**	84.96	56.39	72.93	**87.22**
Temporal reasoning	64.66	54.14	65.41	**67.67**	70.68	41.35	65.41	**75.19**	**94.74**	69.92	86.47	93.23
Knowledge update	51.28	66.67	75.64	**79.49**	65.38	41.03	67.95	**76.92**	94.87	82.05	**97.44**	93.59
Overall	64.20	61.00	67.80	**75.00**	64.60	42.40	66.20	**77.80**	90.80	73.80	85.40	**91.40**

Performance by question type. Tables 2 and 3 report Top-200 results; Appendix C.7 gives Top-50 breakdowns. Qwen's largest LongMemEval gains over Direct are in knowledge update, preference, and multi-session questions; LMI trails Direct on single-session assistant and MMR on single-session user questions. GPT-5 SC leads on knowledge update but trails LMI overall. Among Direct, MMR, and LMI on LoCoMo, LMI leads on adversarial and single-hop questions across backbones. Llama MMR leads on multi-hop and temporal questions and ties Direct on open-domain. With GPT-5, SC has the highest temporal accuracy. On open-domain questions, LMI achieves lower accuracy than Direct and MMR.

Table 3: LoCoMo accuracy (%) by question type at Top-200.

Question type	Qwen2.5-7B				Llama-3.1-8B				GPT-5			
	Direct	MMR	SC	LMI	Direct	MMR	SC	LMI	Direct	MMR	SC	LMI
Multi-hop	**73.05**	68.79	66.31	70.57	56.74	**75.18**	71.63	69.50	90.07	90.07	86.88	**92.55**
Temporal reasoning	78.82	76.01	79.13	**86.92**	69.78	**84.74**	77.88	79.44	94.08	95.33	**95.95**	95.64
Open-domain	30.21	**46.88**	23.96	22.92	**46.88**	**46.88**	12.50	30.21	**65.62**	57.29	56.25	42.71
Single-hop	74.79	71.34	74.79	**84.90**	73.25	77.29	71.82	**82.52**	92.27	88.82	90.84	**94.53**
Adversarial	40.83	52.52	59.63	**77.29**	47.48	44.27	42.89	**57.80**	90.83	90.83	92.43	**93.81**
Overall	65.54	66.40	68.47	**78.49**	63.36	69.43	63.51	**72.17**	90.64	88.97	89.78	**91.75**

4.3 Efficiency Analysis

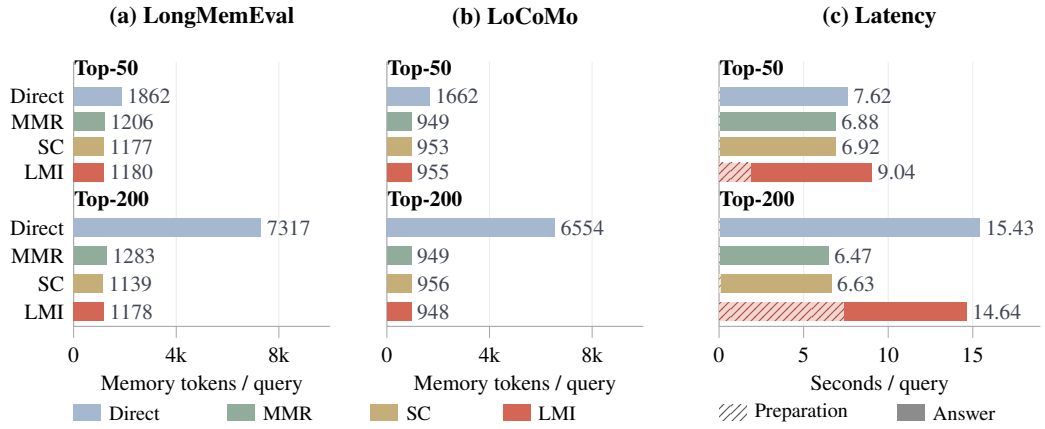

Figure 3: Memory use and latency. (a,b) Mean compiled memory tokens per query (`o200k_base`). (c) Mean preparation (hatched) and answer (solid) latency in a LongMemEval pilot with Qwen2.5-7B-Instruct, excluding retrieval and grading.

Injected-memory tokens. At Top-200, LMI reduces injected-memory tokens by 83.90% on LongMemEval and 85.54% on LoCoMo relative to Direct (Figure 3(a,b)). On LongMemEval, LMI uses fewer tokens than MMR at both depths, but slightly more than SC. On LoCoMo, MMR, SC, and LMI have similar counts; MMR is lowest at Top-50 and LMI at Top-200 (Appendix C).

Post-retrieval latency. In the separate pilot (Figure 3(c)), memory preparation outweighs LMI's answer-time savings at Top-50, increasing latency by 18.67% over Direct. At Top-200, latency falls by only 5.13%, with preparation accounting for 7.369 of 14.636 s. MMR is fastest at both depths, followed by SC. These single-run measurements exclude retrieval and grading and do not establish full-lifecycle or monetary savings, or speedups beyond this setting (Appendix C.6).

4.4 Ablation Studies

Table 4 reports ablations on LongMemEval Top-200 with Qwen2.5-7B-Instruct. Independent selection lowers accuracy at nearly unchanged memory-token counts, despite retaining the graph and learned scores. This supports adding or replacing memories according to the information already represented in the selected set, rather than relying on individual rankings alone. Removing learned utility also lowers accuracy at a similar memory footprint, suggesting that feedback-based scores improve candidate priorities within the same graph-guided selection procedure.

Table 4: Ablation results on LongMemEval at Top-200.

LMI variant	Acc. (%)	Δ Acc. (pp)	Tokens/query	Δ Tokens
Independent selection	66.60	−8.40	1,178.42	+0.04
w/o structured rendering	67.00	−8.00	1,022.38	−156.00
w/o learned utility (static)	71.00	−4.00	1,178.06	−0.32
Full	**75.00**	—	**1,178.38**	—

Δ: change from Full (accuracy in percentage points). Tokens/query: mean injected-memory tokens.

Replacing structured context with a plain list of the same selected memories also lowers accuracy, supporting the role of presentation in this configuration. This ablation removes formatting tokens, so it tests the whole presentation module rather than formatting at matched token counts (Appendix C.8). All variants retain the graph; these comparisons therefore do not isolate its contribution to the full-pipeline gains.

4.5 Case Study

Figure 4 traces the retrospective case in Figure 1, using the same frozen Top-50 candidates with Qwen2.5-7B-Instruct (Appendix A).

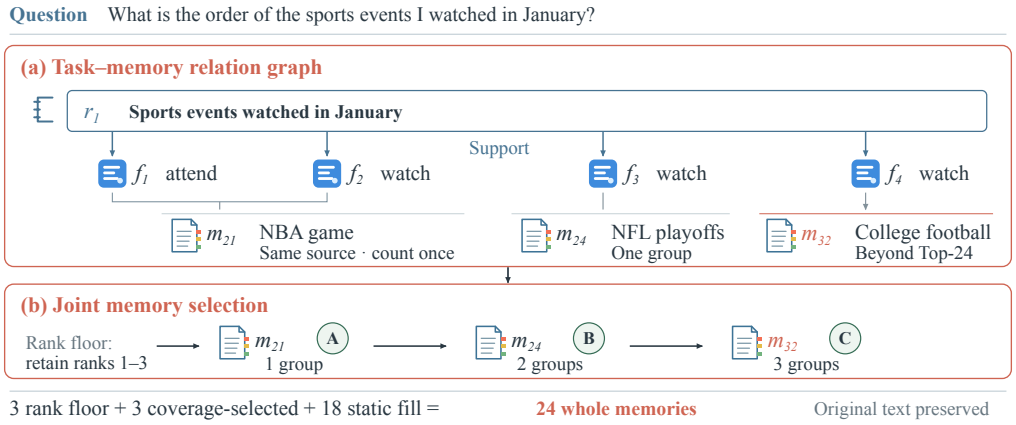

Figure 4: Requirement-scoped Support subgraph and source memories (top); cumulative group coverage during whole-memory selection (bottom). Memory labels indicate retrieval ranks.

The two NBA facts share a source proposition and count once; rank 32 adds college-football information omitted by Top-24. The Direct answer omits the NBA event, whereas LMI returns all three events in reference chronology. Coverage remains incomplete, with unresolved relations. Top-24 and LMI match item counts, not token budgets; component and learning effects are not isolated.

5 Conclusion

LMI jointly selects whole memories using task requirements and source-linked facts within a budget; utility learned from historical execution feedback guides candidate priorities. It organizes selected memories as model context. Across three LLM backbones evaluated on LongMemEval and LoCoMo, LMI uses fewer memory tokens than Direct at equal or higher aggregate accuracy.

Limitations

Whole-memory selection can retain irrelevant text. Errors or uncertainty in requirement parsing, fact extraction, or relation labeling can omit useful content or retain irrelevant content. Source grouping can count paraphrases separately or jointly count distinct facts sharing source spans or composition lineage. Coverage neither enforces fact combinations within undecomposed requirements nor guarantees sufficient information to answer. Evaluation uses fixed candidates for question answering; long-horizon interactions with memory updates or changing goals remain untested.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/pdf?id=jbNjgmE0OP.

Jaime Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 335–336, 1998. doi: 10.1145/290941.291025. URL https://www.cs.cmu.edu/afs/cs/Web/People/jgc/publication/MMR_DiversityBased_Reranking_SIGIR_1998.pdf.

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

Yiming Du, Bingbing Wang, Yang He, Bin Liang, Baojun Wang, Zhongyang Li, Lin Gui, Jeff Z. Pan, Ruifeng Xu, and Kam-Fai Wong. MemGuide: Intent-driven memory selection for goal-oriented multi-session LLM agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 30584–30592, 2026. doi: 10.1609/aaai.v40i36.40313. URL https://ojs.aaai.org/index.php/AAAI/article/view/40313.

Manu Ghulyani, Arunabh Singh, Karan Bharadwaj, Ankit Nath, and Suranjan Goswami. PACMS: Submodular context selection as a pluggable engine for LLM agents, 2026. URL https://arxiv.org/abs/2606.20047. arXiv preprint arXiv:2606.20047.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, et al. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783. arXiv preprint arXiv:2407.21783.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength natural language processing in Python, 2020. URL https://doi.org/10.5281/zenodo.1212303.

Mehmet Iscan. Learning when to remember: Risk-sensitive contextual bandits for abstention-aware memory retrieval in LLM-based coding agents, 2026. URL https://arxiv.org/abs/2604.27283. arXiv preprint arXiv:2604.27283.

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 (NAACL)*, 2024. URL https://aclanthology.org/2024.naacl-long.389/.

Eric Hanchen Jiang, Zhi Zhang, Yuchen Wu, Levina Li, Dong Liu, Xiao Liang, Rui Sun, Yubei Li, Edward Sun, Haozheng Luo, Zhaolu Kang, Aylin Caliskan, Kai-Wei Chang, and Ying Nian Wu. Memory as a controlled process: Learned adaptive memory management for LLM agents, 2026. URL https://arxiv.org/abs/2607.13591. arXiv preprint arXiv:2607.13591.

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 (EMNLP)*, 2023. URL https://aclanthology.org/2023.emnlp-main.825/.

Lilly Kumari and Jeff Bilmes. Submodular span, with applications to conditional data summarization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 12344–12352, 2021. doi: 10.1609/aaai.v35i14.17465. URL https://ojs.aaai.org/index.php/AAAI/article/view/17465.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pp. 661–670, 2010. doi: 10.1145/1772690.1772758.

Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 510–520, 2011. URL https://aclanthology.org/P11-1052/.

Jun Liu, Zhenglun Kong, Changdi Yang, Fan Yang, Tianqi Li, Peiyan Dong, Joannah Nanjekye, Hao Tang, Geng Yuan, Wei Niu, Wenbin Zhang, Pu Zhao, Xue Lin, Dong Huang, and Yanzhi Wang. RCR-Router: Efficient role-aware context routing for multi-agent LLM systems with structured memory, 2025. URL https://arxiv.org/abs/2508.04903. arXiv preprint arXiv:2508.04903.

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. doi: 10.1162/tacl_a_00638.

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. doi: 10.18653/v1/2024.acl-long.747. URL https://aclanthology.org/2024.acl-long.747/.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. A survey of context engineering for large language models, 2025. URL https://arxiv.org/abs/2507.13334.

OpenAI. GPT-5 system card. Model system card, 2025. URL https://openai.com/index/gpt-5-system-card/.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems, 2023. URL https://arxiv.org/abs/2310.08560. arXiv preprint arXiv:2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. 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 (UIST)*, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Zhiruo Wang, Jun Araki, Zhengbao Jiang, Md Rizwan Parvez, and Graham Neubig. Learning to filter context for retrieval-augmented generation, 2023. URL https://arxiv.org/abs/2311.08377. arXiv preprint arXiv:2311.08377.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.10813.

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 (NeurIPS)*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

Yue Xu, Yutao Sun, Yihao Liu, Mengyu Zhou, Jiayi Qiao, Lu Ma, Kai Tang, Wenjie Wang, Xiaoxi Jiang, and Guanjun Jiang. From passive retrieval to active memory navigation: Learning to use memory as a structured action space, 2026. URL https://arxiv.org/abs/2607.05794. arXiv preprint arXiv:2607.05794.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Haozhen Zhang, Haodong Yue, Tao Feng, Quanyu Long, Jianzhu Bao, Bowen Jin, Weizhi Zhang, Xiao Li, Jiaxuan You, Chengwei Qin, and Wenya Wang. Learning query-aware budget-tier routing for runtime agent memory. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2602.06025.

A Motivating Case: Replay Protocol and Scope

Figure 1 uses the LongMemEval question `gpt4_45189cb4`: "What is the order of the sports events I watched in January?" The reference order is NBA, College Football National Championship, and NFL playoffs. The displayed events correspond to retrieval ranks 21, 32, and 24, respectively, and are arranged chronologically rather than by retrieval rank. The date attached to the NFL excerpt is approximate.

Conditions. All three conditions share the same frozen Mem0 Top-50 candidate list. *Inject all* uses all 50 MemoryItems; *Top-24* retains original ranks 1–24; and *LMI* uses a previously frozen 24-item selection and its context organization. This historical case uses static utility, unlike the learned-utility configuration in Section 4.2. It illustrates structured selection and context organization, not the benefit of feedback learning. For the answer comparison, we replayed the downstream answer model, not retrieval or the LMI decision process. The selected ranks for LMI were 1, 2, 3, 4, 9, 11, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, and 39. Each injected item retained its complete memory text; the figure abbreviates the excerpts and answers. Unfilled cells denote omitted injections, not deletions from the host memory store.

Replay and verification. We used the same Qwen2.5-7B-Instruct model revision, BF16 precision, greedy decoding, and a fresh prompt/KV cache for every request. This illustrative replay is separate from the FP16 aggregate runs in Table 6. Three independent process rounds ran each condition once, giving nine answer generations. The three full outputs within each condition were byte-identical. All conditions terminated normally rather than reaching the output limit. Prompt content outside the memory span was identical; LMI-specific guidance, labels, and organization inside that span remain part of the system treatment. We checked the answers against the benchmark reference and original sessions manually; no official judge was run for this replay.

Offline decision-trace verification. For Section 4.5, we separately reconstructed the decision process using the frozen inputs and a historical execution snapshot whose source, resource, and parser-environment hashes match the original run. Replaying its original 100-question batch reproduced the stored graph/planning and compiled-output hashes, the full freeze manifest, and this case's exact selection, context, and answer prompt. We then observed the original selector without modifying its decisions; the resulting plan hash and selected sets also matched. These are reconstructed records checked against historical hashes, not originally persisted step-by-step logs. No retrieval, answer-model, or judge calls were made during this audit, and neither reference answers nor historical selections were inputs to the decision algorithm.

The case contains one requirement and 120 facts. The 120 requirement–fact pairs comprise four Support, one Related, 43 Mismatch, and 72 Unresolved judgments. At the memory level, nine candidates remain plausibly relevant but unresolved. Although the three identified Support groups are all selected, both candidate coverage and the planner outcome retain an `abstained` completeness status. This is an internal uncertainty marker, not a refusal by the answer model. The NBA group joins two facts from the same source proposition; no composed facts or explicit cross-memory temporal edges occur in this trace. January remains in the requirement's target fields but is not extracted as a separate temporal constraint.

Selection and compilation details. Figure 4 reports the three coverage additions at ranks 21, 24, and 32; its displayed facts f_1–f_4 correspond to the two NBA facts and the single NFL and college-football facts, respectively.

The frozen profile allows 24 items and a planner memory-token budget of 1280, with a rank-safe floor of three. The final selection costs 770 planner tokens, so it reaches the item cap, not the token cap. The coverage phase adds ranks 21, 24, and 32; static-utility filling adds the other 18 items. The frozen compiler places the first eight selected items by retrieval rank in its primary section and sorts the remaining items by record date and rank in its background section. Thus, the three directly supporting memories are not all placed first. All 24 memory texts are preserved. Including guidance and formatting, the compiled context contains 1134 tokens under the shared tokenizer; the complete model-native answer inputs contain 4529, 3742, and 3944 tokens for inject-all, Top-24, and LMI, respectively. Planner cost, compiled-context size, and full answer-input size are distinct measures.

The case was selected retrospectively from a fixed six-question screen and is illustrative rather than an estimate of average performance. LMI retains the repeated NFL information at ranks 24 and 29; the figure does not imply that every duplicate is removed.

B Method Execution and Feedback Protocol

This appendix specifies semantic extraction, relation classification, support grouping, and the selection and feedback records underlying Section 3. The selection details distinguish graph-guided set updates from ranking and source-retention rules; they do not extend the learner's scope.

B.1 Semantic Extraction and Provenance

Source-bound extraction. The frontend uses a frozen local spaCy parser, dependency rules, and lexical resources; it does not call the answer model. Query requirements retain subject, predicate, target, metric, and temporal bindings when recognized. For memory text, event verbs and metric-bearing copulas supply predicate candidates. The primary dependency-based branch keeps at most four such candidates per MemoryItem; fallback branches can retain partial facts. Each fact retains its predicate span, parent source span, and original MemoryItem identifier. Extraction need not enumerate every proposition; failure to extract a fact does not remove the original candidate.

Selection consumes these source-bound views separately from the extraction procedure. A replacement frontend would need to preserve the requirement, fact, provenance, and uncertainty fields and the selection-time information boundary. Alternative frontends require separate evaluation; the reported configuration does not use an LLM fallback for unresolved relations.

Composition is restricted to adjacent source spans within one MemoryItem. The left span must end in a continuation marker (a colon, semicolon, or dash) and provide a bound predicate and subject; the right must add a metric, temporal binding, or list content. Known content roles and states must not conflict. A composed fact retains the left predicate and subject, the combined source span, and links to its component facts; the original facts remain. No cross-memory proposition is synthesized.

B.2 Relation Classification

Relation checks and decision order. Every requirement–fact pair receives seven Boolean checks (Table 5), using source-bound arguments, frozen lexical families, type matching, and explicit composition links. The classifier applies the following order:

1. Assign SUPPORT when all seven checks pass.
2. Otherwise, assign RELATED when subject, target, and content role pass. Failed predicate, metric, state, or temporal checks therefore do not by themselves exclude this category, which does not establish answer support.
3. Otherwise, assign MISMATCH only under a decisive mismatch rule. If either extracted item is marked abstained, this rule does not fire. A failed state or content-role check, or a subject mismatch without a bound user agent, is decisive. A target, predicate, or metric mismatch is decisive only when no recorded ambiguity remains.
4. Assign UNRESOLVED to the remaining pairs.

Only SUPPORT-labeled facts enter answer-support groups; the separate context-history grouping branch is specified below. Compact storage may omit explicit MISMATCH records, but not their classification from the complete requirement–fact candidate space. Paper labels SUPPORT, RELATED, MISMATCH, and UNRESOLVED correspond to the unchanged implementation values `direct_support`, `context_only`, `not_support`, and `abstained`, respectively; the negative default remains compactly stored.

These checks have implementation limits. Temporal matching is heuristic: it uses lexical matching and relative-window cues, includes a permissive branch for observed temporal strings containing digits, and does not fully normalize date intervals. Extraction status alone also does not determine the relation label. Neither SUPPORT nor MISMATCH is therefore an independently verified entailment judgment.

Table 5: Operational checks for requirement–fact classification. Passing these checks is a rule-based judgment, not a semantic guarantee.

Check	Source-bound information tested
Subject	Requested entity or a locally resolved user-agent reference.
Predicate	Requested action or property, including permitted lexical-family and metric/state-based entailment rules.
Target	Requested object, type, and scope qualifiers, using bound arguments and allowed composition links.
Metric	Requested quantity or attribute attached to the relevant content; counting and enumeration use their designated rules.
State	Required status, such as current, completed, or planned.
Temporal	Recognized temporal bindings and the frozen matching rules.
Content role	Compatibility with user facts, user plans, or prior recommendations; this is not an agent-routing role.

State and temporal checks assess a fact's applicability to a requirement. They do not construct general plan–execution–completion chains, infer cross-memory update relations, or resolve contradictions by choosing the latest record. Record-time ordering in the compiler is a presentation rule, not a conflict-resolution procedure.

B.3 Support Grouping and Coverage Status

Source groups and requested counts. For an answer-support requirement r_j, initialize its SUPPORT facts $\mathcal{F}_{t,j}$ as separate components. Construct the undirected links $E^{ff}_{t,j}$ by joining facts with equal source-proposition text hashes, and a composed fact to its recorded components within one memory. Both endpoints must be eligible under the same requirement. Their connected components, including isolated facts, give H_{jk}; projecting through E^{fm}_t gives the provider sets P_{jk} in Equation (4). Hash equality can join exact source-text copies across memories, but need not join paraphrases. Composition links express shared source lineage, not arbitrary semantic equivalence. The requirement-tagged links collectively form the derived relation E^{ff}_t in Equation (3); they neither introduce additional requirement–fact labels nor score memory–memory similarity. Grouping is requirement-specific, not a global partition of facts, and g_j is not a verified count of distinct real-world events.

Shared memory membership alone does not merge facts. However, the native extractor can bind several predicate-level facts to the same sentence span. If eligible under the same requirement, these facts share a text hash and are counted together even when they express different information. Composition links can likewise join distinct components through a composed fact. Conversely, semantically equivalent paraphrases without either source link can remain separate groups. Thus grouping can undercount distinct information or overcount semantic repetition; it controls recognized source-based repetition rather than establishing semantic equivalence or complementarity. These links do not delete facts or rewrite memories: selecting a provider injects its whole original text.

Under the separate-unit definition in Section 3.4, each requirement unit r_j has its own target τ_j. The target is the requested number for a fixed-count unit and one for a single-answer unit. A question requiring both endpoints of the same event or object requires two units, one per endpoint, each with target one. A pooled target of two is not equivalent to these separate targets. For all-matching or ordered-set units, the target is $\max(1, g_j)$ and is explicitly candidate-bounded. Selection caps coverage credit at $\kappa_j = \min(\tau_j, g_j)$; this cap accounts for available groups, not budget feasibility. The separate pre-selection cardinality record is *met* if $g_j \geq \tau_j$, *partial* if $0 < g_j < \tau_j$, and *missing* if $g_j = 0$. Clipping the selection target does not erase a shortfall.

Counting versus information completeness. The count in Equation (5) applies within each unit, using its own fact matches, groups, and target. Groups matched to one unit cannot compensate for missing support in another: for example, two start-time groups cannot satisfy the end-time unit. A memory can provide groups for several units, so separate evaluation does not require separate mem-

ories. When a requirement is decomposed, only its parts contribute to the coverage sum; the parent is not counted again. The composite requirement is considered covered only when the selection meets every required unit's original target and no required unit remains unresolved.

The cap bounds selection credit without lowering the original target. If $\tau_j = 3$ but $g_j = 2$, selecting both groups reaches $\kappa_j = 2$ while the cardinality record remains partial and the target remains unmet. For all-matching requirements, an unknown information need may not appear as a numerical shortfall because the target is itself candidate-bounded.

This coverage definition depends on the frontend identifying the necessary units and matching facts to their respective roles. The fixed parser does not guarantee exhaustive decomposition for arbitrary questions, and group counts do not check required fact combinations left within one unit. Coverage describes recognized support in the fixed candidates, subject to extraction, classification, and grouping errors; it does not establish semantic distinctness or answer sufficiency.

Personal-history context groups. The `personal_input_context` branch instead admits RELATED facts carrying an explicit source-bound personal-history qualification. It applies the same source-link grouping and provider projection, but records context-history groups rather than answer-support groups. These groups enter the planner's group counts with $\tau_j = \max(1, g_j)$; the requirement's overall status remains unresolved, even when all available groups are selected. Thus their contribution denotes retained historical context, not an independent answer fact or established answer completeness. The RELATED label alone does not qualify a fact for this branch.

Unresolved coverage. The uncertainty check considers UNRESOLVED pairs with compatible state and content role. Among active subject, predicate, target, metric, and temporal checks, at least one must match and at most one may fail; a failed check needs a corresponding ambiguity record. If all active checks match, incomplete extraction or missing decisive proof must explain the abstention. Within a memory, Support relations take precedence over Related, and either takes precedence over such unresolved relations. A requirement with a remaining plausibly relevant unresolved memory has overall coverage status *abstained*, independently of its cardinality record. Otherwise its status is missing, partial, or candidate-bounded complete according to the counts above. Generic extraction failures need not pass this relevance test. Candidate-bounded completeness describes recognized groups, not exhaustive extraction or guaranteed answer sufficiency. This aggregate status is distinct from the coverage increment in Equation (5). Outside the context-history branch, non-Support labels add no groups, but their source-bound records can constrain removal. Related records with additional source-qualified justification can also make a memory eligible for a set update; the Related label alone is insufficient. Neither use changes coverage counts or triggers supplementary retrieval. The learned utility component uses per-memory label proportions and related coverage statistics as features. Utility learning does not change the relation labels or the coverage-counting rule.

B.4 SELECTION AND COMPILATION PROTOCOL

Source and selection records. Each extracted fact retains its original MemoryItem identifier, text span, and content hash, so relation judgments can be checked against the retrieved text. The semantic view does not replace the original memory. If the text does not establish a relation, the frontend preserves that uncertainty. The graph records query–requirement links as `requires`, fact–memory links as `sourced_from`, and neutral requirement–fact links as `candidate_for`. Relation labels and the derived links E_t^{ff} complete the logical graph; the implementation stores their judgments and group memberships in separate analysis records rather than materializing every derived fact edge in the base graph. Selection consumes the resulting provider sets and retention records, not a path-search or graph-neural score. Query–requirement links organize the per-requirement objectives; fact–memory links make those objectives actionable at the whole-memory level. Group membership determines coverage gains, while source-retention records constrain admissible swaps. For each attempted set update, the planner records the candidate, the before/after selections, added groups or qualified context, and any removed item or rejected trial with its reason. These records distinguish an accepted addition, a one-item exchange, and a blocked update.

Initialization and graph-guided updates. The initial set preserves S_t^{safe}, the first $\min(r_{\text{safe}}, n)$ candidates in the host's returned order, independently of their relation labels. The default $r_{\text{safe}} = 3$;

this prefix must fit both budgets in full or planning raises an error. Initial filling visits all remaining candidates in descending individual score order, skipping those that do not fit, without a relation filter or utility threshold. A meaningful rank safeguard depends on the host's ordering; an arbitrary enumeration is not a relevance ranking.

Graph-guided updates consider unselected memories with positive marginal coverage gain or an existing Related record carrying source-qualified context justification. Candidates are ordered by k_{cov} below. For each candidate, the planner first tries adding it without removal, then tries replacing one selected item at a time in ascending individual priority. The first valid trial is accepted. Every trial must remain within both budgets, preserve the rank prefix and all represented groups, and retain required source information. In particular, recorded temporal, metric, or state information, contextual or unresolved records, and incomplete extraction can prevent removal even when group counts would be unchanged. Newly admitted preference context must retain the source memory that justifies its relevance. For new-advice questions, a narrow exception permits removal of unresolved prior advice only when every fact and retention record in that memory qualifies and no explicit temporal or numeric information is present; known user facts and plans remain protected. These are source-based checks, not a proof that every relevant detail is preserved.

After an accepted update, all remaining candidates are reconsidered against the new set. The procedure stops when no candidate has an admissible addition or one-item exchange. Token cost determines feasibility rather than normalizing gain. Stopping does not certify task completeness. This heuristic does not guarantee a minimum-size, duplicate-free, or globally optimal set.

Context construction retains selected identifiers and source mappings. The frozen Primary limit determines how many of the highest-ranked selected items enter Primary; they retain retrieval-rank order. The rest enter Background in ascending record-timestamp order, with retrieval rank as the tie-breaker. Every selected item is rendered exactly once. These formatting rules are shared by Full LMI and its static ablation; they are not learned or derived from Support labels.

Ranking scores. The static key is $\mathbf{s}_0(m_i) = (s_{\text{sem}}, s_{\text{BM25}}, s_{\text{entity}}, 1/\operatorname{rank}(m_i))$, using semantic similarity, BM25, entity match, and rank prior in that order, not a weighted scalar score. Full LMI prepends learned utility, using $\mathbf{s}_t(m_i) = (u_t(m_i), \mathbf{s}_0(m_i))$. The ablation without learned utility uses $\mathbf{s}_t = \mathbf{s}_0$. Graph-update candidates and initial filling use the following descending lexicographic keys, respectively:

$$\begin{aligned} k_{\text{cov}}(m_i \mid S) &= \big(\Delta_{\text{cov}}(m_i \mid S), \mathbf{s}_t(m_i), -\operatorname{rank}(m_i)\big), \\ k_{\text{fill}}(m_i) &= \big(\mathbf{s}_t(m_i), -\operatorname{rank}(m_i)\big). \end{aligned} \tag{10}$$

Vector entries are expanded in these comparisons; retrieval rank breaks remaining ties. The keys order trials but do not override their retention or budget checks. Fixed inputs and tie-breaking make selection replayable. Learned utility therefore orders initial budget filling and breaks coverage-gain ties within the same structured selection procedure; it does not replace coverage gains or retention constraints.

Frozen context template and token accounting. The current configuration counts selection-stage text costs with `o200k_base`: $c_t(m_i)$ is the token count of the exact string `- ` followed by the original memory text. The sum of these independently counted costs is checked against T_t. It does not include all compiled-context overhead and is not the token count of the concatenated final context. The separately measured T_t^{ctx} tokenizes the entire compiled memory section, including guidance, section headings, roles, dates, and newlines. Full answer-input tokens additionally include the unchanged host prompt outside that section. These counts must not be interchanged.

The following fixed prefix starts the memory section. Typesetting may wrap a line; each bullet below corresponds to one output line.

```
[Memory-use guidance]
- Use Primary evidence first; use Background only when
it directly supports the question.
- Plans and historical recommendations are not proof
of completion or current facts.
```

The frozen query view conditionally appends the following lines, in the listed order; several may apply:

1. Advice or decision required: `- For advice or a decision, prioritize established user facts and current constraints.`
2. Temporal reasoning required: `- Use explicit event dates; a memory record timestamp is source metadata, not event time.`
3. Comparison required: `- Use both relevant endpoints and keep different entities or contexts separate.`

The memory records then use this template, repeating each record line once per item in the ordering specified above:

```
[Primary evidence]
- [roles] [YYYY-MM-DD] original memory text
[Background evidence]
- [roles] [YYYY-MM-DD] original memory text
```

Bracketed roles are the frozen source-role strings `established fact`, `user plan`, `prior recommendation`, or `other history`; multiple roles are joined by ` + `. The date is the source record's date, not an inferred event date. Background and its heading are omitted when empty. All output lines are joined with a single newline, and this section replaces only the host's original memory span.

B.5 Feedback Credit and Learning Boundary

Submission-aligned feedback. For every final model request, the submission record identifies the memories, text spans, and token counts actually sent to the model. A planned selection alone is not an exposure record. Feedback must be aligned with the submitted memory and its original selection-time feature version; content that was selected but never submitted cannot produce a learner observation. Reported token use is likewise measured from the final request. The compiled prompt contains the full planned set; the submission record distinguishes actual exposure from successful compilation rather than granting an additional selection step.

Consistent evaluation of paired runs. During training, the paired removal comparison in Equation (7) keeps the question, candidate set, answer model, prompt template, and decoding configuration fixed, changing only the presence of the target memory. Answer generation is completed before judging. After official answer normalization, the question, reference answer, normalized prediction, exact judge prompt, judge model, and judge decoding configuration jointly define an evaluation identity. The evaluation pipeline requests one verdict per identity and reuses it across all corresponding experimental cells. Different judge samples for an identical input therefore cannot be counted as a treatment effect.

Credit, confidence, and abstention. When repeated removal effects agree in direction, their mean forms the proxy-credit target y_h and receives the full pre-specified confidence weight. Partially informative agreement, such as one zero effect and another with a stable nonzero direction, receives a smaller w_h. Opposing effects, invalid requests, and indeterminate outcomes cause abstention and produce no learner observation. The target estimates a removal contribution conditional on the submitted set, not a memory's intrinsic value. Interchangeable memories can each have zero removal effect despite providing necessary information; repetition reduces execution noise but cannot resolve this attribution ambiguity. Token use, latency, and request cost are recorded separately rather than incorporated into correctness credit.

Update and evaluation boundary. During training, only valid, submission-aligned observations update A_t and b_t in Equation (8). The resulting read-only snapshot is published after the current episode and may affect only later episodes. Validation and test use a frozen learned snapshot with $\beta = 0$ and no updates; freezing retains the learned utility rather than reverting to static scores. Reference answers supervise training scores, but are not selection features or answer-model inputs.

Deployment with the frozen model requires no reference answer. Further learning from user ratings or task outcomes requires a separate protocol; an observed outcome alone is not a paired removal target. Learned utility orders budget filling and resolves coverage ties, but it cannot change graph relations, content groups, coverage targets, rank-safe items, hard selection limits, or source text.

C Experimental and Reproducibility Details

This appendix consolidates the execution and accounting details behind Section 4. It covers the three answer models reported in the paper: Qwen2.5-7B-Instruct, Llama-3.1-8B-Instruct, and GPT-5. The grading model is part of the evaluation protocol and is not treated as a fourth answer model.

C.1 Benchmark Scope and Scoring

For each benchmark, answer model, and retrieval depth, we compare Direct injection, Budgeted MMR (MMR), semantic coverage (SC), and LMI on the same questions and frozen candidate scope wherever results are available. The methods use the same accuracy denominator within each comparison; outputs that cannot be graded remain in the denominator without correctness credit.

LongMemEval uses all 500 evaluation questions at Top-50 and Top-200. Direct injection places all retrieved Mem0 candidates in the memory span. MMR, SC, and LMI select whole MemoryItems from the same frozen list and compile their original text, without supplementary retrieval or memory rewriting.

LoCoMo uses all ten conversations and the same 1,986 questions at both retrieval depths, with a shared GPT-4o-mini-derived Mem0 corpus. This scope includes `conv-30` and `conv-42`, which were used during method development; it is not a wholly held-out conversation split. The input set contains 1,540 ordinary and 446 adversarial questions. Ten adversarial questions have semantically equivalent answer choices under our choice-based protocol and are excluded from accuracy scoring. The accuracy denominator is therefore 1,976: 282 multi-hop, 321 temporal, 96 open-domain, 841 single-hop, and 436 adversarial questions. These question-level exclusions differ from ungradable model outputs, which remain in the scored denominator without correctness credit. SC results are reported for all three answer models at both depths. Scoring and output exceptions are specified in Section C.4; aggregate accuracies appear in the main tables and type-level results in Section C.7.

C.2 Models, Environments, and Run Provenance

Table 6: Answer-model configurations for the aggregate accuracy experiments.

Answer model	Runtime	Generation configuration	Identity boundary
Qwen2.5-7B-Instruct	Remote vLLM 0.16.0 on four V100 GPUs; FP16	Temperature 0; 4,096-token output limit; 20,480-token context window	Frozen revision `a09a35458c`
Llama-3.1-8B-Instruct	Remote vLLM 0.16.0 on four V100 GPUs; FP16	Temperature 0; 4,096-token output limit; 20,480-token context window	Frozen revision `0e9e39f249`
GPT-5	OpenRouter, OpenAI provider only	No fallback; 8,192-token output limit; service-default temperature and reasoning effort	Floating `openai/gpt-5` alias, not an immutable snapshot

Full revisions: Qwen `a09a35458c702b33eeacc393d103063234e8bc28`; Llama `0e9e39f249a16976918f6564b8830bc894c89659`.

LoCoMo run records. Table 7 preserves the earlier Direct/MMR/LMI run records; these dates do not identify the subsequent full-data SC results. The Qwen SC aggregate and type-level results use the September 25 full-data summary. Its Top-50/Top-200 correct counts are 1,375/1,353 out of

1,976. GPT-5 SC answers 1,781/1,774 correctly at Top-50/Top-200 (90.13%/89.78%), with 195/202 incorrect and no ungradable outputs in the scored set. Llama SC answers 1,315/1,255 correctly (66.55%/63.51%), with 661/715 incorrect and 0/6 ungradable outputs. All six ungradable Top-200 outputs remain in the accuracy denominator without correctness credit, but are recorded separately from semantic errors. These summaries do not establish repeated-run stability.

Table 7: Earlier three-method LoCoMo run records (2026).

Answer model	Run date	Completion and recovery
Qwen2.5-7B	Sep. 17	Both depths complete, including the protocol-authorized Top-200 replacement (Section C.4).
Llama-3.1-8B	Sep. 18	All six arms complete; no unknown or unsettled submissions.
GPT-5	Sep. 19	Both depths complete; all scored answers evaluable, no unsettled submissions.

LongMemEval run completion. The Qwen Top-50 MMR run resumed across September 15–16 and passed replay checks without new model calls. Llama's interrupted MMR run completed through a separately audited continuation; both the Llama and GPT-5 MMR runs also passed replay checks without new model calls. Submission exceptions and their accounting are specified in Section C.4.

SC runs. The original LongMemEval SC runs completed on September 25 at both depths for all three answer models. Qwen's frozen SC answers were rejudged with GPT-5; Llama and GPT-5 generated answers from the same SC contexts. The answer-model configurations follow Table 6. The reported Llama SC scores use the September 26 explicit-answer recovery and GPT-5 regrading described in Section C.4; no new Llama answers were generated. The encoder and selection parameters below are fixed across answer models; SC does not fit or load a learned utility model.

C.3 Frozen Retrieval, Selection, and Prompts

Top-50 and Top-200 denote frozen ordered Mem0 candidate lists of at most 50 and 200 MemoryItems. Candidate text, retrieval order, and query metadata are fixed before applying Direct injection, MMR, SC, or LMI; non-memory prompts remain bound to the archived runs. Direct and LMI also share answer configurations and grading procedures within their paired runs. The aggregate LMI comparisons use structured joint selection with learned utility, a protected rank prefix, an approximately 24-item selection cap, and the source-preserving compiler in Section 3. Training and frozen evaluation follow Section C.9; ablation definitions and controls are in Section C.8. Exact candidate snapshots, prompt identifiers, selection caps, and run-specific limits remain bound to the archived run records; results from different bindings are not pooled.

The answer-generation prompts follow each benchmark's frozen evaluation templates used by the Mem0 suite. Prompt construction, candidate snapshots, and compiled memory contexts are reused across answer models where the paper reports shared-context token counts. Such reuse controls the memory input; it does not make the model executions or their tokenizers identical.

Budgeted MMR comparison. The Qwen Top-200 MMR result uses a frozen quantized ONNX export of `bge-small-en-v1.5`, normalized embeddings, cosine similarity, and $\lambda = 0.5$, with caps of 24 whole memory items and 1,280 selection tokens. The token cap counts item text and bullet prefixes; the reported memory span also includes compiled guidance and formatting. Selected text is not rewritten, and MMR uses the same presentation policy as LMI. Answer generation follows Table 6.

The Llama and GPT-5 MMR runs reuse the exact Qwen MMR selected texts and compiled contexts at both depths, with the configurations in Table 6. GPT-5 also reuses the frozen prompts.

Semantic coverage comparison. SC adapts the combination of facility-location coverage and modular query relevance used by MixModSub (Kumari & Bilmes, 2021). Its encoder, normalization, and dual-budget greedy selection are adapted to memory injection; it is not a full reproduction of MixModSub. SC uses a frozen quantized ONNX export of `BAAI/bge-small-en-v1.5`,

with 384-dimensional, CLS-pooled, L2-normalized embeddings. The query prefix is "Represent this sentence for searching relevant passages: "; memories have no prefix. For query embedding $\mathbf{q}$ and candidate embeddings $\mathbf{e}_i$, let $r_i = \max(0, \cos(\mathbf{q}, \mathbf{e}_i))$, $w_j = r_j / \sum_\ell r_\ell$, and $s_{ji} = \min(1, \max(0, \cos(\mathbf{e}_j, \mathbf{e}_i)))$. Weights are uniform when all r_i are zero. With item cap $K = 24$, SC selects whole memories using the fixed objective

$$F_{\mathrm{SC}}(S) = \frac{1}{2} \sum_j w_j \max_{i \in S} s_{ji} + \frac{1}{2K} \sum_{i \in S} r_i, \qquad F_{\mathrm{SC}}(\varnothing) = 0. \tag{11}$$

Here, j and ℓ range over all retrieved candidates, and S contains the indices of the selected memories. The first term represents query-weighted candidate content; the second retains relevance of the selected items themselves. This embedding-based coverage is not a test of task-requirement completeness.

Both component scores, before multiplication by $1/2$, lie in $[0, 1]$ for feasible sets. We use equal weights as a simple default that retains both candidate representativeness and direct query relevance, not as an optimum selected by a hyperparameter search. The weights are fixed across questions, answer models, and retrieval depths; equal coefficients do not imply equal marginal contributions at each selection step.

The caps of 24 items and 1,280 selection tokens match those used by LMI and MMR rather than being tuned separately for SC. They control available selection resources, not the exact tokens used by each method. The reported SC version (V3) retained these caps from the first SC version and froze its objective and equal weights before its answer-generation evaluation. Earlier-version results and failure cases had informed method development; this freeze does not imply that the evaluation questions were unseen during development.

Starting from the empty set, SC greedily adds the feasible item with the largest marginal objective gain, not gain per token, under caps of 24 items and 1,280 selection tokens. Gains and relevance scores are rounded to 12 decimal places for comparison; ties favor higher query relevance, earlier retrieval rank, then lexicographically smaller item ID. Selection stops at the item cap, when no item fits, or when no rounded gain is positive. The result is compared with the best budget-feasible singleton under the same objective; objective ties favor lower token cost, then sorted rank–ID order. SC has no protected rank prefix or preliminary text deduplication. Invalid or misaligned embeddings are rejected, not silently replaced.

Selection cost uses `o200k_base` on each original memory text with its bullet prefix. The shared compiler adds guidance, roles, dates, and formatting outside this selection budget, retaining each selected original text once. SC does not use LMI's task–memory graph, joint planner, learned utility, or graph-based replacement constraints. It makes no selection-time LLM calls and does not access reference answers or task outcomes.

C.4 Output Handling and Grading

LongMemEval. Output handling follows the frozen protocol for each reported run. For the reported Llama Direct/LMI cells, the recovery procedure reads only an existing model response, preserves a valid answer verbatim, and permits at most one short final-answer recovery. It has no access to the question, memories, reference answer, or score. The GPT-5 Top-50 record includes one protocol-authorized replacement of a missing direct-injection response. The GPT-5 Top-200 Direct/LMI runs share format-screening and recovery rules. These operations concern answer serialization and do not rerun retrieval or selection; the Top-200 Direct record includes two format-recovered answers. Llama MMR instead uses the ordinary-text answer contract without additional final-answer recovery: its 15 Top-50 and 18 Top-200 answers that reach the output limit remain under the frozen grading rules. GPT-5 MMR uses the same verbatim answer extraction as the other GPT-5 conditions. For Llama, the accuracy gaps relative to MMR in Table 1 do not isolate the effect of memory selection from differences in output handling.

For Llama SC, we recover an explicit conclusion only when official cleaning returns an empty answer, the archived response ends normally (`finish_reason=stop`), and the entire response consists of one complete, non-nested `mem_thinking` block. We extract the original text after the last explicit conclusion marker (`Answer:`, `Final answer:`, `Conclusion:`, `Therefore,`, or `the answer is:`), trimming only surrounding whitespace. The same rule applies at both

depths and reads neither the question nor the reference answer or previous score. Nonempty answers and truncated responses remain unchanged.

This restores 131/140 answers at Top-50/Top-200. Only these 271 answers are regraded with GPT-5 under the same frozen judge prompt; the other 729 scores are retained. Correct counts rise from 248/236 to 342/331 out of 500 at each depth (68.40%/66.20%). These are locally recovered results, not scores under the unmodified official cleaning rule. The increase reflects extraction of existing conclusions, not a change to memory selection. Candidates, selected memories, prompts, original responses, and injected-memory token counts remain unchanged.

All 500 questions are evaluable in the Qwen Top-50 MMR cell and in each Llama and GPT-5 MMR cell at both depths. Four original Qwen Top-50 MMR attempts with uncertain submission status remain separately recorded; missing usage is not treated as zero.

Following the official default configuration, all accuracy cells in the main LongMemEval table have completed common GPT-5 judging, separate from answer generation, using the frozen LongMemEval grading templates from the Mem0 suite. Accuracy uses the fixed denominators in Section C.1. Relative gain is $(A_{\mathrm{LMI}}/A_{\mathrm{direct}} - 1) \times 100\%$, computed before rounding.

For GPT-5 at Top-200, Direct, MMR, and LMI results are aligned by question ID under this shared evaluation protocol. The evaluation includes ten questions used during selector development. Excluding these questions, Direct and LMI answer 444/490 and 447/490 correctly, respectively.

LoCoMo. For ordinary questions, GPT-5 judges correctness against reference answers using the frozen Mem0 LoCoMo judge prompt, rather than the LongMemEval grading templates. Adversarial questions instead use deterministic matching against the designated answer choice, without an LLM judge. Overall accuracy sums correct answers from both categories over the 1,976 scored questions in Section C.1; it is not an unweighted average of question-type accuracies. Although all ten conversations are covered, the reported accuracies use this adapted mixed scoring protocol, not an unmodified reproduction of the official LoCoMo evaluation.

Qwen permits one 8,192-token recovery after reaching its 4,096-token output limit, but none for semantic errors. For the Top-200 unknown submission, scoring uses the protocol-authorized replacement under the same frozen prompt. The original is audit-only, with unknown token usage; the question counts once in the denominator. GPT-5 answers are not retried for truncation or semantic errors. Qwen LMI retains one ungradable single-hop output at each depth. For Llama, Direct/MMR/LMI have 2/0/2 ungradable outputs at Top-50 and 40/2/1 at Top-200. All remain in the accuracy denominator, so LMI's Top-200 gain over Direct includes output-evaluability differences and cannot be attributed solely to memory selection. All scored GPT-5 outputs are evaluable.

C.5 Token Accounting and Claim Boundary

Reported injected-memory tokens measure the frozen compiled memory contexts, including guidance and formatting, rather than the selection-stage text cost. Counts use the shared `o200k_base` tokenizer. Aggregate reduction is calculated from the summed token counts within each benchmark, rather than by averaging per-question percentages. This measure is not a model-native billing count. Complete answer-input tokens additionally include the fixed non-memory prompt and chat template. Recovery output and offline judging are separate answer-side and evaluation costs. On LongMemEval, complete answer-input reductions at Top-50/Top-200 are 14.00%/61.38% for Qwen and 15.42%/62.03% for Llama. For LongMemEval, we do not infer an unreported GPT-5 total-input reduction from the shared memory-context counts.

At Top-50/Top-200 on LongMemEval, MMR uses 1,206.19/1,282.66 memory tokens per question (35.22%/82.47% below Direct), while LMI uses 1,180.28/1,178.38 (2.15%/8.13% below MMR). Qwen and Llama share the MMR contexts. On LoCoMo, LMI uses 954.84/947.90 tokens per question, reductions of 42.56%/85.54% from Direct; MMR uses 949.28/948.86 tokens. The averages include guidance and formatting over the complete frozen inputs.

SC uses 588,744/569,643 compiled memory tokens at Top-50/Top-200 on LongMemEval, or 1,177.49/1,139.29 per question over 500 questions, measured with the same `o200k_base` tokenizer.

For Llama and GPT-5 SC on LoCoMo, the corresponding totals are 1,892,974/1,898,447 tokens over all 1,986 inputs, including the ten questions excluded from accuracy scoring. Mean compiled memory lengths are therefore 953.16/955.91 tokens per input, not totals divided by the 1,976-question accuracy denominator.

The GPT-5 LoCoMo Direct/MMR/LMI runs reuse their corresponding frozen memory contexts and have the same `o200k_base` memory-token counts. At Top-50/Top-200, LMI reduces complete answer-input tokens relative to Direct by 24.95%/72.55%, and answer-plus-grading tokens by 13.16%/57.71% under the scoring protocol in Section C.4. Answer-output tokens instead increase by 10.59%/9.04% and include provider-reported reasoning tokens. These request totals exclude earlier memory construction, retrieval, and selection costs; they are not full-lifecycle costs or memory-token counts.

The evaluated LMI path makes no additional answer-model call, but retrieval, host construction, local selection, and compilation still incur computation. Token reductions alone therefore do not imply latency or monetary savings. The separate latency pilot below measures the post-retrieval path only; it does not establish a full memory-lifecycle cost reduction. The model identity, dataset scope, candidate snapshot, prompt binding, selection limits, output policy, and judge binding must all be preserved when reproducing an individual result cell.

C.6 Post-Retrieval Latency Measurement

Table 8 uses the first 100 questions in the existing ordered LongMemEval-S cohort, not a newly sampled held-out set. In the original three-method measurement, Top-50 and Top-200 were measured for Direct, MMR, and LMI in separate runs with the same frozen LMI source and model configuration. Each method was executed once per question and depth. All three methods share the corresponding frozen candidates and non-memory prompt; each depth retains its original host ordering rather than slicing the Top-200 list. MMR and LMI use at most 24 whole items and 1,280 selection tokens. Selection and compilation are recomputed for every request; frozen outputs serve only as equivalence checks. MMR also recomputes each query embedding, while candidate embeddings are precomputed. Input-dependent parsing caches are cleared; fixed resources remain loaded.

Answering uses Qwen2.5-7B-Instruct in FP16 with vLLM 0.16.0, with model revision `a09a35458c702b33eeacc393d103063234e8bc28`. Unlike the four-GPU aggregate runs in Table 6, this pilot uses three V100-32GB GPUs, each hosting one replica with prefix caching disabled. Methods rotate across replicas by question, giving each method a 34/33/33 allocation; each GPU has at most one request in flight. CPU preparation is serial, answer requests run concurrently, and a per-question barrier waits for all three responses. The frozen official plain-text answer prompt uses temperature 0 and a 4,096-token output limit. Six warm-up calls per run are excluded from timing summaries. All 300 measured answers per depth completed without errors, retries, unknown submissions, or output truncation. No judge was called, so this pilot supplies no new accuracy result.

SC was measured on September 25 using the same ordered 100 questions per depth, answer-model configuration, three replicas, disabled prefix cache, and timing boundaries. Each request recomputes the query embedding, selection, and compilation; candidate embeddings remain precomputed, and compiled contexts are checked against the frozen SC outputs. Each depth assigns 34/33/33 requests to the GPUs, with serial CPU preparation and at most one request in flight per GPU; batches contain different questions, not three methods for one question. All 200 measured SC answers completed without errors, retries, unknown submissions, or truncation; 12 warm-up calls are excluded and no judge is called. SC shares the measurement definition but is not part of the original simultaneous three-method run.

For each request, one client-side monotonic clock records t_0 when processing retrieved candidates starts, t_1 immediately before HTTP submission, and t_2 when the complete response returns. Preparation is $t_1 - t_0$, answering is $t_2 - t_1$, and post-retrieval total latency is $t_2 - t_0$. Preparation includes local analysis, selection, compilation, request construction, token counting, validation, persistence, and submission handoff. Answer time includes network and serving overhead as well as generation; it is not pure GPU compute time. Memory ingestion, shared retrieval, resource cold starts, offline candidate embedding, and grading are excluded. Means are computed over individual requests, not from batch wall time divided by the number of questions.

The paired mean total-latency difference, LMI minus Direct, is $+1.4225$ s at Top-50 and -0.7909 s at Top-200. Question-level 95% bootstrap intervals (10,000 resamples, seed 20260917) are $[1.0020, 1.8159]$ s and $[-1.4174, -0.1655]$ s, respectively. These intervals describe this fixed sample and single-run serving condition, not cross-run stability. MMR has lower total latency at both depths. We do not extrapolate the pilot to the 500-question scope of Figure 3(a), infer a quality ranking from latency, or treat unbilled self-hosted GPU execution as free.

Table 8: Post-retrieval latency on the same 100 LongMemEval questions with Qwen2.5-7B-Instruct. Values are mean seconds per query; total includes preparation and the complete answer request, but excludes retrieval and grading.

Candidates	Method	Preparation	Answer	Total
Top-50	Direct	0.011	7.608	7.618
	MMR	0.024	6.857	**6.881**
	SC	0.030	6.887	6.917
	LMI	1.839	7.201	9.041
Top-200	Direct	0.019	15.408	15.426
	MMR	0.045	6.421	**6.467**
	SC	0.072	6.563	6.634
	LMI	7.369	7.266	14.636

C.7 Performance by Question Type

Table 9 supplements the Top-200 breakdown in Table 2. Both join archived per-question judgments from the executions in Table 1 to the six dataset types by question ID, without new model calls. Each type uses its full question count, including ungradable outputs; overall accuracy uses all 500 questions rather than an unweighted mean of the rows. Abstention questions remain within their original types. Output recovery and scoring follow Section C.4.

Table 9: LongMemEval accuracy (%) by question type at Top-50. N is the number of questions.

Question type	N	Qwen2.5-7B				Llama-3.1-8B				GPT-5			
		Direct	MMR	SC	LMI	Direct	MMR	SC	LMI	Direct	MMR	SC	LMI
Single-session user	70	90.00	**94.29**	**94.29**	91.43	84.29	57.14	87.14	**94.29**	95.71	95.71	95.71	**97.14**
Single-session assistant	56	**75.00**	69.64	67.86	71.43	73.21	60.71	73.21	**76.79**	85.71	**87.50**	80.36	85.71
Single-session preference	30	63.33	50.00	50.00	**66.67**	**73.33**	60.00	56.67	**73.33**	**96.67**	86.67	93.33	93.33
Multi-session	133	53.38	61.65	71.43	**72.93**	60.15	50.38	59.40	**72.18**	**85.71**	75.94	81.95	84.21
Temporal reasoning	133	67.67	68.42	**71.43**	68.42	66.92	46.62	66.92	**73.68**	90.23	85.71	86.47	**90.98**
Knowledge update	78	73.08	76.92	78.21	**82.05**	**71.79**	51.28	70.51	**71.79**	93.59	91.03	91.03	**94.87**
Overall	500	68.40	70.60	74.00	**75.20**	69.40	52.20	68.40	**76.20**	**90.20**	85.60	87.00	**90.20**

For Qwen at Top-50, LMI has the highest reported accuracy on multi-session, preference, and knowledge-update questions, whereas SC leads on temporal reasoning. LMI remains below Direct on single-session assistant questions and below MMR and SC on single-session user questions, consistent with the non-uniform pattern at Top-200. These descriptive results do not isolate the effects of graph construction, selection, or feedback learning.

LoCoMo. Table 10 supplements the Top-200 comparison in Table 3, using the same evaluation scope and scoring rules in Sections C.1 and C.4. Each percentage uses the scored questions of its type as the denominator; ungradable outputs remain in that denominator, and overall accuracy is computed across questions rather than by averaging type-level percentages. Bold values indicate the maximum among the reported methods within each backbone, including ties. GPT-5 LMI leads on multi-hop, single-hop, and adversarial questions at both depths. At Top-50, it ties SC on temporal questions; SC leads on temporal questions at Top-200. On open-domain questions, LMI remains below Direct and MMR for all three answer models at both depths; it also remains below SC except

with Llama-3.1-8B, where it is higher. Aggregate gains therefore do not imply uniform improvements across types.

Table 10: LoCoMo accuracy (%) by question type at Top-50. Best values within each backbone are bold.

Question type	Qwen2.5-7B				Llama-3.1-8B				GPT-5			
	Direct	MMR	SC	LMI	Direct	MMR	SC	LMI	Direct	MMR	SC	LMI
Multi-hop	67.38	**77.30**	69.86	72.34	75.53	69.15	69.86	**79.43**	89.72	91.84	88.65	**93.97**
Temporal reasoning	77.57	77.57	77.57	**86.92**	**87.85**	81.93	81.31	83.80	91.28	91.28	**94.39**	**94.39**
Open-domain	38.54	38.54	**39.58**	21.88	39.58	**55.21**	12.50	30.21	**64.58**	**64.58**	61.46	41.67
Single-hop	79.19	78.00	75.62	**81.81**	85.02	82.88	77.65	**86.92**	89.77	90.37	90.84	**92.75**
Adversarial	57.57	61.01	58.49	**73.62**	50.92	**55.05**	44.04	54.59	91.97	91.97	92.89	**93.58**
Overall	70.50	72.17	69.59	**76.57**	74.39	73.28	66.55	**75.46**	89.27	89.83	90.13	**90.89**

C.8 Ablation Definitions and Controls

Table 4 defines four variants on LongMemEval at Top-200 with Qwen2.5-7B-Instruct. Full follows the same method definition as LMI in Section 4.2, including learned utility.

Variant definitions. **Full** combines the task–memory graph, learned utility, set-dependent selection, and structured rendering. **Independent selection** retains the graph and learned utility, but stops at the initial individually ranked selection. It disables subsequent additions and replacements based on what the selected set already covers; it is not a no-graph ablation. **Without structured rendering** retains the selection policy and selected memories, replacing the structured context with a plain list of the same original texts. This tests the presentation module as a whole, not an individual label or formatting choice. **Without learned utility (static)** removes the learned ranking signal while retaining the graph, set-dependent selection, and structured rendering. It uses only the static ranking key $\mathbf{s}_0$ defined in Section B.4, rather than Full's key that includes learned utility.

Comparison controls and metrics. All four variants were evaluated on the same questions and frozen candidate snapshots, using the same graph construction, answer model, non-memory prompt, selection budgets, output limits, and grading configuration. The three learning-enabled variants used the same fitted parameter values, held fixed during evaluation. The rendering ablation preserved Full's selected items, original text, dates, and item order. Accuracy differences are in percentage points relative to Full. Tokens/query counts the injected memory span, including formatting, not the whole request; its reduction without structured rendering is 13.24%.

C.9 Utility Training and Frozen Evaluation

This section gives the training configuration for the LMI results in Tables 1 and 4. The paired-removal objective and update boundaries follow Section 3.5 and Appendix B.5.

Training data and separation. Utility training used a pool of 250 questions with reference answers, separate from the reported 500-question evaluation set. Before feedback collection, screening excludes source-session overlap, duplicate questions, and repeated memory text across the training and evaluation sets. The pool is divided into five source-grouped folds, with approximately 200 questions for fitting and 50 for validation in each fold; intact source groups may produce uneven fold sizes. Seed 17 controls the split and intervention sampling through separate deterministic hash namespaces, with seeds 29 and 43 used for sensitivity checks.

Feedback collection and weighting. Training feedback was collected using Qwen2.5-7B with Mem0 Top-200 candidates and GPT-5 grading. Paired runs fix the question, candidates, answer model, prompt template, and decoding settings, changing only the target memory's presence. Reference answers supervise binary correctness scores but are excluded from selection features and answer-model inputs. A frozen static selector provides the submitted set. For each question, the protocol samples at most two selected, non-protected memories and repeats each paired removal twice without resampling the target. The mean of the two correctness differences (Equation (7)) defines y_h, a contribution conditional on the submitted set rather than a memory's intrinsic value.

Identical differences, including two zeros, receive confidence $c_h = 1$; one zero and one nonzero difference receives $c_h = 0.5$. Opposing signs or an invalid comparison yield no observation, rather than zero credit. Identical grading inputs reuse one verdict as specified in Appendix B.5. For 250 questions, these limits allow at most 500 intervention targets and 1,000 paired comparisons, not a predetermined number of valid training examples.

Features and fitting. The weighted ridge model in Equation (8) is fitted to valid observations (ϕ_h, y_h, w_h). The feature vector has 53 dimensions: 29 item-level and 24 requirement-linked features, combining retrieval and static-ranking signals with task requirements, relation-label proportions, and coverage state. Features are recorded before selection and exclude model outputs and task outcomes. Each fold computes RMS scaling from its fitting data only, without centering or an intercept; scale values below 10^{-12} are replaced by 1. To average losses within each question, Equation (8) uses $w_h = c_h/n_q$, where n_q is the number of valid targets for question q. Questions without a valid target are excluded from fitting. The protocol selects $a \in \{0.01, 0.1, 1, 10\}$ with $\lambda = aN$, where N counts questions with valid feedback in the fitting partition. The validation criterion is mean per-question squared feedback prediction error, with predictions clipped to $[-1, 1]$ and ties resolved in favor of the smaller a.

Frozen model and reuse. After selecting a, we recomputed RMS scaling and refitted one utility model on all eligible training/development questions. All 12 LMI settings in Table 1 used this same frozen model across the two benchmarks, three answer models, and two candidate depths, without benchmark-, model-, or depth-specific refitting. The three learning-enabled ablation variants also used this model. Parameters and feature scaling remained fixed during evaluation, with $\beta = 0$ and no feedback updates. Table 11 summarizes this reuse and the ablation controls. Direct injection, MMR, and SC do not use the LMI utility model.

Table 11: Frozen utility model reuse across the reported results.

Reported results	Evaluation setting	Utility model use
Table 1: LMI	LongMemEval; all three answer models; Top-50/200	Shared frozen model; no refitting or updates.
Table 1: LMI	LoCoMo; all three answer models; Top-50/200	Same frozen model as LongMemEval; no refitting or updates.
Table 4: Full	LongMemEval; Qwen2.5-7B; Top-200	Same frozen model as the main comparisons.
Independent selection	Same ablation setting	Same frozen model as Full; no set-dependent additions or replacements.
w/o structured rendering	Same ablation setting	Same frozen model and selected items as Full; plain-list rendering.
w/o learned utility	Same ablation setting	Static ranking; no learned utility score.

The static ablation disables the learned score while preserving the other components (Section C.8). Learned scores order initial budget filling and break coverage-gain ties without changing the graph or selection constraints (Appendix B.5). Offline paired-removal calls, tokens, and fitting cost are accounted for separately and excluded from the reported inference memory-token and latency measurements.