From Monolithic Prompts to Scaffolded Execution for Memory Agents with Small Language Models

Seunghan Yang[1] **Jihwan Bang**[1] **Sungha Choi**[2*†] **Juntae Lee**[1]
[1]Qualcomm AI Research[‡] [2]Kyung Hee University
{seunghan, jihwbang, juntlee}@qti.qualcomm.com sunghac@khu.ac.kr

Abstract

Personal memory agents process sensitive user data, making on-device deployment with small language models (SLMs) attractive. However, existing memory-agent prompts are often monolithic specifications written for high-capacity LLMs, overloading a single inference with many requirements while leaving necessary intermediate operations implicit. This mismatch creates two bottlenecks for SLMs: prompt complexity and logical incompleteness. We introduce **CAST** (**C**ompiling **A**gent **S**pecifications into Scaffolded **T**asks), a framework that converts a monolithic memory-agent prompt into a scaffolded execution procedure. **CAST** segments the original prompt into fixed, semantics-preserving main tasks, then synthesizes and places auxiliary scaffolding tasks that produce reusable intermediate artifacts; a trajectory-feedback step then adapts these scaffolds to the target SLM. At inference time, only the target SLM executes the compiled procedure. Across two memory-agent frameworks on LoCoMo and MemoryAgentBench, **CAST** outperforms monolithic prompting, prompt optimization, and reasoning-based baselines, while offering a stronger quality-latency trade-off than unconstrained reasoning. The results show that adapting memory agents to SLMs requires restructuring execution, not merely rewriting prompts.

1 Introduction

Large language models (LLMs) have increasingly been used as agents that maintain context, make decisions, and support long-running user interactions (Gemini Team, Google, 2025; OpenAI, 2024; Li et al., 2025a; Singh et al., 2025b; Yao et al., 2022). Such agents are typically controlled by prompt-based task specifications, which adapt off-the-shelf models to particular agent functions. As agentic services become more complex, these specifications often grow longer, more structured, and more difficult to execute reliably.

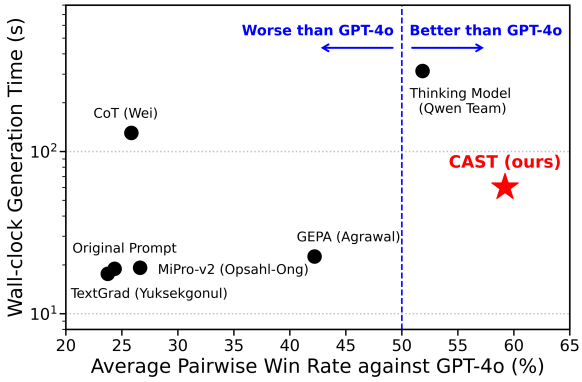


Figure 1: **Quality-efficiency trade-off of prompting strategies for SLMs on Qwen3-4B models.** The x-axis shows the average pairwise win rate against GPT-4o in memory quality evaluation (Table 1), and the y-axis shows the average wall-clock generation time measured on Snapdragon 8 Elite Gen 5. We compare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

This challenge is especially pronounced in personal memory agents. These agents maintain persistent user information across interactions and support information extraction, memory creation, updating, consolidation, and retrieval. Existing memory-agent systems (Xu et al., 2025b; 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) commonly specify these functions with monolithic prompts, bundling multiple memory-management requirements into a single instruction. This design is largely written for high-capacity, server-side LLMs, which can often

*Work completed while employed at Qualcomm AI Research.
†Corresponding author.
‡Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 20471–20498
October 24-29, 2026 ©2026 Association for Computational Linguistics

infer unstated intermediate operations internally.

However, personal memory data is often captured on edge devices such as smartphones, AR glasses, and personal assistants. Since such data can be highly sensitive, memory construction should ideally be performed on device, where privacy, cost, and latency constraints favor small language models (SLMs). The difficulty is that monolithic prompts written for high-capacity LLMs transfer poorly to SLMs. As shown in Fig. 1, Qwen3-4B-Instruct with the original prompt achieves an average pairwise win rate below 25% against GPT-4o in memory-quality evaluation.

We attribute this gap to two bottlenecks: *prompt complexity* and *logical incompleteness*. First, a single prompt often asks the SLM to perform many operations at once, such as extracting facts, comparing them with prior memories, detecting overlap or contradiction, selecting a memory operation, and producing a structured update. Second, memory-agent prompts often assume intermediate operations that are necessary for correct execution but never explicitly specified. For example, deciding whether to delete a memory may require identifying candidate memory IDs, grounding contradictions with evidence, and separating unsafe deletions from safe entries; linking memories may require selecting relevant neighbors and preserving their original order. While strong LLMs may recover such missing steps implicitly, SLMs often fail when these operations remain hidden inside one instruction.

Natural remedies are insufficient. Prompt optimization methods such as TextGrad (Yuksekgonul et al., 2025), MiPro-v2 (Opsahl-Ong et al., 2024), and GEPA (Agrawal et al., 2026) refine the surface text of a prompt, but they do not change the underlying execution structure and may further lengthen an already complex prompt. Reasoning-based methods, including chain-of-thought prompting (Wei et al., 2022), tree-of-thought reasoning (Yao et al., 2023), and reasoning-oriented models (Guo et al., 2025), expose intermediate reasoning, but the reasoning process remains unconstrained: the model decides which operations to perform, in what order, and at what level of detail. This can be unreliable for SLMs and can greatly increase token usage and latency. Fig. 1 shows that prompt optimization and CoT remain limited, while the thinking model improves quality at much higher generation cost.

This suggests that adapting memory agents to SLMs requires changing not only the wording of a prompt, but also how the prompt is executed. Our controlled motivation experiment in Appendix A.1 supports this view. Strengthening a monolithic prompt provides little improvement, and executing only the decomposed main tasks improves reliability only modestly. The largest gain appears when missing intermediate operations are externalized as explicit structured artifacts that subsequent steps can reuse. Thus, the core adaptation target is the execution procedure, not merely the prompt surface form.

We introduce **CAST** (**C**ompiling **A**gent **S**pecifications into Scaffolded **T**asks), a framework that converts a monolithic memory-agent prompt into a scaffolded execution procedure for a target SLM. The main-task sequence is obtained by a lightweight LLM-assisted segmentation of the original prompt, where each main task preserves the original instruction content. **CAST** then synthesizes auxiliary scaffolding tasks that materialize reusable intermediate artifacts, selects where to insert them, and refines their instructions using trajectory-level feedback. The resulting scaffolds produce artifacts such as evidence spans, candidate memory IDs, contradiction checks, and order-preserving summaries, allowing later main tasks to rely on explicit intermediate information rather than inferring it implicitly. This compilation is performed offline; at deployment time, the compiled procedure is fixed and executed only by the target SLM.

Across two memory-agent frameworks and two long-term memory benchmarks, LoCoMo (Maharana et al., 2024) and MemoryAgentBench (Hu et al., 2026), **CAST** outperforms monolithic prompting, prompt optimization, and reasoning-based baselines among SLM methods. It also provides a stronger quality-latency trade-off than unconstrained reasoning and improves downstream RAG performance. These results suggest that reliable on-device memory agents require moving from monolithic prompts to scaffolded execution procedures that make implicit intermediate operations explicit.

2 Related Work

Memory agents and on-device SLMs. Small language models (SLMs) are increasingly studied as practical alternatives to LLMs under latency, cost, and privacy constraints (Belcak et al., 2025; Sharma and Mehta, 2025; Lee et al., 2025). This is especially relevant for personal memory agents,

where raw user data should ideally remain on edge devices. Recent memory-agent systems, including A-mem (Xu et al., 2025b), Mem0 (Chhikara et al., 2025), HippoRAG (Gutiérrez et al., 2024), HippoRAG2 (Gutiérrez et al., 2025), MemOS (Li et al., 2025b), MemoryOS (Kang et al., 2025), and long-term conversational memory methods (Tan et al., 2025; Wu et al., 2025; Xu et al., 2026), maintain persistent information through extraction, updating, consolidation, and retrieval. However, these pipelines are typically specified with structured prompts designed for high-capacity LLMs. When executed by SLMs, such prompts may bundle many implicit operations, such as overlap detection, contradiction checking, memory-operation selection, and schema-constrained generation. **CAST** targets this mismatch by compiling monolithic memory-agent specifications into scaffolded execution procedures for SLMs.

Prompt optimization, decomposition, and reasoning. Prompt optimization methods, such as TextGrad (Yuksekgonul et al., 2025), MiPro-v2 (Opsahl-Ong et al., 2024), and GEPA (Agrawal et al., 2026), adapt prompts to a target model by rewriting instructions, demonstrations, or constraints. These methods are effective when the task is already well specified, but rewriting a monolithic memory-agent prompt does not necessarily expose the intermediate artifacts needed by SLMs. Task-decomposition methods, such as least-to-most prompting and decomposed prompting (Zhou et al., 2022; Khot et al., 2022), reduce complexity by splitting tasks into smaller steps, but they generally assume that the resulting subtasks are logically complete. Reasoning methods, including chain-of-thought, zero-shot chain-of-thought, self-consistency, tree-of-thought, and reasoning-oriented models (Wei et al., 2022; Kojima et al., 2022; Wang et al., 2022; Yao et al., 2023; Guo et al., 2025), elicit intermediate reasoning, but leave the choice and ordering of operations to the model and can increase latency. In contrast, **CAST** offline-compiles task-specific scaffolding artifacts that are fixed at deployment time and directly reused by later main tasks.

3 Proposed Method

We present **CAST** (**C**ompiling **A**gent **S**pecifications into Scaffolded **T**asks), a framework that converts a monolithic memory-agent prompt into a scaffolded execution procedure for a target SLM. Figure 2 gives an overview: **CAST** first obtains a fixed main-task sequence through lightweight LLM-assisted segmentation of the original prompt, while preserving the original instruction content. It then adds auxiliary scaffolding tasks that explicitly produce intermediate artifacts needed by later steps. During *offline compilation*, a high-capacity LLM is used for main-task segmentation, scaffold synthesis, placement selection, and scaffold refinement using execution trajectories from the target SLM. During *deployment*, the compiled procedure is fixed, and only the target SLM executes the resulting sequence of main and scaffolding tasks.

3.1 Problem Setup

Let P_{mono} denote the original monolithic prompt for a memory-agent function. At each time step, the agent receives an input x, such as a user utterance, a memory note, a document segment, or a query, together with a persistent memory store M. The goal of the agent function is to produce an output y that follows P_{mono}, such as a structured candidate memory, a memory update decision, or a set of memory-management operations.

The target model is a small language model f. During offline compilation, **CAST** uses a high-capacity LLM g and a small training set $\mathcal{D}_{\text{train}}$. At deployment time, neither g nor $\mathcal{D}_{\text{train}}$ is used. Our objective is to compile P_{mono} into a fixed sequential procedure that helps f produce outputs satisfying the original specification while preserving the instruction content of P_{mono}.

3.2 Scaffolded Execution Procedure

Before scaffold optimization, **CAST** performs a lightweight main-task segmentation of the original monolithic prompt. We define a main task c_t as a semantically coherent span of the original instruction that can be executed as a single step without requiring information the agent has not yet produced. Given P_{mono}, we use a fixed LLM-assisted segmentation prompt to identify semantically coherent task boundaries and obtain an ordered main-task sequence

$$C = (c_1, \ldots, c_T). \tag{1}$$

A boundary is placed where the instruction shifts to a new decision or output field, such as between deciding whether to evolve the memory and deciding which action to take. This ensures that each c_t has a self-contained input/output interface. This preparation step is not optimized on $\mathcal{D}_{\text{train}}$ and is

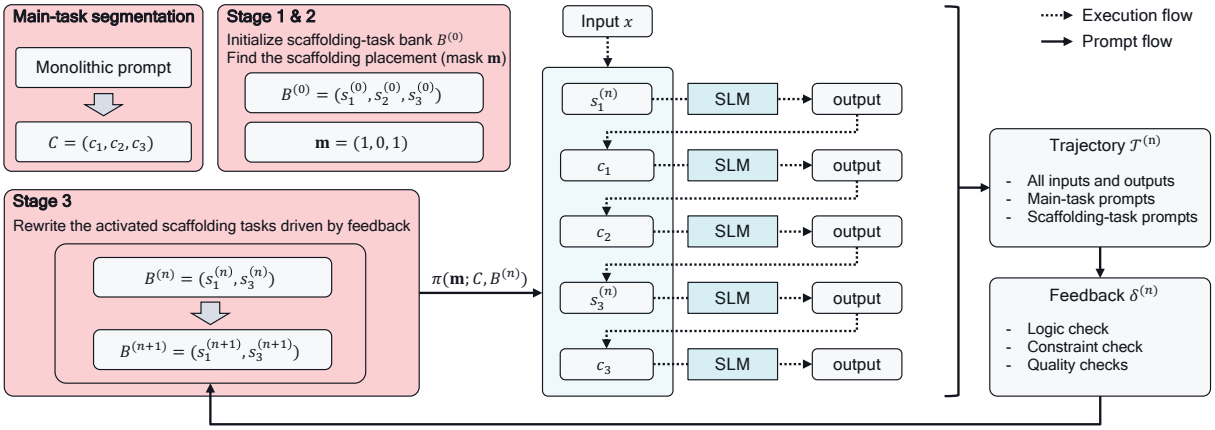


Figure 2: **Overview of CAST.** Given a monolithic memory-agent prompt P_{mono}, **CAST** first obtains fixed main-task spans $C = (c_1, c_2, c_3)$ through LLM-assisted segmentation that preserves the original instruction content. It then initializes a scaffolding-task bank B, selects where to insert scaffolding tasks using a binary placement mask $\mathbf{m}$, and refines the activated scaffolds using trajectory-level feedback. At deployment time, the compiled procedure is fixed and executed only by the target SLM.

not used to introduce new task instructions. Each c_t preserves the original instruction content and does not add requirements, remove constraints, or optimize the wording of the task. For A-mem (Xu et al., 2025b), this yields four main tasks: (i) whether the memory should evolve, (ii) which action to take, (iii) how to update the relationship if strengthening is chosen, and (iv) which update operation to apply if the neighbor is updated. The full worked decomposition for both frameworks is given in Tables 12 and 13. After segmentation, C is fixed; **CAST** optimizes only auxiliary scaffolding tasks inserted around these main tasks.

A *scaffolding task* is an auxiliary instruction inserted before a main task to produce an explicit intermediate artifact. For each main task c_t, **CAST** maintains a candidate scaffolding task s_t. The output of s_t is intended to support c_t by making implicit prerequisite operations explicit. Typical artifacts include candidate memory IDs, grounded evidence spans, overlap rationales, contradiction checks, schema-validity checks, or order-preserving summaries of retrieved memories.

Scaffolding tasks are not allowed to change the original task specification. Instead, they externalize information that is already required implicitly by the original prompt. For example, if a memory-management prompt requires deciding whether an existing memory should be deleted, a scaffolding task may first enumerate candidate memory IDs, separate safe and potentially contradictory entries, and extract evidence for each contradiction. The final delete decision is still made by the original main task, but it can condition on explicit intermediate artifacts.

Not every main task requires scaffolding. **CAST** therefore uses a binary activation mask

$$\mathbf{m} = (m_1, \ldots, m_T) \in \{0, 1\}^T \tag{2}$$

to decide which scaffolding tasks are inserted. If $m_t = 1$, s_t is executed immediately before c_t; otherwise s_t is omitted. Given a scaffolding-task bank $B = (s_1, \ldots, s_T)$, the resulting scaffolded execution procedure is

$$\pi(\mathbf{m}; C, B) = ([s_1]^{m_1}, c_1, \ldots, [s_T]^{m_T}, c_T), \tag{3}$$

where $[s_t]^{m_t}$ denotes executing s_t when $m_t = 1$ and omitting it otherwise. For an input x, the target SLM f executes this procedure sequentially. The full execution trajectory $\tau_{\mathbf{m}}(x)$ contains the input, the main-task prompts, the activated scaffolding-task prompts, their intermediate outputs, and the final output.

3.3 CAST: Offline Compilation

Given the fixed main-task sequence C obtained by main-task segmentation, **CAST** compiles P_{mono} into a scaffolded execution procedure through three optimization stages: scaffold initialization, placement selection, and feedback-driven refinement. The main tasks remain fixed throughout these stages. **CAST** only synthesizes, places, and refines auxiliary scaffolding tasks around them.

Components. **CAST** runs *offline* using a high-capacity LLM g to perform the initial main-task

segmentation, draft and rewrite scaffolding-task instructions, and judge execution trajectories. At deployment time, only the target SLM f and the compiled procedure are used.

Stage 1: Scaffold initialization. For each main task c_t, the high-capacity LLM g synthesizes one candidate scaffolding task conditioned on c_t and the original monolithic prompt P_{mono}:

$$s_t^{(0)} \sim g(c_t, P_{\text{mono}}; P_{\text{init}}). \tag{4}$$

This produces the initial scaffolding-task bank

$$B^{(0)} = (s_1^{(0)}, \ldots, s_T^{(0)}). \tag{5}$$

The initialization prompt encourages each $s_t^{(0)}$ to produce concise, structured intermediate artifacts that can be reused by the subsequent main task.

Stage 2: Placement selection. **CAST** next determines which candidate scaffolding tasks should actually be inserted. Given $B^{(0)}$, we evaluate all binary masks $\mathbf{m} \in \{0,1\}^T$, including the all-zero mask. For each $\mathbf{m}$, the target SLM f executes the procedure $\pi(\mathbf{m}; C, B^{(0)})$ on the training set $\mathcal{D}_{\text{train}}$, producing a set of trajectories:

$$\mathcal{T}_{\mathbf{m}}^{(0)} = \{\tau_{\mathbf{m}}(x) \mid x \in \mathcal{D}_{\text{train}}\}. \tag{6}$$

The judge model g evaluates these trajectories against the original monolithic specification P_{mono} and returns a scalar score and narrative feedback:

$$(r_{\mathbf{m}}^{(0)}, \delta_{\mathbf{m}}^{(0)}) \sim g(\mathcal{T}_{\mathbf{m}}^{(0)}, P_{\text{mono}}; P_{\text{fb}}), \tag{7}$$

where $r_{\mathbf{m}}^{(0)} \in [0,1]$ is the training score and $\delta_{\mathbf{m}}^{(0)}$ is the corresponding feedback. We select the highest-scoring mask:

$$\mathbf{m}^\star = \arg\max_{\mathbf{m} \in \{0,1\}^T} r_{\mathbf{m}}^{(0)}. \tag{8}$$

In our experiments, the number of main tasks is small, so exhaustive mask enumeration is tractable. The all-zero mask serves as a no-scaffold baseline, allowing **CAST** to omit scaffolding when it does not improve execution. This top-1 selection is a special case of a top-K variant, which can retain and refine multiple high-scoring masks independently.

Stage 3: Feedback-driven refinement. After selecting $\mathbf{m}^\star$, **CAST** keeps the placement fixed and refines only the activated scaffolding tasks. Let

$$A^\star = \{t \mid m_t^\star = 1\} \tag{9}$$

be the set of activated indices. Starting from $B^{(0)}$, **CAST** iteratively rewrites one activated scaffolding task at a time using the current execution trajectories and trajectory feedback.

At refinement iteration n, suppose index $i \in A^\star$ is selected. The synthesizer g proposes a rewrite of s_i:

$$\tilde{s}_i \sim g(i, \mathcal{T}^{(n-1)}, P_{\text{mono}}, \delta^{(n-1)}; P_{\text{refine}}), \tag{10}$$

where $\mathcal{T}^{(n-1)}$ and $\delta^{(n-1)}$ are the trajectories and feedback from the current compiled procedure. **CAST** forms a proposal bank $\tilde{B}$ by replacing s_i with $\tilde{s}_i$, then re-executes the proposed procedure $\pi(\mathbf{m}^\star; C, \tilde{B})$ on $\mathcal{D}_{\text{train}}$. The judge returns a new score and feedback:

$$(\tilde{r}, \tilde{\delta}) \sim g(\tilde{\mathcal{T}}, P_{\text{mono}}; P_{\text{fb}}), \tag{11}$$

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

CAST accepts the rewrite only if it improves the current score:

$$\text{accept } \tilde{B} \quad \text{if} \quad \tilde{r} > r. \tag{12}$$

Otherwise, the proposal is discarded and the previous scaffolding task is retained. To reduce order bias, **CAST** reshuffles the activated indices at the beginning of each refinement round. Optimization stops when a full round produces no improvement or when the refinement budget is exhausted. The final output of offline compilation is the fixed scaffolded procedure

$$\pi(\mathbf{m}^\star; C, B^\star). \tag{13}$$

The algorithm pseudo-code is given in Appendix A.2, and all prompts used in our experiments are provided in Appendix A.3.

At deployment time, **CAST** does not use the high-capacity LLM g, the trajectory feedback prompt, or $\mathcal{D}_{\text{train}}$. The target SLM f executes the fixed procedure $\pi(\mathbf{m}^\star; C, B^\star)$ on each new input. Because the procedure is executed sequentially in a single interaction context, scaffold outputs remain available to later main tasks, allowing the SLM to rely on explicit artifacts rather than inferring all intermediate operations inside one monolithic prompt.

4 Experiments

We evaluate whether **CAST** enables SLMs to execute memory-agent specifications more reliably. We focus on memory quality, efficiency, downstream utility, and the contribution of scaffolded execution.

	A-mem		Mem0		Efficiency	
Method	**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
CAST (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.

4.1 Setup and Evaluation Protocol

Datasets and agents. We evaluate on LoCoMo (Maharana et al., 2024) and MemoryAgentBench (Hu et al., 2026), two benchmarks for long-term memory agents. We instantiate two representative memory-agent frameworks, A-mem (Xu et al., 2025b) and Mem0 (Chhikara et al., 2025). At each time step, the agent receives an input segment, constructs a candidate memory representation, retrieves related memories, and predicts a structured memory operation. Details of the datasets and agent tasks are provided in Appendix A.4 and A.5.

Implementation and baselines. Each framework defines two core specifications: one for information extraction and one for memory management. We apply **CAST** separately to each specification. For each monolithic prompt, we first obtain $C = (c_1, \ldots, c_T)$ using a fixed LLM-assisted segmentation prompt that preserves the original instruction content. The main-task sequence is then fixed, and **CAST** synthesizes auxiliary scaffolding tasks, searches over placement masks, and refines only the activated scaffolds using trajectory-level feedback.

All offline compilation is performed on a single training set D_{train}, the first 50 memory inputs of one LoCoMo conversation; the same D_{train} is used for the prompt-optimization baselines. Evaluation covers both benchmarks: on LoCoMo we evaluate on held-out inputs disjoint from D_{train}, and MemoryAgentBench is never used at any stage of optimization, so it additionally serves as a zero-shot cross-benchmark transfer setting. Appendix A.4 details the split, and Appendix A.11 reports the reverse direction, where the procedure is optimized on MemoryAgentBench and applied to LoCoMo.

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 CAST for A-mem.** The scaffolds externalize evidence selection, neighbor-linking candidates, and order-preserving update inputs.

Our primary target SLM is Qwen3-4B-Instruct (Qwen Team, 2025). We compare against the original monolithic prompt, prompt optimization methods including TextGrad (Yuksekgonul et al., 2025), MiPro-v2 (Opsahl-Ong et al., 2024), and GEPA (Agrawal et al., 2026), and reasoning baselines including CoT (Wei et al., 2022) and Qwen3-4B-Thinking. As large-model references, we report GPT-4o (Hurst et al., 2024) and GPT-5.2 (Singh et al., 2025a) under the original agent prompts.

Memory-quality evaluation. For each method, we compare its memory-management output against the output produced by GPT-4o under the same framework and original prompt. The judge receives the input, the original specification, and two candidate outputs, and selects which output better satisfies adherence to the original prompt, faithfulness, logical consistency, hallucination avoidance,

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
	CAST (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 3: **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.

and structured-output validity. GPT-4o is used as a reference system, not an oracle: it also executes the original monolithic prompt and may fail when intermediate operations remain implicit. We swap candidate order, repeat each comparison 10 times, and report the non-tied pairwise win rate against GPT-4o. We use GPT-5.2 as the primary judge and additionally report Gemini 3 Pro and Claude Opus as independent judges (Section 4.5). The full pairwise evaluation prompt is provided in Appendix A.8. In addition to the automated judge, we conduct a human evaluation with 50 annotators (Section 4.5).

Efficiency and downstream evaluation. We report the average number of generated tokens and wall-clock generation time per input, measured on Snapdragon 8 Elite Gen 5 including both prefill and decoding. For downstream RAG, we construct a memory store with each method and answer benchmark questions using retrieved memories and the same GPT-4o response generator. 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.

4.2 Memory Quality and Efficiency

Table 1 reports specification-conditioned pairwise win rates against the GPT-4o reference, together with token usage and wall-clock latency. Across both memory-agent frameworks and both benchmarks, **CAST** achieves the strongest memory quality among all SLM-based methods. It outperforms monolithic prompting, prompt optimization, and reasoning-based baselines, showing that the main bottleneck is not merely prompt wording but the execution structure of the agent specification.

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 4: **Deletion-oriented scaffolding task discovered by CAST for Mem0.** The scaffold externalizes contradiction evidence and safe deletion candidates before the final delete decision.

CAST also provides a stronger quality-efficiency trade-off than unconstrained reasoning. Qwen3-4B-Thinking improves over the original prompt, but generates substantially more tokens and incurs much higher latency. In contrast, **CAST** introduces short, structured, task-specific steps whose intermediate outputs remain available to later tasks in the same context. This allows **CAST** to improve memory quality while remaining substantially more efficient than thinking-style decoding.

Discovered scaffolding tasks. Table 2 shows representative scaffolding tasks optimized for A-mem. Rather than adding generic "think step by step" instructions, **CAST** discovers task-specific artifacts such as evidence spans, grounded candidate keywords, candidate neighbor links with overlap evidence, and order-preserving neighbor context summaries. These artifacts reduce prompt complexity by distributing the original monolithic requirements across multiple steps and mitigate logical incompleteness by materializing intermediate operations that were previously implicit.

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
	CAST (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.

4.3 Downstream RAG Performance

Mem0 results. Table 3 reports downstream QA performance for Mem0. **CAST** improves over monolithic prompting and prompt-optimization baselines, following the same trend as the memory-quality evaluation.

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. As shown in Table 4, **CAST** inserts a dedicated scaffold that enumerates existing memory IDs, separates delete candidates from safe items, and grounds each contradiction with evidence. The gains are especially pronounced on fact-consolidation subsets such as FC-SH and FC-MH, where the agent must merge, preserve, or delete stored facts when new evidence arrives. By externalizing contradiction detection and deletion evidence into structured scaffolding artifacts, **CAST** improves memory consolidation and downstream QA.

A-mem results. Table 5 reports downstream QA performance for A-mem. **CAST** again outperforms Qwen3-4B prompting baselines on average. The gain is particularly strong on the Adv subset of LoCoMo, which tests whether the system can distinguish answerable from unanswerable adversarial questions based on stored memory. This suggests that scaffolded execution produces more accurate and better-calibrated memory representations for retrieval and generation.

Relating memory quality to downstream gains. The improvement in memory quality is larger than the improvement in end-task QA, and the two metrics measure different objects. Memory quality scores every element of a memory-management output at each instance, *i.e.*, the action choice, the extracted keywords/tags, the neighbor edits, and the linked connections, so a single incorrect element lowers the score. Downstream QA instead checks only the final answer, which draws on the few facts a given question needs. Many specification-level errors therefore never surface in the answer: performing ADD where the specification requires DELETE leaves a redundant entry in the store, yet the retriever still returns the relevant fact and the answer is unchanged (Appendix A.9 gives a concrete case).

Consistent with this account, **CAST**'s downstream gains concentrate in exactly the question types that depend on well-managed memory. Multi-hop questions require correctly linked memories and adversarial questions penalize hallucinated content, and these are where **CAST** improves most among the Qwen3-4B methods: on A-mem it attains the best SLM multi-hop score (28.32) and an adversarial score of 49.04, above all other SLM methods (37.5–42.3) and above GPT-4o (39.99); on Mem0 it is again best among SLMs on both subsets (25.46 and 37.26). The overall average moves less because most questions do not require finely managed memory, which is why we report both metrics.

4.4 Structure Ablation

We evaluate whether **CAST**'s gains come from stronger prompting, task decomposition, or scaffolded execution. Table 6 compares the origi-

Method	A-L	A-R	M-L	M-R	Avg
Original	14.58	35.41	7.40	40.00	24.35
Strengthened	22.20	36.80	20.10	43.00	30.53
Main tasks only	23.75	39.80	24.40	45.00	33.24
Init. scaffolds	38.20	45.80	55.60	82.50	55.53
CAST full	40.63	49.47	59.26	87.50	59.22

Table 6: **Structure ablation on benchmarks.** A-L/A-R denote A-mem on LoCoMo/Ruler; M-L/M-R denote Mem0 on LoCoMo/Ruler. Values are pairwise win rates against the GPT-4o reference.

nal monolithic prompt, a strengthened monolithic prompt, sequential main tasks without scaffolding, initialized scaffolds without feedback-driven refinement, and full **CAST** under the same protocol.

The results show that prompt strengthening and decomposition alone are insufficient. Sequential main tasks reduce prompt complexity, but the largest single gain comes from inserting artifact-producing scaffolds and selecting where to place them (33.24→55.53); trajectory-level refinement then contributes a further improvement (→59.22). The value of refinement is moreover model-dependent: for weaker SLMs from other families it is not marginal but decisive (Appendix A.11). This supports our claim that **CAST** is not merely prompt rewriting or task decomposition; its main contribution is compiling the original specification into a scaffolded execution procedure.

Additional analysis. Appendix A.6 shows that finer main-task decomposition improves memory quality, and Section 4.5 and Appendix A.10 examine judge reliability. Appendix A.11 further evaluates Qwen3-0.6B and other model families, Appendix A.12 compares efficient-reasoning baselines (Xu et al., 2025a; Han et al., 2025), and Appendix A.13 reports a preliminary transfer result on NaturalPlan Trip Planning (Zheng et al., 2024).

4.5 Human Evaluation and Judge Reliability

Because our primary memory-quality metric is an LLM-based pairwise comparison, we verify that the reported ranking does not depend on the judge used during development. We therefore add (i) a human evaluation that uses none of the rubric or automated judge involved in optimization, and (ii) re-evaluation by two additional judge models.

We sample 100 memory-management instances from A-mem/LoCoMo and split them into five sets. Each annotator is assigned two sets, giving 80 judgments (40 instances × two comparisons: **CAST** vs. GPT-4o and Thinking vs. GPT-4o), for a total of 50 annotators and 4,000 judgments (2,000 per comparison). For each instance the annotator sees the input together with its five nearest-neighbor memories and two candidate outputs, presented blind and in randomized order, and selects the higher-quality memory: concretely, whether the appropriate neighbors were selected for linking and whether the keywords/tags were extracted well. Excluding ties, **CAST** is preferred over GPT-4o on 41.9% of comparisons, versus 22.4% for the Thinking baseline; the full vote counts and annotation instructions are given in Appendix A.15.

Evaluator	GPT-5.2	Gemini 3 Pro	Opus	Human
CAST win rate (%)	40.63	45.41	45.70	41.90

Table 7: Memory quality of **CAST** against GPT-4o under four independent evaluators (A-mem, LoCoMo).

Table 7 compares **CAST**'s win rate against GPT-4o under four independent evaluators. All four fall within a narrow 40–46% band, so the ranking is not an artifact of any single evaluator; in particular, the human study shares no rubric or judge with the optimization procedure.

5 Conclusion

We introduced **CAST**, a framework that adapts prompt-specified memory agents to SLMs by compiling monolithic prompts into scaffolded execution procedures. **CAST** first obtains fixed main-task spans through lightweight LLM-assisted segmentation that preserves the original instruction content, and then inserts scaffolding tasks that externalize implicit intermediate operations, further refined with trajectory-level feedback. Across two memory-agent frameworks and two benchmarks, **CAST** consistently improves memory quality while maintaining reasonable efficiency. Downstream RAG results further show that the resulting memory stores improve retrieval and generation. Our experiments suggest that adapting agent pipelines to SLMs requires more than prompt refinement or longer reasoning traces: it often requires restructuring the execution procedure so that hidden intermediate operations become explicit and reusable. By keeping the original task specification fixed while moving scaffold synthesis and refinement to offline compilation, **CAST** provides a practical approach for deploying memory agents on resource-constrained models.

6 Limitations

CAST targets memory-agent settings where recurring agent functions have stable specifications and are applied repeatedly over many inputs and long input streams. The cost of generating the scaffolded execution procedure is therefore incurred only during offline compilation. After deployment, the fixed compiled procedure is reused without additional synthesis or verification. Thus, **CAST** is not intended to replace ad hoc prompting in settings where each input requires a substantially different task specification. In addition, **CAST** preserves the original memory-agent specification and inserts scaffolding tasks around it without redefining the memory objective, schema, or retrieval module. This keeps the adaptation controlled and faithful to the existing pipeline. Accordingly, **CAST** focuses on improving how the target SLM executes a given specification, while assuming that the specification itself defines the desired memory-agent behavior.

For memory quality, each pairwise comparison is judged ten times (five per presentation order), so the reported win rates are averages rather than single draws; in addition, three independent judge models and 50 human annotators all place **CAST** in a narrow 40–46% band against GPT-4o. Our downstream QA results, however, are single-run point estimates at temperature 0, and a systematic variance study across decoding temperatures is left to future work.

7 Potential Risks

CAST is designed to support on-device memory-agent deployment, but its offline compilation stage may still involve sensitive data if real user trajectories are used for scaffold optimization. In our setting, the high-capacity LLM is used only during offline compilation and is not queried at deployment time. Nevertheless, if the optimization set contains private user conversations or memory states, sending these trajectories to an external model could introduce privacy risks. This risk can be mitigated by compiling **CAST** with public or de-identified calibration data, using a locally hosted or enterprise-controlled LLM for offline compilation, and applying redaction or data-minimization policies before trajectory feedback is generated.

A second risk is that scaffolding tasks make intermediate memory operations explicit. While this improves reliability, intermediate artifacts such as candidate memory IDs, contradiction evidence, or safe deletion sets may expose sensitive information if they are logged, stored, or shown to downstream components unnecessarily. To reduce this risk, scaffold outputs should be treated as transient execution artifacts rather than persistent memory entries. In practice, deployment systems can discard scaffold outputs after each memory update, restrict logging to final structured outputs, enforce access control over memory stores, and apply schema or rule-based validators to prevent unsafe updates such as unsupported deletion or hallucinated memory creation.

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.

Google. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

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*.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24842–24855.

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*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

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*.

Microsoft. 2024. Phi-3 technical report: A highly capable language model locally on your phone.

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*.

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

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*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

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*.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025a. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025b. 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*.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V Le, Ed H Chi, et al. 2024. Natural plan: Benchmarking llms on natural language planning. *arXiv preprint arXiv:2406.04520*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

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 8: 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 9: 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. Appendix A.1 presents the controlled motivation experiment. Appendix A.2 details **CAST** and its top-K extension. Appendix A.3 summarizes the optimization-time prompt families. Appendix A.4 describes the benchmarks. Appendix A.5 details the agent tasks and main-task segmentation. Appendix A.6 reports the decomposition-granularity ablation. Appendix A.7 presents the optimization setup. Appendix A.8 provides the pairwise evaluation prompt. Appendix A.9 presents qualitative comparisons of memory-construction outputs. Appendix A.10 and A.11 report additional evaluation results. Appendix A.12 compares **CAST** with efficient-reasoning baselines. Appendix A.13 reports a preliminary transfer result on NaturalPlan Trip Planning. Long-form prompt listings are collected in Appendix A.14.

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 8 lists the intermediate operations and corresponding main decisions used in this study.

We compare four prompting structures that progressively increase task structure. In the single-prompt setting, the model solves all main decisions in one monolithic prompt. In strengthened prompting, the monolithic prompt is augmented with additional guidance, but the model is still not required to produce explicit intermediate outputs. In sequential execution, the monolithic prompt is decomposed into main tasks, reducing prompt complexity but introducing no dedicated scaffolding prompts. Finally, sequential execution with scaffolding tasks explicitly externalizes intermediate operations as structured outputs that subsequent main tasks can directly condition on.

Table 9 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. Strengthened prompting yields almost no improvement (0.41), indicating that surface-level prompt refinement alone

Stage	Description
Initialize	For each fixed 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 feedback score on $\mathcal{D}_{\text{train}}$.
Refinement	Keeping the selected mask fixed, iteratively rewrite only the activated scaffolding tasks using execution trajectories and trajectory feedback.

Table 10: Three optimization stages of **CAST** after main-task segmentation.

Prompt	Role
P_{mono}	Original monolithic prompt that defines the target agent behavior. It remains fixed throughout optimization.
P_{seg}	Segmentation prompt used to identify semantically coherent main-task spans in P_{mono} while preserving the original instruction content.
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 feedback from P_{fb}.
P_{fb}	Trajectory feedback prompt used during offline compilation to score execution trajectories and return narrative feedback for scaffold optimization.

Table 11: Optimization-time prompt families used by **CAST**.

does not resolve the 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.

A.2 Algorithm Details of CAST

Before scaffold optimization, **CAST** obtains a fixed main-task sequence from the original monolithic prompt using a lightweight LLM-assisted segmentation prompt. This preparation step identifies semantically coherent task spans while preserving the original instruction content; it does not add new requirements, remove constraints, or optimize task wording. Given the resulting main-task sequence, Table 10 summarizes the three optimization stages of **CAST**, 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, **CAST** 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 $\pi(\mathbf{m}; C, B_{\mathbf{m}}^{\star})$. Our experiments use the $K = 1$ instantiation, which was sufficient in practice and keeps the optimization procedure simple.

A.3 Optimization-Time Prompt Families

We use five prompt families during training-time optimization. Throughout optimization, the original monolithic prompt P_{mono} is kept fixed; **CAST** only synthesizes and refines auxiliary scaffolding tasks around it. Table 11 summarizes the role of each prompt family.

Concrete instances of P_{mono} correspond to the original core specifications of A-mem and Mem0, used verbatim without manual selection or tuning on our part. 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.14.

A.4 Datasets

LoCoMo. LoCoMo is a conversational benchmark consisting of temporally ordered dialogues spanning multiple sessions. 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 persistent memory construction: SH-QA, MH-QA, EventQA, and FactConsolidation-SH/MH.

Train/test split. Scaffold optimization uses only D_{train}, the first 50 memory inputs of a single LoCoMo conversation; **MemoryAgentBench is not used at any stage of optimization**. The memory-quality test set is drawn from the remaining, disjoint inputs and excludes every training instance. For downstream QA, memory is built from the entire conversation, so the early inputs used during optimization do appear in the constructed store; the QA questions and answers were never observed

Algorithm 1 **CAST**: Training-time main-task segmentation, scaffolding-task placement, and instruction refinement

Require: Monolithic prompt P_{mono}, 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 $\pi(\mathbf{m}^\star; C, B^\star)$

$C = (c_1, \ldots, c_T) \leftarrow \textsc{Segment}(g, P_{\text{mono}}; P_{\text{seg}})$
for $t = 1$ **to** T **do**
  $s_t^{(0)} \sim g(c_t, P_{\text{mono}}; P_{\text{init}})$
end for
$B^{(0)} \leftarrow (s_1^{(0)}, \ldots, s_T^{(0)})$
for each $\mathbf{m} \in \{0,1\}^T$ **do**
  $\mathcal{T}_{\mathbf{m}}^{(0)} \leftarrow \textsc{Execute}(f, \pi(\mathbf{m}; C, B^{(0)}), \mathcal{D}_{\text{train}})$
  $(r_{\mathbf{m}}^{(0)}, \delta_{\mathbf{m}}^{(0)}) \leftarrow g(\mathcal{T}_{\mathbf{m}}^{(0)}; P_{\text{fb}})$
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**
      $\tilde{s}_i \sim g(i, \mathcal{T}, P_{\text{mono}}, \delta; P_{\text{refine}})$
      $\tilde{B} \leftarrow \textsc{Replace}(B, i, \tilde{s}_i)$
      $\tilde{\mathcal{T}} \leftarrow \textsc{Execute}(f, \pi(\mathbf{m}^\star; C, \tilde{B}), \mathcal{D}_{\text{train}})$
      $(\tilde{r}, \tilde{\delta}) \leftarrow g(\tilde{\mathcal{T}}; P_{\text{fb}})$
      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 $\pi(\mathbf{m}^\star; C, B^\star)$

during optimization, and the exposure is small, covering 50 of 5,882 memory inputs (0.85%), with 24 of 1,986 QA items (1.21%) answerable from those inputs.

A.5 Agent Tasks and Main-Task Segmentation

We apply **CAST** 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.

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 each agent specification, we obtain the main-task sequence through LLM-assisted segmentation of the original monolithic prompt. This segmentation preserves the original instruction content and does not introduce new task requirements. Table 13 shows an example decomposition for the A-mem memory-management specification.

A.6 Ablation on Decomposition Granularity

We examine how the granularity of main-task decomposition affects memory quality. We compare three variants of the A-mem memory-management

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 12: Agent tasks optimized by **CAST** in our experiments.

Main task	Description
c_1	Should this memory be evolved?
c_2	What specific actions should be taken, such as strengthening the connection or updating 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 13: Main-task segmentation of a monolithic agent prompt for A-mem memory management.

	$c_1c_2c_3c_4$	$c_1c_2,\ c_3c_4$	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 14: 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.

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 variant $(c_1c_2c_3c_4)$.

The fully decomposed variant is preferred over both grouped variants (Table 14), while the fully grouped variant performs worst. This trend indicates that even when scaffolding tasks are available, residual prompt complexity in grouped main tasks can still hinder SLM execution. Fine-grained main tasks provide clearer interfaces for scaffolding artifacts and enable more reliable memory decisions.

A.7 Optimization Details

All offline scaffold-placement search and instruction refinement are performed on a training set $\mathcal{D}_{\text{train}}$, defined as the first 50 memory inputs of LoCoMo. For a fixed mask $\mathbf{m}$, fixed main-task sequence C, and scaffolding-task bank B, the compiled procedure $\pi(\mathbf{m}; C, B)$ is executed on each $x \in \mathcal{D}_{\text{train}}$. The trajectory feedback prompt P_{fb} 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.

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

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

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 $\mathcal{D}_{\text{train}}$. After selecting the best mask, **CAST** refines only the activated scaffolding tasks while keeping the original main-task instructions unchanged.

A.8 Pairwise Evaluation 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 adherence to the original specification, faithfulness to the input, hallucination avoidance, logical consistency, and usefulness for future retrieval. The full pairwise evaluation prompt P_{pair} is shown in Table 18.

A.9 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 outputs produced by the original monolithic prompt with Qwen3-4B-Instruct, the same prompt with GPT-4o, and **CAST**.

As shown in Table 16, the monolithic prompting baseline often produces incomplete or overly narrow updates. In contrast, **CAST** generates more

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.
CAST (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 16: Qualitative comparison of memory-construction outputs under the same input. Qwen3-4B-Instruct under-connects the new memory to prior context, while **CAST** recovers broader support-related structure by linking multiple relevant neighbors. GPT-4o is more comprehensive but can include loosely related neighbors.

consistent and better-structured outputs by explicitly leveraging intermediate artifacts through scaffolded execution.

In LoCoMo conv-26, a turn (D4:3) reveals that Caroline's necklace is "a gift from my grandma . . . it stands for love, faith and strength," while memory already holds the sparse entry "Caroline has a necklace." The specification-correct operation is to UPDATE the existing entry; a model that instead ADDs a second overlapping entry leaves both in the store, which a memory-quality judge penalizes. Both associated QA items ("What was grandma's gift to Caroline?", "What does Caroline's necklace symbolize?") are nonetheless answered correctly from the newly added fact, illustrating why memory quality and downstream QA need not move together.

A.10 Judge Robustness

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

We additionally evaluate with a third judge, Claude Opus, alongside the GPT and Gemini-3-Pro judges. Under Opus, CAST attains a 45.70% non-tied win rate against GPT-4o versus 17.4% for the original monolithic prompt, reproducing the main trend. At the instance level both additional judges correlate strongly with the GPT judge (Pearson $r = 0.89$ for Gemini and $r = 0.87$ for Opus). Disagreements concentrate on instances where both candidates satisfy the action–field logic and differ only in tag specificity. We also note that GPT-4o serves as a strong reference point rather than an oracle of ground-truth memory: the comparison measures how close on-device memory comes to cloud-model quality, not whether GPT-4o is correct.

A.11 Additional Generalization Results

Smaller models in the same family. To evaluate generality beyond Qwen3-4B, we apply **CAST** to Qwen3-0.6B. Because this model has much lower capacity than GPT-4o, we compare **CAST** directly against the original Qwen3-0.6B pipeline rather than against the GPT-4o reference. Under this com-

parison, **CAST** is preferred in 81.72% of non-tied cases in the memory-quality evaluation. We also observe that lower-capacity models tend to benefit from more explicit scaffolding tasks: Qwen3-4B-Instruct uses two scaffolding tasks, whereas Qwen3-0.6B requires three in our setting. This trend is consistent with our main claim that the importance of externalized intermediate operations increases as model capacity decreases.

Other model families. To test whether **CAST**'s benefit is specific to Qwen3, we apply the identical pipeline—same planner, seed, and training set—separately to two SLMs from other families, Gemma (Google, 2024) and Phi (Microsoft, 2024). Besides the pairwise win rate against GPT-4o, we report a judge-free *logic-gate pass rate*: a rule-based check of whether the output satisfies the action–field truth table and ID-validation constraints stated in the original specification. On Gemma-2-9B the win rate rises from 5% to 34% and the logic-gate pass rate from 0% to 82%; on Phi-3.5-mini-3.8B the win rate rises from 15% to 31% and the pass rate from 10% to 26%. Under the monolithic prompt both models largely fail the specification's logic, and **CAST** recovers it. The two families fail differently: Gemma almost never satisfies the action–field logic, for instance selecting `strengthen` while filling neighbor-rewrite fields, whereas Phi hallucinates neighbor IDs that are absent from the input. Because the corrective scaffold differs per model, this is where trajectory feedback matters most: a fixed, hand-written scaffold could not adapt to which part of the specification's logic a given SLM violates.

Cross-benchmark transfer. The procedure optimized on LoCoMo is applied as-is to MemoryAgentBench in the main results, which is already a zero-shot transfer. We additionally verify the reverse direction: a procedure optimized on Ruler, the Wikipedia-style document subset of MemoryAgentBench, and applied without modification to LoCoMo (conversations) attains a 35.26% win rate against GPT-4o, well above the original monolithic prompt (14.58%). The scaffolds are thus not benchmark- or topic-specific; **CAST** does not learn new task semantics but reorganizes the given specification so that a small model executes it more reliably, so topical coverage remains a property of the source specification.

Method	Win Rate (%)	Tokens
Original Prompt	14.6	444
CoT	15.2	3459
Chain-of-Draft	13.3	720
TALE-EP	15.1	2076
CAST	40.6	1539

Table 17: **Efficient-reasoning baselines on A-mem LoCoMo.** We report pairwise win rate against the GPT-4o reference and average generated tokens.

A.12 Efficient-Reasoning Baselines

As shown in Table 17, we additionally compare **CAST** with efficient-reasoning baselines on A-mem LoCoMo. Besides standard CoT, we evaluate Chain-of-Draft and TALE-EP, which aim to reduce the cost of reasoning traces. All methods use Qwen3-4B-Instruct and follow the same memory-quality evaluation protocol as the main experiments.

Efficient-reasoning methods can reduce the verbosity of reasoning traces, but they still leave the choice and ordering of intermediate operations to the target SLM. In contrast, **CAST** fixes task-specific scaffolding artifacts after offline compilation and makes them directly reusable by later main tasks.

A.13 Preliminary Transfer to NaturalPlan Trip Planning

To examine whether scaffolded execution can apply beyond memory agents, we conduct a preliminary experiment on the Trip Planning task from NaturalPlan. Using Qwen3-4B-Instruct, the original monolithic prompt achieves an accuracy of 0.25, while applying a CAST-style scaffolded procedure improves accuracy to 0.31. This result suggests that externalizing implicit intermediate operations may also help other prompt-specified agentic tasks. We treat this as preliminary evidence, since the main focus of this work is memory-agent execution.

A.14 Supplementary Prompt Listings

To keep the main appendix narrative compact, we place the long-form prompt listings here. Tables 19-21 show the optimization-time prompt families. Tables 22-23 present the scaffolding tasks discovered by **CAST**. Tables 24-30 provide the optimized prompts produced by the prompt-optimization baselines.

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 18: Pairwise evaluation prompt used for memory-quality evaluation.

A.15 Human Evaluation Details

Annotators were shown no model identity, no rubric, and no reference judgment, and each pair was presented in randomized order. Instances were selected to be non-trivial: topic-bearing messages with five available neighbor memories. Out of 2,000 judgments per comparison, the Thinking baseline won 397, tied 230, and lost 1,373 against GPT-4o, while CAST won 730, tied 259, and lost 1,011. Excluding ties this corresponds to 22.4% and 41.9% respectively. Annotators were volunteers recruited internally, and participation was voluntary with informed consent; no personal data was collected and no annotator identity is recorded in the released data. Each annotator completed 80 judgments.

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 19: 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 20: 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 21: Feedback prompt P_{fb} used by prompt optimization baselines and by **CAST** during offline scaffold refinement.

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 22: 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 23: 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 24: 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 25: 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 26: 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 27: 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 28: 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 29: 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 30: The final prompt optimized by GEPA for A-mem memory management (continued).