GRAM: Empowering Agent with Actively Managed Graph-Structured Memory via Reinforcement Learning

Anonymous ACL submission

Abstract

Large Language Model (LLM) agents face a fundamental bottleneck called *context flooding* in multi-turn information retrieval: as evidence accumulates across interaction turns, their context windows become saturated with redundant, conflicting, and outdated facts. Existing research either compresses interaction history into rolling summaries, sacrificing relational precision, or relies on retrieval-augmented pipelines that lack mechanisms for resolving knowledge conflicts and maintaining a globally consistent belief state. In this work, we introduce GRAM, a reinforcement learning framework designed to train an agent to actively manage and utilize a dynamic graph-structured memory. Rather than passively accumulating retrieved text, the agent learns to incrementally construct and revise a coherent knowledge graph that serves as its persistent belief state, resolving contradictions and preserving logical relationships in memory. Furthermore, We employ Group Relative Policy Optimization to train small language models (under 4B) to master these complex memory-governance behaviors. Experimental results across multiple mainstream question-answering benchmarks demonstrate the superiority of the GRAM framework over existing agentic memory baselines and conventional RAG systems.

1 Introduction

LLMs have demonstrated remarkable capability across a wide range of knowledge-intensive and reasoning tasks, with recent advances in long-context modeling further expanding their applicability to complex scenarios such as multi-document question answering, tool-augmented reasoning, and autonomous agents. However, these capabilities often break down in *incremental knowledge acquisition* scenarios like multi-turn web search, where information arrives incrementally across multiple turns rather than being provided in its entirety at once.

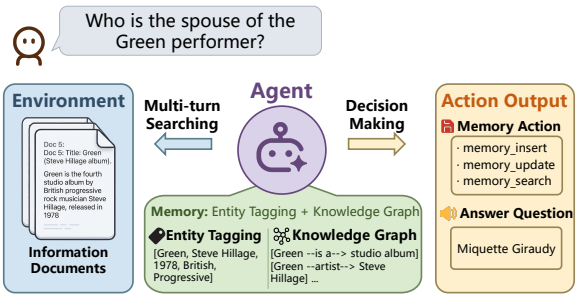

Figure 1: Overview of the multi-document question answering problem.

As information accumulates across turns, the context window becomes saturated with redundant, conflicting, and outdated facts, leading to *context flooding* that degrades reasoning accuracy through the "lost-in-the-middle" effect (Liu et al., 2024; Yi et al., 2025) and increases both inference latency and retrieval cost (Xiao et al., 2023).

To address these challenges, previous research (Zhou et al., 2025; Sun et al., 2025; Wu et al., 2025) primarily focuses on context compression, such as summarization or rolling memory. Such emphasis on context length reduction often leads to information loss, especially in tasks requiring precise fact retrieval or multi-hop reasoning. Additionally, some latest studies employ Retrieval-Augmented Generation (RAG) offers an alternative by dynamically selecting relevant context from an external corpus, but it suffers from a critical limitation: retrieval alone does not guarantee that the *right* information is surfaced at the *right time*. Furthermore, current work aims at (Chhikara et al., 2025; Wang et al., 2025) reconstructing the memory structures in agent from passive storage to an actively maintained belief state. They often rely on a rigid two-stage process that processes a static corpus before reasoning begins, which disconnects information gathering from immediate reasoning needs and fails to facilitate real-time belief revision as new evidence emerges. Across all three

paradigms, a shared deficiency persists: none provides a mechanism for *explicit belief revision*, leaving accumulated contradictions and outdated facts unresolved. Therefore, the key capability of an agent in long-horizon tasks is not only *how to read*, but also *how to retain*, *how to revise*, and *how to discard*.

In this work, we propose **GRAM** (**G**raph-**R**etrieval **A**gentic **M**emory), a framework that equips an LLM agent with a dynamic graph-structured memory and a set of explicit cognitive actions: `Insert`, `Update`, and `Search`. As is shown in Figure 1, instead of passive storage, the agent actively constructs and maintains a persistent belief state by integrating new data and resolving contradictions in real time. Crucially, the `Update` operation allows the agent to prune outdated facts or reconcile inconsistencies, ensuring the context remains concise and logically consistent throughout the process of incremental acquisition. Furthermore, training agent to master such memory-governance behaviors poses a fundamental challenge: the optimal sequence of memory operations is rarely annotated, and supervised learning provides no clear signal for long-horizon memory decisions. Therefore, we formulate GRAM as a reinforcement learning problem and train the agent using Group Relative Policy Optimization (GRPO) (Shao et al., 2024). This allows the agent to learn effective memory management strategies directly from task-level supervision. Experiments show that the trained agent indeed develops a strategy of proactive information consolidation, minimizing the need for expensive searches.

Our main contributions are summarized as follows:

- We introduce GRAM, a reinforcement learning framework that enables LLM agents to actively manage a dynamic graph-structured memory in incremental knowledge acquisition environments, treating memory governance as a sequential decision-making problem.
- We design a set of explicit cognitive actions, including `Insert`, `Update`, and `Search` that allow the agent to acquire knowledge incrementally by constructing, revising, and querying a coherent knowledge graph as its persistent belief state, with `Update` providing explicit conflict resolution absent from prior approaches.
- Experimental results demonstrate that our GRAM driven by small language models, training through only task-level reward, can acquire sophisticated memory governance behaviors and substantially outperform existing methods on complex multi-hop reasoning benchmarks.

2 Related Work

2.1 Graph-Augmented LLMs

Integrating Knowledge Graphs (KGs) with LLMs has emerged as a promising direction to mitigate hallucinations and enhance multi-hop reasoning capabilities (Pan et al., 2024). Early approaches, such as RAG (Retrieval-Augmented Generation) (Lewis et al., 2020), primarily rely on vector similarity to retrieve relevant text chunks. However, these methods often struggle with "reasoning across documents" or detecting structural patterns that are not semantically adjacent.

To address this, Graph-Augmented methods have been proposed. Systems like GraphRAG (Edge et al., 2024) and G-Retriever (He et al., 2024) leverage pre-constructed community summaries or subgraph retrieval to provide structured context. Think-on-Graph (ToG) (Sun et al., 2023) and MindMap (Wen et al., 2024) further enable LLMs to perform beam search or random walks over static KGs to reason explicitly. However, a common limitation of these approaches is their reliance on static, pre-defined graphs. The graph is typically assumed to be a read-only external database provided beforehand. In contrast, our work focuses on the incremental knowledge acquisition scenarios, where the agent must dynamically construct and evolve the graph memory from scratch while processing sequential information, handling conflicts and redundancy in real-time.

2.2 LLM Agents with Active Memory

Standard LLMs are constrained by fixed context windows, leading to the development of agents with active memory management. MemGPT (Hong et al., 2023) introduces an OS-like hierarchy, allowing agents to manage context via virtual memory paging, effectively extending the context window indefinitely. Generative Agents (Park et al., 2023) utilize a reflection mechanism to store and retrieve synthesized text logs in a vector database. More recently, Mem1 (Zhou et al., 2025) trains agents to maintain a "working memory" in the form of a

compressed text summary, updating it iteratively via reinforcement learning.

While these methods enable long-horizon interactions, their memory representations are predominantly unstructured or semi-structured. Textual summaries often suffer from information loss or "fuzziness" when handling precise, multi-hop dependency chains. Furthermore, most agents lack explicit mechanisms to correct stored errors once written. Our method, GRAM, differentiates itself by employing a structured graph memory. By training with **PPO** (Schulman et al., 2017) or **GRPO** (Shao et al., 2024), our agent learns discrete actions to strictly manage knowledge triplets. This structure offers superior precision for multi-hop reasoning and enables specific "surgical" updates to resolve conflicts, a capability often absent in summary-based memory systems.

3 Method

Given a user query q and a list of documents $D = \{d_1, d_2, \ldots, d_L\}$, the agent functions as a policy $\pi_\theta(a_t|s_t)$ that interacts with an evolving Knowledge Graph $\mathcal{G}$. At each time step t, the agent observes a state s_t, which encapsulates the query, a textual summary of the current graph topology, and the current document d_k. Our GRAM framework, characterizing the dynamic graph memory, the discrete agentic action space, and the Group Relative Policy Optimization (GRPO) training pipeline, aims at generating an optimal trajectory of actions to construct a coherent answer, as illustrated in Figure 2.

3.1 Dynamic Graph Memory

Unlike traditional RAG systems that rely on static vector databases or linear context buffers, GRAM maintains a dynamic knowledge graph for actively memory management. Formally, at any time step t, the memory is denoted as a directed graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, where $\mathcal{V}_t$ represents the set of unique entities and $\mathcal{E}_t \subseteq \mathcal{V}_t \times \mathcal{R} \times \mathcal{V}_t$ denotes the set of semantic relations. This structure supports two primary cognitive functions: topology evolution for belief revision and structure-aware retrieval for multi-hop reasoning.

Topology Evolution. The topology of the graph is designed for self-organization through non-monotonic state transitions. Let $\mathcal{T}_{in}$ be the set of new knowledge triplets extracted from the current document. The state transition from $\mathcal{G}_t$ to $\mathcal{G}_{t+1}$ is governed by a policy that first performs Online Entity Alignment to resolve entity aliasing. For a new mention m, a linking function $\phi : \mathcal{V}_{new} \to \mathcal{V}_t$ is employed; if the semantic similarity between m and an existing node $v \in \mathcal{V}_t$ exceeds a threshold δ, the mention is merged. This process ensures that fragmented attributes are aggregated into a unified representation:

$$v_{new} = \text{Aggr.}\left(\{m \mid \text{sim}(m, v) > \delta\} \cup \{v\}\right) \quad (1)$$

Furthermore, to address the contradictions inherent in incremental documents, the policy facilitates explicit conflict resolution. Unlike standard memory architectures that accumulate facts monotonically ($\mathcal{G}_{t+1} \supseteq \mathcal{G}_t$), GRAM allows the agent to identify a set of edges to add (Δ_t^+) and a set of obsolete or conflicting edges to prune (Δ_t^-). This results in a refined state transition:

$$\mathcal{G}_{t+1} = (\mathcal{G}_t \cup \Delta_t^+) \setminus \Delta_t^- \quad (2)$$

This mechanism allows the agent to actively resolve conflicts and maintain a globally consistent state that reflects the latest valid information.

Structure-Aware Retrieval. To mitigate the information loss common in compressed contexts, GRAM utilizes a structure-aware retrieval protocol. When the agent executes a search action, it triggers a query to retrieve a local subgraph $\mathcal{G}' \subset \mathcal{G}_t$ rather than relying on isolated vector similarity. This protocol is specifically designed to uncover latent connections through multi-hop bridging. By defining $\mathcal{P}_{i,k}$ as the set of relational paths connecting entities v_i and v_k up to a maximum length L, the retrieval function traverses the graph to identify logical dependencies:

$$\mathcal{P}_{i,k} = \{(v_i \xrightarrow{r_1} v_j \ldots \xrightarrow{r_L} v_k) \in \mathcal{G}_t\} \quad (3)$$

By recovering these paths, the agent identifies logical dependencies between entities that never co-occurred in the same source document. The retrieved topological information $\mathcal{G}'$ is then linearized into a coherent summary sequence $S = \text{Linearize}(\mathcal{G}')$, augmenting the agent's working memory with precise, logic-oriented evidence independent of the document stream's temporal order.

3.2 Agentic Action Space

The information flow within GRAM is governed by a discrete action space:

$$\mathcal{A} = \{\texttt{Insert}, \texttt{Update}, \texttt{Search}, \texttt{Answer}\} \quad (4)$$

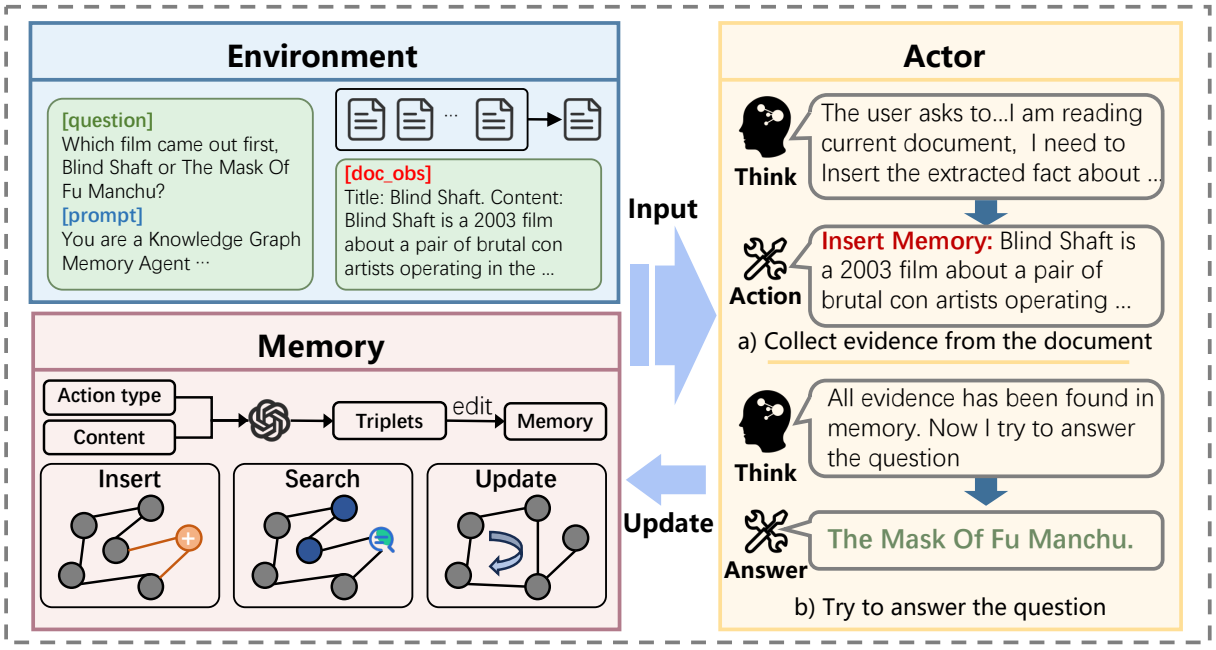

Figure 2: **Overview of the GRAM Framework.** The agent processes a sequential document stream to construct a dynamic Knowledge Graph (KG) for complex reasoning. At each step, the agent observes the query, the current document, and a textual summary of the graph state. Following the ReAct mode, the agent first generates a think trace to analyze information gaps, and then executes a symbolic action.

At each step, the agent follows the ReAct mode (Yao et al., 2022): generating a reasoning trace z_t to evaluate the current state before sampling a single atomic action $a_t \in \mathcal{A}$. These actions are categorized into knowledge maintenance and information navigation.

Knowledge Maintenance Action facilitate the integration of new evidence from the current document d_k into the graph $\mathcal{G}$. The `Memory_Insert` action extracts atomic triplets from d_k to expand the topology of the graph. Alternatively, the `Memory_Update` action allows the agent to issue natural language instructions to revise or prune outdated nodes and edges in $\mathcal{G}_{t-1}$. This ensures the consistency of the belief state when conflicting information is encountered. Once either maintenance action is completed, the system assumes the current document has been processed and advances the pointer to d_{k+1}.

Information Navigation Action allow the agent to pause the traversal of the document to gather context or terminate the task. The `Memory_Search` action performs a retrieval operation on $\mathcal{G}_{t-1}$. Critically, this action does not advance the document pointer but enriches the context of the agent with retrieved relational paths to support the understanding of d_k in subsequent steps. Finally, the `Answer` action terminates the interaction trajectory and generates a response based on the accumulated evidence.

3.3 Reinforment Learning Pipeline

While most optimal sequence of memory operations is not explicitly annotated, We model the interaction as a Markov Decision Process (MDP) and optimize it through Reinforcement Learning (RL). Specifically, we employ Group Relative Policy Optimization (GRPO) as the RL algorithm. For each query Q, we sample a group of trajectories $\{\tau_1, \ldots, \tau_K\}$ from the current policy π_θ. The objective function maximizes the advantage of the generated outputs relative to the group average:

$$\mathcal{J}(\theta) = \mathbb{E}_{Q\sim\mathcal{D},\epsilon\sim\pi_\theta} \left[\frac{1}{K} \sum_{i=1}^{K} \min \left(\frac{\pi_\theta(\tau_i)}{\pi_{old}(\tau_i)} \hat{A}_i, \text{clip}(\ldots) \hat{A}_i \right) \right] \tag{5}$$

where $\hat{A}_i$ is the standardized reward advantage.

Reward Formulation We design a compound reward using two components: an outcome reward and a process reward. The $\lambda > 0$ term is a tunable knob balancing the preference for answer correctness and process quality, and we set $\lambda = 0.1$. The reward is calculated as $R = R_{\text{outcome}} + \lambda R_{\text{process}}$.

- **Outcome Reward (R_{outcome})**: Evaluates the final answer correctness using the token-level F1-score against the groundtruth. It is defined as:

$$R_{\text{outcome}} = \frac{2 \cdot \text{Precision}(\hat{y}, y^*) \cdot \text{Recall}(\hat{y}, y^*)}{\text{Precision}(\hat{y}, y^*) + \text{Recall}(\hat{y}, y^*)} \tag{6}$$

where $\hat{y}$ is the predicted answer generated by the agent and y^* is the groundtruth.

- **Process Reward (R_{process})**: To prevent reward hacking and ensure structural compliance, we primarily rely on a format reward. It assigns a strict penalty if the agent generates invalid XML tags or outputs multiple action tags in a single turn. This guides the SLM to adhere to the strict single-action protocol defined in Section 3.2. The process reward is formulated as a step penalty function:

$$R_{\text{process}} = \begin{cases} 1, & \text{if action tags are valid,} \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

4 Experiment

4.1 Experiment Setup

Dataset We evaluate GRAM on four question answering benchmarks spanning single-hop and multi-hop reasoning: TriviaQA (Joshi et al., 2017), HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), and MuSiQue (Trivedi et al., 2021). To simulate incremental information settings, we adapt these datasets by presenting supporting and distractor documents sequentially rather than providing full context at once. As shown in Table 1, TriviaQA serves as a single-hop baseline with well-localized context, while multi-hop datasets like MuSiQue impose substantially greater demands. These benchmarks range from simple factoid retrieval to complex multi-step reasoning, rigorously testing GRAM's ability to maintain coherent graph memory under long-horizon retrieval scenarios.

Evaluation Metrics For answer quality evaluation, we use the *F1 Score*, which measures the average word overlap between the prediction and the groundtruth. For cost-effectiveness evaluation, we track the *Average Interaction Turns (AIT)* to evaluate efficiency not by absolute turn counts alone, but by how well the agent allocates its interaction budget relative to task complexity, avoiding both redundancy on simple queries and premature termination on complex ones.

Dataset	# Examples	Avg # Docs	Avg # Steps
TriviaQA	16853	1	1
HotpotQA	7405	9.94	2.43
2WikiMultihop	12576	10	2.44
Musique	2417	20	2.65

Table 1: Statistics of the Evaluation Datasets. **Avg # Docs** denotes the average number of documents per sample, and **Avg # Steps** represents the average number of supporting facts required to deduce the final answer.

Baselines We compare GRAM against a range of baselines, from direct prompting to advanced agentic frameworks. The baselines fall into three categories: i) direct generation methods (Zero-shot and CoT (Wei et al., 2022)), which test the model's parametric knowledge without any retrieval; ii): retrieval-augmented generation methods (Standard RAG and IRCoT (Trivedi et al., 2023)), which ground the model in external document chunks; iii): agentic methods (Search-R1 (Jin et al., 2025) and MEM1 (Zhou et al., 2025)), which give the model active control over its information-gathering process. Note that all methods use Qwen2.5 as the backbone LLM to ensure a fair comparison. Please refer to the appendix for more implementation details, e.g., Appendix A for detailed baseline decription and training setups and Appendix C for used prompts.

4.2 Overall Performance

Table 2 presents the comparative performance of GRAM against state-of-the-art baselines across four benchmarks.

Performance Comparison against Baselines. GRAM consistently outperforms traditional retrieval (RAG), linear reasoning (IRCoT), and agentic baselines, demonstrating its most significant advantage on complex multi-hop datasets. On the highly challenging Musique benchmark, GRAM-3B achieves an F1 score of 0.316, reflecting a 6.7× improvement over standard RAG (0.047) and substantially outperforming Search-R1-3B (0.146). Similarly, it surpasses Search-R1 by over 11 points on 2Wiki. These gains highlight how GRAM's active memory management and explicit graph structure effectively mitigate the "lost-in-the-middle" phenomenon and "reasoning drift" inherent to passive context stuffing, successfully preserving rela-

Method	Model Size	TriviaQA	HotpotQA	2Wiki	Musique
Zero-shot	3B	0.288	0.149	0.244	0.020
	7B	0.408	0.183	0.250	0.031
RAG	3B	0.544	0.255	0.226	0.047
	7B	0.585	0.299	0.235	0.058
CoT (Wei et al., 2022)	3B	0.185	0.092	0.111	0.022
	7B	0.032	0.021	0.021	0.002
IRCoT (Trivedi et al., 2023)	3B	0.312	0.164	0.171	0.067
	7B	0.478	0.133	0.149	0.072
Search-R1 (Jin et al., 2025)	3B	0.545	0.324	0.319	0.103
	7B	0.610	0.370	0.414	0.146
Mem1 (Zhou et al., 2025)	3B	0.767	0.638	0.301	0.129
	7B	**0.819**	**0.709**	0.321	0.146
Ours	3B	0.792	0.637	**0.528**	**0.316**

Table 2: Evaluation results on four datasets. Best in **Bold** and second best in Underline.

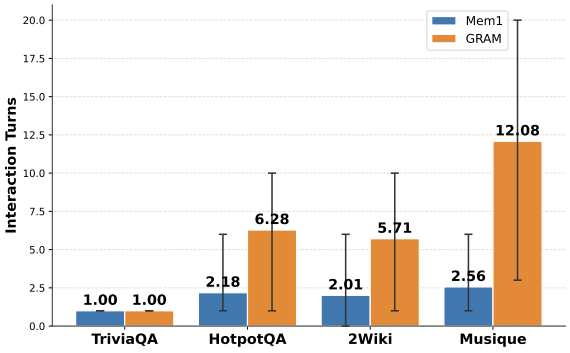

Figure 3: Comparison of Average Interaction Turns across different benchmarks. GRAM adapts its trajectory length based on task complexity, demonstrating efficiency on simple tasks and persistence on complex multi-hop queries.

Setting	TriviaQA	HotpotQA	2Wiki	Musique
w/o Update	0.763	0.628	**0.592**	**0.447**
w/ Update	**0.792**	**0.637**	0.528	0.316

Table 3: Ablation study on the impact of the Graph Update mechanism. w/o Update denotes an insert-only strategy.

Setting	F1 Score	Token Consumption
w/o Graph	0.568	11,240
w/ Graph	**0.637**	**6,450**

Table 4: Ablation Study on Memory Structure. Comparison between a linear text buffer (*w/o Graph*) and graph memory (*w/ Graph*).

tional bridges across fragmented evidence.

Parameter Efficiency and Structural Fidelity. Despite using a compact 3B backbone, GRAM achieves performance competitive with considerably larger models. On TriviaQA, GRAM-3B (0.792) is comparable to 7B-scale Mem1 (0.819), indicating that structured graph representations retain the fine-grained information necessary for precise fact extraction, a property that rolling summary approaches tend to compromise. On HotpotQA, GRAM substantially narrows the gap against larger or more specialized models, suggesting that equipping smaller LLMs with a persistent, dynamic belief state is an efficient route to strong multi-hop reasoning without proportional increases in computational cost.

Adaptive Interaction Efficiency. Beyond accuracy, we examine agent behavior through Average Interaction Turns (AIT), illustrated in Figure 3. GRAM's reasoning depth scales with task complexity. On single-hop tasks such as TriviaQA, it matches the efficiency of summary-based baselines (averaging 1.00 turn), showing that graph maintenance does not add unnecessary overhead for simple factual queries. On complex multi-hop tasks such as Musique, GRAM invests substantially more turns (12.08) compared to Mem1 (2.56). This difference reflects GRAM's resistance to premature termination and hallucination: rather than collapsing the reasoning process early, it continues to chain intermediate steps until a complete evidence path is constructed, which directly accounts for its superior F1 performance on these tasks.

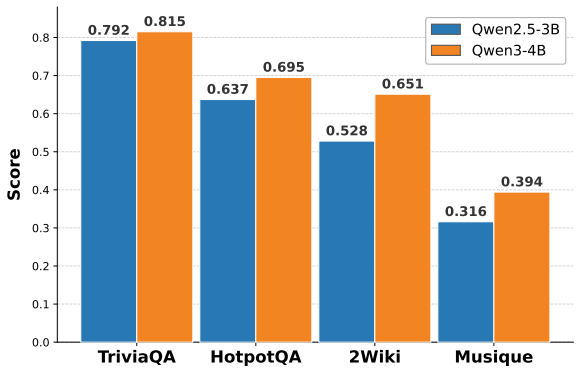

Figure 4: Performance of our GRAM framework with different sizes of base models.

4.3 Ablation Studies

To dissect the contribution of each component in GRAM, we conduct ablation studies on the HotpotQA dataset of our GRAM framework driven by Qwen2.5-3B.

Impact of Memory Update Mechanism. To investigate the necessity of dynamic graph modification, we compare our full model (GRAM) against a variant without the `<memory_update>` action (denoted as *w/o Update*), which adopts an append-only strategy for knowledge accumulation. Table 3 summarizes the results. We observe a distinct performance divergence across different dataset types:

- Benefits for Fact Retrieval: Enabling the update mechanism yields performance gains on TriviaQA (+2.9%) and HotpotQA (+0.9%). These datasets often require precise answer extraction where conflicting or outdated information in the graph can be detrimental. The update action allows the agent to correct errors and refine existing nodes, leading to higher precision in final answers.

- Challenges in Complex Multi-hop Reasoning: Conversely, on datasets requiring extensive multi-hop reasoning like 2WikiMultihop and Musique, the append-only strategy (*w/o Update*) outperforms the full model. We hypothesize that frequent updates during complex reasoning chains may inadvertently disrupt the graph's structural integrity. In multi-hop scenarios, "history" or seemingly redundant nodes often serve as crucial intermediate bridges. Aggressive updating might prune these bridges, breaking the reasoning path and causing the significant drop observed in Musique (-13.1%).

These results suggest that while the update mechanism is essential for maintaining a concise and consistent memory state (vital for long-term interacting agents), it introduces a risk of structural instability in static, complex reasoning benchmarks. An append-only strategy provides a safer, albeit potentially noisier, context for deep multi-hop traversal.

Impact of Memory Structure. To isolate the specific contribution of the memory topology, we conduct an ablation study by replacing the graph memory with a linear text buffer (denoted as *w/o Graph*). The full GRAM model is referred to as *w/ Graph*. In the *w/o Graph* variant, the `Memory_Insert` action simply appends raw text segments (or summaries) to a list, and `Memory_Search` performs dense retrieval over these text chunks, while maintaining the same RL training pipeline and action interface.

- Improved Accuracy with Superior Efficiency: As detailed in Table 4, the *w/ Graph* setting outperforms the *w/o Graph* variant in accuracy, validating that structured topology supports more effective reasoning than unstructured text buffers. More importantly, a critical disparity emerges in Token Consumption.

- High Information Density: The *w/ Graph* model reduces the average token usage per query by approximately 42.6% compared to the *w/o Graph* setting. This efficiency stems from the graph's nature as a form of symbolic compression.

We attribute this efficiency gain to two inherent advantages of the graph structure over linear text:

- Deduplication: The graph naturally merges redundant mentions of the same entity into a single canonical node. In contrast, the *w/o Graph* setting must store and retrieve multiple repetitive text chunks to capture the same information.

- Precise Retrieval: A graph query allows the agent to traverse and retrieve specific semantic paths, which are informationally dense. The *w/o Graph* setting, conversely, relies on retrieving entire document passages to convey

Query: *"Who is the spouse of the Green performer?"*
Context (Step 1): [Doc 1] Title: Grant's First Stand. Content: Grant's First Stand is the debut album by American jazz guitarist Grant Green featuring performances by Green recorded... **Agent: [Thought]** The document mentions "Grant Green", but this refers to a jazz guitarist. The query uses "The Green performer" which likely refers to a specific work titled "Green" rather than a surname. **[Action]** `<memory_insert>None</memory_insert>` (Skips insert)
...
Context (Step 5): [Doc 5] Title: Green (Steve Hillage album). Content: Green is the fourth studio album by British progressive rock musician Steve Hillage, released in 1978... **Agent: [Thought]** Found a critical link. "Green" is an album by Steve Hillage. Therefore, "The Green performer" likely refers to Steve Hillage. I will save this raw fact into memory. **[Action]** `<memory_insert>Green is the fourth studio album by British progressive rock musician Steve Hillage</memory_insert>`
Context (Step 6): [Doc 6] Title: Miquette Giraudy. Content: ...Miquette Giraudy is a keyboard player... best known for her work in Gong and with her partner Steve Hillage... **Agent: [Thought]** This document identifies "Miquette Giraudy" as the partner (spouse) of "Steve Hillage". My memory index currently lists the entity "Steve Hillage", but I must search the graph to retrieve the exact facts confirming he is the "Green" performer before drawing a final conclusion. **[Action]** `<memory_search>Steve Hillage Green</memory_search>`
Context (Step 7): [Memory Search Results]: [Steve Hillage] –[performer of]–> [Green (album)]. **Agent: [Thought]** The search results explicitly confirm that Steve Hillage is the performer of the album "Green". Splicing this retrieved memory with the clue from the previous step (his partner is Miquette Giraudy), the logical chain is complete. **[Action]** `<answer>Miquette Giraudy</answer>`

Table 5: Case study of robustness to noise, multi-hop linking, and active memory retrieval (Qwen2.5-3B).

the same simple relationship. This introduces significant "token noise," inflating the prompt context with irrelevant details that do not contribute to the answer.

4.4 Further Analysis of Model Size

To verify the generality of our GRAM framework and its applicability across models of varying sizes, we conduct comparative experiments utilizing the Qwen (Yang et al., 2025) model family, specifically Qwen2.5-3B and Qwen3-4B, as alternative base models. As is shown in 4, Qwen3-4B consistently outperforms Qwen2.5-3B across all four benchmarks, with improvements of 2.3, 5.8, 12.3, and 7.8 percentage points on TriviaQA, HotpotQA, 2WikiMultiHopQA, and Musique, respectively. The performance gap is particularly pronounced on the more challenging multi-hop reasoning benchmarks (2WikiMultiHopQA and Musique), suggesting that a larger model capacity provides a greater benefit when the task demands complex compositional reasoning. Notably, both models achieve competitive results under the GRAM framework, demonstrating that GRAM is not tailored to a specific model scale but generalizes effectively across different parameter sizes. However, we ultimately chose Qwen2.5-3B as the base model because it is commonly used in most existing fine-tuning-based methods, ensuring fairness by avoiding discrepancies arising from differences in the capabilities of base models.

4.5 Case Study

Table 5 illustrates a reasoning trajectory of our model answering a complex multi-hop query. The case demonstrates the agent's robustness to noise by correctly ignoring irrelevant distractors (Step 1). More importantly, it highlights the agent's active retrieval capability: rather than relying on hallucinated connections, the agent explicitly searches its memory index (Step 6) to verify past facts, successfully splicing the retrieved evidence with new context to deduce the final answer (Step 7). We present a more detailed analysis of specific cases in Appendix B.

5 Conclusion

In this paper, we propose GRAM, a framework that equips LLM agents with an actively managed graph-structured memory to address the bottleneck of context flooding in multi-turn information retrieval. Specifically, we introduce a set of explicit cognitive actions: Insert, Update, and Search, which enable the agent to incrementally construct and revise a coherent knowledge graph as its persistent belief state. We further employ GRPO algorithm to train the agent to master these memory-governance behaviors, allowing it to resolve knowledge conflicts and maintain relational precision. Our experiments on multiple mainstream question-answering benchmarks demonstrate that GRAM substantially outperforms existing agentic memory baselines and conventional RAG systems, particularly on complex multi-hop reasoning tasks.

Limitations

While GRAM demonstrates significant advantages, our in-depth ablation studies reveal an important trade-off between memory consistency and structural preservation. Although belief revision is essential for maintaining a concise and accurate memory state, overly aggressive updates can inadvertently disrupt critical reasoning paths required in complex multi-hop scenarios. This finding highlights that memory governance should be viewed as a nuanced control problem rather than a purely engineering concern. Promising directions for future work to address these limitations include developing uncertainty-aware memory updates and designing adaptive policies conditioned on task complexity to better balance graph stability with real-time knowledge revision.

References

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

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Advances in Neural Information Processing Systems*, 37:132876–132907.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2023. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *Preprint*, arXiv:2503.09516.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

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.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M Ni, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. *arXiv preprint arXiv:2307.07697*.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. 2025. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2021. Musique: Multihop questions via single-hop question composition, 2022. *URL https://arxiv. org/abs/2108.00573*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 10014–10037.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. 2025. Mem-alpha: Learning memory construction via reinforcement learning. *Preprint*, arXiv:2509.25911.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yilin Wen, Zifeng Wang, and Jimeng Sun. 2024. Mindmap: Knowledge graph prompting sparks graph of thoughts in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10370–10388.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, and 1 others. 2025. Resum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Zihao Yi, Delong Zeng, Zhenqing Ling, Haohao Luo, Zhe Xu, Wei Liu, Jian Luan, Wanxia Cao, and Ying Shen. 2025. Attention basin: Why contextual position matters in large language models. *arXiv preprint arXiv:2508.05128*.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. 2025. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*.

A Implementation Details

A.1 Baseline Descriptions

We detailedly introduce the compared baseline methods as follows:

Direct Generation. These baselines evaluate the intrinsic capabilities of the backbone LLM without external memory or retrieval tools, establishing a lower bound on how much the model can accomplish using only its parametric knowledge and the immediate input context.

- **Direct Inference** feeds only the question to the LLM, which generates an answer in a single forward pass.
- **Chain-of-Thought (CoT)** (Wei et al., 2022) prompts the model to produce step-by-step reasoning traces before generating the final answer, using its inherent reasoning ability to connect relevant clues.

Retrieval-Augmented Generation. To assess the effect of external information access, we include standard retrieval-based approaches. These methods use a retriever to select relevant document chunks from the stream, reducing context window pressure and limiting noise passed to the generator. They represent the prevailing paradigm for knowledge-intensive tasks.

- **Standard RAG** uses a dense vector retriever to fetch the top-k most relevant chunks based on the query; these chunks are concatenated and passed to the generator.
- **IRCoT** (Interleaved Retrieval CoT) (Trivedi et al., 2023) interleaves retrieval and reasoning steps: the model generates intermediate search queries from partial reasoning chains to retrieve additional context, enabling multi-step information gathering.

Agentic Baselines. We also compare against agentic frameworks that give the LLM control over its information-gathering process. Unlike passive RAG, these methods allow the model to decide dynamically when and what to retrieve, providing a strong competitive reference for our active memory management approach.

- **Search-R1** (Jin et al., 2025) is an RL-trained agent that interleaves chain-of-thought reasoning with multi-turn search queries, optimizing

the reasoning trajectory to decide when to issue a search. Unlike GRAM, Search-R1 holds all retrieved information in the LLM's context window without a persistent external memory module, making it vulnerable to context flooding on very long document streams.

- **MEM1** (Zhou et al., 2025) is a memory-efficient agent trained via RL to maintain a compact internal state. At each turn, it folds new observations into a rolling textual summary and discards the raw interaction history. This baseline represents a compression-based memory paradigm. Comparing against MEM1 lets us assess whether GRAM's structured graph retention preserves precise relational details better than generative summarization, which may suffer from information loss or hallucinations.

A.2 Training Details

We implement the GRAM framework and conduct all Reinforcement Learning (RL) experiments using the `verl` library. For the base policy model, we initialize our agent using the Qwen instruction-tuned series (e.g., Qwen2.5-3B-Instruct and Qwen3-4B). The RL optimization is driven by Group Relative Policy Optimization (GRPO). To ensure stable policy updates and mitigate severe reward hacking, we apply a low-variance KL divergence penalty between the active policy and the reference model.

The key hyperparameters utilized during our RL training phase are summarized in Table 6. We use the AdamW optimizer with a warmup step ratio of 0.285 and a peak learning rate of 1×10^{-6}, sampling a group size of $G = 5$ trajectories per prompt.

Hyperparameter	**Value**
RL Algorithm	GRPO
Sampled Group Size (G)	5
Optimizer	AdamW
Peak Learning Rate	1×10^{-6}
Warmup Step Ratio	0.285
KL Divergence Type	Low-variance
KL Penalty Coefficient (β)	0.001

Table 6: Summary of key hyperparameters for GRPO training.

Hardware & Distributed Setup. All training procedures are executed on a single computing node equipped with 4 NVIDIA RTX 3090 (24GB) GPUs. Given the strict VRAM constraints of consumer-grade hardware, we employ several memory-saving techniques to prevent out-of-memory (OOM) errors during the memory-intensive rollout phase. We utilize SGLang as the asynchronous rollout backend and configure a Tensor Parallelism (TP) size of 2 alongside a Data Parallelism (DP) size of 2. To fit the 24GB memory budget, the global mini-batch size is set to 8 with a reduced micro-batch size per GPU, seamlessly aligning the forward and backward passes to minimize inter-GPU communication overhead. Furthermore, we mandate gradient checkpointing and aggressive optimizer state offloading to CPU memory to ensure stable training.

B Case Study

B.1 Compressed Summary Memory Case

In this section, we analyze the reasoning behaviors of the strongest baseline, Mem1-3B. We present two representative cases: a successful retrieval-augmented inference and a failure caused by "associative drift" in noisy environments.

The contrast between the two cases reveals a critical vulnerability in linear memory agents like Mem1.

- **Success Condition:** As is shown in Table 7, although the retrieved text does not explicitly state "Caroline LeRoy is married to Daniel Webster," the model uses its internal parametric knowledge to make the correct link. The retrieved evidence (Fletcher is Daniel's son) is relevant and low-noise, allowing the rolling summary to function correctly.

- **Failure Mode (Associative Drift):** Table 8 demonstrates the danger of unstructured summarization. When the retrieval engine returns a distractor document (about *Stadio Giuseppe Grezar*) due to vector similarity or scarcity of information, Mem1 indiscriminately incorporates this irrelevant entity into its working memory. Lacking a structured topology to verify if *Stadio Giuseppe Grezar* is an alias of the target *Stadio Ciro Vigorito*, the agent treats it as valid evidence. This triggers a chain reaction: it pivots to investigating the wrong

Case Study: Successful Reasoning (Mem1)
Query: *"Who is the child of Caroline LeRoy's spouse?"*
Context (Step 1): [Search Results] Title: Fletcher Webster. Content: Daniel Fletcher Webster ... was the son of renowned politician **Daniel Webster** and Grace Fletcher Webster. **Agent: [Thought]** Based on the information provided, the child of Caroline LeRoy's spouse, Daniel Webster, is Fletcher Webster. **[Action]** `<answer> Fletcher Webster</answer>` **[Correct]**

Table 7: Case study of successful reasoning and accurate retrieval (Mem1).

Case Study: Reasoning Drift due to Noise (Mem1)
Query: *"What league does the team that plays in **Stadio Ciro Vigorito** play for?"*
Context (Step 1): [Search Results] Title: **Stadio Giuseppe Grezar**. Content: Stadio Giuseppe Grezar ... was initially used as the stadium of **U.S. Triestina Calcio** matches. **Agent: [Thought]** The information indicates Stadio Giuseppe Grezar ... was the stadium of U.S. Triestina Calcio. ... I will search for this team's league. **[Action]** `<search> "U.S. Triestina Calcio" current league</search>` **Context (Step 2):** [Search Results] Title: Piacenza Calcio 1919. Content: Piacenza Calcio ... play their home matches at ... Stadio Leonardo Garilli. **Agent: [Thought]** ... U.S. Triestina Calcio no longer plays at Stadio Ciro Vigorito ... it is now known as Piacenza Calcio 1919 ... which is in Serie B. **[Action]** `<answer> Serie B</answer>` **[Wrong]**

Table 8: Case study of reasoning drift due to noise and hallucination (Mem1).

team (*Triestina Calcio*) and eventually hallucinates a relationship with a third unrelated entity (*Piacenza Calcio*), drifting completely away from the original question.

This failure highlights a characteristic vulnerability of the Mem1 baseline, which operates by synthesizing externally retrieved evidence with its internal parametric knowledge. The error in Case 2 suggests that the model can become overconfident in its internal associations when faced with noisy or partial evidence. Instead of recognizing the mismatch between the query (Stadio Ciro Vigorito) and the retrieved document (Stadio Giuseppe Grezar), the agent seemingly relied on hallucinated connections to bridge the gap. This indicates that within a limited number of reasoning turns, the agent is prone to "premature closure," where it allows irrelevant context to trigger false internal priors rather than seeking further verification.

B.2 Qualitative Case Studies of GRAM

In this section, we present two qualitative examples of GRAM's reasoning process shown in Table 9 and Table 10. These cases illustrate how the graph-structured memory acts as a symbolic filter, preventing the "associative drift" observed in baseline models and enabling precise information retention over long context streams.

Precision in Multi-hop Attribute Filtering This example demonstrates GRAM's ability to distinguish between entities that share high semantic similarity (same songwriters) but differ in specific topological relations (different albums).

Robustness to Distractors in Streaming In streaming settings, the answer often appears early in the stream, followed by numerous irrelevant "distractor" documents. This case shows GRAM's ability to retain early facts in its graph memory without being overwritten or confused by subsequent noise.

C Used Prompts

C.1 Agent System Prompt

Figure 5 presents the complete system prompt used to define the behavior, strict action space, and reasoning constraints of the Agent.

C.2 Graph Operation Prompt

Figure 6 outlines the specialized prompts employed for dynamic graph operations, guiding the agent through robust entity extraction, strict graph maintenance, and standardized relation extraction.

Case Study: Entity Disambiguation via Graph Structure (GRAM)

Query: *"Which song from Taylor Swift's 'Red' album was written by Swift, Max Martin, and Shellback?"*

Context (Step 1): [Doc 1] "Bad Blood" is a song from the album "1989"... written by Swift, Max Martin, and Shellback.
Agent: [Thought] The document mentions "Bad Blood" has the correct writers, but it is linked to the album "1989", not "Red". I will record this but keep looking. **[Action]** `<memory_insert>"Bad Blood" is from the album "1989" and written by Swift, Max Martin, and Shellback.</memory_insert>`
(Background Graph Update: `(Bad Blood, is from album, 1989)`, `(Bad Blood, written by, Max Martin)`*...)*

Context (Step 2): [Doc 2] "I Knew You Were Trouble" is a song... for her fourth studio album, "Red"... written by Swift, Max Martin and Shellback.
Agent: [Thought] Found the matching song for the "Red" album with the correct writers. I will insert this critical fact. **[Action]** `<memory_insert>"I Knew You Were Trouble" is from the album "Red" and written by Swift, Max Martin and Shellback.</memory_insert>`
(Background Graph Update: `(I Knew You Were Trouble, is from album, Red)`*...)*

Context (Step 3): [System Prompt] Requesting final action based on updated memory.
Agent: [Thought] The graph clearly separates the writers' work by album. The song from "Red" is "I Knew You Were Trouble". **[Action]** `<answer>I Knew You Were Trouble</answer>`

Analysis: A standard RAG model might be confused by the high keyword overlap between "Bad Blood" and the query (both contain the target writers). GRAM, however, builds a structured representation where `Bad Blood` $\rightarrow$ `1989` and `I Knew You Were Trouble` $\rightarrow$ `Red`. By querying the graph for nodes connected to *both* "Red" and the writers, GRAM precisely isolates the correct entity.

Table 9: Case study illustrating how GRAM uses structured graph memory to disambiguate entities with high keyword overlap.

Case Study: Fact Retention amidst Noise (GRAM)

Query: *"When did the Stadio Luigi Ferraris open?"*

Context (Step 1): [Doc 1] Stadio Comunale Luigi Ferraris... opened in 1911... one of the oldest stadiums in Italy.
Agent: [Thought] Found the exact opening year of the requested stadium. I will save this fact into the memory graph. **[Action]** `<memory_insert>Stadio Comunale Luigi Ferraris opened in 1911.</memory_insert>`
(Background Graph Update: `(Stadio Luigi Ferraris, opened in, 1911)`*...)*

Context (Steps 2-4): [Docs 2-4] *(Stream of distracting documents about "1989 Hockey East Tournament", "Potgietersrus Rugby Stadium", and "Stadium Arena (Norrkoping)".)*
Agent: [Thought] These documents contain distractors about other venues. I need to verify my memory index to ensure I still have the correct opening year for Luigi Ferraris. **[Action]** `<memory_search>Stadio Luigi Ferraris opened</memory_search>`

Context (Step 5): [Memory Search Results]: `[Stadio Luigi Ferraris] -[opened in]-> [1911]`.
Agent: [Thought] The graph search confirms the specific opening year (1911) from the earlier document is safely stored. The distracting stream has not overwritten it. **[Action]** `<answer>1911</answer>`

Analysis: While linear memory models often suffer from "recency bias" or forget early information when the context window fills up with distractors, GRAM's symbolic storage is persistent. Once the fact `(Stadio Luigi Ferraris, 1911)` is inserted into the graph, it remains strictly accessible regardless of how many irrelevant documents follow, demonstrating superior robustness in long-context streaming.

Table 10: Case study demonstrating GRAM's ability to retain early facts over long reasoning trajectories despite heavy noise.

System Prompt

You are a Knowledge Graph Memory Agent answering complex questions by processing a sequential document stream.
Your task is to collect clues from the multi-turn document stream to maintain a knowledge graph. At each step, you receive <memory_obs>...</memory_obs> (containing Search Results, Graph Memory, and Progress Hint) and <document_obs>...</document_obs> (Current Document).

1. Perform concise reasoning within <think>...</think> to decide your action. Keep your thoughts brief (max 3-5 sentences) and avoid repetitive checking.

2. Then you must choose exactly ONE of the following actions to update your observation:

   - If <document_obs> contains any new relevant info for solving the question (Any entity of the document that is shown in the question), you MUST use <memory_insert>...</memory_insert> to add it to the knowledge graph. The content must be comprehensive and include all relevant clue sentences found in the text.
   - If <document_obs> contains information that conflicts with, corrects, or updates specific attributes of an entity ALREADY existing in <memory_obs>, use <memory_update>...</memory_update> to modify the graph node or edge accordingly.
   - If <document_obs> does NOT contain any relevant info AND <memory_obs> is NOT empty, use <memory_search> to query the knowledge graph for missing connections.
   - If all evidence has been collected within <memory_obs> (Search Results & Graph), and there's nothing to be added, don't analyze the <document_obs>, just output the final answer inside <answer>...</answer>. The answers must be concise, contain only essential words, and avoid any explanations.

Important:

- If the document helps, <memory_insert> or <memory_update> is mandatory. Do not search if you are holding a useful document, even if the document clues are partial.
- If current clues in <memory_obs> cannot answer the question completely, you cannot guess the answer directly, you must try to insert new clues or search more evidence.
- <memory_obs> is your ONLY source of knowledge to answer the question, NOT <document_obs> (unless you Insert/Update it first).
- Always follow this structure: <think>...</think><memory_insert>...</memory_insert> OR <think>...</think><memory_update>...</memory_update> OR <think>...</think><memory_search>...</memory_search> OR <think>...</think><answer>...</answer>.

Figure 5: The System Prompt used for the Knowledge Graph Memory Agent.

Entities Extraction Prompt

You are an expert Knowledge Graph extraction assistant. Output ONLY valid JSON lists. Do not include markdown formatting, preambles, or explanations.
Your task is to identify all salient named entities (e.g., Persons, Locations, Organizations, Dates, Events, and key Concepts) in the text below.
Guidelines:
1. Extract entities in their most specific and canonical form.
2. Resolve obvious pronouns to their exact referents if possible.
3. Ignore generic nouns, stop words, or overly long descriptive phrases.
Output ONLY a JSON list of strings. Example: ["Steve Jobs", "Apple Inc.", "April 1, 1976"]
Text: {{INPUT_TEXT}}

Insert Prompt (Graph Maintenance)

You are a strict Knowledge Graph maintenance assistant responsible for keeping the graph accurate and up-to-date. Based on the new observation or correction text, identify how the existing knowledge base should be modified.
Guidelines:
1. Identify obsolete, conflicting, or incorrect facts (triplets) that are explicitly invalidated by the text and need to be REMOVED.
2. Identify new factual relationships introduced in the text that need to be ADDED.
3. Ensure all extracted triplets follow the ["subject", "predicate", "object"] format.
4. If the text only adds new information without contradicting past facts, the 'remove' array must be left empty.
Correction Text: "{{CORRECTION_TEXT}}"
Output ONLY a valid JSON object with this exact structure (No markdown):
{
"remove": [["subject", "predicate", "object"], ...],
"add": [["subject", "predicate", "object"], ...]
}

Update Prompt (Relation Extraction)

You are an expert Knowledge Graph relation extraction assistant. Output ONLY valid JSON lists. Do not include markdown formatting or explanations.
{{CONTEXT_INSTRUCTION}}
Given the input text and a verified list of entities: {{LIST_OF_ENTITIES}}, your task is to extract all factual, directed relationships between these specific entities.
Guidelines:
1. Formulate relationships as semantic triplets: ["subject", "predicate", "object"].
2. The "subject" and "object" MUST strictly be selected from the provided LIST_OF_ENTITIES. Do not hallucinate new entities.
3. The "predicate" should be a concise, standardized, and lowercase relation type (e.g., "founder of", "located in", "released on").
Output ONLY a JSON list of triplets: [["subject", "predicate", "object"], ...]
Text: {{INPUT_TEXT}}

Figure 6: Detailed prompts used by the GRAM agent for robust Entity Extraction, Graph Maintenance (Insert/Correct), and Relation Extraction (Update).