MoMem: A Mixture-of-Memory Architecture for Balancing Factual Recall and Experiential Reuse in LLM Agents

Anonymous ACL submission

Abstract

As LLM agents move from short isolated interactions to sustained deployment, their effectiveness increasingly depends on memory that can retain important information, adapt to evolving users and environments, and make use of what has been learned from past interactions, but realistic workloads make it difficult to balance reliable fact preservation with the reuse of accumulated experience. To address this challenge, we propose **Mixture of Memory** (**MoMem**), a hybrid architecture that separates memory for facts from memory for reusable experience into textual and parametric pathways. During inference, a router decides which textual records to inject and whether to activate parametric memory using logic-matched trajectory evidence; after each interaction, a decoupler separates factual signals from reusable reasoning experience and updates the two pathways independently. Experiments show that, in the main mixed-workload setting, MoMem performs strongly on both sides of this balance, reaching 51.4 on LongMemEval, 11.7 points above the strongest baseline, and 86.4 on MATH-500, 3.2 points above the backbone model. Ablation studies show that these gains come from routing and decoupling rather than hybrid storage alone. Code is available at https://anonymous.4open.science/r/MoMem_a-01FE.

1 Introduction

Large language model (LLM) agents are increasingly expected to operate over long horizons, interact across tasks, and improve through continued use. In such settings, memory is not only a mechanism for extending context length, but also the basis for preserving prior knowledge, adapting to users, and accumulating experience over time. Realistic agent workloads place different demands on memory: some tasks require accurate retention of explicit facts such as user preferences, entity attributes, and environment states, while others require reuse of strategies for reasoning, planning, coding, and tool use. Following Hu et al. (2026), we refer to these two functional roles as *factual memory* and *experiential memory*. This creates a central challenge for memory design in realistic long-horizon settings. To provide a memory-level foundation for lifelong learning, a memory system must preserve factual precision while continuously accumulating reusable experience under mixed workloads.

However, most existing memory architectures focus on only one of these two aspects, making it difficult to support both functions effectively. One prominent line of work stores memory externally in textual form, often referred to as *explicit memory* or *external memory* (Zhang et al., 2025b; Huang et al., 2026). These systems excel at factual recall, preference tracking, and long-horizon consistency (Zhong et al., 2024; Packer et al., 2024; Chhikara et al., 2025; Xu et al., 2025; Li et al., 2025; Wang et al., 2024b; Zhang et al., 2025a; Fang et al., 2026), and can also support aspects of experiential memory that are representable as structured workflows or execution traces (Wang et al., 2024b; Zhang et al., 2025a; Fang et al., 2026). Yet they remain weaker on reasoning-intensive tasks such as mathematics, where useful signals are often logically rather than semantically related (Liu et al., 2025). When mismatched information is injected into long contexts, it can become noise rather than useful reasoning support (Liu et al., 2024; Zhou et al., 2025). Consequently, prompt-level reuse in external memory functions more like retrieving static past traces than dynamically synthesizing reusable experience (Wei et al., 2026).

Conversely, another direction stores memory directly within model parameters, commonly described as *implicit memory* or *internal memory* (Zhang et al., 2025b; Huang et al., 2026). This strategy is better suited to reasoning-dense experience that is difficult to externalize as compact text, because it can internalize reusable behavior

rather than replay prior traces through context (Yuksekgonul et al., 2026; Zuo et al., 2025; Wei et al., 2026). But it is a less reliable substrate for newly emerging facts. Prior work shows that retrieval is often preferable to fine-tuning for factual injection (Ovadia et al., 2024; Balaguer et al., 2024), while factual edits in parameter space remain constrained by the trade-off among reliability, generalization, and locality (Wang et al., 2024a). Under repeated updates, this difficulty is further amplified by interference between edits, especially for sparse long-tail knowledge (Nguyen et al., 2026; Yi et al., 2025).

More importantly, the challenge extends beyond the fact that no single mechanism is equally well-suited to both functions. Forcing factual and experiential memory into a single mechanism inherently couples how memory is invoked during inference and how it is updated post-interaction. In mixed workloads, explicit retrieval can return memories in which factual details and strategy traces are entangled, bloating the context and introducing low-relevance noise (Liu et al., 2024; Zhou et al., 2025). For parametric memory, if the same parameter space must absorb both factual details and transferable experience, updates become more frequent and more interfering, increasing cost and catastrophic forgetting (Kirkpatrick et al., 2017; Wang et al., 2024a; Luo et al., 2025). Taken together, these observations suggest that the problem is not only one of representational mismatch, but also one of interaction-level interference between different memory functions.

We address these challenges with **Mixture of Memory** (**MoMem**), a hybrid memory architecture designed to support factual and experiential memory in a more balanced manner under mixed workloads. To achieve this, MoMem incorporates three coordinated design principles. First, it allocates memories based on representational fit, preserving factual knowledge and verbalizable procedural experience within explicit memory while reserving implicit memory exclusively for hard-to-externalize reasoning capabilities. Second, it introduces a *memory router* that dynamically determines which explicit memories to retrieve and whether to activate the implicit memory for a given query. Third, it employs a *memory decoupler* that disentangles post-interaction feedback signals, ensuring that different types of information are updated through their respective specialized mechanisms.

Our main contributions are summarized as follows:

- We conceptualize agent memory in long-horizon, mixed-task settings as a *balance problem*. Rather than maximizing a single memory function in isolation, the objective is to simultaneously support both factual and experiential memory within a unified system while mitigating mutual interference.
- We introduce **MoMem**, a hybrid memory architecture that coordinates heterogeneous mechanisms through adaptive routing and decoupling. This design ensures that factual knowledge, verbalizable procedural traces, and hard-to-externalize reasoning capabilities are stored, invoked, and updated via their respective optimal pipelines.
- We comprehensively evaluate MoMem across factual-intensive, implicit-experience-dense, and multi-task mixed settings. Experimental results demonstrate that MoMem achieves a strong overall balance and stable performance under mixed workloads compared to single-mechanism baselines.
- Through ablation studies on the memory router and decoupler, we verify that MoMem's improvements do not stem from architectural hybridization alone, but from explicitly segregating how heterogeneous memory signals are selected and updated.

2 Related Work

Recent surveys distinguish agent memory by both function and form: factual and experiential memory describe what memory is used for, while token-level, parametric, and latent memory describe how it is represented (Hu et al., 2026). Following this distinction, we organize related work by substrate: *explicit* memory maintained outside the model and *implicit* memory absorbed into parameters or parameter-efficient modules (Huang et al., 2026; Zhang et al., 2025b).

2.1 Explicit Memory for LLM Agents

Explicit or external memory stores inspectable artifacts outside the model and retrieves them at inference time (Huang et al., 2026; Zhang et al., 2025b). This substrate is commonly used for factual memory, as in MemoryBank (Zhong et al.,

2024), MemGPT (Packer et al., 2024), Mem0 (Chhikara et al., 2025), A-Mem (Xu et al., 2025), and CAM (Li et al., 2025), where readability and editability support preference tracking, dialogue facts, and precise updates. It can also serve experiential functions when experience is verbalized into workflows, traces, cheatsheets, or reasoning memories, as in Agent Workflow Memory (Wang et al., 2024b), G-Memory (Zhang et al., 2025a), Memp (Fang et al., 2026), Dynamic Cheatsheet (Suzgun et al., 2025), and ReasoningBank (Ouyang et al., 2026). These systems remain bounded by retrieval and prompt injection: long, weakly relevant, or logically mismatched context can become noise rather than useful memory (Liu et al., 2024; Zhou et al., 2025; Liu et al., 2025; **?**).

2.2 Implicit Memory and Parametric Adaptation

Implicit or *internal* memory stores reusable information in model parameters, parameter-efficient adapters, or other non-textual internal representations (Huang et al., 2026; Zhang et al., 2025b). Test-time learning and parametric adaptation methods such as Learning to Discover at Test Time (Yuksekgonul et al., 2026) and TTRL (Zuo et al., 2025) update the model so useful behavior can be reused without replaying prior traces in the prompt, making this substrate well aligned with reasoning-dense experiential memory. Yet parametric memory is weaker for precise factual maintenance: retrieval often outperforms fine-tuning for factual injection (Ovadia et al., 2024; Balaguer et al., 2024), and repeated updates face locality, stability, interference, and forgetting challenges (Wang et al., 2024a; Nguyen et al., 2026; Yi et al., 2025; Kirkpatrick et al., 2017; Luo et al., 2025). MoMem builds on these complementary strengths by asking how factual and experiential demands can be routed and updated when they are interleaved in the same long-horizon workload.

3 Method

3.1 Overview

Figure 1 illustrates the overall architecture of MoMem. Given a user request, MoMem first retrieves textual candidates from the explicit memory module. For implicit memory, MoMem does not retrieve trajectories by raw semantic similarity. Instead, it first performs a lightweight pre-reasoning to judge whether the request exhibits nontrivial reasoning structure and is therefore likely to benefit from using parametric memory. In MoMem, the parametric memory is implemented by the parameters of LoRA adapter. Only in that case does it derive a structured logic sketch of the methods and rough reasoning steps likely to be needed, and retrieve logic-matched trajectory evidence from the Logic-Indexed Trajectory Bank. These retrieved results are passed to the memory router, which selects the explicit memories to inject into the prompt and makes the final decision about whether the current request should activate the parametric memory. The agent then generates a response using the routed memory configuration. After the interaction is completed, MoMem sends the full trajectory and feedback to the memory decoupler, which separates textual memory content from reasoning experience and writes them back through the corresponding memory modules.

We denote the completed interaction at round t as τ_t, which includes the request x_t, the response y_t, and any intermediate reasoning or tool-use traces when available. We denote the feedback signal used for memory update as u_t, which may be a verifier score, execution result, task-level reward, or other feedback provided by benchmark.

The two control points in this process are the router and the decoupler. The router operates before generation. It does not simply append all retrieved information to the context; instead, it filters explicit memories according to the current request and uses logic-matched trajectory evidence to judge whether the parametric memory is likely to help. The decoupler operates after generation. It determines which parts of the trajectory should become textual memory records and which parts should instead be internalized as reusable reasoning behavior.

3.2 Explicit Memory Module

The explicit memory module stores memory signals that can be represented and reused as text. In MoMem, this module contains two lightweight memory banks with complementary roles. The first bank stores textual memory records extracted from prior interactions. These records cover factual content and explicitizable experiential content while omitting much of the surrounding interaction noise. The second bank stores the corresponding raw interaction records. Raw records are less compact and may contain irrelevant context, so they are not always well matched by semantic re-

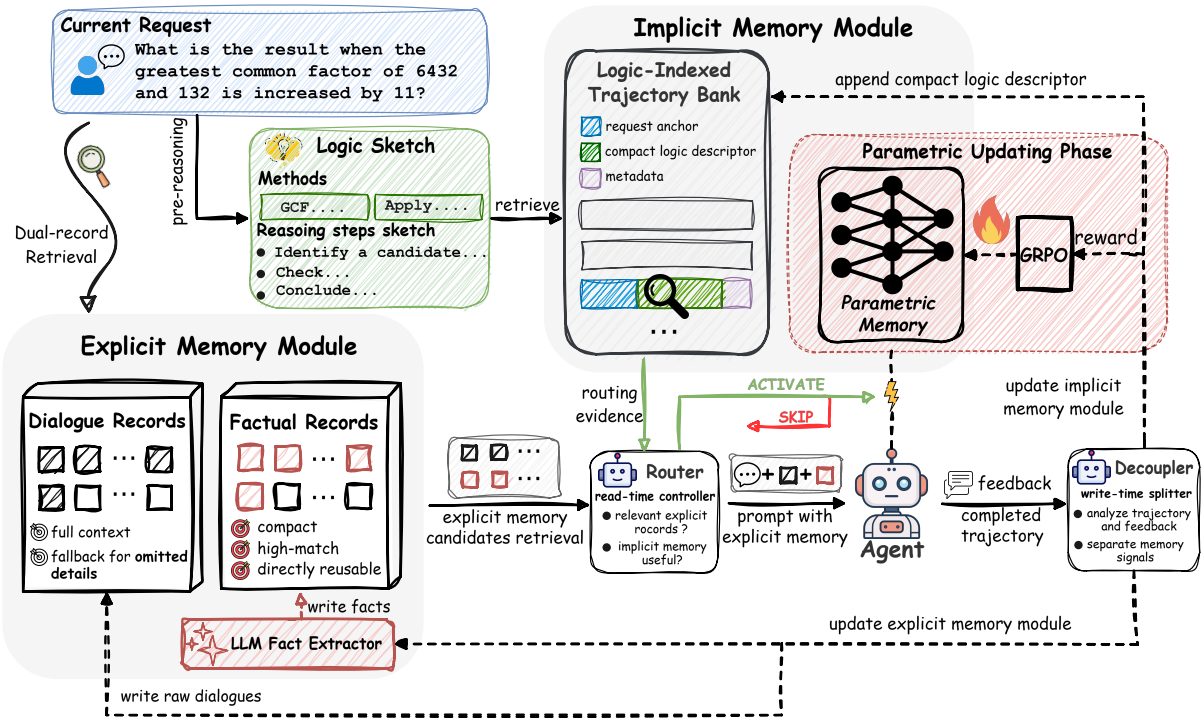

Figure 1: Overview of MoMem. During inference, the agent first derives a logic sketch from the current request. The router then retrieves explicit memory candidates from dialogue and factual records, obtains logic-matched routing evidence from the Logic-Indexed Trajectory Bank, and decides which explicit records to inject and whether to activate the parametric memory. The trajectory bank provides routing evidence rather than prompt content. After interaction, the decoupler analyzes the completed trajectory and feedback, separating explicit memory signals from implicit method experience. Explicit signals are written back through the dialogue/factual memory pathway, while implicit signals update the trajectory bank and the parametric memory.

trieval; however, they preserve information that may be lost during extraction and therefore serve as a completeness-preserving fallback. During retrieval, MoMem queries both banks and returns relevant textual memory records and relevant raw interaction records.

Formally, the explicit memory module is written as

$$\begin{aligned} \mathcal{M}_t^{\text{ext}} &= (\mathcal{V}_t, \mathcal{H}_t), \\ \mathcal{V}_t &= \{(v_i, \mu_i)\}_{i=1}^{N_t^V}, \\ \mathcal{H}_t &= \{(h_j, \mu_j)\}_{j=1}^{N_t^H}. \end{aligned} \tag{1}$$

Here, $\mathcal{V}_t$ is the textual memory bank and $\mathcal{H}_t$ is the raw interaction history bank. Each v_i is a textual record extracted from prior interactions, covering either factual content or explicitizable experiential content, and μ_i stores metadata for retrieval and management. Each h_j is a raw interaction record with metadata μ_j. This design makes the explicit module both retrievable and information-preserving: textual memory records improve matching and direct reuse, while raw interaction records protect against omissions introduced by the extraction step.

3.3 Implicit Memory Module

The implicit memory module targets reusable reasoning experience whose value is not reliably captured as a textual memory record. It uses LoRA weights Δ_t as parametric memory to internalize transferable reasoning behavior, and maintains a Logic-Indexed Trajectory Bank $\mathcal{P}_t$ to help decide when the parametric memory should be activated.

Rather than storing trajectories for prompt reuse, the bank stores a compact logic descriptor extracted from each trajectory selected by the memory decoupler for implicit memory update. This allows MoMem to retrieve matched logic descriptors as routing evidence, rather than relying on surface semantic similarity. In this sense, $\mathcal{P}_t$ is not a second explicit memory store for task solving; it is an index that helps identify when the parametric memory is likely to be useful.

Each entry in the Logic-Indexed Trajectory Bank

is represented as

$$p_i = (x_i, \ell_i, \mu_i), \qquad \ell_i = (m_i, s_i), \tag{2}$$

where x_i is the original request, m_i denotes the methods used in the trajectory, s_i denotes a rough sketch of the reasoning steps, and μ_i stores metadata such as time. Here, m_i records the high-level method or methods, while s_i preserves the coarse order of the reasoning steps without full detail. Together, ℓ_i denotes the logic descriptors that describes the reusable reasoning structure of the trajectory at a compact level.

3.4 Inference-Time Memory Routing

The memory router controls which textual memories are injected into the prompt and whether the parametric memory are used during generation. Given a new request x_t, MoMem first performs a lightweight pre-reasoning step:

$$q_t^R = \mathcal{A}_{\text{LLM}}(x_t), \tag{3}$$

where q_t^R is either a structured logic sketch aligned with the bank descriptor $\ell_i = (m_i, s_i)$, or $\varnothing$ when the current request does not exhibit enough reasoning structure. When produced, q_t^R summarizes the methods and rough reasoning steps required by the current request. This step does not solve the task itself.

MoMem then gathers two retrieval results:

$$\begin{aligned} \mathcal{C}_t^{\text{ext}} &= \text{Ret}_{\text{ext}}(x_t, \mathcal{M}_t^{\text{ext}}), \\ \mathcal{C}_t^{P} &= \begin{cases} \text{Ret}_P^{\text{logic}}(x_t, q_t^R, \mathcal{P}_t), & q_t^R \neq \varnothing, \\ \varnothing, & q_t^R = \varnothing. \end{cases} \end{aligned} \tag{4}$$

Here, Ret_{ext} retrieves explicit-memory candidates from both $\mathcal{V}_t$ and $\mathcal{H}_t$, with the former providing compact extracted records and the latter preserving details that extraction may omit. In contrast, $\text{Ret}_P^{\text{logic}}$ is invoked only when logic sketch is produced and not empty. In that case it matches q_t^R and request x_t with the entries $p_i = (x_i, \ell_i, \mu_i)$, stored in $\mathcal{P}_t$, so the resulting set $\mathcal{C}_t^P$ reflects similarity of reasoning sketch rather than raw semantic similarity between requests.

These two retrieval results play different roles: $\mathcal{C}_t^{\text{ext}}$ provides possible prompt context, while $\mathcal{C}_t^P$ provides evidence for the router to decide whether to activate the parametric memory rather than context to be directly injected into the prompt. MoMem uses an LLM-based router $\mathcal{R}_{\text{LLM}}$ to decide which explicit memories to inject and whether the parametric memory should be activated:

$$(\tilde{\mathcal{C}}_t^{\text{ext}}, z_t) = \mathcal{R}_{\text{LLM}}(x_t, \mathcal{C}_t^{\text{ext}}, \mathcal{C}_t^P), \tag{5}$$

where $\tilde{\mathcal{C}}_t^{\text{ext}} \subseteq \mathcal{C}_t^{\text{ext}}$ is the subset of selected explicit memories and $z_t \in \{\text{SKIP}, \text{ACTIVATE}\}$ is the router's activation decision for the parametric memory.

After routing, the generation context is constructed as

$$\text{Prompt}_t = [x_t; \tilde{\mathcal{C}}_t^{\text{ext}}], \tag{6}$$

and the response is generated by

$$y_t \sim \begin{cases} \pi_\theta(\cdot \mid \text{Prompt}_t), & z_t = \text{SKIP}, \\ \pi_{\theta,\Delta_t}(\cdot \mid \text{Prompt}_t), & z_t = \text{ACTIVATE}. \end{cases} \tag{7}$$

This design reduces context clutter by filtering explicit memories before prompt injection, while using logic-matched trajectory descriptors only as routing evidence for the parametric memory.

3.5 Post-Interaction Memory Decoupling

If routing determines how memory is used during inference, decoupling determines how memory is formed after interaction. Rather than writing the completed interaction into a unified memory pool, MoMem applies an LLM-based decoupler $\mathcal{D}_{\text{LLM}}$ to separate textual memory content from reasoning experience to be internalized into the parametric memory:

$$(E_t^{\text{ext}}, d_t^{\text{ext}}, E_t^{\text{int}}, d_t^{\text{int}}) = \mathcal{D}_{\text{LLM}}(\tau_t, u_t), \tag{8}$$

where E_t^{ext} denotes textual memory content to be written to the explicit module, while E_t^{int} denotes reasoning experience to be internalized into the parametric memory. The variables d_t^{ext} and d_t^{int} are the corresponding write decisions.

Given the interaction trajectory and feedback, the decoupler produces textual memory records, reasoning experience, and write-or-skip decisions for the two modules. This step is more than deciding whether a trajectory is worth keeping: a single interaction can contain both factual or explicitizable experiential signals and reasoning experience, and these signals should be written back through different mechanisms.

The decoupler therefore decouples these signals so that explicit memory module and implicit memory module can be updated separately:

$$\mathcal{M}_{t+1}^{\text{ext}} = \begin{cases} \mathcal{U}_{\text{ext}}(\mathcal{M}_t^{\text{ext}}, E_t^{\text{ext}}), & d_t^{\text{ext}} = \text{write}, \\ \mathcal{M}_t^{\text{ext}}, & \text{otherwise}, \end{cases} \tag{9}$$

$$(\Delta_{t+1}, \mathcal{P}_{t+1}) = \begin{cases} \mathcal{U}_{\text{int}}(\Delta_t, \mathcal{P}_t, E_t^{\text{int}}, \tau_t, u_t), & d_t^{\text{int}} = \text{write}, \\ (\Delta_t, \mathcal{P}_t), & \text{otherwise}. \end{cases} \tag{10}$$

The two update operators implement different write mechanisms. $\mathcal{U}_{\text{ext}}$ writes extracted textual memory records while also preserving the corresponding raw interaction records. $\mathcal{U}_{\text{int}}$ uses reasoning experience selected by the decoupler to update the LoRA weights and to append compact logic descriptors to the trajectory bank. Let the preserved samples for parametric memory up to time t be

$$\mathcal{B}_t = \{(E_i^{\text{int}}, \tau_i, u_i) \mid i \leq t,\ d_i^{\text{int}} = \text{write}\}, \quad (11)$$

In our implementation, the update of parametric memory is instantiated with GRPO:

$$\max_{\Delta} \mathbb{E}_{(E^{\text{int}},\tau,u)\sim\mathcal{B}_t}\left[J_{\text{GRPO}}(\theta, \Delta; E^{\text{int}}, \tau, u)\right]. \quad (12)$$

By decoupling these updates, MoMem avoids two common failure modes of single-substrate memory. It prevents factual and explicitizable experiential content from being inappropriately compressed into parameter space, and it prevents low-quality reasoning traces from being stored in the parametric memory as long-term memory. Thus, the decoupler gives MoMem differentiated update dynamics rather than merely combining two storage substrates.

4 Experiments

We evaluate MoMem as an online memory system for long-horizon agents. The experiments cover factual memory workloads, experiential memory workloads, mixed workloads that interleave the two under the same memory state, ablation studies, and analysis of routing and decoupling decisions.

4.1 Experimental Settings

Datasets. We use LongMemEval (Wu et al., 2024) and two factual settings from the official MemBench FirstAgent split (Tan et al., 2025), FirstAgent-Simple and FirstAgent-Conditional, to evaluate factual memory. These datasets require precise retention and retrieval of user- or interaction-specific information. To evaluate experiential reuse, we use MATH-500 (Hendrycks et al., 2021; Lightman et al., 2023), Omni-MATH-500[1], a subset of Omni-MATH (Gao et al., 2024), BigCodeBench (Zhuo et al., 2025), and AMC23 (Mathematical Association of America, 2025)[2] under the same online protocol. For compact presentation in tables, we abbreviate FirstAgent-Simple as MB-S, FirstAgent-Conditional as MB-C, and BigCodeBench as BigCode.

Online protocol. All methods are evaluated as online memory systems. At round t, the agent receives the current request, may read from memory, generates an answer, receives feedback when the benchmark provides it, and updates memory before round $t+1$. Memory is never reset within a stream. This protocol makes the evaluation about memory dynamics under continued use rather than one-shot retrieval with a fixed memory snapshot.

Mixed workloads. To study interference between memory demands, we construct mixed online streams by interleaving a target dataset with datasets from the other group while keeping a single evolving memory state. For each factual target, we mix its stream with requests from MATH-500, Omni-MATH-500, BigCodeBench, and AMC23. For each experiential target, we mix its stream with LongMemEval, MB-S, and MB-C. Scores are computed only on the target dataset instances, but all interleaved non-target interactions still update the same memory state. Therefore, the gap of performance between single workload and mixed workload reflects the effect of interactions on the same memory process.

Evaluation metrics. Except for LongMemEval, each dataset has a unique reference answer or executable test, so we extract the agent's final answer and compare it with the gold answer. For LongMemEval, where valid responses can be phrased differently, we use GPT-5.4 as a judge to determine whether the agent's answer is consistent with the reference answer. For every benchmark, we sample five independent outputs per item and report **pass@1**, **pass@5**, and **5-run accuracy**. Here, pass@k counts an item as correct if at least one of k sampled attempts is judged correct, while 5-run accuracy is the mean accuracy over five independently sampled runs. In tables, p@1 and p@5 denote pass@1 and pass@5.

Compared methods. We compare MoMem with eight mixed-workload baselines: Vanilla, SimpleMem (Liu et al., 2026), Mem0 (Chhikara et al., 2025), A-Mem (Xu et al., 2025), Agent Workflow Memory (AWM) (Wang et al., 2024b), Dynamic Cheatsheet (DC) (Suzgun et al., 2025), ReasoningBank (Ouyang et al., 2026), and SFT+GRPO. Vanilla is the basic model without memory system. SimpleMem, Mem0, and A-Mem represent

[1] https://huggingface.co/datasets/reliable-agents/Omni-MATH-500

[2] We use the Hugging Face evaluation split: https://huggingface.co/datasets/math-ai/amc23.

Method	LongMemEval	MB-S	MB-C	MATH-500		Omni-MATH-500		BigCode		AMC23	
				Acc.	Δ	Acc.	Δ	Acc.	Δ	Acc.	Δ
Vanilla	–	–	–	83.2	0.0	48.8	0.0	48.8	0.0	67.5	0.0
SimpleMem	39.7	86.9	71.5	82.3	-0.7	46.3	-2.2	45.3	+0.9	64.5	-1.5
Mem0	35.7	85.6	72.6	85.2	+0.8	49.1	+0.5	47.9	+2.0	65.5	-7.0
A-Mem	33.9	90.3	67.1	84.2	+0.5	48.5	-1.3	41.3	-0.9	66.5	-5.0
AWM	31.0	80.6	47.4	83.3	-1.4	48.1	-1.8	41.9	-0.3	71.0	-1.0
DC	38.0	**93.5**	63.1	82.8	-2.7	48.0	-1.1	41.1	+0.6	73.0	-1.5
ReasoningBank	32.9	82.5	21.4	82.8	-0.9	48.4	-0.4	40.8	+3.5	69.5	+0.5
SFT+GRPO	9.9	40.4	34.3	86.0	-0.1	48.6	-1.9	35.2	-14.0	71.0	-1.0
MoMem	**51.4**	90.0	**78.1**	**86.4**	-0.1	**49.9**	-0.4	**49.0**	+0.1	**73.5**	+0.5

Table 1: Main mixed-workload results. Acc. reports 5-run accuracy. Δ is mixed-workload 5-run accuracy minus the corresponding single-workload 5-run accuracy; red negative values indicate degradation under mixing, while green positive values indicate improvement. Full pass@1/pass@5/5-run results are in Appendix A.

explicit factual memory systems; AWM, DC, and ReasoningBank represent explicit experience or trace reuse. For DC, we use its CumulativeRetrieval variant. For mixed workloads, supervised fine-tuning (SFT) and Group Relative Policy Optimization (GRPO) are reported as one parametric baseline: SFT handles explicit supervision, while GRPO updates the implicit experiential component. Reporting them separately would cover only one side of the mixed stream.

4.2 Main Results

Mixed workload evaluation. Table 1 gives the main comparison under mixed workloads. We report 5-run accuracy in the main text and leave the full pass@1/pass@5/5-run tables to Appendix A. The results show two main patterns. First, MoMem performs well across the two memory functions rather than specializing in only one side. It is the strongest method on LongMemEval and MB-C among the factual memory targets, while also obtaining the best 5-run accuracy on all four experiential memory targets. Second, this performance remains stable when factual and experiential interactions share a single evolving memory state: MoMem shows near-zero degradation on the experiential targets relative to the corresponding isolated workloads. These results indicate that MoMem can preserve strong cross-function memory performance under mixed workloads, rather than improving one memory function at the cost of the other.

The comparison also clarifies where the mixed setting differs from isolated evaluation. Explicit memory baselines remain competitive on factual targets, but they do not consistently transfer to reasoning-dense experiential targets once the stream also contains factual updates. Conversely, the parametric SFT+GRPO baseline improves some experiential targets but is weak on factual memory, especially when user- or interaction-specific facts must remain editable. MoMem's advantage is therefore not only a higher score on individual targets, but a more balanced behavior across targets that place different demands on memory.

4.3 Ablation Study

The ablation study separates hybrid storage from the control mechanisms that make the hybrid useful. We focus on mixed workloads whose targets are MATH-500, Omni-MATH-500, BigCodeBench, and AMC23 after interleaving with factual-memory streams, because this is where inappropriate factual injection and incorrect activation of parametric memory are most likely to hurt reasoning reuse. *w/o Router* removes query-dependent invocation, so explicit memories and activation of parametric memory are no longer selected according to the current request. *w/o Decoupler* keeps the two substrates but removes post-interaction separation, so textual records and reasoning experience are written back through less differentiated update paths. *w/o Router & Decoupler* tests whether merely having both substrates is sufficient.

Figure 2 shows the effect of removing each control component. A drop without the router suggests

that query-dependent memory invocation is important. A drop without the decoupler suggests that separating textual records from reasoning experience during update is also important. The combined ablation tests whether hybrid storage alone is enough.

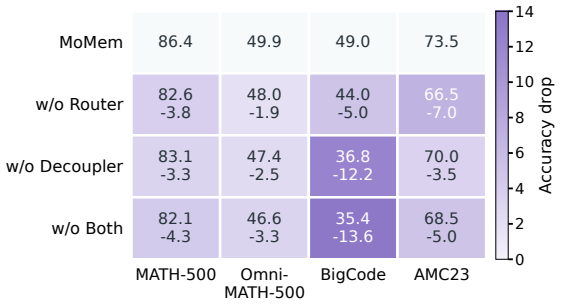

Figure 2: Mixed-workload ablation on reasoning-dense experiential benchmarks. Cells report 5-run accuracy; the second line in each ablated row reports the drop from MoMem, with darker shading indicating a larger drop.

These drops are largest when the task requires reusable reasoning rather than direct factual recall, which is consistent with MoMem's design. Without the router, implicit memory may be skipped when it is useful or activated when the retrieved evidence is not logic-matched. Without the decoupler, trajectories that should become textual records and trajectories that should become implicit method experience are no longer cleanly separated. The ablation results therefore support the claim that routing and decoupling are functional parts of the memory design, not only implementation details.

4.4 Decoupling and Routing Analysis

We further inspect the two control decisions made by MoMem. The decoupler acts after each interaction and assigns the update signal to one of four outcomes: *Explicit*, *Implicit*, *Both*, or *Neither*. Explicit means that only textual memory is updated; Implicit means that only the implicit memory receives the update; Both means that the interaction contains separable signals for both memory pathways; Neither means that the interaction is not written as persistent memory.

Figure 3 summarizes the update and routing decisions. Factual-memory datasets are dominated by explicit updates, while reasoning-dense experiential datasets are dominated by implicit updates. Mixed update signals are rare but appear in LongMemEval, indicating that a single interaction can still contain separable factual and experiential content. The routing panel shows the complementary inference-time pattern: parametric memory is activated for most reasoning-dense queries when logic-matched evidence is available.

This analysis connects the empirical gains to the intended memory assignment. The decoupler usually sends factual streams to explicit memory and reasoning-dense streams to implicit memory, while still allowing both pathways when an interaction contains separable signals. The router then applies the same separation during inference by activating implicit memory mainly when the current request has matching reasoning structure. Together, these decisions reduce unnecessary cross-use between memory pathways.

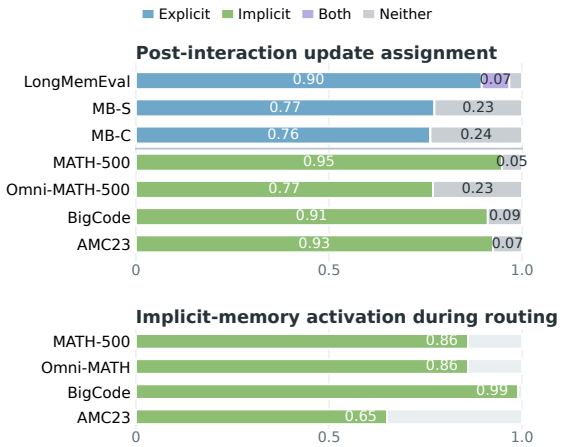

Figure 3: Decoupling and routing analysis. The upper panel reports post-interaction update assignment by the memory decoupler; the lower panel reports activation of parametrc memory during routing on reasoning-dense experiential datasets.

5 Conclusion

We presented MoMem, a hybrid memory architecture for long-horizons LLM agents that decouples factual memory and experiential memory across storage, invocation, and update. MoMem combines explicit records, parametric memory indexed by a Logic-Indexed Trajectory Bank, and a memory router and decoupler that control activation and update. Experiments show that MoMem performs well across factual and experiential memory and remains stable under mixed workloads with a single evolving memory state. Ablations show that the gains come from routing and decoupling, not hybrid storage alone, suggesting that memory works better when different memory functions are handled by separate mechanisms rather than a single growing memory store.

Limitations

MoMem introduces additional system complexity relative to single-substrate memory systems. It requires router and decoupler calls, explicit-memory management, trajectory indexing, and parametric memory updates, which may increase latency and maintenance cost in deployment. Its effectiveness also depends on the quality of the LLM-based routing and decoupling decisions, as well as the feedback signal used to select implicit-memory updates. The implicit module may still face stability and forgetting issues under long continual update sequences. Finally, our mixed workloads are controlled benchmark proxies rather than full real-world deployments. Different interleaving schedules, task ratios, feedback sources, or longer interaction horizons may change the observed trade-offs, so future work should test MoMem under more open-ended agent environments.

References

Angels Balaguer, Vinamra Benara, Renato Luiz de Freitas Cunha, Roberto de M. Estevão Filho, Todd Hendry, Daniel Holstein, Jennifer Marsman, Nick Mecklenburg, Sara Malvar, Leonardo O. Nunes, Rafael Padilha, Morris Sharp, Bruno Silva, Swati Sharma, Vijay Aski, and Ranveer Chandra. 2024. Rag vs fine-tuning: Pipelines, tradeoffs, and a case study on agriculture. *Preprint*, arXiv:2401.08406.

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

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2026. Memp: Exploring agent procedural memory. *Preprint*, arXiv:2508.06433.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. *Preprint*, arXiv:2410.07985.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *Preprint*, arXiv:2103.03874.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, and 28 others. 2026. Memory in the age of ai agents. *Preprint*, arXiv:2512.13564.

Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, Shanglin Wu, Ruiyao Xu, Liangwei Yang, Rui Yang, Wooseong Yang, Chin-Yuan Yeh, Hanrong Zhang, Haozhen Zhang, Siqi Zhu, and 41 others. 2026. Rethinking memory mechanisms of foundation agents in the second half: A survey. *Preprint*, arXiv:2602.06052.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, and 1 others. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Rui Li, Zeyu Zhang, Xiaohe Bo, Zihang Tian, Xu Chen, Quanyu Dai, Zhenhua Dong, and Ruiming Tang. 2025. Cam: A constructivist view of agentic memory for llm-based reading comprehension. In *Advances in Neural Information Processing Systems*, volume 38, pages 113381–113406. Curran Associates, Inc.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

Hao Liu, Zhengren Wang, Xi Chen, Zhiyu Li, Feiyu Xiong, Qinhan Yu, and Wentao Zhang. 2025. HopRAG: Multi-hop reasoning for logic-aware retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 1897–1913, Vienna, Austria. Association for Computational Linguistics.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. 2026. Simplemem: Efficient lifelong memory for llm agents. *Preprint*, arXiv:2601.02553.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2025. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *Preprint*, arXiv:2308.08747.

Mathematical Association of America. 2025. American mathematics competitions. `https://maa.org/student-programs/amc/`. Official website, accessed 2026-05-26.

Duy Nguyen, Hanqi Xiao, Archiki Prasad, Elias Stengel-Eskin, Hyunji Lee, and Mohit Bansal. 2026. Conflict-resolving and sharpness-aware minimization for generalized knowledge editing with multiple updates. *Preprint*, arXiv:2602.03696.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. 2026. Reasoningbank: Scaling agent self-evolving with reasoning memory. *Preprint*, arXiv:2509.25140.

Oded Ovadia, Menachem Brief, Moshik Mishaeli, and Oren Elisha. 2024. Fine-tuning or retrieval? comparing knowledge injection in LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 237–250, Miami, Florida, USA. Association for Computational Linguistics.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Memgpt: Towards llms as operating systems. *Preprint*, arXiv:2310.08560.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. 2025. Dynamic cheatsheet: Test-time learning with adaptive memory. *Preprint*, arXiv:2504.07952.

Haoran Tan, Zeyu Zhang, Chen Ma, Xu Chen, Quanyu Dai, and Zhenhua Dong. 2025. MemBench: Towards more comprehensive evaluation on the memory of LLM-based agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19336–19352, Vienna, Austria. Association for Computational Linguistics.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2024a. Wise: Rethinking the knowledge memory for lifelong model editing of large language models. *Advances in Neural Information Processing Systems*, 37:53764–53797.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024b. Agent workflow memory. *Preprint*, arXiv:2409.07429.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. 2026. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *Preprint*, arXiv:2511.20857.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2024. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *Adaptive Foundation Models: Evolving AI for Personalized and Efficient Learning*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. In *Advances in Neural Information Processing Systems*, volume 38, pages 17577–17604. Curran Associates, Inc.

Xinhao Yi, Jake Lever, Kevin Bryson, and Zaiqiao Meng. 2025. Can we edit LLMs for long-tail biomedical knowledge? In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 25662–25679, Suzhou, China. Association for Computational Linguistics.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. 2026. Learning to discover at test time. *Preprint*, arXiv:2601.16175.

Guibin Zhang, Muxin Fu, Kun Wang, Frank Wan, Miao Yu, and Shuicheng Yan. 2025a. G-memory: Tracing hierarchical memory for multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 38, pages 12988–13018. Curran Associates, Inc.

Zeyu Zhang, Yang Zhang, Haoran Tan, Rui Li, and Xu Chen. 2025b. Explicit v.s. implicit memory: Exploring multi-hop complex reasoning over personalized information. *Preprint*, arXiv:2508.13250.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, 17, pages 19724–19731.

Yang Zhou, Hongyi Liu, Zhuoming Chen, Yuandong Tian, and Beidi Chen. 2025. GSM-∞: How do your LLMs behave over infinitely increasing reasoning complexity and context length? In *Forty-second International Conference on Machine Learning*.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, and 14 others. 2025. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *Preprint*, arXiv:2406.15877.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. 2025. Ttrl: Test-time reinforcement learning. *Preprint*, arXiv:2504.16084.

A Full Results

Method	LongMemEval			MB-S			MB-C			MATH-500			Omni-MATH-500			BigCode			AMC23		
	p@1	p@5	5-run	p@1	p@5	5-run	p@1	p@5	5-run	p@1	p@5	5-run	p@1	p@5	5-run	p@1	p@5	5-run	p@1	p@5	5-run
Vanilla	–	–	–	–	–	–	–	–	–	81.8	90.9	83.2	49.4	60.8	48.8	48.9	55.0	48.8	75.0	85.0	67.5
SimpleMem	39.8	44.0	40.2	87.3	87.6	87.1	71.5	72.1	71.5	83.9	90.6	83.0	48.0	59.8	48.5	44.6	51.1	44.4	70.0	90.0	66.0
Mem0	36.6	38.6	35.8	85.6	86.6	85.6	72.4	73.5	72.7	84.5	91.5	84.4	48.0	63.3	48.6	46.4	52.5	45.9	75.0	90.0	72.5
A-Mem	33.2	37.4	34.0	90.4	90.9	90.5	66.9	67.9	67.1	84.2	91.5	83.7	51.2	62.5	49.8	41.8	48.2	42.2	75.0	85.0	71.5
AWM	30.8	32.6	30.7	80.4	81.9	80.9	47.3	47.6	47.3	87.6	93.6	84.7	48.8	64.8	49.9	42.5	49.5	42.2	67.5	85.0	72.0
DC	38.8	46.0	38.3	93.1	95.3	93.3	61.4	72.4	62.1	85.8	92.7	85.5	48.8	61.7	49.1	39.8	52.2	40.5	72.5	90.0	74.5
ReasoningBank	34.0	37.8	34.2	82.4	83.1	82.6	20.9	21.4	21.1	83.9	91.8	83.7	47.7	61.3	48.8	42.2	50.1	37.3	70.0	85.0	69.0
SFT	10.0	20.0	10.3	42.9	64.0	43.6	36.6	62.6	37.7	–	–	–	–	–	–	–	–	–	–	–	–
GRPO	–	–	–	–	–	–	–	–	–	86.1	92.7	86.1	47.7	60.6	50.5	48.9	53.3	49.2	65.0	90.0	72.0
MoMem	51.0	53.8	50.2	90.4	90.9	90.0	78.0	78.8	77.8	86.9	93.2	86.5	48.9	61.8	50.3	48.7	56.1	48.9	70.0	85.0	73.0

Table 2: Full isolated single-workload results on factual and experiential benchmarks.

Method	LongMemEval			MB-S			MB-C			MATH-500			Omni-MATH-500			BigCode			AMC23		
	p@1	p@5	5-run	p@1	p@5	5-run	p@1	p@5	5-run	p@1	p@5	5-run	p@1	p@5	5-run	p@1	p@5	5-run	p@1	p@5	5-run
Vanilla	–	–	–	–	–	–	–	–	–	81.8	90.9	83.2	49.4	60.8	48.8	48.9	55.0	48.8	75.0	85.0	67.5
SimpleMem	39.0	43.0	39.7	86.6	87.3	86.9	71.5	72.0	71.5	81.8	90.6	82.3	45.7	57.0	46.3	44.5	51.6	45.3	65.0	80.0	64.5
Mem0	36.2	38.0	35.7	85.6	86.2	85.6	72.7	73.3	72.6	87.0	91.8	85.2	50.4	59.4	49.1	47.6	54.5	47.9	65.0	80.0	65.5
A-Mem	34.2	36.8	33.9	90.5	90.8	90.3	67.2	67.9	67.1	84.5	91.2	84.2	51.6	62.9	48.5	41.6	46.8	41.3	72.5	85.0	66.5
AWM	31.2	32.0	31.0	80.7	81.5	80.6	47.4	47.9	47.4	84.8	91.8	83.3	48.4	61.3	48.1	42.7	48.9	41.9	75.0	90.0	71.0
DC	38.0	45.8	38.5	93.6	95.4	93.5	62.6	74.0	63.1	83.9	90.6	82.8	48.8	62.1	48.0	41.4	53.9	41.1	72.5	90.0	73.0
ReasoningBank	32.6	37.0	32.9	82.3	83.0	82.5	21.5	21.8	21.4	81.8	91.8	82.8	49.2	60.2	48.4	40.8	47.6	40.8	65.0	82.5	69.5
SFT+GRPO	10.4	22.2	9.9	40.6	68.8	40.4	37.0	63.7	34.3	87.0	94.2	86.0	47.7	59.4	48.6	35.4	45.6	35.2	70.0	90.0	71.0
MoMem	51.1	53.4	51.4	90.1	90.6	90.0	78.1	78.8	78.1	87.6	93.6	86.4	48.8	60.5	49.9	48.7	55.6	49.0	70.0	85.0	73.5

Table 3: Full mixed-workload results after interleaving factual and experiential tasks under the same evolving memory state. SFT+GRPO combines supervised fine-tuning with GRPO.

This appendix provides the complete metrics behind Table 1. For each benchmark, we report pass@1, pass@5, and 5-run accuracy. Table 2 gives the isolated single-workload setting and serves as the reference for the delta values in the main table. Table 3 gives the corresponding mixed-workload setting, where factual and reasoning-oriented interactions are interleaved under one evolving memory state. Thus, negative deltas in Table 1 indicate drops from the isolated scores, while positive deltas indicate improvements under mixing.

The full tables are included to make three comparisons auditable. First, they show whether a method is strong in isolation before heterogeneous interactions are introduced. Second, they show whether the same method remains stable after factual and experiential streams share one memory state. Third, pass@1 and pass@5 expose whether an apparent gain is robust across samples or depends mainly on one successful attempt among five. This distinction is useful for memory systems because mixed workloads can change both average accuracy and sampling stability.

The two tables are separated because the mixed setting changes the memory state throughout evaluation. A method may preserve its isolated score on one benchmark while degrading on another after heterogeneous interactions have been interleaved. We keep pass@1 and pass@5 here to expose sampling sensitivity; the main paper uses 5-run accuracy and the difference from Table 2 as compact indicators of mixed-workload robustness.

For factual targets, the most relevant question is whether reasoning-heavy interactions pollute or distract explicit recall. For experiential targets, the corresponding question is whether factual-memory updates weaken reusable reasoning behavior. MoMem is intended to reduce both types of interference: explicit records remain available for factual recall, while reasoning experience is routed through the implicit pathway only when the current request matches reusable method evidence.

B Logic Sketch Examples

Figures 5–6 show complete examples of the logic sketches extracted for implicit memory. Each example keeps the original request, the reusable method, and a coarse reasoning-step sketch. These cases are written to implicit memory rather than explicit memory because they contain transferable handling patterns, not durable user facts or external state that

should be injected verbatim into later prompts.

The examples are meant to clarify what is stored in the Logic-Indexed Trajectory Bank. A stored sketch is not a full solution trace and is not injected into the prompt as an answer. Instead, it records the handling structure that may later support routing: the high-level method, the order of important reasoning moves, and any checks that prevent a superficially similar but incorrect reuse. This keeps the bank useful for identifying when the LoRA-based memory should be activated while avoiding a second long-context retrieval store.

The four examples cover different forms of experience. AMC23 illustrates algebraic transformation over graph constraints; BigCodeBench illustrates an implementation workflow; MATH-500 shows that even a short arithmetic task can contain a reusable verification habit; Omni-MATH-500 shows a longer geometry construction with candidate checking. Together, they demonstrate that implicit memory is selected by transferability of handling logic rather than by answer length, task domain, or surface similarity alone.

Source	Explicit	Implicit	Reusable signal
AMC23	False	True	Algebraic handling of midpoint constraints on a function graph.
BigCodeBench	False	True	DataFrame construction followed by a requested visualization workflow.
MATH-500	False	True	Short arithmetic procedure with an explicit coprimality check.
Omni-MATH-500	False	True	Coordinate-geometry construction with candidate-root verification.

Table 4: Decoupler decisions and reusable signals for the logic-sketch examples.

C Prompt Templates

Figures 8–12 show the verbatim English prompt templates used by the router and decoupler. The router prompts decide whether to construct an experience query, which explicit records can be injected, and whether retrieved implicit evidence should activate the LoRA-based memory. The decoupler prompts decide whether a completed interaction should be written as a directly reusable text record, an implicit method experience, both, or neither.

The prompt templates are grouped by control stage. The pre-reason prompt asks for an abstract experience query, not for a task answer, so that retrieval keys describe reusable handling logic without encoding problem-specific conclusions. The explicit-memory filter then handles only directly injectable records, such as user facts, constraints, definitions, and compact workflows. The implicit-activation prompt uses logic-matched evidence only as an activation signal for parametric memory, not as context to be copied into the answer.

The two decoupler prompts enforce the complementary write policy after an interaction is completed. The explicit branch is conservative: it accepts stable background context that can safely appear in future prompts and rejects one-off reasoning products. The implicit branch stores reusable handling methods, including decomposition, verification, repair, and negative lessons, while rejecting durable facts that belong in explicit memory. This separation is what connects the prompt templates to the decoupling analysis in the main paper.

Prompt	Stage	Output role
Pre-reason	Before retrieval	Builds an abstract experience query without answering the task.
Explicit filter	During routing	Selects directly reusable text records for context injection.
Implicit activation	During routing	Decides whether retrieved method evidence should activate implicit memory.
Explicit truth	After interaction	Saves stable background context as explicit memory.
Implicit truth	After interaction	Saves transferable handling methods as implicit experience.

Table 5: Operational roles of the prompt templates shown in Figures 8–12.

MATH-500

GCF arithmetic check

Request

```
What is the result when the greatest common factor of 6432 and 132 is increased by 11?
```

Methods used

```
- Compute GCF by factoring out a common divisor and checking remaining coprimality
- then apply the requested arithmetic adjustment (+11).
```

Rough reasoning-step sketch

```
1. Identify a candidate common divisor shared by both numbers and divide both numbers by
it.
2. Check whether the reduced pair has any additional common factor.
3. Conclude the divisor is the GCF if no further common factor exists.
4. Add 11 to the GCF and report.
```

Figure 4: Complete logic-sketch example from MATH-500.

AMC23

Function-graph midpoint

Request

```
Points $A$ and $B$ lie on the graph of $y=\log_{2}x$. The midpoint of $\overline{AB}$ is
$(6, 2)$. What is the positive difference between the $x$-coordinates of $A$ and $B$?
The final answer can be written in the form $m \sqrt{n}$, where $m$ and $n$ are
relatively prime positive integers. What is $m+n$?
```

Methods used

```
- Parameterize points on a function as \((x,f(x))\); apply midpoint formula to get equations
on x-values and y-values;
- combine log terms using product rule to convert an average-of-logs constraint into a
product constraint;
- reduce to solving for two numbers given sum and product via a quadratic;
compute coordinate difference using root separation (or directly from roots), simplify
radical form.
```

Rough reasoning-step sketch

```
1. Represent the unknown points as \((a,\log_2 a)\), \((b,\log_2 b)\).
2. Apply midpoint formula to set \((a+b)/2\) and \((\log_2 a+\log_2 b)/2\) equal to the
given midpoint
coordinates.
3. Convert the log equation to \(\log_2(ab)=\text{constant}\) to obtain \
(ab=\text{value}\).
4. With sum \(a+b\) and product \(ab\), form the quadratic \(x^2-(a+b)x+ab=0\).
5. Solve via discriminant to get \(a,b\) and take the positive difference \(|a-b|\);
express in \(m\sqrt n\)
form and compute \(m+n\).
```

Figure 5: Complete logic-sketch example from AMC23.

Omni-MATH-500

Coordinate-geometry construction

Request

```
In right triangle $A B C$, a point $D$ is on hypotenuse $A C$ such that $B D \perp A C$.
Let $\omega$ be a circle with center $O$, passing through $C$ and $D$ and tangent to
line $A B$ at a point other than $B$. Point $X$ is chosen on $B C$ such that $A X \perp
B O$. If $A B=2$ and $B C=5$, then $B X$ can be expressed as $\frac{a}{b}$ for
relatively prime positive integers $a$ and $b$. Compute $100 a+b$.
```

Methods used

```
- Choose coordinates aligned with the right triangle to simplify equations.
- Compute the altitude foot \(D\) by intersecting the hypotenuse with the perpendicular
through \(B\).
- Use tangency to a line to parametrize the circle center as having a fixed distance to
that line.
- Enforce equal radii through two known points on the circle, solve the resulting system,
and check roots.
- Use the perpendicularity condition \(AX \perp BO\) to write the line through \(A\), then
intersect it with
\(BC\) to get \(X\).
- Verify candidate solutions exactly to eliminate any extraneous root.
```

Rough reasoning-step sketch

```
1. Set \(B=(0,0)\), \(A=(0,2)\), \(C=(5,0)\) so the triangle becomes algebra-friendly.
2. Find \(D\) as the foot from \(B\) to \(AC\) using line intersection.
3. Let the circle center be \((r,y)\) because tangency to \(AB\) fixes the center's
horizontal relation to
the tangent line.
4. Write equations from \(OC=OD\) and \(OC=r\), solve the quadratic for \(r\), and check
which root truly
satisfies the circle conditions.
5. Use the resulting center \(O\) to get the slope of \(BO\), draw the perpendicular
through \(A\),
intersect with \(BC\), and read off \(BX\).
```

Figure 6: Complete logic-sketch example from Omni-MATH-500.

BigCodeBench

DataFrame-to-plot workflow

Request

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


def task_func(list_of_pairs):
"""
Create a Pandas DataFrame from a list of pairs and visualize the data using a bar
chart.
- The title of the barplot should be set to 'Category vs Value'`.

Parameters:
list_of_pairs (list of tuple): Each tuple contains:
- str: Category name.
- int: Associated value.

Returns:
tuple:
- DataFrame: A pandas DataFrame with columns 'Category' and 'Value'.
- Axes: A matplotlib Axes displaying a bar chart of categories vs. values.

Requirements:
- pandas
- matplotlib.pyplot
- seaborn

Example:
>>> list_of_pairs = [('Fruits', 5), ('Vegetables', 9)]
>>> df, ax = task_func(list_of_pairs)
>>> print(df)
Category Value
0 Fruits 5
1 Vegetables 9
"""
```

Methods used

```
- Convert a list of `(category, value)` tuples into a `pandas.DataFrame` with explicit column
names.
- Use `seaborn.barplot` with the DataFrame's category column on `x` and numeric column on `y`.
- Set the plot title before returning both the DataFrame and the Axes object.
```

Rough reasoning-step sketch

```
1. Read the requested output types and required libraries.
2. Infer that each tuple maps naturally to `Category` and `Value` columns.
3. Build the DataFrame from the raw list, then create a bar chart from those columns.
4. Add the required title and return the data object and visualization handle together.
```

Figure 7: Complete logic-sketch example from BigCodeBench.

Router Prompt: Pre-Reason

```
Task: you will see a user request. Decide whether an experience query should be constructed; if so,
write a very brief experience-query sketch. Do not answer the request itself.

Here, "experience-query sketch" only means writing an abstract description of the handling logic the
current request may need, so it can later be compared with method summaries from past tasks. The
sketch must describe how to handle this kind of problem, without including the current answer,
concrete numbers, problem-specific details, or intermediate computation.

Focus on whether the request can yield reusable handling logic, implementation order, check points, or
negative lessons. Do not judge by response length, number of steps, or surface complexity; a short
answer, direct implementation, or basic calculation may still contain reusable experience. Even if the
handling is lightweight, write a sketch when it could remind a future task how to proceed, what to
check first, or what mistake to avoid.

Do not write a sketch when the request is only factual lookup, translation, rewriting, summarization,
format conversion, or casual chat, or when you can only name the topic but cannot describe any
handling logic, check point, implementation order, or negative lesson.

If True, the sketch must include:
- Methods used: likely methods, strategies, implementation choices, check points, or issues to avoid,
without named entities, concrete numbers, file names, or final answers.
- Rough reasoning-step sketch: 2 to 5 coarse ordered steps, without solving the task.

Output format must be exactly:
Build experience query: $$$True$$$
or
Build experience query: $$$False$$$
Experience-query sketch: <<<
[if True, write:
Methods used: ...
Rough reasoning-step sketch: ...
if False, write NONE]
>>>

Current request:
{query}
```

Figure 8: Prompt for constructing an experience-query sketch before retrieval.

Router Prompt: Explicit-Memory Filter

```
Task: you will see a current request and several saved text records from the past. Output only the
record content that can be directly used in the current answer.

Nature of the candidate records: they are directly reusable text records, such as user facts,
preferences, constraints, environment state, prior commitments, definitions, formulas, theorems, short
rules, checklists, or compact workflows.

Output relevant content when:
- The records contain concrete information that is required or clearly helpful for answering the
current request.
- The request refers to prior information, such as "what I said before", "use my preference", "reuse
the previous setting", or "continue the earlier topic".
- Injecting the records would improve factual accuracy, consistency, or constraint following.

Output NONE when:
- The records are empty, irrelevant, or only topically similar without materially helping the answer.
- The records are mainly long proofs, full derivations, solution transcripts, or only similar in topic
rather than directly reusable facts/rules/procedures.
- The request needs new reasoning, coding, planning, or proof, but the records provide no directly
reusable information.

Requirements:
- Do not output the decision rationale.
- Do not output irrelevant candidate records.
- You may merge, deduplicate, or lightly rewrite relevant records, but do not invent information
absent from the candidates.
- Do not decide whether to use learned problem-solving/debugging/planning experience; this task only
filters text records that can be placed directly into context.

Current request:
{request}

Candidate records:
{explicit_memories}

Output format must be exactly:
Directly usable text records: <<<
[relevant content; write NONE if there is no relevant content]
>>>
```

Figure 9: Prompt for filtering directly usable explicit text records.

Router Prompt: Implicit-Experience Activation

```
Task: you will see the current request, an experience-query sketch for the current request, and
several method summaries from past tasks. Decide whether to activate past experience.

The past-task method summaries are not context to place in the answer, and they are not hints for
solving the current problem. Use them only to decide whether the current request may benefit from
learned handling experience, check points, or negative lessons.

Do not require the past task and the current request to be fully isomorphic. A past summary does not
need to cover the full solution of the current request; it is enough if it can help at one local
stage, such as suggesting a modeling angle, decomposition order, verification habit, boundary check,
or assumption that is easy to get wrong. The decision is whether the model should be allowed to refer
to these handling experiences, not whether the past experience can directly solve the current request.

Do not use past experience when the summaries are empty, or when they only share domain words or
keywords but cannot suggest any handling action, checking action, or error-avoidance cue; also do not
use it when the current request is only factual lookup, translation, rewriting, summarization, format
conversion, or casual chat.

Current request:
{request}

Current request experience-query sketch:
{request_logic_sketch}

Past-task method summaries:
{logic_trajectory_evidence}

Output exactly one line:
Activate experience: $$$True$$$
or
Activate experience: $$$False$$$
```

Figure 10: Verbatim English prompt for deciding whether to activate retrieved implicit experience.

Decoupler Prompt: Explicit Truth

```
Task: you will see a completed user-assistant interaction. Decide whether it should be saved as a
directly reusable text record.

A text record stores stable information that can be used later as background context. Such information
should come from the user, the input material, or outside context itself, such as user preferences,
environment state, project settings, dataset descriptions, long-term constraints, or a task state that
may be continued later. Its value is that a future answer can directly know this information.

Do not treat content produced by the assistant's reasoning, calculation, proof, coding, debugging, or
verification in this interaction as a text record. Even if an answer, conclusion, implementation
result, or problem condition is specific, queryable, and writable as one sentence, it should not be
saved when it is mainly part of the current task instance rather than independent background state for
future use.

If the value of the information is mainly "how to obtain it, how to handle similar problems, or how to
avoid mistakes", it does not belong in this text-record store.

Output requirements:
- First give a brief rationale in 1 to 3 sentences
- Then output one field:
Reusable text record: <<<
[directly reusable text record; write NONE if False]
>>>
- Final line must be exactly: Save text record: $$$True$$$ or Save text record: $$$False$$$

Conversation Messages:
{formatted_messages}
```

Figure 11: Prompt for the explicit-memory branch of the decoupler.

Decoupler Prompt: Implicit Truth

```
Task: you will see a completed user-assistant interaction. Decide whether it demonstrates a handling
method worth reusing later, and extract its methods and step summary.

Reusable handling experience is a way of solving, planning, debugging, verifying, or using tools that
could help on future tasks with a similar structure but different surface details. It stores the
experience of how to handle the task, not the final answer to this task.

Focus on whether the interaction shows an input-to-output handling structure: how the problem was
decomposed, how a method was chosen, how reasoning was organized, how results were checked, how
failures were located, or how a solution was repaired. Correct solutions can be saved; wrong or
incomplete solutions should not be dismissed automatically, because they may reveal transferable
pitfalls, missing checks, wrong assumptions, or paths to avoid. Negative experience does not require
the interaction to complete the repair; if a reusable error type, failure cause, or missing check is
visible, save it as negative experience rather than presenting the invalid method as correct.

Even if no complete reproducible method can be extracted, save negative experience when you can name
an error pattern, unverified assumption, skipped inference, or missing check that future tasks should
avoid.

Do not judge reusability by response length, number of steps, or surface complexity. A short answer,
direct implementation, or basic calculation may still contain reusable handling experience. The key is
whether you can extract an underlying logic, check point, implementation order, or negative lesson
that could guide future tasks.

Do not save user facts, environment state, or stable text records that can be placed directly into
future context. If the interaction only recalls facts, translates or rewrites text, changes format, do
not save it.

If True, the two fields below must summarize the transferable structure. If False, write NONE for both
fields.

Output requirements:
- First provide Evidence snippets (up to 3 short phrases)
- Then explain in 1 to 3 sentences why the signal is or is not worth storing as transferable
processing experience
- Then output three fields:
The question mainly cause reason: <<<
[question that cause reason]
>>>
Methods used: <<<
[methods used; write NONE if False]
>>>
Rough reasoning-step sketch: <<<
[rough reasoning-step sketch; write NONE if False]
>>>
- Final line must be exactly: Save method experience: $$$True$$$ or Save method experience:
$$$False$$$

Conversation Messages:
{formatted_messages}
```

Figure 12: Prompt for the implicit-memory branch of the decoupler.