ISYS: Interleaved Synthesizing of Scaffolding Tasks to Leverage SLMs in Memory Agents

Anonymous ACL submission

Abstract

Personal memory agents manage highly sensitive user data and therefore increasingly require on-device deployment, where tight cost and latency budgets favor small language models (SLMs). Memory agents often specify multiple requirements within a single monolithic prompt. As these prompts are typically written for high-capacity LLMs, they often transfer poorly to SLMs. We identify two key bottlenecks in this mismatch: excessive prompt complexity and logical incompleteness. First, each task prompt overloads a single inference with many requirements, exceeding the limited capacity of SLMs. Second, these prompts frequently presuppose intermediate operations that are never stated explicitly. To mitigate these problems, we introduce **ISYS**, a feedback-driven prompt synthesizer that decomposes a monolithic task prompt into sequential main tasks and interleaves externalized scaffolding tasks. Specifically, **ISYS** proposes candidate scaffolding tasks, optimizes their placement, and iteratively refines their instructions using trajectory-level feedback. At inference time, the main tasks are executed sequentially with interleaved optimized scaffolding tasks, enabling the SLM to rely on explicit intermediate outputs. Across memory agent frameworks and benchmarks such as LoCoMo and MemoryAgentBench, **ISYS** outperforms SLM baselines, including prompt optimization and reasoning baselines, while achieving performance comparable to LLMs (e.g., GPT-4o and GPT-5.2).

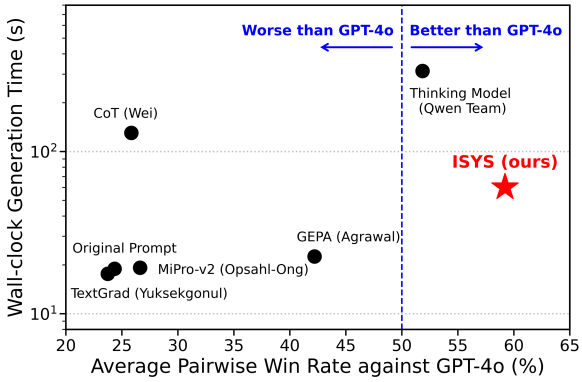

Figure 1: **Quality-efficiency trade-off of prompting strategies for SLMs on Qwen3-4B models.** The x-axis shows the average pairwise win rate against GPT-4o in memory quality evaluation (Table 1), and the y-axis shows the average wall-clock generation time measured on Snapdragon 8 Elite Gen 5. We compare prompt optimization methods (TextGrad, MiPro-v2, GEPA) and reasoning variants (CoT, Thinking Model). **ISYS** achieves the highest memory quality while remaining substantially more efficient than reasoning approaches. Prompt optimization and CoT baselines use Qwen3-4B-Instruct, while the Thinking Model uses Qwen3-4B-Thinking.

1 Introduction

The evolution of Large Language Models (LLMs) (Gemini Team, Google, 2025; OpenAI, 2024; Li et al., 2025a; Singh et al., 2025b; Yao et al., 2022) has shifted their role from passive information retrieval systems to proactive agents. To support such applications, LLM agents (Xu et al., 2025; Chhikara et al., 2025) typically rely on prompt-based task specifications to leverage off-the-shelf models. As agentic services become increasingly complex, these task specifications often grow longer and more intricate.

This challenge is particularly pronounced in memory agents, which emulate the complexities of human memory and thus require exceptionally long and sophisticated task specifications for context maintenance and data processing. Existing memory agent methods (Xu et al., 2025; Chhikara et al., 2025; Gutiérrez et al., 2025; Kang et al., 2025; Li et al., 2025b; Tan et al., 2025; Wu et al., 2025; Xu et al., 2026) design their prompts in a monolithic manner, bundling multiple intricate tasks (e.g., memory management operations) into a single long prompt, under the assumption of high-capacity, server-side LLMs.

However, this paradigm often overlooks the origin of the source data: much of the data used to build personal memory is captured on personal edge devices such as AR glasses and smartphones. To safeguard data privacy and stay within compute budgets, it is therefore important to prioritize on-device processing. In such settings, memory construction and management must be handled by SLM-based agents running on edge devices. The challenge arises because the complex monolithic prompts often exceed the limited reasoning capacity of SLMs, leading to significant performance degradation. As shown in Fig. 1, Qwen3-4B-Instruct with the original monolithic prompt achieves an average pairwise win rate below 25% against GPT-4o in memory quality evaluation.

We attribute this failure to two fundamental issues in monolithic task prompts: *complexity* and *incompleteness*. First, a single monolithic prompt often bundles too many requirements into a single inference, straining the limited capacity of SLMs. Second, since these requirements are typically authored for high-capacity LLMs, they often presuppose intermediate operations that are never explicitly specified. While LLMs can often infer these missing logical connections internally, SLMs struggle when such scaffolding tasks remain implicit.

A natural response is prompt optimization, which automatically refines prompts for a target model. However, existing prompt optimization approaches (TextGrad (Yuksekgonul et al., 2025), MiPro-v2 (Opsahl-Ong et al., 2024), GEPA (Agrawal et al., 2026)) mainly optimize the surface text of a single prompt. In memory agents, this usually means rephrasing the original monolithic prompt by adding more constraints, clarifications, or examples. While intended to help, this strategy can further increase prompt complexity, thereby exacerbating the first issue. At the same time, it leaves the underlying incompleteness of the task's logical structure unaddressed. As shown in Fig. 1, prompt optimization methods improve performance but still remain below GPT-4o.

Another common remedy is to elicit explicit reasoning through thinking models (Guo et al., 2025) or chain-of-thought cues (CoT) (Wei et al., 2022) such as "let's think step by step." However, in the SLM regime, this unconstrained reasoning is often unreliable, as capacity-limited models may allocate effort to irrelevant details or perform incorrect intermediate operations. Furthermore, longer deliberation significantly increases token usage and latency, directly conflicting with the strict constraints of on-device deployment. Fig. 1 shows that the thinking model can improve output quality but incur substantially longer generation time.

Given the limitations of SLMs, we propose that the monolithic task prompt should be executed as sequential main tasks, i.e., the major required tasks of the original prompt, supported by explicitly constructed *scaffolding tasks* that ensure task completion and logical continuity. Our controlled motivation experiment (Appendix A.1) supports this view: simply strengthening a monolithic prompt or decomposing only the main tasks is insufficient, whereas explicitly externalizing missing intermediate operations yields the largest reliability gains.

Motivated by these findings, we introduce **ISYS**: **I**nterleaved **SY**nthesizing of **S**caffolding Tasks, a feedback-driven prompt synthesizer that uncovers missing but critical logical steps in a monolithic task prompt and externalizes them as scaffolding tasks. Given a task specification as a single monolithic prompt, **ISYS** first decomposes its core directives into sequential main tasks, then synthesizes an initial scaffolding task for each main task. Next, **ISYS** interleaves scaffolding tasks with the main tasks by selecting their placements and iteratively refining their instructions using trajectory-level feedback from a judge model. The resulting scaffolding tasks produce intermediate artifacts that subsequent main tasks can directly leverage, enabling SLM agents to better understand the target specification while preserving its original intent.

Across multiple memory-agent frameworks and benchmarks like LoCoMo (Maharana et al., 2024) and MemoryAgentBench (Hu et al., 2026), **ISYS** achieves notably improved performance in the SLM-based memory agents, which is comparable (often superior) to the LLM agents.

2 Related Work

SLM and on-device deployment. Recent work has increasingly emphasized SLMs as practical alternatives to LLMs in deployment settings with tight latency and cost constraints (Belcak et al., 2025; Sharma and Mehta, 2025; Lee et al., 2025). This shift is especially relevant for privacy-sensitive applications on personal edge devices, where raw user data is ideally processed on-device. In such settings, limited compute budgets make SLMs a natural choice. However, many task specifications in modern agent systems are still de-

signed for high-capacity, server-side LLMs, creating a growing mismatch between prompts designed for LLMs and those executed by SLMs. This mismatch is particularly important in personal memory agents, where task specifications are often long and structurally complex.

Personal memory agents. A growing body of work studies memory-augmented agents that maintain persistent user information across interactions, including A-mem (Xu et al., 2025), Mem0 (Chhikara et al., 2025), HippoRAG (Gutiérrez et al., 2024), HippoRAG2 (Gutiérrez et al., 2025), MemOS (Li et al., 2025b), and MemoryOS (Kang et al., 2025). These systems support functions such as information extraction, memory creation, memory updating, consolidation, and retrieval, often through task-specific prompts with structured outputs. However, existing memory-agent pipelines are largely designed and evaluated with high-capacity LLMs in mind. In practice, each memory function is often specified by a single prompt that bundles multiple requirements while relying on intermediate logical operations that are never explicitly stated. While strong LLMs can often infer such missing steps, this design becomes brittle when transferred to SLMs.

Prompt optimization and automatic prompt engineering. Prompt optimization aims to automatically adapt prompts to a target model and task, including gradient-based methods such as TextGrad (Yuksekgonul et al., 2025) and search-based methods such as MiPro-v2 (Opsahl-Ong et al., 2024) and GEPA (Agrawal et al., 2026). Most existing approaches improve the surface form of a prompt by rewriting instructions, adding clarifications, inserting demonstrations, or refining constraints. This paradigm is effective when the task is already well specified and the model can reliably perform all required intermediate operations within a single inference. However, these approaches face two limitations in the SLM setting. First, improving a monolithic prompt often makes it longer and more complex, further burdening capacity-limited SLMs. Second, surface-level rewriting does not resolve cases where the prompt is logically incomplete and silently assumes intermediate operations.

Reasoning elicitation. A broad line of work improves model performance by eliciting intermediate reasoning, including chain-of-thought prompting, tree-based reasoning, and reasoning-oriented model variants that generate extended deliberation traces (Wei et al., 2022; Yao et al., 2023; Guo et al., 2025). These approaches show that exposing intermediate reasoning can improve difficult problem solving. However, such reasoning is typically unconstrained: the model decides what intermediate steps to perform, in what order, and at what level of detail. This can be problematic in the SLM regime, where models may spend effort on irrelevant details, omit prerequisite checks, or produce unreliable intermediate conclusions.

3 Proposed Method

3.1 Limitations of SLM Memory Agents

In recent memory agents such as A-mem (Xu et al., 2025) and Mem0 (Chhikara et al., 2025), every memory agent function like information extraction or memory management is specified by multiple core directives, which are combined into a single monolithic prompt designed for high capacity LLMs. For example, an information extraction prompt may jointly require keyword, tag, and summary extraction, while a memory management prompt may jointly require selecting one of add, delete, update, or none.

Directly using such monolithic prompts with SLMs yields significantly degraded results. This degradation stems from two fundamental bottlenecks: prompt *complexity*, which overwhelms the limited capacity of SLMs by bundling multiple core directives into a single inference, and logical *incompleteness*, where the prompt relies on implicit intermediate operations that SLMs struggle to infer internally. **ISYS** mitigates these bottlenecks, enabling SLMs to more reliably process the core directives while preserving the original goal of the agent function.

3.2 Problem Definition and Goal

We represent the core specification of a target memory agent function as C. A memory agent processes an input x to manage a persistent memory store M according to the specification C, where x may be a user utterance, a memory note, a document snippet, or a query. For example, the agent may extract structured information from the input or decide how to use it to update memory.

To reduce the complexity of executing the monolithic core specification on SLMs, we first decompose C into an ordered list of main tasks, $C = (c_1, \ldots, c_T)$, where each c_t is the prompt for

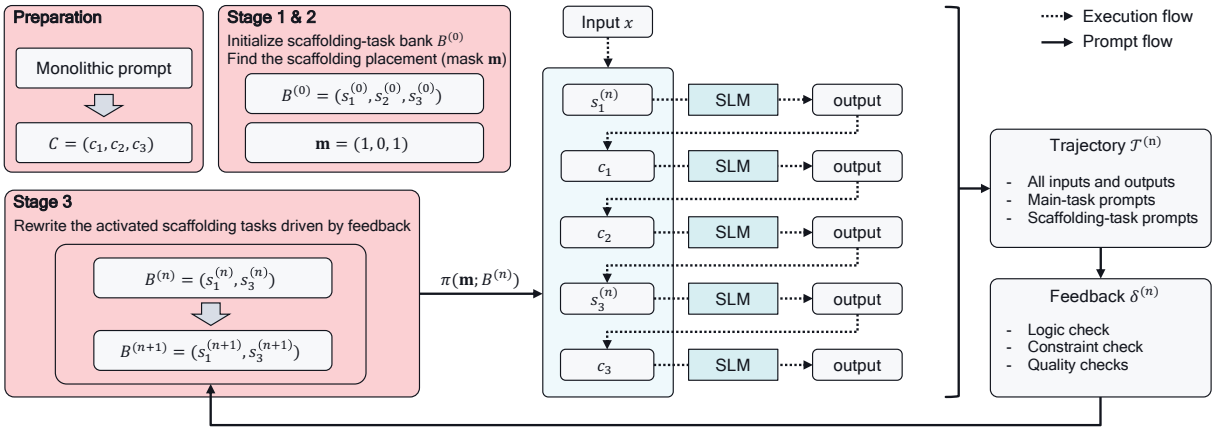

Figure 2: **Overview of ISYS.** A monolithic prompt is decomposed into sequential main tasks $C = (c_1, c_2, c_3)$. A scaffolding-task bank B is initialized and a placement mask $\mathbf{m}$ selects where scaffolding tasks should be inserted (e.g., before c_1 and c_3). The compiled procedure is executed by the target SLM to produce a trajectory, which is evaluated by a judge model. The resulting feedback is used to iteratively rewrite the activated scaffolding tasks.

the t-th core directive. In our experiments, this decomposition is provided by either a human expert or an LLM. We further introduce a bank of candidate scaffolding tasks $B = (s_1, \ldots, s_T)$, where each s_t is associated with main task c_t. Without changing the main tasks, we interleave them with scaffolding tasks and iteratively refine their instructions using the LLM g. At deployment time, the SLM agent sequentially executes the main tasks together with the scaffolding tasks.

3.3 Sequential Execution with Scaffolding Tasks

Given the main-task decomposition $C = (c_1, \ldots, c_T)$, we allow scaffolding tasks to be inserted before main tasks to support each step.

Scaffolding tasks. A scaffolding task s_t is a short auxiliary instruction placed before main task c_t to explicitly produce an intermediate output that supports that step. Typical outputs include candidate sets, grounded evidence snippets, constraint checks, or relations. Scaffolding tasks are intended to be semantics-preserving: they do not alter the original task specification, but instead externalize intermediate information that is already implicit in it, so that later main tasks can directly leverage it.

Binary activation mask. Not every main task requires additional scaffolding; some steps can already be executed reliably by the target SLM alone. To allow scaffolding only where it is beneficial, we use a binary mask to activate only a subset of the candidate scaffolding tasks in B. A binary mask $\mathbf{m} = (m_1, \ldots, m_T) \in \{0, 1\}^T$ specifies which candidate scaffolding tasks are activated. If $m_t = 1$, s_t is inserted immediately before c_t; otherwise it is omitted. This yields the compiled sequential procedure

$$\pi(\mathbf{m}; B) = ([s_1]^{m_1}, c_1, [s_2]^{m_2}, c_2, \ldots, [s_T]^{m_T}, c_T), \tag{1}$$

where $[s_t]^{m_t}$ denotes executing s_t if $m_t = 1$ and omitting it otherwise.

3.4 Feedback-Driven Optimization

Given the main-task sequence, **ISYS** optimizes the inserted scaffolding tasks in three stages, as illustrated in Fig. 2. Stage 1 initializes a candidate scaffolding-task bank, Stage 2 selects scaffold placement via a binary mask, and Stage 3 iteratively rewrites only the activated scaffolding tasks based on execution trajectories and judge feedback. Both scaffold placement search and instruction refinement are performed on the training set.

Components. **ISYS** runs *offline* using a high-capacity LLM g in two roles: (i) a *synthesizer* that drafts and rewrites scaffolding-task instructions, and (ii) a *judge* that evaluates how well the target SLM's execution satisfies the original core specification, returning a scalar score and narrative feedback. At deployment time, only the target SLM f and the compiled procedure are used.

Stage 1: Initialize scaffolding-task bank. For each main task c_t, LLM g synthesizes a candidate scaffolding task s_t given c_t and the original monolithic prompt P_{mono}. This produces the initial scaffolding-task bank $B^{(0)} = (s_1^{(0)}, \ldots, s_T^{(0)})$.

Stage 2: Select the best scaffold placement. Given the initialized bank $B^{(0)}$, we exhaustively evaluate all masks $\mathbf{m} \in \{0,1\}^T$ on the training set $\mathcal{D}_{\text{train}}$, including the all-zero mask. For each input $x \in \mathcal{D}_{\text{train}}$, the target SLM executes $\pi(\mathbf{m}; B^{(0)})$, producing an execution trajectory $\tau_{\mathbf{m}}(x)$ that contains the scaffold outputs and main-task outputs. Let $\mathcal{T}_{\mathbf{m}}^{(0)} = \{\tau_{\mathbf{m}}(x) \mid x \in \mathcal{D}_{\text{train}}\}$ denote the resulting set of training trajectories. The judge model g then returns a scalar score and feedback with a judge prompt $\mathcal{P}_{\text{judge}}$:

$$(r_{\mathbf{m}}^{(0)}, \delta_{\mathbf{m}}^{(0)}) \sim g(\mathcal{T}_{\mathbf{m}}^{(0)}; \mathcal{P}_{\text{judge}}), \tag{2}$$

where $r_{\mathbf{m}}^{(0)} \in [0,1]$ is the training score and $\delta_{\mathbf{m}}^{(0)}$ is the corresponding feedback. In our experiments, we instantiate Stage 2 with top-1 mask selection:

$$\mathbf{m}^\star = \arg\max_{\mathbf{m} \in \{0,1\}^T} r_{\mathbf{m}}^{(0)}. \tag{3}$$

This top-1 selection is a special case of a top-K variant; Appendix A.2 discusses how the top-K masks can be retained and refined independently.

Stage 3: Rewrite the selected scaffolding tasks driven by feedback. Keeping $\mathbf{m}^\star$ fixed, we iteratively refine only the activated scaffolding tasks starting from $B^{(0)}$. Let $A^\star = \{t \mid m_t^\star = 1\}$ be the set of activated indices, and let $B^{(n)}$ denote the scaffolding-task bank after the n-th rewrite. For example, if $\mathbf{m}^\star = (1,0,1)$, only s_1 and s_3 are retained and refined, while the inactive candidate s_2 is discarded (Fig. 2).

To reduce order bias, we randomly choose the refinement order over $A^\star$. At iteration n, let $i \in A^\star$ be the selected scaffolding-task index. Using the current trajectories $\mathcal{T}^{(n-1)}$ and judge feedback $\delta^{(n-1)}$, LLM g rewrites the corresponding scaffolding task based on a refinement prompt $\mathcal{P}_{\text{refine}}$:

$$s_i^{(n)} \sim g(i, \mathcal{T}^{(n-1)}, P_{\text{mono}}, \delta^{(n-1)}; \mathcal{P}_{\text{refine}}), \tag{4}$$

where $\mathcal{T}^{(n-1)}$ is obtained by executing the compiled procedure $\pi(\mathbf{m}^\star; B^{(n-1)})$ on $\mathcal{D}_{\text{train}}$. After updating s_i, we re-execute the compiled procedure and re-evaluate it with a judge prompt $\mathcal{P}_{\text{judge}}$:

$$(r^{(n)}, \delta^{(n)}) \sim g(\mathcal{T}^{(n)}; \mathcal{P}_{\text{judge}}), \tag{5}$$

where $\mathcal{T}^{(n)}$ is the set of trajectories produced by $\pi(\mathbf{m}^\star; B^{(n)})$.

If the score improves, we accept the rewritten scaffolding task and update the bank; otherwise, we discard the proposal and keep the previous instruction. If repeated rewrite attempts for the current index fail to improve the score, we move to the next activated scaffolding task. After refining all activated scaffolding tasks, we reshuffle the refinement order and repeat this process for several refinement rounds. The final compiled procedure is $\pi(\mathbf{m}^\star; B^\star)$. The algorithm pseudo-code is given in Appendix A.2, and all prompts used in our experiments are provided in Appendix A.3.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate on LoCoMo (Maharana et al., 2024) and MemoryAgentBench (Hu et al., 2026), two benchmarks for long-term memory agents. Each benchmark provides a stream of inputs and requires the agent to incrementally construct and maintain a persistent memory store. Details are provided in Appendix A.4.

Memory agents. We instantiate two representative memory agent frameworks, A-mem (Xu et al., 2025) and Mem0 (Chhikara et al., 2025). At each time step, the agent receives an input segment (e.g., a conversation utterance or a document chunk), derives a candidate memory representation, retrieves related memories from the current store, and predicts a structured memory operation according to the framework design.

Implementation details. Each agent defines two fixed core specifications: one for *information extraction* and one for *memory management* (Appendix A.5). We apply **ISYS** separately to each specification. We decompose each monolithic specification into sequential main tasks $C = (c_1, \ldots, c_T)$ while preserving the original instruction content. **ISYS** then synthesizes the scaffolding tasks, searches over their placement masks, and refines the activated scaffolding tasks.

Optimization protocol. To control optimization cost and test generalization, we optimize **ISYS** and prompt-optimization baselines using only the first 50 samples from LoCoMo. Additional implementation details are provided in Appendix A.6.

Baselines. Our primary target SLM is Qwen3-4B-Instruct (Team, 2025). We compare **ISYS** against the following prompting baselines: (i) the original monolithic prompt, (ii) prompt optimization methods including TextGrad (Yuksekgonul

Method	A-mem		Mem0		Efficiency	
	LoCoMo	Ruler	LoCoMo	Ruler	Avg Tokens	Avg Time (s)
Original Prompt	14.58	35.41	7.40	40.00	444.54	18.93
TextGrad	14.33	25.53	10.10	45.00	392.84	17.61
MiPro-v2	17.50	30.30	18.70	40.00	428.12	19.22
GEPA	31.58	40.20	29.54	67.50	494.39	22.54
CoT	15.23	33.20	11.63	43.33	3459.23	130.39
Thinking model	34.40	47.50	50.00	75.50	8422.26	313.85
ISYS (ours)	**40.63**	**49.47**	**59.26**	**87.50**	1539.01	60.29

Table 1: **Memory quality and efficiency comparison against GPT-4o.** We report average pairwise win rates (%) of each method on Qwen3-4B models against GPT-4o across two memory-agent frameworks (A-mem and Mem0) and two benchmarks (LoCoMo and Ruler from MemoryAgentBench). Higher is better. We also report the average number of generated tokens and the average wall-clock generation time per input for memory management. Wall-clock time is measured on Snapdragon 8 Elite Gen 5 and includes both prefill and generation stages.

et al., 2025), MiPro-v2 (Opsahl-Ong et al., 2024), and GEPA (Agrawal et al., 2026), and (iii) reasoning baselines including CoT (Wei et al., 2022) and Thinking model (Qwen3-4B-Thinking). As large-model references, we additionally report GPT-4o (Hurst et al., 2024) and GPT-5.2 (Singh et al., 2025a) under the original agent prompts.

4.2 Evaluation Protocol

We evaluate methods from two complementary perspectives: (i) **memory quality**, which measures whether the constructed memory correctly follows the original agent specification, and (ii) **downstream retrieval-augmented generation (RAG)**, which measures whether the constructed memory improves retrieval and generation.

Memory quality via pairwise judging against GPT-4o. We evaluate the quality of the constructed memory itself in the memory management stage. For each method, we compare its constructed memory against the memory produced by GPT-4o under the same agent framework and prompt specification. Given the input, the original agent prompt, and two candidate memory outputs, a judge model selects which output better satisfies (i) adherence to the original prompt and (ii) memory quality.

To mitigate position bias, we swap candidate order and repeat each comparison 10 times, then aggregate the results by majority vote. We report the resulting pairwise win rate, i.e., the percentage of evaluation instances whose majority vote preferred the method over GPT-4o, excluding ties. We use GPT-5.2 and Gemini 3 Pro as independent judges and observe consistent trends. The full evaluation prompt is provided in Appendix A.7.

Efficiency measurement. Along with memory quality, we report two efficiency metrics for the memory building pipeline: (i) the average number of generated tokens and (ii) the average wall-clock generation time per input. Wall-clock generation time is measured on Snapdragon 8 Elite Gen 5. The reported time includes both the prefill stage (processing the input prompt) and the generation stage (token decoding).

Downstream RAG evaluation. We evaluate whether improved memory construction leads to better downstream retrieval and generation. For each method, we first construct a memory store using the corresponding agent pipeline and prompting procedure, and then answer benchmark questions using retrieved memories. Retrieval uses a fixed k per framework across all methods (top-40 for A-mem and top-100 for Mem0). We report F1 on LoCoMo and EM on MemoryAgentBench following the official evaluation protocols. To isolate the effect of memory construction quality, all methods use the same response generator, GPT-4o.

4.3 Memory Quality

Pairwise judging and efficiency. Table 1 reports specification-conditioned pairwise win rates against GPT-4o, together with the average number of generated tokens and the average wall-clock generation time of the memory building pipeline.

Across both memory agent frameworks and both benchmarks, `ISYS` achieves the strongest memory quality among all SLM-based methods. In several settings, `ISYS` approaches GPT-4o-level performance, and in some cases it is even preferred over GPT-4o under pairwise judging. This result is plausible because GPT-4o is used here only as a

Scaffolding tasks on A-mem
Pre-before Information Extraction.
`evidence`, `keyword_candidates`, `tag_candidates`, `context_skeleton`
Optional `uncertainties` for missing or unclear cases.
Pre-before Neighborhood Linking.
`candidate_neighbor_ids`, `high_priority_candidates`, `overlap_evidence`, `current_memory_keywords`
Pre-before Neighborhood Update.
`neighbor_ids`, `current_contexts`, `current_tags`, `neighbor_count`
Preserve input order and original context/tag structures.

Table 2: **Representative scaffolding tasks discovered by ISYS for A-mem.**

reference system under the same original prompt, not as an oracle; when the original prompt leaves important intermediate operations implicit, even GPT-4o can produce suboptimal memory outputs. We provide qualitative comparisons of memory management outputs under the same input in Appendix A.8.

ISYS executes the compiled procedure in a turn-by-turn sequential manner. Because each step continues from the previous context, KV caches can be reused across steps. As a result, additional scaffolding steps do not incur repeated prefill costs, and the overall latency remains reasonable even for multi-step execution. **ISYS** achieves the gains with lower cost than unconstrained thinking-based prompting. Although scaffolding introduces additional steps, the resulting overhead remains moderate relative to Qwen3-4B-Thinking while yielding substantially stronger and more reliable memory construction.

Discovered scaffolding tasks. **ISYS** improves SLM execution by externalizing missing intermediate operations as explicit scaffolding tasks. Table 2 shows scaffolding tasks optimized for A-mem when targeting Qwen3-4B-Instruct.

Rather than inserting generic "think step by step" style instructions, **ISYS** discovers task-specific intermediate artifacts such as evidence spans, grounded candidate keywords, candidate neighbor links with overlap evidence, and order-preserving neighbor context summaries that later main tasks can directly use.

These results support our hypothesis. The discovered scaffolding tasks reduce *complexity* by distributing the original monolithic requirements across multiple steps, and mitigate logical *incompleteness* by explicitly materializing intermediate operations that were previously left implicit.

Scaffolding task on Mem0
Pre-before Delete.
`ids_in_memory`, `delete_ids`, `safe_ids`, `evidence_by_id`, `ambiguity_notes`
No invention; if contradiction is uncertain or implicit, keep the item in `safe_ids`.

Table 3: **Deletion-oriented scaffolding task discovered by ISYS for Mem0.**

4.4 Downstream RAG Performance

Mem0 results. A representative failure mode of SLMs in Mem0 appears in memory deletion. Deletion requires determining whether newly observed facts explicitly contradict existing memory entries, a decision that is difficult for SLMs when it must be carried out implicitly inside a long monolithic prompt. **ISYS** frequently addresses this issue by inserting a dedicated scaffolding task before the delete step. As shown in Table 3, this intermediate task explicitly enumerates existing memory IDs, separates them into delete and safe candidates, and extracts contradiction evidence for each case. This makes the final delete decision substantially more reliable.

This behavior is closely related to the downstream gains on fact-consolidation subsets such as FC-SH and FC-MH. These subsets test whether the agent can correctly merge or remove previously stored facts when new evidence arrives. Such cases require multiple intermediate operations that are often left implicit in the original prompt, including identifying potentially conflicting memory entries, grounding the contradiction in the current evidence, and selecting the appropriate memory operation. By externalizing these operations into structured scaffolding tasks, **ISYS** enables the target SLM to execute memory consolidation more reliably.

Table 4 reports downstream QA performance for Mem0. We observe the same overall trend as in the memory quality evaluation: **ISYS** consistently improves RAG performance over both implicit prompting and prompt optimization baselines. The gains are especially pronounced on FC-SH and FC-MH, indicating that better handling of contradiction resolution and memory updates directly translates into stronger downstream RAG.

A-mem results. Table 5 reports downstream QA performance for A-mem. Across both benchmarks, **ISYS** consistently outperforms all Qwen3-4B prompting baselines.

Model	Method	LoCoMo (F1)						MemoryAgentBench (EM)					
		MH	Temp	OD	SH	Adv	Avg	SH	MH	EventQA	FC-SH	FC-MH	Avg
GPT-4o	Original Prompt	22.54	30.55	14.94	28.43	33.20	25.93	77.00	67.00	63.40	38.00	8.00	50.68
GPT-5.2	Original Prompt	25.20	38.63	19.73	34.01	33.50	30.21	70.00	64.00	66.00	15.00	3.00	43.60
Qwen3-4B	Original Prompt	21.92	30.39	13.79	29.56	34.12	25.96	64.00	63.00	59.00	15.00	4.00	41.00
	MiPro-v2	19.39	32.46	14.82	27.70	30.40	24.95	70.00	60.00	56.00	16.00	6.00	41.60
	GEPA	22.87	33.95	16.62	29.98	35.33	27.75	70.00	64.00	62.00	21.00	5.00	44.40
	CoT	20.37	33.61	14.83	28.04	34.53	26.28	68.00	62.00	60.00	11.00	2.00	40.60
	Thinking Model	21.11	**33.96**	17.08	29.63	35.36	27.43	75.00	**67.00**	66.00	20.00	8.00	47.20
	ISYS (ours)	**25.46**	33.12	**17.21**	**32.80**	**37.26**	**29.17**	**80.00**	63.00	**67.00**	**27.00**	**9.00**	**49.20**

Table 4: **Mem0 downstream QA performance on LoCoMo and MemoryAgentBench.** GPT-4o and GPT-5.2 use the original monolithic prompt and serve as reference LLM agents. For the Qwen3-4B block, all methods use Qwen3-4B-Instruct except Thinking Model, which uses Qwen3-4B-Thinking. Avg denotes the mean over the corresponding subsets. **Bold values** indicate the best performance among SLM agents.

Model	Method	LoCoMo (F1)						MemoryAgentBench (EM)					
		MH	Temp	OD	SH	Adv	Avg	SH	MH	EventQA	FC-SH	FC-MH	Avg
GPT-4o	Original Prompt	28.19	46.31	18.87	41.13	39.99	34.90	75.00	57.00	67.40	19.00	3.00	44.28
GPT-5.2	Original Prompt	28.17	47.73	17.20	42.69	39.52	35.06	75.00	64.00	69.20	22.00	1.00	46.24
Qwen3-4B	Original Prompt	25.12	45.06	16.77	38.07	39.16	32.84	70.00	57.00	66.00	15.00	4.00	42.40
	MiPro-v2	27.58	44.25	16.88	**40.58**	42.33	34.32	70.00	59.00	**67.00**	16.00	3.00	43.00
	GEPA	27.57	41.75	17.47	39.74	39.50	33.21	65.93	54.00	64.00	**20.00**	3.00	41.39
	CoT	25.56	44.41	19.36	36.53	37.54	32.68	69.00	55.00	60.00	16.00	**5.00**	41.00
	Thinking Model	26.70	42.23	17.80	39.79	40.42	33.39	**72.00**	**60.00**	**67.00**	16.00	4.00	43.80
	ISYS (ours)	**28.32**	**45.55**	**19.91**	40.18	**49.04**	**36.66**	**72.00**	**60.00**	**67.00**	**20.00**	3.00	**44.40**

Table 5: **A-mem downstream QA performance on LoCoMo and MemoryAgentBench.** GPT-4o and GPT-5.2 use the original monolithic prompt and serve as reference LLM agents. For the Qwen3-4B block, all methods use Qwen3-4B-Instruct except Thinking Model, which uses Qwen3-4B-Thinking. Avg denotes the mean over the corresponding subsets. **Bold values** indicate the best performance among SLM agents.

The gains are particularly pronounced on the Adv subset of LoCoMo. Since Adv evaluates whether the system can correctly distinguish answerable from unanswerable adversarial questions based on stored memory, this improvement suggests that **ISYS** produces more accurate and more trustworthy memory representations. Improved adherence to the original memory specification leads to better downstream retrieval and generation.

Additional results are provided in the appendix, including evaluation with an ablation study on core task decomposition (Appendix A.5.3), an alternative judge model (Gemini 3 Pro; Appendix A.9), and generalization to other SLMs (Appendix A.10).

5 Conclusion

We introduced **ISYS**, a framework that improves SLM-based memory agents by externalizing missing intermediate operations as explicit scaffolding tasks. Instead of relying on monolithic prompts originally designed for large language models, **ISYS** decomposes the task into sequential main steps and inserts structured intermediate tasks that are optimized through feedback. Across two memory-agent frameworks and two benchmarks, **ISYS** consistently improves memory quality while maintaining reasonable efficiency. Our experiments show that explicitly structuring intermediate operations allows small language models to execute complex agent logic more reliably. These findings suggest that adapting agent pipelines to SLMs requires more than prompt refinement: it often necessitates restructuring the reasoning process itself. By making implicit operations explicit and optimizing them through feedback, **ISYS** provides a practical approach for deploying memory agents on resource-constrained models.

6 Limitations

Despite its effectiveness, **ISYS** has several limitations. First, the current framework assumes that the monolithic agent specification can be decomposed into a sequence of main tasks. In our experiments, this decomposition is provided by a human expert or a large language model. Automatically discovering optimal task decompositions for the target SLM remains an open problem. Second, **ISYS** relies on LLM-based judging during the optimization process. Although we use multiple judges and observe consistent trends, LLM-based evaluation may still introduce bias or noise in preference judgments. Developing more reliable automated evaluation methods for memory quality remains an important direction for future work. Third, while scaffolding tasks improve reliability, they introduce additional execution steps. Although the overhead remains moderate due to sequential execution and KV cache reuse, the approach may still increase latency in extremely resource-constrained environments. Finally, our experiments focus on memory-agent frameworks. While the underlying idea of externalizing intermediate operations is general, further work is needed to evaluate its effectiveness in other agentic settings such as tool-use agents, planning agents, or interactive assistants.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *ICLR*.

Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Small language models are the future of agentic ai. *arXiv preprint arXiv:2506.02153*.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.

Gemini Team, Google. 2025. Gemini 2.5 pro: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. https://arxiv.org/abs/2507.06261. Technical Report.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. In *NeurIPS*.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*.

Yuanzhe Hu, Yu Wang, and Julian McAuley. 2026. Evaluating memory in llm agents via incremental multi-turn interactions. *ICLR*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory os of ai agent. In *EMNLP*.

Juntae Lee, Jihwan Bang, Seunghan Yang, and Simyung Chang. 2025. CIFLEX: Contextual instruction flow for sub-task execution in multi-turn interactions with a single on-device LLM. In *EMNLP*.

Yangning Li, Weizhi Zhang, Yuyao Yang, Wei-Chieh Huang, Yaozu Wu, Junyu Luo, Yuanchen Bei, Henry Peng Zou, Xiao Luo, Yusheng Zhao, et al. 2025a. Towards agentic rag with deep reasoning: A survey of rag-reasoning systems in llms. *arXiv preprint arXiv:2507.09477*, 2.

Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, et al. 2025b. Memos: A memory os for ai system. *arXiv preprint arXiv:2507.03724*.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of llm agents. In *ACL*.

OpenAI. 2024. Gpt-4o. https://openai.com/index/hello-gpt-4o/. Accessed: 2024-05-13.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *EMNLP*.

Raghav Sharma and Manan Mehta. 2025. Small language models for agentic systems: A survey of architectures, capabilities, and deployment trade offs. *arXiv preprint arXiv:2510.03847*.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. 2025a. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*.

Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025b. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136*.

Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, et al. 2025. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *ACL*.

Qwen Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*.

Yaxiong Wu, Yongyue Zhang, Sheng Liang, and Yong Liu. 2025. Sgmem: Sentence graph memory for long-term conversational agents. *arXiv preprint arXiv:2509.21212*.

Derong Xu, Yi Wen, Pengyue Jia, Yingyi Zhang, Yichao Wang, Huifeng Guo, Ruiming Tang, Xiangyu Zhao, Enhong Chen, Tong Xu, et al. 2026. From single to multi-granularity: Toward long-term memory association and selection of conversational agents. *ICLR*.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *NeurIPS*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *ICLR*.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by backpropagating language model feedback. *Nature*.

Operation	Definition
Intermediate operation 1	Segmentation of the raw input text into candidate memory units.
Main decision 1	Detection and filtering of noisy or irrelevant candidate units from the raw input.
Intermediate operation 2	Identification of relationships between the new memory and neighboring memories.
Main decision 2	Conflict checking between the new memory and its neighboring memories.
Intermediate operation 3	Aggregation of contextual information from neighboring memories.
Main decision 3	Novelty assessment of the new memory against existing memories.

Table 6: Intermediate operations and corresponding main decisions used in the motivation experiment.

Method	Scaffolding prompts	Intermediate outputs	Turns	Score
LLM: GPT-5.2				
Original Prompt	✗	✗	Single	0.81
SLM: Qwen3-4B-Thinking				
Original Prompt	✗	✗	Single	0.62
SLM: Qwen3-4B-Instruct				
Original Prompt	✗	✗	Single	0.40
Strengthened Prompt	✓	✗	Single	0.41
Sequential Execution of Main Tasks	✗	✓	Sequential	0.50
Sequential Execution with Scaffolding Tasks	✓	✓	Sequential	0.89

Table 7: Prompting structures and performance in the motivation experiment. Explicit intermediate artifacts substantially improve SLM reliability.

A Appendix

This appendix follows the overall logic of the paper. Section A.1 presents the controlled motivation experiment. Section A.2 details **ISYS** and its top-K extension. Section A.3 summarizes the optimization-time prompt families. Section A.4 describes the benchmarks. Section A.5 details the agent tasks and their monolithic decompositions. Section A.6 presents the optimization setup. Section A.7 provides the pairwise evaluation prompt. Section A.8 presents qualitative comparisons of memory-construction outputs. Sections A.9 and A.10 report additional evaluation results. Long-form prompt listings are collected in Section A.11.

A.1 Motivation Experiments

We begin with a controlled experiment designed to isolate the two bottlenecks highlighted in the main paper: prompt complexity and logical incompleteness. In many memory-agent prompts, several intermediate operations are required for correct execution but are not stated explicitly. High-capacity LLMs can often recover these missing steps implicitly, whereas SLMs frequently cannot.

Each instance consists of a new memory input together with its top-5 retrieved neighbor memories. Given this input, the agent must make several decisions whose correctness depends on latent intermediate operations. Table 6 lists the intermediate operations and corresponding main decisions used in this study.

We compare four prompting structures that progressively increase task structure.

Single-prompt. The model is asked to solve all main decisions in a single monolithic prompt, so all intermediate reasoning must be carried out implicitly within one inference.

Strengthened prompting. This setting augments the monolithic prompt with additional guidance intended to encourage useful intermediate reasoning. However, the task is still executed in a single prompt and the model is not required to produce explicit intermediate outputs.

Sequential execution of main tasks. Here, the monolithic prompt is decomposed into several main tasks that are executed sequentially. This reduces prompt complexity by distributing the original requirements across multiple turns, but introduces no dedicated scaffolding prompts.

Sequential execution with scaffolding tasks. In the final setting, we insert scaffolding prompts before selected main tasks. These scaffolding tasks explicitly externalize intermediate operations as structured outputs that subsequent main tasks can directly condition on.

Stage	Description
Initialize	For each main task c_t, synthesize one candidate scaffolding task $s_t^{(0)}$ conditioned on the original monolithic prompt P_{mono}.
Placement	Enumerate binary placement masks $\mathbf{m} \in \{0,1\}^T$ and select the best mask according to the aggregate judge score on $\mathcal{D}_{\text{train}}$.
Refinement	Keeping the selected mask fixed, iteratively rewrite only the activated scaffolding tasks using execution trajectories and judge feedback.

Table 8: Three stages of **ISYS**.

Prompt	Role
P_{mono}	Original monolithic prompt that defines the target agent behavior. It remains fixed throughout optimization.
P_{init}	Initialization prompt used to synthesize one candidate scaffolding task for each main task.
P_{refine}	Refinement prompt used to rewrite one activated scaffolding task using trajectories and judge feedback.
P_{judge}	Judge prompt used to score compiled procedures and produce narrative feedback for optimization.

Table 9: Optimization-time prompt families used by **ISYS**.

Table 7 shows a large gap between LLM and SLM behavior under implicit prompting. Under the original monolithic prompt, Qwen3-4B-Instruct performs poorly (0.40), whereas GPT-5.2 reaches 0.81, suggesting that larger models often succeed by implicitly performing intermediate operations that are not explicitly specified in the prompt. Strengthened prompting yields almost no improvement (0.41), indicating that surface-level prompt refinement alone does not resolve the underlying bottlenecks. Sequential execution of main tasks provides a modest gain (0.50), suggesting that decomposition partially alleviates prompt complexity. The strongest improvement occurs only when scaffolding tasks are introduced (0.89), indicating that explicitly externalizing intermediate operations as structured artifacts enables SLMs to execute the task much more reliably.

These results support two conclusions. First, many memory-agent decisions depend on intermediate operations that are often left implicit in monolithic prompts. Second, SLM performance improves substantially when those operations are externalized into explicit artifacts that subsequent steps can directly use.

A.2 Algorithm Details of ISYS

Table 8 summarizes the three stages of **ISYS**, and Algorithm 1 presents the full training-time procedure used in our experiments.

Top-K mask generalization. The Stage 2 selection in the main text uses the top-1 mask for simplicity of presentation and optimization. More generally, **ISYS** can retain the top-K masks under the initialized bank $B^{(0)}$. Let $\mathcal{M}_K \subseteq \{0,1\}^T$ denote the set of the K highest-scoring masks according to $r_{\mathbf{m}}^{(0)}$. Stage 3 can then be applied independently to each $\mathbf{m} \in \mathcal{M}_K$, yielding refined scaffold banks $B_{\mathbf{m}}^{\star}$ and final compiled procedures $\pi(\mathbf{m}; B_{\mathbf{m}}^{\star})$. The final output can be selected using the corresponding post-refinement training score. Our experiments use the $K = 1$ instantiation, which was sufficient in practice and keeps the optimization procedure simple, but the method itself does not rely on top-1 selection.

A.3 Optimization-Time Prompt Families

We use four prompt families during training-time optimization. Throughout optimization, the original monolithic prompt P_{mono} is kept fixed; **ISYS** only synthesizes and refines auxiliary scaffolding tasks around it. Table 9 summarizes the role of each prompt family.

In our experiments, concrete instances of P_{mono} correspond to the original core specifications of A-mem and Mem0. Because these prompts are framework-specific and lengthy, we summarize their task structure in Section A.5 and collect long-form prompt listings in Section A.11. For completeness, we include the concrete templates for P_{init}, P_{refine}, and P_{judge} below.

A.4 Datasets

We evaluate on two benchmarks designed for long-term memory agents.

LoCoMo. LoCoMo is a conversational benchmark consisting of temporally ordered dialogues spanning multiple sessions. Conversations can contain hundreds of turns and several thousand tokens. The evaluation suite includes question types testing single-hop recall (SH), multi-hop reasoning (MH), open-domain questions (OD), temporal reasoning (Temp), and adversarial robustness (Adv).

MemoryAgentBench. MemoryAgentBench provides a unified framework for evaluating persistent memory capabilities in agent systems, including retrieval accuracy, temporal reasoning, and selective forgetting. We focus on subsets aligned with

Algorithm 1 `ISYS`: Training-time scaffolding-task placement and instruction refinement

Require: Monolithic prompt P_{mono}, decomposed main tasks $C = (c_1, \ldots, c_T)$, target SLM f, high-capacity LLM g, training set $\mathcal{D}_{\text{train}}$, maximum refinement rounds N_{round}, per-index rewrite budget R
Ensure: Compiled sequential procedure $\pi(\mathbf{m}^\star; B^\star)$

for $t = 1$ **to** T **do**
  $s_t^{(0)} \sim g(c_t, P_{\text{mono}}; \mathcal{P}_{\text{init}})$
end for
$B^{(0)} \leftarrow (s_1^{(0)}, \ldots, s_T^{(0)})$
for each $\mathbf{m} \in \{0,1\}^T$ **do**
  $\mathcal{T}_{\mathbf{m}}^{(0)} \leftarrow \textsc{Execute}(f, \pi(\mathbf{m}; B^{(0)}), \mathcal{D}_{\text{train}})$
  $(r_{\mathbf{m}}^{(0)}, \delta_{\mathbf{m}}^{(0)}) \leftarrow g(\mathcal{T}_{\mathbf{m}}^{(0)}; \mathcal{P}_{\text{judge}})$
end for
$\mathbf{m}^\star \leftarrow \arg\max_{\mathbf{m} \in \{0,1\}^T} r_{\mathbf{m}}^{(0)}$
$B, \mathcal{T}, r, \delta \leftarrow B^{(0)}, \mathcal{T}_{\mathbf{m}^\star}^{(0)}, r_{\mathbf{m}^\star}^{(0)}, \delta_{\mathbf{m}^\star}^{(0)}$
$A^\star \leftarrow \{t \mid m_t^\star = 1\}$
for $q = 1$ **to** N_{round} **do**
  $\textsc{Shuffle}(A^\star)$
  improved $\leftarrow$ False
  for each $i \in A^\star$ **do**
    for $j = 1$ **to** R **do**
      $\tilde{s}_i \sim g(i, \mathcal{T}, P_{\text{mono}}, \delta; \mathcal{P}_{\text{refine}})$
      $\tilde{B} \leftarrow \textsc{Replace}(B, i, \tilde{s}_i)$
      $\tilde{\mathcal{T}} \leftarrow \textsc{Execute}(f, \pi(\mathbf{m}^\star; \tilde{B}), \mathcal{D}_{\text{train}})$
      $(\tilde{r}, \tilde{\delta}) \leftarrow g(\tilde{\mathcal{T}}; \mathcal{P}_{\text{judge}})$
      if $\tilde{r} > r$ **then**
        $B, \mathcal{T}, r, \delta \leftarrow \tilde{B}, \tilde{\mathcal{T}}, \tilde{r}, \tilde{\delta}$
        improved $\leftarrow$ True
        break
      end if
    end for
  end for
  if improved $=$ False **then**
    break
  end if
end for
$B^\star \leftarrow B$
return $\pi(\mathbf{m}^\star; B^\star)$

persistent memory construction: SH-QA, MH-QA, EventQA, and FactConsolidation-SH/MH.

A.5 Agent Tasks and Monolithic Decomposition

We apply `ISYS` to two fixed core specifications in each framework: information extraction and memory management. At each time step, the agent first constructs a candidate memory representation from the current input segment, retrieves related memories from the current store, and then executes a framework-specific memory-management decision.

A.5.1 Detailed task definitions

For LoCoMo, each utterance is treated as a memory input. For MemoryAgentBench, documents are split into 128-token chunks, and each chunk is treated as a memory input.

For each input x, we first execute the information extraction core specification to produce a candidate memory representation. We then retrieve the top-5 nearest-neighbor memories from the current memory store and execute the memory management core specification conditioned on both the new input and the retrieved neighbors.

For both A-mem and Mem0, we keep the original core specifications unchanged and only interleave scaffolding tasks discovered by `ISYS`. We show the description of the core specifications in Table 10.

A.5.2 Monolithic decomposition for memory management

For each agent specification, we decompose the original monolithic prompt into a sequence of main tasks $C = (c_1, \ldots, c_T)$. Importantly, this decomposition does not introduce new instructions. Each c_t corresponds to a contiguous segment of the original prompt and preserves the original wording.

Table 11 shows an example decomposition for the A-mem memory management specification.

Framework	Information extraction task	Memory management task
A-mem	Produce a candidate structured memory representation from the current input segment.	Decide whether the new memory should evolve the neighborhood and, if so, whether to strengthen links or update neighboring entries.
Mem0	Convert the current input segment into candidate user-specific facts.	Decide whether retrieved facts should be added, preserved, or deleted from the memory store based on overlap and contradiction checks.

Table 10: Agent tasks optimized by **ISYS** in our experiments.

Main task	Description
c_1	Should this memory be evolved?
c_2	What specific actions should be taken (e.g., strengthen the connection or update the neighbor)?
c_3	If strengthening is chosen, determine how the relationship between memories should be updated.
c_4	If updating the neighbor is chosen, determine the appropriate update operation.

Table 11: Main task decomposition of a monolithic agent prompt (memory management in A-mem).

	$c_1c_2c_3c_4$	$c_1c_2,\ c_3c_4$	c_1, c_2, c_3, c_4
$c_1c_2c_3c_4$	–	10.53	10.10
$c_1c_2,\ c_3c_4$	89.47	–	37.20
c_1, c_2, c_3, c_4	**89.90**	**62.80**	–

Table 12: Pairwise win rate (%) of memory-quality evaluation across task decomposition variants. Each cell indicates how often the row variant is preferred over the column variant by the LLM judge (ties excluded).

A.5.3 Planned ablation on decomposition granularity

We conduct an ablation study to examine how the granularity of core task decomposition affects memory quality. Specifically, we compare three variants of the memory-management decomposition: (i) the fully decomposed four-step variant (c_1, c_2, c_3, c_4), (ii) a two-step grouped variant (c_1c_2, c_3c_4), and (iii) the fully grouped monolithic variant $(c_1c_2c_3c_4)$.

Table 12 reports pairwise win rates under the same memory-quality evaluation protocol. The results show a consistent ordering across all comparisons: the fully decomposed variant (c_1, c_2, c_3, c_4) achieves the highest preference rate, followed by the partially grouped variant (c_1c_2, c_3c_4), while the monolithic variant $(c_1c_2c_3c_4)$ performs worst.

This trend indicates that even when scaffolding tasks are available, residual prompt complexity in grouped variants still makes reliable execution difficult for SLMs. Fully separating the core decisions reduces this complexity and enables the scaffolding tasks to guide the model more effectively, resulting in substantially higher memory quality.

A.6 Optimization Details

Training set used for prompt optimization. All offline scaffold-placement search and instruction refinement are performed on a training set $\mathcal{D}_{\text{train}}$, defined as the first 50 samples of LoCoMo. For a fixed mask $\mathbf{m}$ and scaffolding-task bank B, the compiled procedure $\pi(\mathbf{m}; B)$ is executed on each $x \in \mathcal{D}_{\text{train}}$. The judge then evaluates the resulting trajectory set and returns a scalar score and narrative feedback, which are used to select the best mask and accept or reject candidate rewrites.

Placement search. Each core specification contains only a small number of main tasks, making exhaustive enumeration over binary placement masks tractable. We therefore enumerate all masks in $\{0, 1\}^T$ during the scaffold-placement stage and select the single best mask according to the aggregate judge score on $\mathcal{D}_{\text{train}}$.

Optimization procedure. After selecting the best mask, **ISYS** refines only the activated scaffolding tasks while keeping the original main-task instructions unchanged. At each rewrite step, the target SLM executes the compiled procedure on $\mathcal{D}_{\text{train}}$, the judge evaluates the resulting trajectory set, and the synthesizer proposes a rewrite for one selected scaffolding task using the current trajectories and judge feedback. A candidate rewrite is accepted only when it improves the aggregate judge score; otherwise it is discarded. The order of activated scaffolding tasks is reshuffled at each refinement round, and optimization stops when an entire round produces no improvement.

A.7 Pairwise Judging Prompt for Memory-Quality Evaluation

For memory-quality evaluation, we compare a candidate method against GPT-4o under the same original instruction. The judge is asked to assess ad-

Pairwise judging prompt for memory-quality evaluation

```
# Role
You are an expert AI Judge evaluating two Memory Evolution Agents (Model A and Model B).

# 1. The Standard (Original Instruction) containing the input memory and neighbor memories.
[BEGIN INSTRUCTION]
{original_instruction}
[END INSTRUCTION]

# 2. Contestants
## Model A Output:
{model_a_output}

## Model B Output:
{model_b_output}

# Order-Invariance Rules
- The presentation order of Model A and Model B must not affect your evaluation in any way.
- Treat both candidates as if their positions were randomized; ignore labels ("Model A", "Model
    B") when judging quality.
- Base your judgment only on: accuracy, logical consistency, hallucination avoidance, and
    adherence to the original instruction.

# Output Format
Return the result in JSON:
{
  "analysis": {
    "Model_A_Quality": "Reason",
    "Model_B_Quality": "Reason"
  },
  "reasoning": "Explain strictly based on the Action-Field Logic table first. Then discuss
    quality.",
  "winner": "Model A" | "Model B" | "Tie"
}
```

Table 13: Pairwise judging prompt used for memory-quality evaluation.

herence to the original specification, faithfulness to the input, hallucination avoidance, logical consistency, and usefulness for future retrieval. Table 13 presents the pairwise judging prompt template used in this evaluation.

A.8 Qualitative Memory Construction Results

We present qualitative comparisons of memory management outputs under the same input. Each example consists of a single input memory together with its retrieved neighbor memories, and we compare the resulting outputs produced by (i) the original monolithic prompt with Qwen3-4B-Instruct, (ii) the same prompt with GPT-4o, and (iii) **ISYS**.

As shown in Table 14, the monolithic prompting baseline often produces incomplete or overly narrow updates, failing to capture the broader relational structure among memories. In contrast, **ISYS** generates more consistent and better-structured outputs by explicitly leveraging intermediate artifacts through sequential execution with scaffolding tasks.

Compared to GPT-4o, which tends to over-connect by including loosely related neighbors, **ISYS** yields more precise and better-calibrated updates, focusing on the most relevant connections while avoiding unnecessary expansions.

A.9 Judge Robustness: Gemini 3 Pro

Table 15 reports the memory-quality comparison using Gemini 3 Pro as the judge. We observe the same overall trend as in the main results. For brevity, we compare only Qwen3-4B-Instruct and **ISYS** against GPT-4o.

A.10 Additional Generalization Results

Generalization to another target model. To evaluate generality, we apply **ISYS** to another target model: Qwen3-0.6B.

This model has lower capacity than our primary target model. We observe that lower-capacity models tend to benefit from more explicit scaffolding

Input	**Context:** A casual conversational snippet where Melanie compliments a photograph and asks how long Caroline has had such a strong support system. **Content:** `Speaker Melanie says: Wow, that photo is great! How long have you had such a great support system?` **Keywords:** support system, photo, duration, relationship, compliment.
Top-5 retrieved neighbors	**45:** Caroline shares a family photo and describes friends, family, and mentors as her support system. **34:** Melanie expresses emotional support for Caroline during a major life transition. **23:** Caroline encourages Melanie and affirms family well-being and self-care. **43:** Caroline emphasizes mutual support, unity, and resilience. **5:** Melanie reacts to a photo and asks about the impact of Caroline's support group.
Original Prompt (Qwen3-4B-Instruct)	**Suggested connections:** `45` **Behavior:** Connects only to the most obvious neighbor and fails to capture broader support-network structure. **Takeaway:** Under-connected and overly narrow.
ISYS (Ours)	**Suggested connections:** `45, 43, 5` **Behavior:** Recovers a broader support-related structure by linking the input not only to the direct support memory (45), but also to mutual-support context (43) and prior support-group interaction (5). **Takeaway: Broader and more coherent than Qwen3-4B.**
Original Prompt (GPT-4o)	**Suggested connections:** `45, 34, 23, 43, 5` **Behavior:** Produces a broader but less selective update, including additional neighbors that are only loosely related to the query. **Takeaway: More comprehensive but less precise.**

Table 14: Qualitative comparison of memory-construction outputs under the same input. Qwen3-4B-Instruct under-connects the new memory to prior context, while ISYS recovers broader support-related structure by linking multiple relevant neighbors. GPT-4o remains the most comprehensive.

	A-mem		Mem0	
Method	**LoCoMo**	**Ruler**	**LoCoMo**	**Ruler**
Original Prompt	6.50	14.38	13.27	37.07
ISYS (ours)	**45.41**	**40.93**	**58.85**	**54.68**

Table 15: Memory quality comparison against GPT-4o evaluated by Gemini 3 Pro.

tasks and, in some cases, require a larger number of them. For example, Qwen3-4B-Instruct uses two scaffolding tasks, whereas Qwen3-0.6B requires three.

Comparing Qwen3-0.6B directly against GPT-4o is less informative in this setting, so we instead evaluate **ISYS** against the original Qwen3-0.6B pipeline. Under this comparison, **ISYS** is preferred in 81.72% of non-tied cases in the memory-quality evaluation.

This trend is consistent with our main claim that the importance of externalized intermediate operations increases as model capacity decreases.

A.11 Supplementary Prompt Listings

To keep the main appendix narrative compact, we place the long-form prompt listings here. Tables 16-18 shows the optimization-time prompt families. Tables 19-20 present the scaffolding tasks discovered by **ISYS**. Tables 21-27 provide the optimized prompts produced by the prompt-optimization baselines.

Initialization prompt for synthesizing candidate scaffolding tasks

```
You are designing ONE optional Scaffolding task to place immediately BEFORE the fixed main tasks
     below

Baseline main tasks are FIXED and must NOT be changed:
{fixed_main_tasks}

TARGET_MAIN_TASK (the next step):
{target_main_task}

Goal:
- Your pretask must externalize reusable clues as STRICT JSON to help the TARGET_BASELINE_STEP
    succeed.
- It should reduce hallucination and improve the final JSON truth-table correctness.
- It must be general (dataset-agnostic). Do NOT restate the full raw input.

Rules:
- Instruction MUST say: "Answer should be clear and concise. Return ONLY JSON".
- Output must be a JSON object (no markdown).
- Keep instruction short (2-6 lines).
- Prefer extracting: neighbor_ids, overlap/evidence, candidate decisions, length templates (if
    relevant).

MAIN_TASK_DESC:
{main_task_desc}

Return ONLY JSON with schema:
"step": {"name": "pre_before_s1_evolve", "instruction": "Answer should be clear and concise.
    Return ONLY JSON : {...}"}
```

Table 16: Initialization prompt P_{init} for synthesizing candidate scaffolding tasks.

Refinement prompt for rewriting activated scaffolding tasks

```
You are rewriting ONE PRETASK instruction that is placed immediately BEFORE a fixed baseline
    step.

Fixed baseline steps (do NOT modify):
{fixed_main_steps}

TARGET_BASELINE_STEP (next step):
{target_main_step}

CURRENT_SCAFFOLDING_TASK:
{current_scaffolding_task}

FEEDBACK (from evaluator/verifier):
{feedback}

Goal:
- Improve the SMALL model's success on the next step and final truth-table correctness.
- Externalize reusable clues as STRICT JSON so later steps can reuse them.

Rules:
- Rewrite ONLY the "instruction".
- Must say: "Answer should be clear and concise. Return ONLY JSON".
- No markdown / no extra text.
- Keep short (2-6 lines).
- General, dataset-agnostic.
- Prefer extracting what the next step needs (IDs, candidates, length templates, evidence).

Return ONLY JSON:
{
  "instruction": "Answer should be clear and concise. Return ONLY JSON ..."
}
```

Table 17: Refinement prompt P_{refine} for rewriting activated scaffolding tasks.

Feedback Prompt

```
You are an expert AI Judge aligned with this rubric:

Step 1 (Pass/Fail): Action-Field Logic Truth Table.
- Do NOT penalize simply for choosing a different action.
- Once an action is chosen, required fields MUST be populated and forbidden fields MUST be empty.


Step 2 (Pass/Fail): ID Validation for strengthen.
- IDs in suggested_connections must exist in neighborhood IDs from nearest neighbors.

Step 3 (Quality): Evaluate ONLY if Step 1 and Step 2 pass.
- For strengthen: semantic relevance of selected neighbors + specificity/relevance of
    tags_to_update.
- For update_neighbor: quality of new_context_neighborhood/new_tags_neighborhood; copying
    originals
is acceptable if no update needed.

You must return ONLY JSON.

INPUTS:
{inputs}

ALL_EXECUTED_STEPS:
{executed_steps}

TRAJECTORIES:
{trajectories}

FINAL_JSON:
{final_outputs}

Return ONLY JSON:
{
  "step1_action_field_pass": true/false,
  "step2_id_validation_pass": true/false,

  "instruction_score": <float 0..1>,
  "grounding_score": <float 0..1>,
  "logic_score": <float 0..1>,

  "action_field_violations": ["..."],
  "id_validation_violations": ["..."],

  "quality_assessment": ["..."],

  "step_scores": {{"<step_name>": <float 0..1>, "...": 0.0}},
  "step_blame": [{{"step_name":"...", "why":"...", "fix":"REWRITE:..."}}],

  "suggested_plan_changes": ["..."]
}
```

Table 18: Feedback prompt used in TextGrad, MiPro v2, GEPA, and our method for prompt optimization.

Discovered Scaffolding Tasks (A-mem)

Information extractor

Pre-before c_1**:**
```
Answer should be clear and concise. Return ONLY JSON
Extract only directly supported clues for the next step: (a) evidence: [{quote, line_or_offset}]
(b) keyword_candidates: [{term, evidence_index}], (c) tag_candidates: [{tag, evidence_index}]
(d) context_skeleton: one sentence with {blanks}.
Prefer including central entities/places and stated motivations; no invention.
If something is missing/unclear, set it to null and include an uncertainties array.
```

Memory management

Pre-before c_3**:**
```
Extract all available memory IDs from the input that could serve as connection candidates. For
     each
candidate, identify specific overlapping themes, concepts, or keywords between the current memory
      and
that neighbor. Prioritize memories containing first-person narratives, personal reflections, or
direct thematic alignment with the current memory's focus. Answer should be clear and concise.
     Return
ONLY JSON with structure:
{
    "candidate_neighbor_ids": ["id1", "id2"],
    "overlap_evidence": {"id1": "specific reason with theme", "id2": "specific reason with theme
     "},
    "current_memory_keywords": ["keyword1", "keyword2"],
    "high_priority_candidates": ["ids with strongest thematic or narrative alignment"]
}
```

Pre-before c_4**:**
```
Extract neighbor memory IDs, their current contexts, and current tags from the input in the
     exact
order provided. Count the total number of neighbors. For each neighbor, preserve the original
     context
and tag array structure exactly as given. Answer should be clear and concise. Return ONLY JSON
     with
structure:
{
    "neighbor_ids": ["id1", "id2", ...],
    "current_contexts": ["context1", "context2", ...],
    "current_tags": [["tag1", "tag2"], ["tag3"], ...],
    "neighbor_count": <number>
}
```

Table 19: Scaffolding tasks discovered when targeting Qwen3-4B-Instruct for A-mem.

Discovered Scaffolding Tasks (Mem0)

Information extractor

Pre-before c_1**:**
Detect language. Extract 0+ user-specific fact candidates as short strings in that language.
Provide user_spans: minimal exact quotes from user/assistant that support each candidate.
Return {language, candidates:[{fact, span_ids:[...] }],dropped:[{text, reason}]} and do not invent
facts.

Memory management

Pre-before c_1**:**
Given old_memory[] and retrieved_facts[],
output: existing_memory[{id,text}], max_existing_id, next_new_id.
For EACH retrieved_fact produce fact_overlap_analysis[{fact,is_new_vs_memory,
closest_overlapping_memory_ids[],overlap_evidence}].
Then output add_candidates[] (facts with is_new_vs_memory=true) and no_add_candidates[] (false);
keep these consistent.
Do not invent facts; only compute next_new_id from max_existing_id.

Pre-before c_3**:**
Given old_memory[] and retrieved_facts[], detect explicit contradictions only
(negation/opposite claim about same entity+attribute).
Output: ids_in_memory[], delete_ids[] (subset),
evidence_by_id{ id: {memory_text, contradicting_facts[], contradiction_type} }.
Also output safe_ids[] (all other ids) and ambiguity_notes (if contradiction is uncertain/ implicit).
Do not invent facts; if no clear contradiction, keep id in safe_ids and omit from delete_ids."

Table 20: Scaffolding tasks discovered when targeting Qwen3-4B-Instruct for Mem0.

Memory Management Prompt Optimized by TextGrad (1/2)

```
You are an AI memory evolution agent responsible for managing and evolving a knowledge base.

The new memory context (input):
{context}
content: {content}
keywords: {keywords}

The nearest neighbors memories (from knowledge base):
{nearest_neighbors_memories}

The number of neighbors is {neighbor_number}.

You are an expert data generator specialized in creating grounded, high-quality updates for knowledge
graph memories.Your goal is to integrate a new piece of information (`input_text`) into an existing
set of related memories (`neighborhood_context`) to evolve the knowledge graph accurately.

**Task Instructions:**
1.  **Analyze Relationships**: distinctively compare the `input_text` with the nodes in
`neighborhood_context`. Identify which nodes (neighbors) have a semantic relationship, contradiction,
causality, or reinforcement with the new input.
2.  **Determine Evolution**: Decide if the neighborhood structure or content needs to change to
reflect this new information (`should_evolve`).
3.  **Suggest Connections**: List the IDs of the specific neighbors that should be updated or linked
(`suggested_connections`).
4.  **Select Actions**: For each connection, determine the action (e.g., "update_neighbor").
```

Table 21: The final prompt optimized by TextGrad for A-mem memory management.

Memory Management Prompt Optimized by TextGrad (2/2)

```
5.  **Execute Updates (Crucial)**:
    *   **Context Generation**: For each selected neighbor, generate a `new_context_neighborhood`
    string.
    *   **Semantic Rule**: When the action is "update_neighbor", **do NOT replace** the neighbor's
    original memory with the new input. Instead, **rewrite** the neighbor's existing context to
    naturally incorporate the connection to the `input_text` while preserving the neighbor's original
    facts. The goal is to enrich the neighbor, not overwrite it.
6.  **Grounding**: For each new context string, extract `new_tags_neighborhood`. Every tag must be an **exact substring** of
      the corresponding updated context text.

**Output Format:**
You must output a single valid JSON object with no additional markdown or text. Ensure the output
strictly adheres to the following list-based structure to handle multiple updates simultaneously.

**JSON Schema Constraints:**
*   `new_context_neighborhood`: MUST be a **LIST** of strings. Even if updating only one neighbor, use
a list of length 1.
*   `new_tags_neighborhood`: MUST be a **LIST of LISTS** of strings.
*   **Synchronization**: The lengths of `suggested_connections`, `actions`,
`new_context_neighborhood`, and `new_tags_neighborhood` **MUST be exactly equal**.
The item at index `i` in `new_context_neighborhood` corresponds to the neighbor ID at index `i`
in `suggested_connections`.

**Final JSON Structure:**
{
  "should_evolve": true,
  "suggested_connections": ["<Neighbor_ID_1>", "<Neighbor_ID_2>", ...],
  "actions": ["strengthen", "update_neighbor"],
  "tags_to_update": ["tag_1",..."tag_n"],
  "new_context_neighborhood": [
    "<Rewritten text for Neighbor 1 merging original facts with new insight>",
    "<Rewritten text for Neighbor 2 merging original facts with new insight>"
  ],
  "new_tags_neighborhood": [
    ["<exact_substring_of_text_1>", "<exact_substring_of_text_1>"],
    ["<exact_substring_of_text_2>"]
  ]
}
```

Table 22: The final prompt optimized by TextGrad for A-mem memory management (continued).

Memory Management Prompt Optimized by MiPro-v2 (1/2)

```
You are a memory evolution planner for a retrieval-augmented conversational agent. You will be given:
(a) a NEW memory summary (`context`), (b) the raw utterance (`content`), (c) extracted `keywords`,
and (d) a blob of the `nearest_neighbors_memories` containing multiple neighbor memory entries with
at least an ID/index, context, keywords, and tags, plus `neighbor_number` (how many neighbors are
listed).

Job: decide whether and how to update the memory neighborhood so future retrieval is more consistent,
de-duplicated, and semantically searchable.\n\nReturn ONLY the following fields (no extra text):

1) `should_evolve`: True or False.
- True if the new memory overlaps strongly with one or more neighbors (same event/theme/people), adds
an important nuance worth merging into the neighborhood, clarifies meaning/tags, or reveals a stable
preference/identity/value that should improve retrieval.
- False if it's clearly unrelated to neighbors, too trivial/one-off, or cannot be improved by linking

2) `actions`: JSON list containing one or both of:
- "strengthen": create/strengthen links between the new memory and selected neighbors.
- "update_neighbor": revise neighbor contexts/tags (and optionally the new memory's neighborhood
phrasing) to make the cluster more coherent and searchable.
(If `should_evolve` is False, return `actions`: [] and empty lists for the remaining fields.)

3) `suggested_connections`: list of neighbor memory IDs/indexes (as strings) from the provided blob
that should be connected to the new memory.
- Choose the smallest set that best matches (typically 1~3). Prefer the most thematically/event-
aligned neighbors over loosely related ones.
```

Table 23: The final prompt optimized by MiPro-v2 for A-mem memory management.

Memory Management Prompt Optimized by MiPro-v2 (2/2)

```
4) `tags_to_update`: list of canonical, retrieval-friendly tags to add/adjust for this neighborhood.
- Tags should be short, consistent, and semantically meaningful (e.g., Family Values, Gratitude,
Counseling, Visual Sharing, LGBTQ Community, Adoption Journey, Emotional Support).
- Avoid duplicating near-synonyms; unify variants (e.g., pick Quality Time vs Family Time, not both
unless clearly distinct).

5) `new_context_neighborhood`: list of rewritten/augmented context sentences (1~3 items) that
summarize the *shared theme* across the new memory + selected neighbors.
- Keep it grounded in the given content; do not invent facts.
- It should read like a concise memory summary for later retrieval (who/what/why it mattered).
- If an image placeholder appears, mention it as "shares an image of ..." without over-speculating.

6) `new_tags_neighborhood`: list of tag-lists aligned by index to `new_context_neighborhood` (same
length).
- Each inner list: 6~10 tags max, consistent casing, mix of topic + intent (e.g., "Inquiry",
"Positive Reinforcement") when relevant.

Process you should follow (implicitly):
- Parse neighbor entries and identify which ones truly match the new memory's core theme (not just a
shared generic word).
- Decide evolve vs not.
- If evolving, pick connections, propose tag improvements, and produce coherent neighborhood
contexts/tags that reduce redundancy and increase future retrievability.

Output must be valid JSON-compatible field values for the six fields above.

Return your decision in JSON format with the following structure:
{
    "should_evolve": True or False,
    "actions": ["strengthen", "update_neighbor"],
    "suggested_connections": ["neighbor_memory_ids"],
    "tags_to_update": ["tag_1",..."tag_n"],
    "new_context_neighborhood": ["new context",...,"new context"],
    "new_tags_neighborhood": [["tag_1",...,"tag_n"],...["tag_1",...,"tag_n"]],
}
```

Table 24: The final prompt optimized by MiPro-v2 for A-mem memory management (continued).

Memory Management Prompt Optimized by GEPA (1/3)

```
You are an AI memory evolution agent. Decide whether a NEW memory note should (a) be linked more
strongly to one or more existing neighbor memories ("strengthen"), and/or (b) cause edits to the
neighbor memories' context/tags ("update_neighbor"), based on semantic relatedness, shared
entities, and shared themes.

INPUTS
- New memory:
  - context: {context}
  - content: {content}
  - keywords: {keywords}
- Nearest neighbors list (each includes memory index/id, content, context, keywords, tags):
  - {nearest_neighbors_memories}
- neighbor_number: {neighbor_number}

CORE DECISION
1) should_evolve:
- true if the new memory clearly relates to at least one neighbor (same topic, same people, same
event thread, same intent like self-care encouragement, career aspiration, support group reflection,
etc.) OR if it helps refine/clean neighbor tags/contexts.
- false if unrelated/too generic/no meaningful connection.

2) actions: choose any of:
- "strengthen": connect the new memory to specific neighbor memory id(s).
- "update_neighbor": revise one or more neighbor memories' context and/or tags for better retrieval
consistency, entity/topic clarity, or to incorporate a better framing (WITHOUT changing meaning).
```

Table 25: The final prompt optimized by GEPA for A-mem memory management.

Memory Management Prompt Optimized by GEPA (2/3)

```
ABSOLUTE STRUCTURAL RULES (MOST IMPORTANT)
A) suggested_connections
- Only include ids that appear as "memory index:<id>" in the input neighbors.
- If "strengthen" is NOT in actions, suggested_connections MUST be [].
- Output ids as STRINGS (e.g., "78"), not numbers.

B) If actions == ["strengthen"] ONLY
- You MUST NOT modify neighbors.
- new_context_neighborhood MUST be [].
- new_tags_neighborhood MUST be [].

C) If actions includes "update_neighbor" (alone OR with strengthen)
- You MUST output new_context_neighborhood AND new_tags_neighborhood.
- Their lengths MUST EXACTLY equal neighbor_number.
- CRITICAL: They MUST be in the EXACT SAME ORDER as the input nearest_neighbors_memories list.
  - Entry i corresponds to neighbor i in the input list (do not sort by id, do not reorder
  by similarity).
- For any neighbor you do NOT want to change:
  - Copy its original context EXACTLY (no paraphrase).
  - Copy its original tags EXACTLY (same items, same order).
- NEVER insert the NEW memory's context/tags into neighborhood arrays. Those arrays are ONLY for
neighbors.

D) Mapping discipline (to avoid corruption)
- Before writing output, create an internal index map like:
  - i=0 -> neighbor id X
  - i=1 -> neighbor id Y
  ...
- Then fill new_context_neighborhood[i] and new_tags_neighborhood[i] for that same neighbor id.
- Do not drop any neighbor; do not add extra entries.

WHEN TO PICK CONNECTIONS (strengthen)
- Prefer neighbors that share:
  - same main topic (Self-Care + Family Balance; Counseling/Mental Health career; LGBTQ support group
  ;Visual Sharing/Painting)
  - same speakers/entities (e.g., Caroline, Melanie)
  - same conversational thread (greeting/catch-up; question->answer; encouragement->reflection;
  aspiration->validation)
- suggested_connections can include multiple ids when the new memory bridges them.

WHEN TO UPDATE NEIGHBORS (update_neighbor)
- Only update if it materially improves retrieval/consistency:
  - Add missing high-signal categorical tags (e.g., Positive Reinforcement, Emotional Support, Career
  Planning/Career Aspiration, Self-Care Validation, LGBTQ Support Group, Visual Sharing, Painting).
  - Normalize overly generic/raw keyword tags into stable categories.
  - Clarify context to include who is speaking and what the utterance is doing (greeting, inquiry,
  validation, reflection), but DO NOT invent facts.
- Do NOT update neighbors just to mention the new memory; keep each neighbor faithful to its own
content.

TAGGING RULES
- tags_to_update applies ONLY to the NEW memory note (not neighbors).
- If should_evolve is false: keep tags_to_update minimal and generic (e.g., Dialogue, General
Reflection).
- If should_evolve is true: make tags specific and consistent with the memory base.
- Tags should be stable, human-readable categories (not raw keywords).
- Include, when relevant:
  - Conversation type: Dialogue, Inquiry, Greeting, Farewell/Conclusion
  - Topics: Self-Care, Family Balance, Work Stress, Mental Health, Counseling, Career Aspiration/
  Career Exploration, Education, LGBTQ Community, Support Group, Visual Sharing, Painting/Art
  - Interaction: Positive Reinforcement, Encouragement, Admiration, Gratitude, Reflection, Validation
  , Active Listening
  - Entities: Caroline, Melanie (only if present)
- Keep tag lists 6~12, non-redundant, consistent capitalization.
```

Table 26: The final prompt optimized by GEPA for A-mem memory management (continued).

Memory Management Prompt Optimized by GEPA (3/3)

```
CONTEXT REWRITE RULES (neighbors only)
- 1 sentence, concise, factual, grounded in that neighbor's content.
- Preserve original meaning; do not add new events.

OUTPUT FORMAT (STRICT)
- Output ONLY a single JSON object (no markdown, no extra text).
{
  "should_evolve": true/false,
  "actions": ["strengthen" and/or "update_neighbor"],
  "suggested_connections": ["neighbor_memory_ids_as_strings"],
  "tags_to_update": ["tag_1", "..."],
  "new_context_neighborhood": ["...", "..."],
  "new_tags_neighborhood": [[...], [...]]
}

FINAL VALIDATION CHECKLIST (DO THIS BEFORE RESPONDING)
1) If update_neighbor is present:
   - len(new_context_neighborhood) == neighbor_number
   - len(new_tags_neighborhood) == neighbor_number
   - For every i, the i-th entry corresponds to the i-th input neighbor (no reordering).
   - Unchanged neighbors: exact original context string + exact original tags list (same order).
2) If strengthen-only:
   - new_context_neighborhood == []
   - new_tags_neighborhood == []
3) suggested_connections:
   - ids exist in input neighbors
   - strings, not integers
   - [] if no strengthen
4) Output is valid JSON and nothing else.
```

Table 27: The final prompt optimized by GEPA for A-mem memory management (continued).