ROSETTA MEMORY: ADAPTIVE MEMORY FOR CROSS-LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory is the key component for transforming a stateless LLM into a persistent, evolving agent through experience accumulation, long-horizon planning, and continual self-improvement. Existing memory systems typically take the LLM as the center and design memory operations tailored to a specific backbone. In practice, however, users frequently switch between LLMs, for example using Claude for coding and GPT for writing across tasks, or routing different steps to different backbones within a single task for cost-effective trade-offs. As a result, memory written by one model often needs to be consumed by another. Making upstream memory effectively adapt to and activate downstream LLMs remains a critical yet underexplored problem. To bridge this gap, we shift the perspective from LLM-centric memory design to *memory-centric LLM adaptation*. Specifically, we approach the above upstream-downstream memory adaptation problem from both the write and read sides, and design two profile-conditioned operators that are jointly trained to optimize how memory is stored and presented for better task completion. To ensure the learned operators generalize across a broad set of LLMs, we propose a minimum-gain sampling curriculum that prioritizes the least-served LLMs during training. To better measure the operators' actual contribution rather than the LLM's own capability, we design a performance-gap reward that compares against a naive memory baseline. Experiments on HotpotQA, 2WikiMultihopQA, and MuSiQue demonstrate that our model consistently outperforms baselines and remains robust under unseen-model replacement.

1 INTRODUCTION

Recent advances in large language models have positioned LLM-based agents as a promising path toward artificial general intelligence (Yu et al., 2026b; Wang et al., 2024). A central challenge on this path is enabling agents to autonomously accumulate experience and evolve their capabilities over time. Memory is the key component for addressing this challenge, and has consequently attracted significant attention from both academia and industry over the past two years (Bi et al., 2025; Feng et al., 2026a; Hao et al., 2026). Current memory systems are predominantly designed around a fixed LLM, tailoring operations to the specific model that consumes the memory. Representative approaches include memory streams in generative agents (Park et al., 2023), self-reflective verbal memory (Shinn et al., 2023), skill libraries (Wang et al., 2023), hierarchical memory management (Packer et al., 2023), and experience distillation (Zhao et al., 2024). These methods focus on what to store, how to compress, and when to retrieve, but all implicitly assume that the backbone remains unchanged throughout the memory lifecycle.

In practice, however, users often operate agents with entirely different LLMs. Across different tasks, users may prefer Claude for coding and GPT for creative writing; even within a single task, people design routing strategies (Moslem & Kelleher, 2026; Jitkrittum et al., 2026; Wang et al., 2025) that leverage different LLMs at different steps to exploit complementary strengths. As a result, memory written by one LLM often needs to be consumed by another. From the upstream LLM's write perspective, given the same preceding state, different LLMs produce different actions, leading to different content being stored in memory. As shown in our pilot experiment (Figure 1, top row), we vary the number of source LLMs from 2 to 7 and consistently observe increasing output divergence within each dataset. From the downstream LLM's read perspective, different memory entries, even when semantically equivalent, can cause the same LLM to produce different results,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

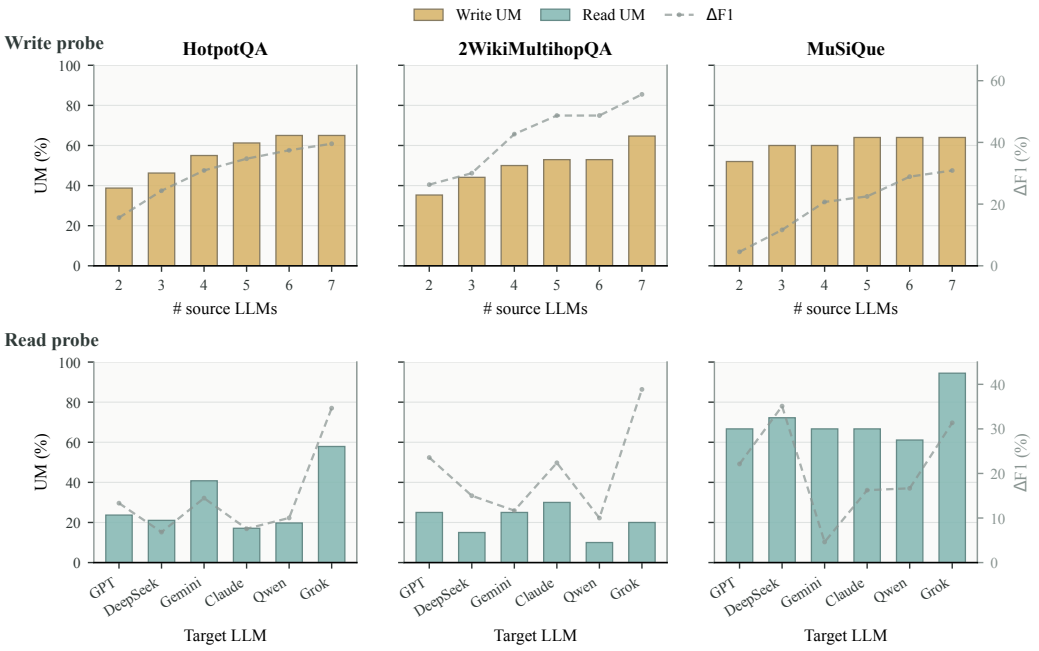


Figure 1: Pilot study of memory mismatch. Each dataset is plotted in a separate panel because absolute mismatch rates are not directly comparable across datasets or probe settings. **Top:** write probe—the same context is processed by K source LLMs; bars show unmatched answer rate (UM) as K grows. **Bottom:** read probe—the same core memory is rewritten into semantically related variants and consumed by different target LLMs. Dashed lines show Δ_{F1} (best–worst F1 gap). Together, the probes show that mismatch can arise both when memories are written by different source models and when similar memories are read by different target models. The same trend appears on HotpotQA, 2WikiMultihopQA, and MuSiQue, covering two-, three-, and four-hop QA settings and suggesting that mismatch is tied to cross-model memory exchange rather than a single dataset or hop length. Please see Appendix D for the full protocol.

thereby affecting agent performance. As shown in Figure 1 (bottom row), semantically related memory variants also lead to different answers across target LLMs within each dataset. These observations motivate the question of this work: *how can agent memory adapt across diverse LLMs, so that memory written by one model effectively activates the downstream LLM that consumes it?*

To answer this question, we shift the perspective from LLM-centric memory design to *memory-centric LLM adaptation*, and formalize the problem of cross-LLM memory adaptation for agent memory systems. To solve this, we propose **Rosetta Memory (RoMem)**, which designs two LLM-aware operators from the write and read sides respectively: each operator takes an LLM profile description and the raw memory as input, and outputs adapted memory tailored to the target LLM. We train these two operators on high-reward mixed-LLM traces. To enable our method to support as many LLMs as possible, we construct a diverse LLM pool and design a minimum-gain sampling curriculum that prioritizes the least-served LLMs during training, progressively improving coverage. To disentangle the contribution of our learned operators from the intrinsic capability of each LLM, we design a performance-gap reward that measures improvement over a naive memory baseline.

In summary, the main contributions of this paper are: (1) We empirically demonstrate, from both the write and read perspectives, the necessity of memory adaptation when diverse LLMs are used in agent tasks, and formalize the problem of cross-LLM memory adaptation. (2) We propose Rosetta Memory (RoMem), which jointly learns LLM-aware write and read operators to adapt upstream memory for downstream LLMs. (3) We design a minimum-gain sampling curriculum and a performance-gap reward to broaden LLM coverage and isolate the operators' contribution. (4) We conduct extensive experiments on HotpotQA, 2WikiMultihopQA, and MuSiQue, demonstrating its effectiveness over multiple-hop QA tasks.

2 Related Work

Memory-augmented LLM agents. Memory has become a central component for transforming stateless LLMs into persistent agents that can reuse observations, reflections, preferences, and interaction traces beyond the immediate prompt. Early systems store natural-language memories for planning, dialogue, or self-reflection (Feng et al., 2026b; Sun et al., 2026; Zhong et al., 2024; Yu et al., 2026a), while recent work improves memory compression, organization, lifecycle management, and online adaptation through gist memories, graph-like notes, scalable memory extraction, and episodic trajectory reuse (Lee et al., 2024; Chhikara et al., 2025; Xu et al., 2025; Zhou et al., 2025; Li et al., 2025; Wang & Chen, 2025). These methods address what to store, how to compress, and when to retrieve, but they generally do not explicitly treat the LLM that writes memory and the LLM that later consumes it as distinct variables, nor evaluate whether memory remains useful when the consuming LLM changes. RoMem instead studies memory as a transferable interface between upstream writer LLMs and downstream reader LLMs within a multi-hop episode.

Context sensitivity and memory interfaces. LLM behavior is sensitive to prompt wording, context order, and evidence placement (Yan et al., 2026; He et al., 2024; Brucks & Toubia, 2025; Feng et al., 2026a), so unchanged facts may still require target-specific presentation. Prefix and prompt tuning further show that compact learned context can steer frozen models (Li & Liang, 2021), motivating profile-conditioned interfaces around fixed black-box LLMs. Closer to our work, RET-LLM introduces explicit memory read/write API calls with symbolic and vector retrieval over extracted triplets (Modarressi et al., 2023), and MemLLM finetunes write and read models to extract relations, query memory, and condition generation on returned relations (Modarressi et al., 2024). However, these methods do not separately model source and target LLM profiles, nor optimize whether memory produced by one LLM remains consumable after the reader LLM is replaced. Instead, RoMem focuses on this cross-LLM memory adaptation problem by jointly learning profile-conditioned write and read operators.

Heterogeneous LLM systems. Model routing and cascading trade off quality and cost (Feng et al., 2026b; Moslem & Kelleher, 2026; Jitkrittum et al., 2026), while ensembles and multi-agent systems aggregate outputs or assign roles (Hao et al., 2026; Wang et al., 2025; Yu et al., 2026b; Hong et al., 2024; Chen et al., 2024); structured memories have also been explored for heterogeneous agents (Yuen et al., 2025). These methods exploit heterogeneity through model selection, output fusion, or role assignment, whereas RoMem adapts shared episode memory to each reader and prioritizes the least-served model–memory pairs.

3 Preliminaries

3.1 Memory-Involved Agent-Environment Interaction

We consider an agent driven by a fixed LLM with policy π. It interacts with an environment over $T+1$ steps indexed by $t = 0, 1, \ldots, T$. At step t, the agent receives an observation $o_t \in \mathcal{O}$ and produces an action a_t. To support long-horizon behavior, it maintains a memory bank, denoted M_t at step t, that evolves across the interaction. Each step follows a read–act–write loop. First, the agent reads a prompt-ready context $\tilde{m}_t$ from M_t conditioned on o_t. Then, it samples an action $a_t \sim \pi(\cdot \mid [\, o_t; \tilde{m}_t \,])$. Finally, it appends a new entry summarizing (o_t, a_t) to form M_{t+1}. Iterating this loop over $t = 0, 1, \ldots, T$ yields a complete trajectory $\tau = (o_0, a_0, \ldots, o_T, a_T)$, which is evaluated by a terminal task reward $R(\tau)$. The agent's objective is to maximize the expected return over trajectories, $\max\ \mathbb{E}_\tau[R(\tau)]$.

3.2 Cross-LLM Memory Adaptation Problem

As mentioned in the introduction, agents often switch backbones across steps within the same trajectory in practice. To formally define this cross-LLM memory adaptation problem, we extend the basic memory-involved agent-environment interaction paradigm in §3.1 by allowing the backbone LLM to vary across steps within the same trajectory. At step t, the agent is driven by a given LLM ℓ_t from a heterogeneous model pool $\mathcal{L} = \{L^{(1)}, \ldots, L^{(K)}\}$, and the model assignment sequence for the trajectory is denoted by $\boldsymbol{\ell} = (\ell_0, \ldots, \ell_T) \in \mathcal{L}^{T+1}$. To make memory effective across this

heterogeneous pool, we parameterize the read and write operations as two learnable, LLM-aware functions: a read function $f_{\text{read}}(\cdot;\, \theta_{\text{r}})$ that takes the target LLM identifier and adapts entries from M_t into a prompt-ready context, and a write function $f_{\text{write}}(\cdot;\, \theta_{\text{w}})$ that takes the source LLM identifier and distills each (o_t, a_t) into a memory entry. The step-level loop becomes:

$$\tilde{m}_t = f_{\text{read}}(o_t,\, M_t,\, \ell_t;\, \theta_{\text{r}}), \tag{1}$$

$$a_t \sim \pi_{\ell_t}(\cdot \mid [\, o_t;\, \tilde{m}_t \,]), \tag{2}$$

$$M_{t+1} = M_t \cup \{ f_{\text{write}}(o_t,\, a_t,\, \ell_t;\, \theta_{\text{w}}) \}. \tag{3}$$

Here ℓ_t acts as the LLM identifier that tells the operators which model will consume the read context and which model writes the current action. We specify its concrete realization in §4.1.

As discussed in Section 1, naive memory operations become suboptimal under diverse LLMs, due to write-side divergence and read-side sensitivity. Given an assignment $\boldsymbol{\ell}$, the read–act–write loop in eqs. (1) to (3) induces a trajectory distribution $P_{\boldsymbol{\ell}}^{\theta_{\text{w}},\theta_{\text{r}}}$. The trajectory-level return is then defined as:

$$J(\boldsymbol{\ell},\, \theta_{\text{w}},\, \theta_{\text{r}}) = \mathbb{E}_{\tau \sim P_{\boldsymbol{\ell}}^{\theta_{\text{w}},\theta_{\text{r}}}}[R(\tau)], \tag{4}$$

where $R(\tau)$ is the terminal task reward on the sampled trajectory τ. Ideally, we want f_{read} and f_{write} to obtain a large J under every possible assignment $\boldsymbol{\ell} \in \mathcal{L}^{T+1}$. However, directly enforcing this across all K^{T+1} assignments is infeasible. We therefore relax it into the following max–min objective:

$$(\theta_{\text{w}}^*,\, \theta_{\text{r}}^*) = \arg\max_{\theta_{\text{w}},\, \theta_{\text{r}}} \min_{\boldsymbol{\ell} \in \mathcal{L}^{T+1}} J(\boldsymbol{\ell},\, \theta_{\text{w}},\, \theta_{\text{r}}). \tag{5}$$

The inner $\min$ targets the worst-case assignment, so maximizing it lifts the lower bound of J across all assignments.

4 METHODOLOGY

The core of our method is to learn the read and write operators f_{read} and f_{write} so that memory written by one LLM remains effective when consumed by another. We first describe the concrete implementation of these two operators (§4.1), and then present their training procedure (§4.2).

4.1 IMPLEMENTATION OF f_{read} AND f_{write}

The write operator f_{write} and the read operator f_{read} are parameterized as two separate language models, and the LLM identifier ℓ_t at each step is concretely realized via a profile embedding. Each LLM $L^{(k)} \in \mathcal{L}$ is associated with a lightweight profile record containing a model identifier i_k, a textual description d_k, and metadata $\boldsymbol{q}_k$ (e.g., backend, family, scale, preferred read style). A learnable mapping function g with parameters θ_g takes this profile record to a condition vector

$$\boldsymbol{e}_{L^{(k)}} = g(i_k,\, d_k,\, \boldsymbol{q}_k) \in \mathbb{R}^d, \tag{6}$$

which is injected into f_{read} and f_{write} as soft prefix tokens (Li & Liang, 2021; Lester et al., 2021). At step t, the reader uses the target-LLM embedding $\boldsymbol{e}_{\ell_t}^{\text{tgt}}$ to rewrite the bank into a prompt-ready context $\tilde{m}_t = f_{\text{read}}(o_t,\, M_t,\, \boldsymbol{e}_{\ell_t}^{\text{tgt}};\, \theta_{\text{r}})$, and the writer uses the source-LLM embedding $\boldsymbol{e}_{\ell_t}^{\text{src}}$ to distill the observation–action pair into a memory entry $m_t = f_{\text{write}}(o_t,\, a_t,\, \boldsymbol{e}_{\ell_t}^{\text{src}};\, \theta_{\text{w}})$. In essence, g acts as a compression function that maps the discrete LLM identity into a continuous embedding space, which improves the model's generalization to unseen LLMs.

4.2 TRAINING OF f_{read} AND f_{write}

RoMem trains f_{read} and f_{write} with an Expert Iteration-style, reward-filtered self-training loop: the current operators generate mixed-LLM rollouts, terminal rewards retain the top-α traces as expert demonstrations, and supervised updates imitate their read/write decisions (Anthony et al., 2017; Gulcehre et al., 2023). To make this loop robust across LLMs, we use two designs throughout training: a counterfactual gain that estimates each LLM's operator-specific improvement over a naive memory baseline, and a sampler that prioritizes LLMs with the lowest gain during rollout collection.

Algorithm 1 RoMem Expert-Iteration Training with Minimum-Gain Sampling

Require: LLMs $\mathcal{L} = \{L^{(1)}, \dots, L^{(K)}\}$, task distribution $p(\mathcal{M})$, hyperparameters η and α
for each $L^{(k)} \in \mathcal{L}$ **do**
  Collect profile record $(i_k, d_k, \boldsymbol{q}_k)$
  Compute profile embedding $\boldsymbol{e}_{L^{(k)}} \leftarrow g(i_k, d_k, \boldsymbol{q}_k)$ via eq. (6)
end for
Initialize $\Delta R(L^{(k)}) \leftarrow 0$ for all $L^{(k)} \in \mathcal{L}$
for each training round until the budget is exhausted **do**
  Sample task $\mathcal{M} \sim p(\mathcal{M})$
  Sequentially sample assignment $\boldsymbol{\ell} = (\ell_0, \dots, \ell_T)$ using eq. (8)
  Initialize memory bank $M_0 \leftarrow \varnothing$
  for each step $t = 0, \dots, T$ **do**
    $\tilde{m}_t \leftarrow f_{\text{read}}(o_t, M_t, \boldsymbol{e}^{\text{tgt}}_{\ell_t}; \theta_{\text{r}})$
    $a_t \sim \pi_{\ell_t}(\cdot \mid [o_t; \tilde{m}_t])$
    $m_t \leftarrow f_{\text{write}}(o_t, a_t, \boldsymbol{e}^{\text{src}}_{\ell_t}; \theta_{\text{w}})$
    $M_{t+1} \leftarrow M_t \cup \{m_t\}$
    Log $(\ell_t, M_t, a_t, m_t, \tilde{m}_t)$
  end for
  Rank completed trajectories by terminal reward and keep the top-α fraction as $\mathcal{D}_{\text{expert}}$
  Build $\mathcal{D}_{\text{train}}$ from logged write/read pairs using eq. (9)
  Update f_{write}, f_{read}, and g by minimizing eq. (10)
  Periodically refresh $\Delta R(L^{(k)})$ for all $L^{(k)}$ using eq. (7)
end for
Return: Trained f_{write}, f_{read}, and profile encoder g

Counterfactual gain. To isolate the operators' contribution from the intrinsic capability of each LLM, we define a *naive baseline* memory system whose write step appends entries to the memory bank without any learned processing and whose read step concatenates all stored entries into the context. For each LLM $L^{(k)}$, the *counterfactual gain* is then

$$\Delta R(L^{(k)}) = R_f(L^{(k)}) - R_{\text{naive}}(L^{(k)}), \tag{7}$$

where $R_f(L^{(k)})$ is the expected reward over entire trajectories with $L^{(k)}$ as the backbone at every step under the current f_{read} and f_{write}, and $R_{\text{naive}}(L^{(k)})$ is the analogous quantity under the naive baseline.[1] We refresh $\Delta R(L^{(k)})$ periodically during training using the latest checkpoints, and a small $\Delta R(L^{(k)})$ identifies an underserved LLM, regardless of its intrinsic strength.

Minimum-gain LLM sampling. To prioritize underserved LLMs during training, we sample the active model at each step according to

$$p(\ell_t = L^{(k)}) \propto d_k^{(t)} \exp\Big(-\eta\, \Delta R(L^{(k)})\Big), \tag{8}$$

where $d_k^{(t)} = 1$ at the first step; otherwise, letting $\mathcal{L}_{\boldsymbol{\ell}_{<t}} = \{\ell_0, \dots, \ell_{t-1}\}$ denote the LLMs already sampled within the current trajectory, we set $d_k^{(t)} = \min_{L^{(j)} \in \mathcal{L}_{\boldsymbol{\ell}_{<t}}} \|\boldsymbol{e}_{L^{(k)}} - \boldsymbol{e}_{L^{(j)}}\|_2$, i.e., the embedding distance from the candidate $L^{(k)}$ to its nearest already-sampled neighbor. The $d_k^{(t)}$ term thus promotes coverage of behaviorally diverse models within a trajectory, the exponential weight biases sampling toward those with small ΔR, and $\eta > 0$ controls how strongly the sampler favors underserved LLMs.

[1] We measure ΔR at the trajectory level because many tasks provide only a terminal reward with no intermediate per-step signal; if per-step rewards were available, our method could be straightforwardly extended to use them.

Training objective. Consistent with the Expert Iteration loop above, we train f_{write}, f_{read}, and the profile encoder on high-reward mixed-LLM rollouts. At each step, we sample ℓ_t via eq. (8), drawing both behaviorally diverse and underserved models. During each rollout, we log the active model ℓ_t, memory bank M_t, action a_t, written entry m_t, and read context $\tilde{m}_t$, rank trajectories by cumulative reward, and keep the top-α fraction as $\mathcal{D}_{\text{expert}}$. The training dataset aggregates the logged steps:

$$\mathcal{D}_{\text{train}} = \{(x_t^w,\, m_t,\, x_t^r,\, \tilde{m}_t) : (o_t, a_t, \ell_t) \in \tau,\ \tau \in \mathcal{D}_{\text{expert}}\}, \tag{9}$$

where $x_t^w = (o_t, a_t, \boldsymbol{e}_{\ell_t}^{\text{src}})$ and $x_t^r = (o_t, M_t, \boldsymbol{e}_{\ell_t}^{\text{tgt}})$. We jointly optimize θ_{w}, θ_{r}, and θ_g by minimizing:

$$\mathcal{L}_{\text{train}} = -\mathbb{E}_{\mathcal{D}_{\text{train}}}[\log p_{\theta_{\text{w}}}(m_t \mid x_t^w) + \log p_{\theta_{\text{r}}}(\tilde{m}_t \mid x_t^r)]\,. \tag{10}$$

Algorithm 1 summarizes the full RoMem training pipeline.

5 Experiments

In this section, we first evaluate the overall task performance of RoMem against existing baselines, and then conduct a component-level ablation over f_{read}, f_{write}, and the profile embedding to identify each part's contribution. We next study generalization to held-out LLMs and the effect of the minimum-gain LLM sampling schedule introduced in §4.2. Finally, we present a qualitative inspection of the read/write outputs. We summarize these aspects as five research questions: **RQ1:** Does RoMem improve overall task performance compared with existing prompting, accumulation, and memory baselines? **RQ2:** Which RoMem components (f_{read}, f_{write}, and the profile embedding) contribute most to the final performance? **RQ3:** Does RoMem generalize to held-out LLMs that are unseen during training? **RQ4:** How effective is minimum-gain LLM sampling? **RQ5:** What do the learned read and write operators actually produce at the text level?

5.1 Experimental Setup

Datasets. We take multi-hop QA as a representative testbed to validate RoMem, since it naturally involves intermediate evidence accumulation across steps and thus exposes cross-LLM memory effects clearly. Specifically, we use HotpotQA, 2WikiMultihopQA, and MuSiQue (Yang et al., 2018; Ho et al., 2020; Trivedi et al., 2022), each adapted into fixed-length multi-hop QA trajectories. For all the datasets, we filter out samples with fewer than the required number of hops, and use the same filtered set across all methods for a fair comparison. After filtering and sampling, the training/test splits contain 1,851/159 examples for HotpotQA, 642/67 examples for 2WikiMultihopQA, and 481/62 examples for MuSiQue, respectively. Following recent API-intensive multi-hop RAG evaluations that use randomly selected subsets to balance stable comparison with experimental feasibility (Asl et al., 2025), we keep the evaluation subsets fixed across all methods. We repeat each experiment with four random seeds and report the mean performance.

Baselines. We group the baselines by how they handle memory across reasoning steps. **(1) Memory-free baselines** access only the retrieved context without any persistent memory: Direct (answers directly from retrieved evidence), Chain-of-Thought (CoT) (Wei et al., 2022), Scratchpad (Nye et al., 2021), Self-Ask (Press et al., 2023), and Least-to-Most (Zhou et al., 2022); the latter four organize retrieved evidence in-context with their respective reasoning prompts but do not persist it across steps. **(2) Raw-buffer memory baselines** keep a running buffer of intermediate state without any learned read or write processing: Step-Accumulate concatenates step-level evidence and notes into a growing buffer, while Running-Summary (Lee et al., 2024) compresses the history into a compact factual summary. **(3) Structured memory baselines** maintain a dedicated memory store with custom organization but without backbone-aware adaptation: Mem0 (Chhikara et al., 2025) writes per-step episodic memory entries, A-Mem (Xu et al., 2025) builds an associative network of linked notes, and Memento (Zhou et al., 2025) stores trajectory checkpoints as cases for answer generation.

Implementation Details. The write and read operators f_{write} and f_{read} are both initialized from the google/flan-t5-small checkpoint (Chung et al., 2024; Raffel et al., 2020) for all runs. The LLM pool of RoMem contains six API profiles: gpt-5.4-mini, gpt-5.4-nano, gemini-2.5-flash, gemini-2.5-flash-nothinking, claude-sonnet-4-6, and claude-sonnet-4-6-thinking (Singh et al., 2025; Kasireddy et al., 2026; Team et al., 2023). We tune the minimum-gain sampling coefficient η over

Table 1: Main results under sample-context retrieval. Values are percentages; higher is better, and the best result in each column is shown in **bold**.

Method	HotpotQA				2Wiki				MuSiQue			
	EM↑	F1↑	ACC↑	ACC_{LLM} ↑	EM↑	F1↑	ACC↑	ACC_{LLM} ↑	EM↑	F1↑	ACC↑	ACC_{LLM} ↑
Memory-free baselines												
Direct	28.3	41.9	34.6	50.9	38.8	40.4	43.3	43.3	8.1	10.4	8.1	11.3
CoT	39.0	54.0	48.4	65.4	49.3	51.4	49.3	49.3	11.3	15.9	11.3	14.5
Scratchpad	34.6	48.9	44.0	56.0	23.9	28.4	28.4	28.4	9.7	15.0	11.3	17.7
Self-Ask	39.6	53.7	46.5	61.6	43.3	46.2	44.8	44.8	4.8	14.4	11.3	16.1
Least-to-Most	36.5	52.0	42.8	59.1	25.4	29.4	25.4	25.4	6.5	17.3	8.1	21.0
Raw-buffer memory baselines												
Step-Accumulate	37.7	53.1	46.5	64.2	38.8	41.3	38.8	38.8	12.9	21.5	16.1	21.0
Running-Summary	34.0	46.6	42.8	55.3	29.9	32.6	31.3	31.3	6.5	14.6	8.1	16.1
Structured memory baselines												
Mem0	35.8	52.5	47.8	62.3	44.8	48.3	44.8	44.8	11.3	17.4	11.3	17.7
A-Mem	39.0	53.9	48.4	66.0	38.8	41.0	40.3	40.3	8.1	16.4	8.1	19.4
Memento	38.4	55.2	48.4	65.4	47.8	50.8	47.8	47.8	12.9	20.8	14.5	24.2
RoMem	**50.9**	**65.8**	**60.4**	**79.9**	**58.2**	**61.0**	**59.7**	**59.7**	**16.1**	**34.2**	**24.2**	**37.1**

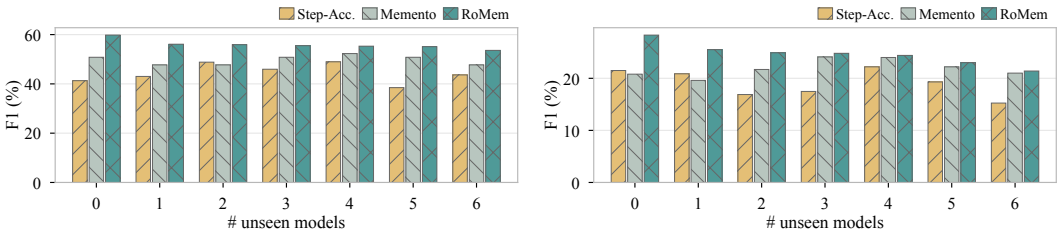
Table 2: Ablation results on the three multi-hop QA datasets. Values are percentages; higher is better, and the best result in each column is shown in **bold**.

Method	HotpotQA				2Wiki				MuSiQue			
	EM↑	F1↑	ACC↑	ACC_{LLM} ↑	EM↑	F1↑	ACC↑	ACC_{LLM} ↑	EM↑	F1↑	ACC↑	ACC_{LLM} ↑
Step-Accumulate	37.7	53.1	46.5	64.2	38.8	41.3	38.8	38.8	12.9	21.5	16.1	21.0
w/o f_{write}	30.2	45.5	37.7	52.8	46.3	47.9	46.3	46.3	12.9	25.1	14.5	29.0
w/o f_{read}	44.7	60.4	54.1	74.2	46.3	49.0	47.8	47.8	16.1	27.2	19.4	30.6
w/o profiles	30.8	46.0	35.8	51.6	44.8	47.0	44.8	46.3	11.3	22.6	14.5	30.6
RoMem	**50.9**	**65.8**	**60.4**	**79.9**	**58.2**	**61.0**	**59.7**	**59.7**	**16.1**	**34.2**	**24.2**	**37.1**

$\{0.4, 0.8, 1.2, 1.6, 2.0\}$. We report Exact Match (EM), token-level F1, script accuracy (Acc), and LLM-judged semantic accuracy (Acc_{LLM}) (Zheng et al., 2023), where Acc is a normalized containment score and Acc_{LLM} is a binary semantic judgment from gpt-5.4-mini.

5.2 RQ1: Overall Comparison

Table 1 reports the main results on the three multi-hop QA datasets. RoMem obtains the best result in every reported column across all the datasets. On HotpotQA, the gains are especially large under Acc_{LLM}, suggesting that learned read/write adaptation improves semantic agreement beyond surface-form matching. On 2WikiMultihopQA, RoMem also leads the strongest prompting and memory baselines, indicating that backbone-aware memory remains useful even when short answers make direct context-based methods competitive. For 2WikiMultihopQA, Acc and Acc_{LLM} are often identical in Table 1 because the answers in our filtered subset are mostly short entity strings, for which normalized containment largely agrees with semantic judging. On MuSiQue, which is the most difficult setting in our evaluation, RoMem achieves the clearest margin on F1 and Acc_{LLM}, showing that learned memory transformation helps preserve partial evidence across harder multi-hop chains. Overall, the results support the central premise of RoMem: storing evidence is not sufficient by itself; the memory must also be written and read in a form that is adapted to the LLM that will consume it.



(a) 2WikiMultihopQA (b) MuSiQue

Figure 2: Model replacement on 2WikiMultihopQA and MuSiQue. Bars are grouped by the number of unseen test-time backbone models and report F1.

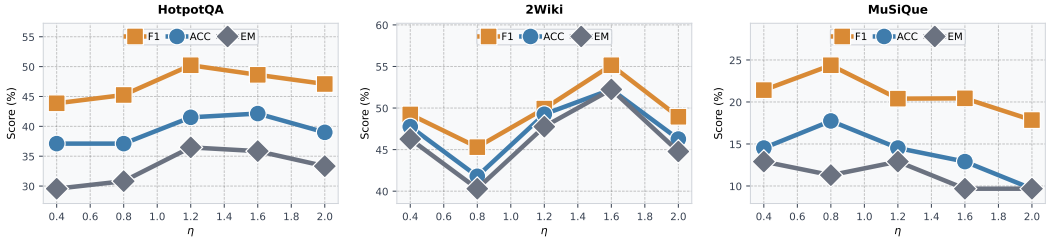


Figure 3: Sensitivity to the minimum-gain sampling coefficient η on HotpotQA, 2WikiMultihopQA, and MuSiQue. Each panel uses η as the x-axis and reports RoMem F1, ACC, and EM.

5.3 RQ2: Ablation Study

We ablate the main RoMem components under the same multi-hop evaluation protocol as RQ1. The w/o f_{write} variant stores raw step evidence, w/o f_{read} directly concatenates memory into the final context, and w/o profiles replaces source and target model profiles with a shared profile identifier. Step-Accumulate serves as a no-read/write reference and is included as the representative non-RoMem baseline because it is consistently strong in our main comparison (Table 1); the RoMem row is copied from the main comparison. Table 2 shows that model profiles and the write operator are important for stable memory transfer. Removing profiles consistently hurts performance, indicating that a shared profile is insufficient for heterogeneous backbones. Removing f_{write} also weakens most settings by leaving raw evidence less structured and less portable. The w/o f_{read} variant is competitive in some settings but falls behind on 2Wiki, suggesting that learned reading is most useful when raw memory needs target-specific presentation. Together, the ablations show that profiles and learned writing are especially important, with learned reading adding robustness under harder cross-LLM transfer. Appendix E.1 further examines writer–reader compatibility across model pairs.

5.4 RQ3: Transfer to Unseen Models

We test whether RoMem generalizes to six held-out LLMs unseen during training: `gpt-5-mini`, `deepseek-reasoner`, `gemini-2.5-flash-lite`, `grok-4-fast`, `qwen3.5-plus`, and `glm-5`. At evaluation time, we replace $k \in \{0, \ldots, 6\}$ training-pool models with held-out API models and use lightweight profiles without fine-tuning. All methods use the same candidate-context source, answer model, and metrics; we compare RoMem with Step-Accumulate and Memento. Figure 2 reports F1 on 2WikiMultihopQA and MuSiQue. On 2WikiMultihopQA, RoMem achieves the best F1 across all replacement levels, suggesting that the learned read/write interface transfers well to new LLMs. On MuSiQue, RoMem is strongest in most settings and remains close to the best baseline otherwise, indicating that profile-conditioned adaptation does not collapse under held-out replacement. The trends are not strictly monotonic because increasing k also changes the concrete mixture of model abilities.

5.5 RQ4: Effectiveness of Minimum-Gain Sampling

This section studies the minimum-gain sampling coefficient η, which controls how strongly model sequences are biased toward backbones with smaller counterfactual gain ΔR through $\exp(-\eta \Delta R)$.

Logged source memory plus evidence → normalized written memory

Dataset. 2WikiMultihopQA.
Question. Which film whose director is younger, *Hell to Pay* or *Darwin's Nightmare*?
Verbatim source memory. GPT-5.4-mini: "Need compare directors' ages: Jay Jennings was born in 1965, and Hubert Sauper's birth year is missing—look up Hubert Sauper's birth date/year to determine which director is younger."
Verbatim retrieved evidence. "Jay Robert Jennings(born August 23, 1965)"; "Hubert Sauper(born 27 July 1966)".
Normalized written memory. *Hell to Pay*: director Jay Jennings, born 1965. *Darwin's Nightmare*: director Hubert Sauper, born 1966. Need compare which director is younger.
Effect: removes the search instruction and most source-model uncertainty, keeps the two comparable facts, and avoids writing the final film answer.

Same memory → profile-conditioned read contexts

Target: `gpt-5.4-nano`. *Hell to Pay* – Jay Jennings – 1965. *Darwin's Nightmare* – Hubert Sauper – 1966. Pick the younger director's film; answer as a title.
Target: `claude-sonnet-4-6-thinking`. Use the memory as an evidence chain: map each film to its director, compare 1965 vs. 1966, then decide which film has the younger director. Do not rely on outside facts.
Target: `gemini-2.5-flash-nothinking`. Table: *Hell to Pay* | Jay Jennings | 1965. *Darwin's Nightmare* | Hubert Sauper | 1966. Short answer only.
Effect: keeps the same facts but varies ordering, explicitness, and answer-format constraints according to the target profile.

Figure 4: Representative read/write inspection from a held-out 2Wiki validation sample.

We evaluate $\eta \in \{0.4, 0.8, 1.2, 1.6, 2.0\}$ under F1, ACC, and EM on all three datasets. Figure 3 shows dataset-specific optima: HotpotQA favors $\eta = 1.2$ for F1 and 1.6 for ACC, 2WikiMultihopQA peaks at 1.6 across all metrics, and MuSiQue favors 0.8 for F1 and ACC. The variation across datasets also indicates that η should be selected on validation data: it trades broad model-sequence coverage against corrective rollout exposure for the least-improved backbones. Thus, moderate reweighting is useful, whereas larger values can over-concentrate sampling on low-gain backbones. Appendix E.2 reports the worst-case target-reader analysis.

5.6 RQ5: Qualitative Read/Write Analysis

Finally, we inspect a held-out 2Wiki validation example to make the intended behavior of f_{write} and f_{read} concrete. This example is illustrative rather than a separate benchmark. Figure 4 contrasts heterogeneous source memory and evidence with normalized memory and target-conditioned read contexts; Table 2 quantifies the operators' contribution. The trace shows complementary behavior: f_{write} removes source-specific instructions and uncertainty while retaining the bridge facts without answer leakage, whereas f_{read} preserves those facts but adjusts their ordering, explicitness, and answer-format cues to the target profile. Appendix E.3 evaluates polluted memory banks.

Failure modes. We also observe several common failure cases. The writer can inherit incomplete source memories, preserve a source model's uncertainty, or copy a final-answer statement into memory. The reader can over-compress the stored evidence and remove a bridge fact needed by weaker target models. These cases suggest that f_{write} should favor neutral evidence over conclusions, while f_{read} should adapt presentation without discarding task-critical facts.

6 Conclusion

This paper studies cross-LLM memory adaptation, where memory written under one backbone must be consumed by another. We propose Rosetta Memory (RoMem), a profile-conditioned interface that learns to write transferable evidence and read it in a target-aware form. Across three multi-hop QA datasets, RoMem improves over prompting, accumulation, and memory baselines, supporting a memory-centric view of heterogeneous LLM agents. Future work can extend this interface to longer-horizon tasks and richer behavioral profiles.

AI USE STATEMENT

Generative AI assistance was used to inspect the existing references, organize the experimental protocol, help to implement and polish the manuscript. We have not used generative AI tools for the core intellectual contributions of this work, including the theoretical derivations and algorithm design. We have carefully reviewed and edited all AI-generated content to ensure accuracy and clarity.

REPRODUCIBILITY STATEMENT

To support reproducibility, we describe the proposed method and experimental setup in the main text. Further details needed to reproduce the experiments, including dataset descriptions, preprocessing procedures, model configurations, hyperparameter selection, training and evaluation protocols, and computational resources, are provided in the experimental sections and the appendix. We have also provided the code and data used in our experiments in the supplementary material.

REFERENCES

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *Advances in neural information processing systems*, 30, 2017.

Mohammad Aghajani Asl, Majid Asgari-Bidhendi, and Behrooz Minaei-Bidgoli. Fair-rag: Faithful adaptive iterative refinement for retrieval-augmented generation. *arXiv preprint arXiv:2510.22344*, 2025.

Zhenyu Bi, Daniel Hajialigol, Zhongkai Sun, Jie Hao, and Xuan Wang. Stoc-tot: Stochastic tree-of-thought with constrained decoding for complex reasoning in multi-hop question answering. In *Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing*, pp. 141–151, 2025.

Melanie Brucks and Olivier Toubia. Prompt architecture induces methodological artifacts in large language models. *PloS one*, 20(4):e0319159, 2025.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pp. 20094–20136, 2024.

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

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

Tao Feng, Chongrui Ye, Tianyang Luo, Jingjun Xu, Xueqiang Xu, Haozhen Zhang, Ge Liu, and Jiaxuan You. Elasticmem: Latent memory as a learnable resource for llm agents. *arXiv preprint arXiv:2605.30690*, 2026a.

Tao Feng, Haozhen Zhang, Zijie Lei, Peixuan Han, and Jiaxuan You. Graphplanner: Graph memory-augmented agentic routing for multi-agent llms. *arXiv preprint arXiv:2604.23626*, 2026b.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Guangya Hao, Yunbo Long, and Zhuokai Zhao. Self-evolving multi-agent systems via decentralized memory. *arXiv preprint arXiv:2605.22721*, 2026.

Jia He, Mukund Rungta, David Koleczek, Arshdeep Sekhon, Franklin X Wang, and Sadid Hasan. Does prompt formatting have any impact on llm performance? *arXiv preprint arXiv:2411.10541*, 2024.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, et al. Universal model routing for efficient llm inference. In *International Conference on Learning Representations*, volume 2026, pp. 10169–10218, 2026.

Ellen Kasireddy, Cuthbert Chow, Jun Collet, Mir-Masoud Pourrahmat, and Mir Sohail Fazeli. Evaluating the performance of claude 3.7 sonnet in data extraction automation for systematic literature reviews. *Value in Health Regional Issues*, 53:101539, 2026.

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint arXiv:2402.09727*, 2024.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 3045–3059, 2021.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597, 2021.

Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, et al. Memos: An operating system for memory-augmented generation (mag) in large language models. *arXiv preprint arXiv:2505.22101*, 2025.

Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Ret-llm: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322*, 2023.

Ali Modarressi, Abdullatif Köksal, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Memllm: Finetuning llms to use an explicit read-write memory. *arXiv preprint arXiv:2404.11672*, 2024.

Yasmin Moslem and John D Kelleher. Dynamic model routing and cascading for efficient llm inference: A survey. *arXiv preprint arXiv:2603.04445*, 2026.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 2383–2392, 2016.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

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

Haoran Sun, Shaoning Zeng, and Bob Zhang. H-mem: Hierarchical memory for high-efficiency long-term reasoning in llm agents. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 341–350, 2026.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Y Zou. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations*, volume 2025, pp. 33944–33963, 2025.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Yu Wang and Xi Chen. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 17577–17604, 2025. doi: 10.52202/085713-0593.

Sikuan Yan, Ahmed Bahloul, Ercong Nie, Susanna Schwarzmann, Riccardo Trivisonno, Volker Tresp, and Yunpu Ma. Memory-r2: Fair credit assignment for long-horizon memory-augmented llm agents. *arXiv preprint arXiv:2605.21768*, 2026.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. *arXiv preprint arXiv:2601.01885*, 2026a.

Zhongming Yu, Naicheng Yu, Hejia Zhang, Wentao Ni, Mingrui Yin, Jiaying Yang, Yujie Zhao, and Jishen Zhao. Multi-agent memory from a computer architecture perspective: Visions and challenges ahead. *arXiv preprint arXiv:2603.10062*, 2026b.

Sizhe Yuen, Francisco Gomez Medina, Ting Su, Yali Du, and Adam J Sobey. Intrinsic memory agents: Heterogeneous multi-agent llm systems through structured contextual memory. *arXiv preprint arXiv:2508.08997*, 2025.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024.

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

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, et al. Memento: Fine-tuning llm agents without fine-tuning llms. *arXiv preprint arXiv:2508.16153*, 2025.

A LIMITATIONS

RoMem is a first step toward memory-centric adaptation for heterogeneous LLM agents, and our study has several limitations.

- We evaluate controlled multi-hop QA episodes with fixed evidence-chain lengths. This setting exposes cross-LLM memory effects clearly, but it does not fully cover open-ended web navigation, tool use, dialogue, or very long-horizon planning.
- Our experiments use filtered evaluation subsets and a finite pool of seen and held-out API LLMs because each example requires many writer, reader, baseline, and judge calls. Larger model pools and larger evaluation sets would further strengthen the empirical picture.
- The learned reader and writer can still fail by over-compressing memory, preserving source-model uncertainty, or producing weak target-specific contexts.

B BROADER IMPACT

RoMem aims to make heterogeneous LLM systems more reliable when intermediate evidence is shared across models. Better cross-LLM memory transfer could reduce unnecessary recomputation, make lower-cost models more useful inside multi-step workflows, and improve robustness when systems switch models because of latency, availability, or cost constraints. The write/read interface may also improve inspectability by converting source-specific traces into compact evidence and by exposing the context presented to each target LLM.

The same capability also creates risks. If memory contains incorrect, biased, private, or copyrighted content, a learned writer may preserve it and a learned reader may rephrase it in a more authoritative form. Cross-model transfer can further spread such errors across systems that otherwise would not share the same internal state. Practical deployments should therefore use data minimization, access control, deletion mechanisms, audit logs, and human inspection for persistent memories. For high-stakes domains, automatically rewritten memory should be treated as supporting evidence rather than as a trusted source of truth, and final decisions should rely on independently verifiable information.

C EXPERIMENTAL SETTING DETAILS

C.1 DATASET CONSTRUCTION AND EPISODE FORMAT

We convert each QA instance into an episode with a fixed number of reasoning hops. HotpotQA, 2WikiMultihopQA, and MuSiQue (Yang et al., 2018; Ho et al., 2020; Trivedi et al., 2022) are instantiated as 2-hop, 3-hop, and 4-hop episodes, respectively, matching their typical evidence-chain lengths in our processed files. Examples with fewer usable hops than the target episode length are filtered out before training and evaluation. After filtering and sampling, the training/test splits contain 1,851/159 examples for HotpotQA, 642/67 examples for 2WikiMultihopQA, and 481/62 examples for MuSiQue, respectively. All methods use the candidate context attached to the current example rather than an external Wikipedia dump, and gold supporting sentences are not directly inserted as prompts during evaluation. At hop t, the model receives the question, the current candidate evidence, and the accumulated state produced by the method being evaluated. The final answer is generated only after the last hop, so all methods share the same episode length and answer interface.

C.2 BACKBONE LLM POOL AND PROFILES

The seen backbone pool contains six API models: **gpt-5.4-mini**, **gpt-5.4-nano**, **gemini-2.5-flash**, **gemini-2.5-flash-nothinking**, **claude-sonnet-4-6**, and **claude-sonnet-4-6-thinking** (Singh et al., 2025; Kasireddy et al., 2026; Team et al., 2023). The held-out pool used in the model-replacement experiment contains six unseen API models: **gpt-5-mini**, **deepseek-reasoner**, **gemini-2.5-flash-lite**, **grok-4-fast**, **qwen3.5-plus**, and **glm-5**. When k unseen models are evaluated, the first k models in this fixed held-out list replace the corresponding seen models. Each model is represented by a lightweight profile record consisting of its model name, a short textual description, and metadata

such as backend, model family, scale bucket, and preferred read style. For held-out models in the model-replacement experiment, we construct the same type of profile record and use it directly at test time. No task rollouts or gradient updates are performed for the held-out models.

C.3 Training Details

RoMem is trained with Expert Iteration-style supervised fine-tuning from mixed-model rollouts. Each iteration collects read/write traces, filters high-reward trajectories, and uses minimum-gain scheduling to sample more often from models whose current read–write gain is low. We run two outer Expert-Iteration rounds. Each round samples 10 mixed-model rollouts per training example, keeps the top-α fraction of reward-ranked trajectories with $\alpha = 0.3$ and a minimum of one retained trajectory per example, and then trains the write/read operators for two SFT epochs. The rollout assignments use the minimum-gain sampler with $\eta = 1.2$ unless otherwise specified.

The profile encoder g maps each profile to a 256-dimensional condition vector. It combines three inputs: a learned model-ID embedding, a mean-pooled embedding of the textual profile description, and an MLP over profile metadata. The metadata consists of the backend type, model family, scale bucket, preferred read style, a log-scale feature, a scale-present flag, a normalized model-name hash, and normalized description length. The metadata MLP uses hidden size 128; the description token embedding size is 128 with a maximum description length of 48 tokens. The concatenated representation is passed through a two-layer fusion MLP with GELU activations and layer normalization. For both f_{write} and f_{read}, the resulting profile vector is projected into eight soft prefix tokens and prepended to the encoder input embeddings of the corresponding `google/flan-t5-small` sequence-to-sequence model. The writer and reader use separate FLAN-T5-small backbones and separate prefix projections.

We optimize the trainable read/write and profile-conditioned components with AdamW using batch size 4, gradient accumulation 4, weight decay 0.01, warmup ratio 0.03, maximum source length 512, and maximum target length 160. We sweep learning rates in $\{1\times10^{-5}, 2\times10^{-5}, 3\times10^{-5}\}$ and tune the minimum-gain sampling coefficient η over $\{0.4, 0.8, 1.2, 1.6, 2.0\}$. All experiments are run on NVIDIA A800-SXM4-80GB GPUs.

C.4 Baseline Descriptions

Prompting and decomposition baselines.

- **Direct** answers the question directly from the shared candidate context without explicit decomposition or memory.
- **CoT** (Wei et al., 2022) prompts the model to produce intermediate reasoning before answering. We instantiate it with the same candidate context and final-answer interface as the other context-using baselines.
- **Scratchpad** (Nye et al., 2021) externalizes intermediate computation as textual notes. We use it as a multi-step scratch space over the retrieved candidate evidence.
- **Self-Ask** (Press et al., 2023) decomposes a complex question into follow-up subquestions and answers them before producing the final answer. We implement the follow-up process over the per-example candidate context rather than external search.
- **Least-to-Most** (Zhou et al., 2022) solves simpler subproblems first and uses their answers to solve the original question. We instantiate the decomposition within the fixed hop budget of each dataset.

State accumulation baselines.

- **Step-Accumulate** keeps the raw evidence and notes collected at previous hops. In our setting, it appends hop-level evidence without learned write or read operators.
- **Running-Summary** (Lee et al., 2024) maintains a compact summary of the reasoning history. We update this summary across hops and pass the final summary to the answer model.

Memory baselines.

- **Mem0** (Chhikara et al., 2025) treats useful past information as retrievable memories for later use. We adapt it to the episode-local QA setting by storing hop evidence as memories within the current question.
- **A-Mem** (Xu et al., 2025) organizes agent experience as structured notes that can be retrieved in later steps. We adapt each hop into a note-style memory entry.
- **Memento** (Zhou et al., 2025) represents experience as an episodic trajectory that can guide later decisions. We instantiate it by treating the hop sequence of the current QA instance as the trajectory memory.

C.5 Evaluation Protocol

All methods are evaluated on the same dataset subsets, candidate-context source, answer model, and metric implementation. We report EM, token-level F1 (Rajpurkar et al., 2016; Yang et al., 2018), Acc, and $\mathrm{Acc}_{\mathrm{LLM}}$ (Zheng et al., 2023). EM and F1 follow standard QA normalization. Acc is a normalized containment score that checks whether the gold answer appears in the normalized prediction. $\mathrm{Acc}_{\mathrm{LLM}}$ is a binary semantic judgment from gpt-5.4-mini that compares the question, gold answer, and prediction.

D Pilot Study Details

We run the controlled mismatch study on the same three multi-hop QA datasets used in the main experiments. Gold hop sentences are used only in this diagnostic study to construct controlled intermediate states and canonical memories. They are not inserted into the prompts of the main evaluation.

Write-side probe. The goal is to test whether different source LLMs write interchangeable memory when they receive the same multi-hop state. For each example, we build a penultimate state from the question and all supporting hops except the last one. Each source LLM is then asked to produce the next compact action or memory entry for the missing bridge. The source pool is ordered as **gpt-5.4-mini**, **deepseek-reasoner**, **gemini-2.5-flash**, **claude-sonnet-4-6**, **qwen3.5-plus**, **grok-4-fast**, and **gpt-5-mini**. We vary the active source pool size from 2 to 7 by taking the first k models in this fixed order. Each written memory is paired with the same known evidence and passed to a fixed answer model, **gpt-5.4-mini**, so answer variation mainly reflects differences in the source-written memory rather than differences in the final answerer.

Read-side probe. The goal is to test whether a target LLM is sensitive to semantically equivalent memory presentations. For each example, we first construct a canonical memory from the question and the full set of supporting hops. A rewrite model, **gpt-5.4-mini**, generates exactly four alternative memory rewrites that should preserve the same entities, numbers, dates, and relations. We keep only rewrites that pass local semantic-preservation filters based on key-term coverage and token-overlap with the canonical memory. The default thresholds are 0.25 for key-term coverage and 0.20 for token-overlap, with at most three rewrite attempts. Each accepted rewrite is then fed separately to six target LLMs: **gpt-5.4-mini**, **deepseek-reasoner**, **gemini-2.5-flash**, **claude-sonnet-4-6**, **qwen3.5-plus**, and **grok-4-fast**. Unlike the write-side probe, there is no separate fixed answer model here; the target LLM both reads the memory variant and produces the final answer.

Metrics and aggregation. We report unmatched rate (UM) and Δ_{F1}. UM is the fraction of examples for which at least two variants produce non-matching final answers. The matching rule is local rather than LLM-judged: it normalizes punctuation, articles, and whitespace, checks dataset answer aliases, treats matching numeric answers as equivalent, and also accepts high token-overlap matches. Δ_{F1} is the difference between the best and worst token-level F1 among variants for the same example, computed against the gold answer and its aliases. The reported pilot curves average these quantities over the selected examples, with at most two repeated runs under different random seeds for robustness.

Results. Figure 1 shows that memory mismatch appears from both the write and read sides. In the write-side probe, increasing the number of source LLMs from 2 to 7 raises UM from 38.8 to 65.0 on HotpotQA, from 35.3 to 64.7 on 2WikiMultihopQA, and from 52.0 to 64.0 on MuSiQue. The corresponding Δ_{F1} also increases, reaching 39.6, 55.6, and 30.9 points on the three datasets, respectively. In the read-side probe, semantically related memory rewrites still lead to substantial answer variation across target LLMs, with average UM of 30.0 on HotpotQA, 20.8 on 2WikiMultihopQA, and 71.3 on MuSiQue. These trends support the motivation that both writing memory and reading memory should be conditioned on the LLM that produces or consumes it.

E Additional Experimental Results

E.1 Writer–Reader Compatibility Heatmap

Experimental Setup. This diagnostic experiment measures whether memory written by one LLM backbone remains easy to consume by another backbone. We run the experiment on HotpotQA and 2WikiMultihopQA. To make the writer–reader mismatch visible, we use seven heterogeneous API backbones: **gpt-5.4-mini**, **claude-sonnet-4-6**, **gemini-2.5-flash**, **deepseek-reasoner**, **qwen3.5-plus**, **gpt-4.1-mini**, and **gpt-4o-mini**. For each dataset, we sample 34 evaluation examples. For each example and each ordered writer–reader pair, the writer receives the question together with retrieved context paragraphs rather than oracle gold evidence, and is instructed to store only intermediate evidence without stating the final answer or final choice. The reader then consumes that memory and generates the final answer. This gives a 7×7 compatibility matrix, where rows correspond to memory writers and columns correspond to memory readers. We evaluate two memory conditions. In the raw-memory condition, the reader directly consumes the writer's unmodified memory. In the RoMem condition, the same raw memory is first canonicalized by the SFT-trained f_{write} using the writer profile and then adapted by the SFT-trained f_{read} using the reader profile. Each heatmap cell reports the average token-level F1, so the visualization directly shows both absolute answer quality and writer–reader compatibility. We also compare the diagonal and off-diagonal averages: a strong diagonal indicates that memories are easiest for the same backbone to read, while a smaller diagonal–off-diagonal gap indicates more uniform cross-backbone consumption.

Results. Figure 5 visualizes whether a memory written by one backbone can be reliably consumed by another. Under raw memory, the heatmaps show a clear same-backbone preference: the diagonal/off-diagonal F1 gaps are 13.7 points on HotpotQA and 25.9 points on 2WikiMultihopQA. This indicates that raw memory is much easier to reuse when the writer and reader are the same or closely aligned. RoMem produces a more even compatibility matrix while also improving average F1, from 29.7 to 57.3 on HotpotQA and from 38.8 to 75.1 on 2WikiMultihopQA. The corresponding RoMem gaps are close to zero (-0.4 and -0.2 points), suggesting that the learned write/read interface substantially reduces the writer-specific diagonal pattern.

E.2 Worst-Case Gain Analysis

Experimental Setup. This diagnostic experiment complements the average scores in Table 1 by measuring whether RoMem improves the worst served target readers under the minimum-gain motivation in §4.2. We run the analysis on HotpotQA and 2WikiMultihopQA with 34 held-out evaluation examples per dataset. The model pool is the same six seen API profiles used in the main experiments: **gpt-5.4-mini**, **gpt-5.4-nano**, **gemini-2.5-flash**, **gemini-2.5-flash-nothinking**, **claude-sonnet-4-6**, and **claude-sonnet-4-6-thinking**. For each target reader t, we fix the final model in the assignment sequence to t and randomly sample three source-to-target assignment sequences from the remaining seen profiles using a fixed seed. HotpotQA uses two-step assignments of the form source $\rightarrow$ target, while 2WikiMultihopQA uses three-step assignments of the form source$_1$ $\rightarrow$ source$_2$ $\rightarrow$ target, with no repeated source model within a sequence. For each sequence and each example, we evaluate Step-Accumulate and RoMem under the same assignment. Step-Accumulate passes the accumulated raw step notes to the final target reader, while RoMem stores each source step through f_{write} and adapts the memory bank to the final target through f_{read}. For each target reader, we compute token-level F1 for each sampled assignment and report the worst F1 obtained by Step-Accumulate and RoMem across the three assignments.

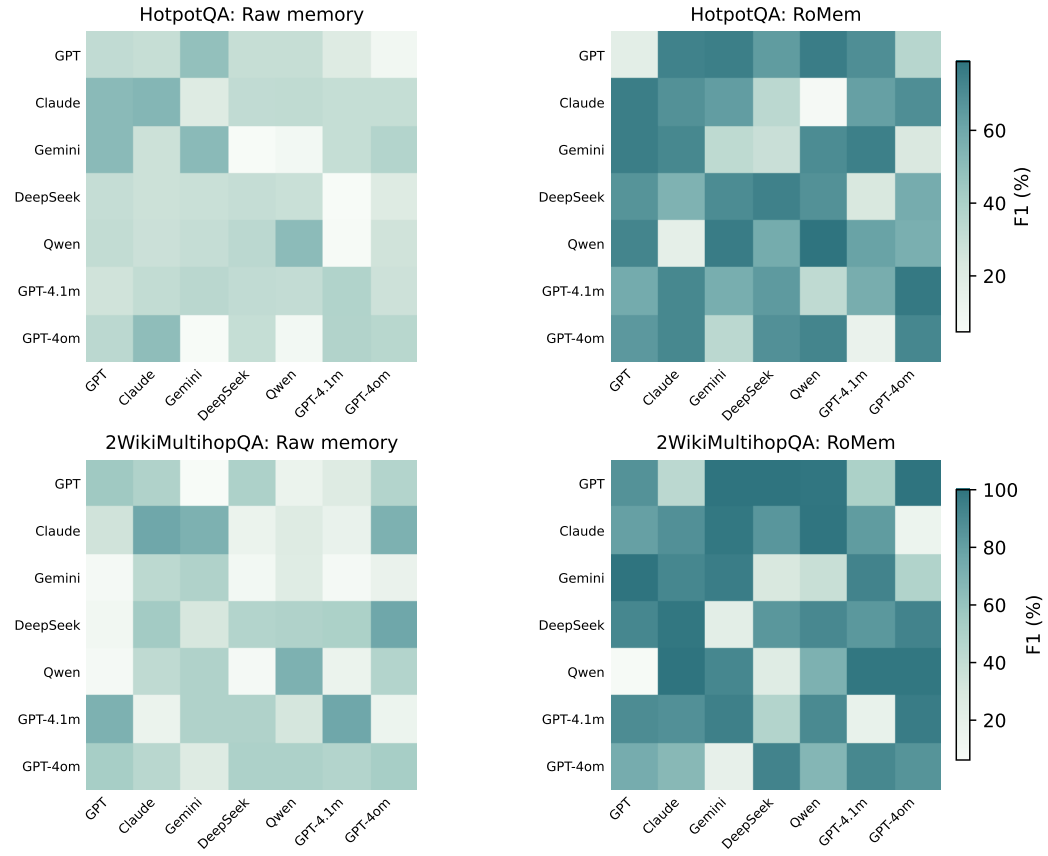


Figure 5: Writer–reader compatibility heatmaps on HotpotQA and 2WikiMultihopQA. Each cell reports average token-level F1 under retrieved-context memory writing. Rows denote source memory writers and columns denote target readers.

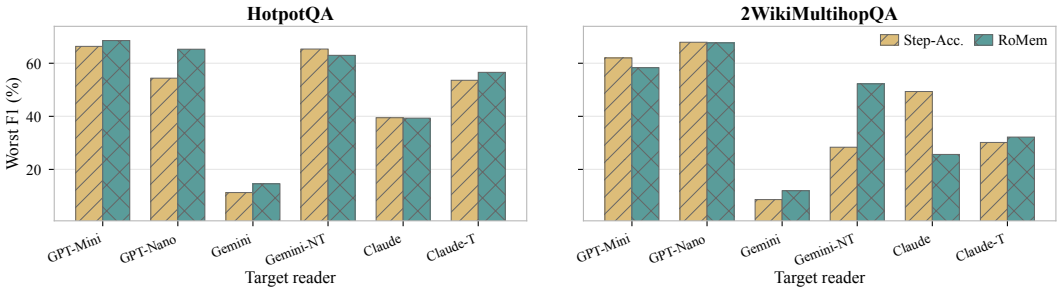


Figure 6: Worst-case target-reader performance on HotpotQA and 2WikiMultihopQA. For each target reader, we report the lowest F1 across three sampled source-to-target assignment sequences.

Results. Figure 6 shows that RoMem improves the worst-sequence F1 for four of six target readers on HotpotQA and three of six target readers on 2WikiMultihopQA. On HotpotQA, RoMem raises the worst-case score for GPT-Mini, GPT-Nano, Gemini, and Claude-Thinking, with the largest gain appearing for GPT-Nano. On 2WikiMultihopQA, the largest improvement appears for Gemini-NT, where RoMem substantially raises the weakest assignment, followed by smaller gains for Gemini and Claude-Thinking. The result also reveals residual failure cases: GPT-Mini and Claude on 2Wiki, together with a few nearly tied targets, still favor Step-Accumulate. Thus, RoMem improves many worst-served readers, but the learned interface does not eliminate all assignment-level regressions.

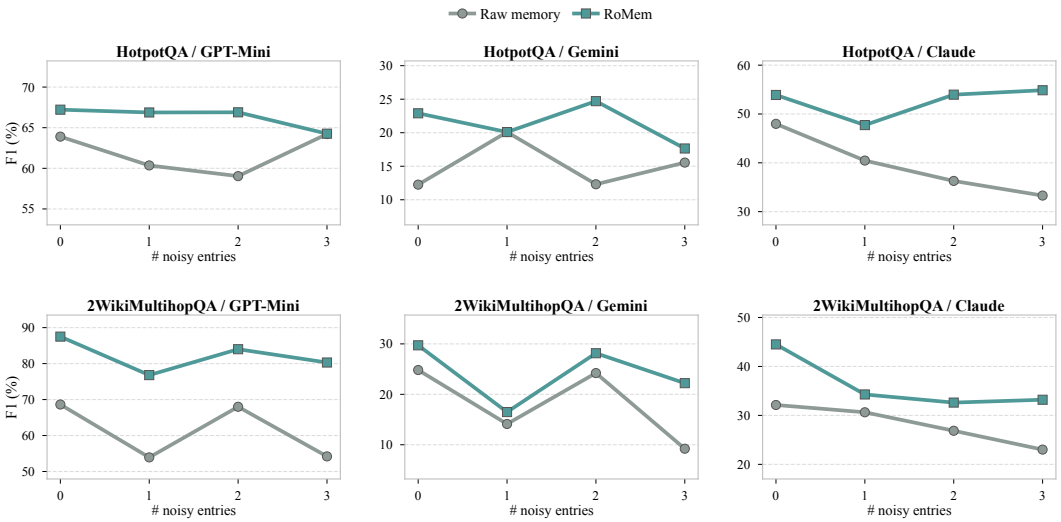


Figure 7: Noisy memory robustness on HotpotQA and 2WikiMultihopQA across three target readers. The x-axis is the number of injected noisy memory entries, and the y-axis reports F1.

E.3 Noisy Memory Robustness

Experimental Setup. This diagnostic experiment tests whether RoMem's read operator can filter useful evidence from a polluted memory bank, a common failure mode in long-running agent memory. Following the motivation of noisy-memory evaluations in agent memory studies, we inject distractor, stale, and conflicting facts into the memory bank and compare raw memory consumption against RoMem's profile-conditioned reading. We run the experiment on HotpotQA and 2WikiMultihopQA with 34 held-out examples per dataset. For the target readers, we use three representative seen API profiles from the main model pool: **gpt-5.4-mini**, **gemini-2.5-flash**, and **claude-sonnet-4-6**. For each example, the clean memory bank is initialized from the gold hop evidence. We then inject $n \in \{0, 1, 2, 3\}$ noisy memory entries and shuffle the resulting memory order. The injected entries cycle through three noise types: distractor facts sampled from non-supporting retrieved context paragraphs, stale facts produced by perturbing dates, numbers, or entity attributes in clean evidence, and conflicting facts that contradict a key relation or answer-bearing statement. Both methods receive the same shuffled noisy memory bank. The raw-memory baseline directly concatenates the memory bank and passes it to the target reader, while RoMem first applies f_{read} with the target profile and then passes the adapted context to the same target reader. For each dataset, target reader, and noise level, we report the average token-level F1 over the 34 examples.

Results. Figure 7 shows that RoMem achieves higher F1 than raw memory for nearly all target-reader and noise-level combinations. On HotpotQA, average gains over the four noise settings are 4.4, 6.3, and 13.1 F1 points for GPT-Mini, Gemini, and Claude, respectively. On 2WikiMultihopQA, the gains are 21.0, 6.1, and 8.0 points, again with the clearest and most stable margin on GPT-Mini. The RoMem curves generally stay above raw memory as more distractor, stale, or conflicting entries are added, although they are not strictly monotonic with the amount of injected noise. These trends support the role of f_{read} as a target-conditioned filter that can reduce the impact of distractor, stale, and conflicting memory entries.