

EvoMemKG: An Evolvable Memory Agent for Multi-hop Knowledge Graph Reasoning

Shiyu Tian¹, Shuyue Xing¹, Zhuoxin Han¹, Caixia Yuan¹, Xiaojie Wang^{1,*}

¹Beijing University of Posts and Telecommunications

{tiansy, xsy84160158, hanzhuoxin, yuancx, xjwang}@bupt.edu.cn

Abstract

Integrating knowledge graphs (KGs) with large language models (LLMs) enhances factual accuracy and interpretability in question answering (QA). However, existing agent-based methods rely on static memory mechanisms, which fail to address the state-space explosion in multi-hop reasoning and lack self-evolution capabilities. To overcome these limitations, we propose **EvoMemKG**, an agent framework with an evolvable memory mechanism for multi-hop KG reasoning. EvoMemKG introduces a dual-layer memory: (1) a working memory that manages and compresses intermediate states using a relation-based compression algorithm, effectively reducing context usage; and (2) an experience memory that distills historical reasoning paths into reusable, generalized strategies, enabling cross-task knowledge transfer and self-evolution. We further design a double-loop workflow that orchestrates the LLM, the dual-layer memory, and KG environment to enable end-to-end autonomous reasoning. Extensive evaluations on three KGQA datasets across two KGs demonstrate that EvoMemKG achieves state-of-the-art performance without requiring additional training or specialized tools, with improvements of up to 20% over strong baselines on complex multi-hop queries.

1 Introduction

The integration of knowledge graphs (KGs) with large language models (LLMs) (Zhao et al., 2023) offers a promising path toward factually-grounded and interpretable reasoning (Hu et al., 2024a; Some et al., 2025). KGs provide structured and human-readable knowledge (Pan et al., 2024) that can mitigate the limitations of LLMs, such as hallucination (Rawte et al., 2023) and knowledge deficiencies (Sun et al.; Schick et al., 2023). However, leveraging KGs for multi-hop question answering remains

*Corresponding authors.

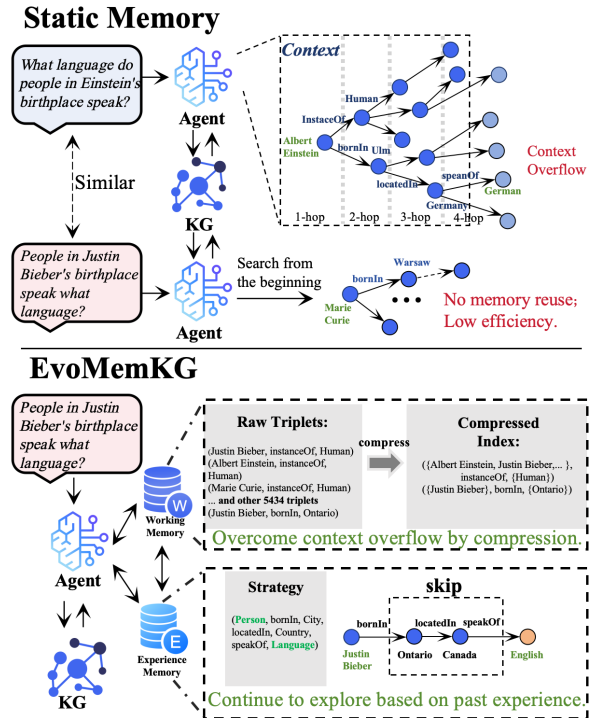


Figure 1: The key limitations of static memory, and how our EvoMemKG overcomes them by a dual-layer memory.

challenging, as it requires multiple rounds of exploration within a vast and structured knowledge space.

Recent methods focus on developing LLM agent frameworks (Wang et al., 2024) to handle multi-hop KG question answering, where an LLM agent interacts with the KG to find answers. These methods can be categorized into indirect interaction (e.g., using retrieval tools or generating SPARQL queries) (Li et al., 2025; Xiong et al., 2024; Jiang et al., 2023a) and direct interaction (e.g., path-finding on the graph) (Jiang et al., 2025; Sun et al., 2024; Markowitz et al., 2024).

Although the agent-based paradigm avoids costly LLM fine-tuning and excels in flexibility and generalizability, most existing methods em-

ploy static memory mechanisms, which suffer from two limitations (as shown in Figure 1): (1) **State-space explosion**: during multi-hop exploration, the number of intermediate states grows exponentially. Static memory cannot store all information within the finite context of LLMs, resulting in information loss or truncated reasoning chains; (2) **Lack of evolution**: each query is processed independently, unable to retain or reuse successful reasoning experiences across tasks, leading to redundant exploration and prevents self-evolution.

To overcome these limitations, we introduce **EvoMemKG**, an agent framework equipped with a dynamic, evolvable memory mechanism designed for multi-hop KG reasoning. EvoMemKG features a dual-layer memory architecture: a **Working Memory (WM)** that compresses retrieved triplets to manage immediate exploration states, and an **Experience Memory (EM)** that distills historical reasoning paths into reusable general strategies. A dedicated double-loop workflow orchestrates the interaction between the LLM, memory, and KG environment, enabling the agent to select appropriate action at each step to solve multi-hop queries autonomously.

We evaluate EvoMemKG on three common KGQA datasets across two KGs. Without any task-specific training or the use of additional tools, our framework achieved state-of-the-art performance, showing particularly significant improvements on complex multi-hop queries. Additional ablation and analysis experiments further illustrate the effectiveness and efficiency of the memory mechanism.

Our contributions are summarized as follows:

- We propose EvoMemKG, an agent framework specifically designed for multi-hop KG reasoning. It features a dynamic dual-layer memory mechanism and is adaptable to any KG and base model, effectively addressing the issues of state explosion and lack of evolution in multi-hop KG reasoning.
- We introduce a dual-layer memory: working memory uses relationship-based compression algorithms to compress and manage intermediate states during the exploration process; experience memory distills historical experiences into reusable strategies, enabling the agent to self-evolve. Additionally, we design a dual-loop workflow to coordinate all components and achieve efficient end-to-end task solving.
- We conduct extensive experiments demonstrating state-of-the-art performance and providing in-depth analysis of the memory mechanism’s effectiveness and efficiency, particularly for multi-hop reasoning.

2 Related Work

Our work lies at the intersection of KG-augmented LLM reasoning and memory-augmented LLM agents. Below, we review relevant advances in both lines of research and highlight the gap that EvoMemKG aims to fill.

2.1 KG-Augmented LLM Reasoning

Incorporating KGs into LLM reasoning has emerged as a key strategy to enhance factual grounding and interpretability. Existing approaches can be categorized along two dimensions: (1) how the KG is accessed, and (2) whether the LLM is fine-tuned.

Fine-tuning based methods (Sun et al., 2018; Saxena et al., 2020; Zhang et al., 2022; Jiang et al., 2023c) often incorporate KG-specific modules into pre-trained models and perform task-specific fine-tuning. While effective on particular benchmarks, these methods sacrifice generalization and require substantial training data and computation.

To preserve generality, agent-based frameworks have gained popularity. Here, the LLM acts as a controller that decides whether and how to access the KG, without being retrained. These frameworks can be further divided into:

Indirect interaction methods, which rely on intermediate tools such as vector retrievers (Guo et al., 2023; Faldu et al., 2024; Hu et al., 2024b) or structured query generators (e.g., SPARQL) (Jiang et al., 2023a; Agarwal et al., 2024; Xiong et al., 2024). While effective, their performance is bounded by the quality of tools, and orchestrating multi-tool workflows remains challenging (Qu et al., 2025). Recent work, such as DoG (Li et al., 2025), uses KG to build a graph-aware decoding strategy to regulate the decoding process of LLMs.

Direct interaction methods, which treat KGQA as a path-finding task on the graph (Markowitz et al., 2024; Sun et al., 2024; Xu et al., 2024). By allowing the agent to traverse edges step-by-step, this paradigm offers greater generalizability and transparency. However, it faces a fundamental bottleneck: the LLM’s finite context window cannot accommodate the exponentially growing

state space during multi-hop exploration. Recent work like ToG-3 (Wu et al., 2025a), ORT (Liu et al., 2025), and LMP (Wan et al., 2025) attempts to mitigate this by dynamically constructing task-specific graph representations (such as graph summarization or pruning). However, they still cannot achieve lossless compression of retrieved information or self-evolution based on historical experience. Our work adopts the direct interaction paradigm and introduces a dynamic memory system to overcome these limitations.

2.2 Memory-Augmented LLM Agents

Memory is essential for agents operating in sequential decision-making environments (Wang et al., 2024; Xi et al., 2023), enabling experience accumulation (Zheng et al., 2023), information abstraction (Zhao et al., 2024), and long-term adaptation (Zhang et al., 2024).

Early agent designs treated memory as a passive record of interactions, storing it within the LLM context, which leads to context overflow in multi-hop tasks. Recent research explored structured memory systems. For example, utilizing temporal KGs to integrate situational and semantic memory (Rasmussen et al., 2025); constructing graph-based memory systems that model long-context information as graphs to enable associative retrieval (Wu et al., 2025b); and adopting hypergraph-based memory systems to model higher-order interactions and handle complex reasoning (Zhou et al., 2026).

Although these studies highlight the value of structured memory, they were not specifically designed for the iterative, graph-based decision-making processes required for direct KG interaction. Thus, they fail to address the unique challenges of multi-hop KG reasoning tasks: (1) intra-session state compression to handle exponentially growing exploration branches and states; and (2) cross-session strategy evolution to reuse successful reasoning strategies.

EvoMemKG bridges this gap through a dual-layer memory architecture that manages state space explosion (via working memory) while distilling transferable reasoning strategies (via experience memory), thus tailoring to multi-hop KG reasoning tasks.

3 Methodology: EvoMemKG

We begin by formalizing multi-hop KG reasoning as a sequential decision-making problem (3.1),

then present an overview of EvoMemKG (3.2), followed by detailed descriptions of its dual-layer memory (3.3) and double-loop workflow (3.4).

3.1 Problem Formulation

Given a natural language query q and its corresponding query entity $e_q \in \mathcal{E}$ within a KG $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$, where \mathcal{E} , \mathcal{R} , and \mathcal{T} denote the sets of entities, relations, and triplets, respectively. The goal is to find the answer entity $e_a \in \mathcal{E}$, which equivalent to finding a valid reasoning path \mathcal{P} in \mathcal{G} :

$$\begin{aligned} \mathcal{P} &= (e_q, r_1, e_1, r_2, e_2, \dots, r_n, e_a), \\ \text{s.t. } &\forall i, (e_i, r_i, e_{i+1}) \in \mathcal{T} \end{aligned} \quad (1)$$

The process of searching for \mathcal{P} can be defined as a Markov decision process. At each step t , the agent observes a state s_t (including retrieved triplets, the current entity, and history) and selects an action a_t (e.g., exploring neighboring nodes or answering). After executing a_t , the environment updates the state to s_{t+1} . An episode ends when the agent submits an answer or reaches the maximum step T_{\max} . The objective is to learn an optimal policy π that maximizes the probability of reaching the correct answer:

$$\arg \max_{\pi} \mathbb{E}_{\mathcal{P} \sim \pi} [\mathbb{I}(e_a \in \mathcal{P}) \mid q, e_q, \mathcal{G}] \quad (2)$$

where \mathbb{I} is the indicator function, and the expectation is taken over paths \mathcal{P} generated by the policy π interacting with \mathcal{G} starting from e_q .

3.2 Overview

As illustrated in Figure 2, EvoMemKG consists of three core components: (1) LLM Agent, the central controller that interprets states, selects actions and generate answers; (2) Dual-Layer Memory, comprising Working Memory (WM) for state management and compression within a single task and Experience Memory (EM) for cross-task experience reuse; (3) KG Environment, provides interfaces for the agent to access KG.

Above these components, a Double-Loop workflow organizes them into a complete agent framework: the **inner loop** handles each reasoning (exploration) steps, where the agent uses current state (from WM) and relevant experience (from EM) to select an action executed in the KG environment; the **outer loop** manages the entire task, initializing memory, running the inner loop until termination, and then distilling successful trajectories into experience memory.

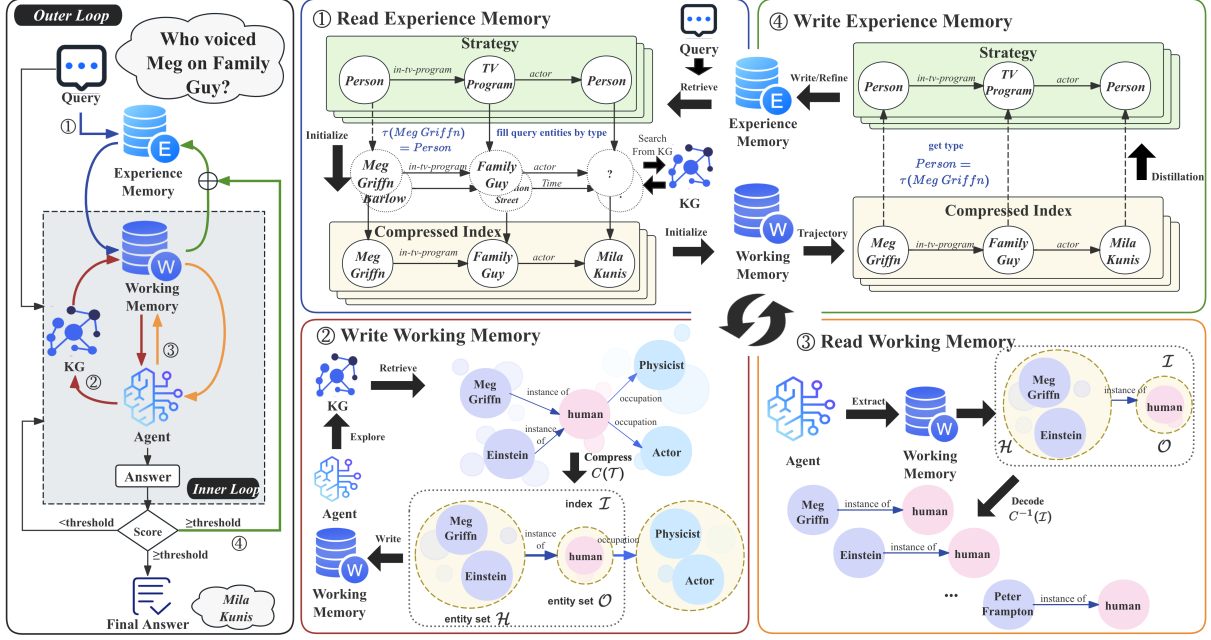


Figure 2: Left: the double-loop workflow (inner and outer loop) and main components; Right: the memory operations; (1) retrieve historical strategies from experience memory to initialize the current state; (2) explore KG and compress the raw triplets into the index of working memory; (3) reconstruct the index of working memory into triplets; (4) distill reusable strategy from trajectories.

This design enables the agent framework to handle state-space explosion via WM while continuously evolving through EM, achieving both efficient reasoning and adaptive learning.

For formal definitions of actions and workflows, see Appendix A.

3.3 Dual-Layer Memory

The Dual-Layer Memory is the core of EvoMemKG. It consists of two modules: **Working Memory (WM)**, which manages and compresses the intermediate states of a single exploration process, and **Experience Memory (EM)**, which enables the reuse of experience across tasks.

3.3.1 Working Memory

WM manages the intermediate state of each exploration step. It compresses the exponentially growing retrieved triplets through a relation-based compression algorithm, enabling the LLM to grasp the complete reasoning chain within a limited context.

Relation-based compression algorithm. At each exploration step, the agent may retrieve numerous triplets from the KG. Directly feeding them into the LLM would quickly exhaust the context limit. Therefore, WM employs a relation-based compression algorithm to compress retrieved

triplets with the same relation into an index entry.

Specifically, at step t , the agent retrieves a set of triplets $\mathcal{T}_t = \{(s_i, r_i, o_i)\}$. The compression process consists of four specific steps:

1. **Grouping:** let \mathcal{R}_t be the distinct relations in \mathcal{T}_t . For each relation $r \in \mathcal{R}_t$, find the triplets in \mathcal{T}_t that contain r :

$$\mathcal{T}_t^{(r)} = \{(s, r, o) \in \mathcal{T}_t\}, \quad (3)$$

2. **Clustering:** Next, the head entities and tail entities in $\mathcal{T}_t^{(r)}$ are clustered to two entity sets:

$$\begin{aligned} \mathcal{H}_t^{(r)} &= \{s \mid (s, r, o) \in \mathcal{T}_t^{(r)}\} \\ \mathcal{O}_t^{(r)} &= \{o \mid (s, r, o) \in \mathcal{T}_t^{(r)}\} \end{aligned} \quad (4)$$

3. **Textualization:** Then, generate a structured text snippet for each relation r and its clusters:

$$\begin{aligned} \phi(\mathcal{H}_t^{(r)}, r, \mathcal{O}_t^{(r)}) &= \text{“Head Set: } \{s_1, s_2, \dots, s_k\}, \\ &\text{Relation: } r, \text{ Tail Set: } \{o_1, o_2, \dots, o_k\}\text{”} \end{aligned} \quad (5)$$

where the function $\phi(\cdot)$ formalizes a relation and its associated entity sets into compact text snippets as an index, displaying only the first k entities from each set as examples.

4. **Concatenation:** Finally, concatenate the text fragments corresponding to all relations $r \in \mathcal{R}_t$ to

obtain the compressed index entry:

$$I_t = \mathcal{C}(\mathcal{T}_t) = \bigoplus_{r \in \mathcal{R}_t} \phi(\mathcal{H}_t^{(r)}, r, \mathcal{O}_t^{(r)}) \quad (6)$$

where \bigoplus denotes concatenation operation and $\mathcal{C}(\cdot)$ is the relation-based compression algorithm.

After compression, the index is fed into the context of LLM instead of all triplets, which is significantly smaller, allowing the LLM to maintain a global awareness of long reasoning chains within the finite context window. Furthermore, this compression is reversible, as all triplets are stored in an external memory pool and can be fully reconstructed via the decoding mapping $\mathcal{C}^{-1}(I_t) \equiv \mathcal{T}_t$.

To enable the agent to use WM, the framework provides two basic actions:

Write: Compress the new retrieved triplet \mathcal{T}_t and concatenate it with the existing index:

$$I_t = \text{WRITE}(I_{t-1}, \mathcal{T}_t) = I_{t-1} \bigoplus \mathcal{C}(\mathcal{T}_t) \quad (7)$$

Read: LLM can invoke the read operation to reconstruct specific portions of the index into triplets for fine-grained reasoning:

$$\mathcal{T}' = \text{READ}(I_t, \text{condition}) = \{(s, r, o) \mid (s, r, o) \in \mathcal{C}^{-1}(I_t) \text{ and match condition}\} \quad (8)$$

3.3.2 Experience Memory

EM stores generalized reasoning strategies distilled from past trajectories, enabling cross-task experience reuse and self-evolution.

Strategy Distillation. After the agent completes a multi-hop reasoning (exploration), it obtains a reasoning path $\mathcal{P}_c = (e_q, r_1, e_1, \dots, r_n, e_a)$. The EM abstracts this into a general strategy S using the strategy distillation function D :

$$S = (k, v, s) = \mathcal{D}(\mathcal{P}_c, q, \mathcal{G}) \quad (9)$$

where S is a tuple containing three elements:

1. key: $k = \text{Emb}(q)$ is the embedding of the original query, serving as a key for future retrievals.
2. value: v is the distilled strategy content. It is a generalized reasoning path obtained by replacing the concrete entities in \mathcal{P}_c with their types $\tau(e)$:

$$v = (\tau(e_q), r_1, \tau(e_1), \dots, r_n, \tau(e_a)) \quad (10)$$

For example, the path (Albert Einstein, bornIn, Ulm, locatedIn, Germany) is abstracted to (Person, bornIn, City, locatedIn, Country), forming a reusable strategy for “birth country” queries.

3. score: $s = \text{LLM}(\mathcal{P}_c, q, \text{ans})$ is the self-assessment score generated by the LLM for the quality of this reasoning path and generated answer ans , which is used for subsequent strategy refinement.

Finally, to enable the agent to use the EM, we also provide two memory management actions:

Read: For a new query q' , compute its embedding $\text{Emb}(q')$ and retrieve the top K strategies with the highest similarity from the experience memory \mathcal{M}_{EM} :

$$S_{\text{top}K} = \text{Retrieve}(\mathcal{M}_{EM}, q') = \{(k_i, v_i) \mid \text{sim}(\text{emb}(q'), k_i) \text{ is top-}K\} \quad (11)$$

where $\text{sim}(\cdot, \cdot)$ is a similarity function (e.g., cosine similarity in the embedding space).

Write/Refine: For a new strategy $S' = (k', v', s')$. If a similar strategy exists ($\text{sim}(k', k_i) \geq \theta, k_i \in \mathcal{M}_{EM}$), keep the one with higher score; otherwise, add the new strategy. This mechanism ensures the quality and diversity of the strategies in the EM, enabling it to evolve dynamically.

3.4 The Double-Loop Workflow

The double-loop workflow coordinates LLM Agent, working memory (WM), experience memory (EM), and KG environment to enable end-to-end autonomous reasoning, allowing EvoMemKG not only to process queries efficiently but also to continuously evolve its reasoning strategies through past experience.

3.4.1 Inner Loop

The inner loop is responsible for executing each specific exploration step. At step t , the agent receives the following information as the current state (observation): (1) the user query q ; (2) the strategies retrieved from the EM and instantiated for the current query; and (3) the compressed index I_t in the current WM.

Based on this information, the LLM selects one of the following three types of actions through a carefully designed prompt (which includes the thought process and action selection):

Explore: Used to explore one hop on the KG to obtain more information; it consists of two sub-actions executed sequentially.

Extract: Used to read from WM, filter information, or perform logical operations on WM. This action comprises a series of specific functions that

the agent can select and provide parameters for as needed.

Answer: Executed when the agent determines that the current information is sufficient to generate an answer or when the maximum number of steps has been reached. It consists of three sub-actions executed in sequence.

See Appendix A.2 for details about these actions.

3.4.2 Outer Loop

The outer loop manages the execution of a complete task and evolution of the memory system, consisting of three phases:

Preparation: When receiving a new query q , the system first retrieves the K ($K = 3$) most relevant strategies from the EM. These strategies are then instantiated based on the current query by replacing the type $\tau(e)$ in the strategy with a corresponding entity in q and KG, forming an initial query sketch. These sketches are written to the WM as the initial exploration state I_0 . This phase provides the agent with high-level prior knowledge, enabling it to avoid blind exploration from scratch. See Algorithm 1 in Appendix A.1.

Exploration: Once the preparation phase is complete, the inner loop will begin, allowing the agent to autonomously and iteratively explore the KG until it executes the *Answer* action and provides a high-confidence answer, or until the maximum steps is reached. Throughout the exploration process, the WM is continuously updated, providing the agent with a global, compressed view of the states.

Evolution: After the task is completed, the reasoning path is converted into a general strategy by strategy distillation. Subsequently, the new strategy is stored in the EM, transforming successful exploration into the system’s experience memory and providing guidance for similar queries in the future. See Algorithm 2 in Appendix A.1.

4 Experimental Setup

4.1 Datasets

We evaluate on three widely-used KGQA datasets that cover diverse complexity levels: WebQuestionsSP(WQSP) (Yih et al., 2016) is a relatively simple dataset where 80% of questions are within 2 hops; ComplexWebQuestions(CWQ) (Talmor and Berant, 2018) extends WQSP with multi-hop and compositionally complex questions; KQAPro (Cao et al., 2022) is the most challeng-

ing benchmark with numerous complex multi-hop queries and six distinct compositional types. They are based on various KGs, where WQSP and CWQ use Freebase (Bollacker et al., 2008) and KQAPro uses WikiData (Vrandečić and Krötzsch, 2014).

See Appendix B for more details about dataset statistics and random sampling methods.

4.2 Evaluation Metrics

We adopt Hits@1 as our primary evaluation metric, consistent with prior work (Luo et al., 2024; Sun et al., 2024; Xu et al., 2024). It evaluates whether any correct answers appear in the top-1 response generated by the model. See Appendix C for details.

4.3 Baselines

To thoroughly assess EvoMemKG, we compare against 20 baseline methods spanning four categories: (1) full fine-tuned language models (LMs): early methods that incorporate KG-specific modules and require full fine-tuning; (2) LLM baselines: standard LLMs without KG access or task-specific adaptation; (3) Indirect interaction agents: LLM-agents that access KGs through intermediate tools; (4) Direct interaction agents: same as our method that treat KGQA as direct graph traversal. More details are provided in Appendix D.

4.4 Implementation Details

Models. We use *gpt-3.5-turbo-0125*, *gpt-4o-2024-08-06*, *gpt-4.1-mini-2025-04-14* and *gpt-5-mini-2025-08-07* as our base models.

Hyperparameters. We set the random seed to 42, temperature to 0, top_p to 0.2 and max_tokens to 2048. All experiments use greedy decoding and zero-shot settings (no task-specific examples are provided). See Appendix E for the detailed settings of each experiment.

5 Results and Analysis

We present comprehensive experimental results to evaluate EvoMemKG’s performance, followed by ablation studies and in-depth analyses of its behavior across different query hops and memory configurations. Additionally, we conduct efficiency analysis, results on more models, and case studies in Appendix F.

5.1 Main Results

Overall Performance. Table 1 compares EvoMemKG against 20 baseline methods on three

Type	Methods	WQSP	CWQ	KQA	Base Model	#Adapt
LM w/ Full-Fine Tune	EmbedKGQA (2020)	66.60	44.70	20.27	RoBERTa-base	add-module
	SR (2022)	69.50	49.30	22.82	RoBERTa-base	add-module
	ReasoningLM (2023b)	78.50	69.00	—	RoBERTa-base	add-module
	UniKGQA (2023c)	75.10	50.70	—	RoBERTa-base*3	add-module
LLM Baselines	Llama-2	64.40	34.60	31.20	Llama-2-7B	—
	GPT-3.5	66.80	39.90	24.96	GPT-3.5	—
	GPT-4	73.20	55.60	37.43	GPT-4	—
	GPT-4.1-mini	73.53	50.31	32.92	GPT-4.1-mini	—
	GPT-5-mini	78.17	65.43	46.73	GPT-5-mini	—
Agent-Based w/ Indirect	StructGPT (2023a)	72.60	54.30	—	ChatGPT	tool-use
	SymKGQA (2024)	71.20	—	51.10	CodeLlama	tool-use
	Interact-KBQA (2024)	72.47	59.17	71.89	GPT-4	tool-use
	DoG(2025)	91.38	76	—	Llama 3.1-8B	add-module
Agent-Based w/ Direct	ROG (2024)	85.70	62.60	—	Llama-2-7B	fine-tune
	EffiQA (2024)	82.90	69.50	—	GPT-4+RoBERTa	fine-tune
	KG-Agent (2025)	83.3	72.2	—	Llama-2-7B	fine-tune
	ToG(2024)	82.60	67.60	46.83*	GPT-4	static-mem
	GoG(2024)	84.40	75.20	41.82*	GPT-4	static-mem
	ORT(2025)	89.43	72.91	—	DeepSeek-v3	static-mem
	LMP(2025)	90	82	—	GPT-4	static-mem
EvoMemKG (Ours)	w/ GPT-3.5	84.83	63.61	70.44	GPT-3.5	evo-mem
	w/ GPT-4o	86.69	73.26	80.33	GPT-4o	evo-mem
	w/ GPT-4.1-mini	94.40	84.33	87.00	GPT-4.1-mini	evo-mem
	w/ GPT-5-mini	94.57	89.47	93.62	GPT-5-mini	evo-mem

Table 1: Main Results: the Hits@1 of different methods on three datasets. * denotes we reproduced these methods on the new dataset, and all other results are from their original paper. #Adapt indicates how to adapt to KGs.

KGQA datasets. Our framework achieves state-of-the-art performance across all benchmarks without any task-specific fine-tuning or additional tools. Notably, with GPT-5-mini as the base model, EvoMemKG reaches 94.57% Hits@1 on WQSP, 89.47% on CWQ, and 93.62% on KQA Pro, surpassing all previous methods.

Outperform fine-tuned methods. EvoMemKG outperforms fine-tuned LLMs (e.g., RoG, KG-Agent) methods, demonstrating the effectiveness of our memory-augmented agent framework.

Advantage over agent baselines. Compared to recent agent-based methods, EvoMemKG shows more gains on complex datasets (CWQ and KQA Pro). For instance, it outperforms ToG by 33.5 points on KQA Pro, highlighting the critical role of memory in handling multi-hop reasoning.

Model scalability. Performance consistently improves with more capable base models (GPT-3.5 → GPT-5-mini), indicating that EvoMemKG effectively leverages base model capabilities and has strong generalizability.

These results demonstrate that our dynamic dual-layer memory is effective for multi-hop KG reasoning, enabling efficient exploration and continuous learning without sacrificing generalization capabilities.

5.2 Ablation Study

To understand the contribution of each component, we conduct ablation experiments in Table 2. Removing EM causes an average performance drop of 5.75% across datasets and models, with larger degradation on complex queries (up to 6.13% on KQA). This suggests that the strategies abstracted by EM can guide exploration and reduce redundant search. Removing WM leads to more degradation (18% average drop), validating that without state compression, the exponential growth of search states quickly exhausts the LLM’s context window, causing reasoning breakdown. Finally, removing WF prevents the agent from accessing the KG and memory, causing it to degrade into a single-round LLM question-answering system.

Method	CWQ	KQA	Base Model
EvoMemKG	84.33	87.00	GPT-4.1 mini
w/o EM	81.32	79.85	
w/o EM+WM	74.69	71.88	
w/o EM+WM+WF	50.31	32.92	
EvoMemKG	89.47	93.62	GPT-5 mini
w/o EM	84.98	87.12	
w/o EM+WM	78.63	75.13	
w/o EM+WM+WF	65.43	46.73	
EvoMemKG	74.42	72.06	Qwen3 8B
w/o EM	69.45	70.65	
w/o EM+WM	62.76	60.33	
w/o EM+WM+WF	39.53	24.33	

Table 2: Ablation Study: the performance of removing Experiential Memory (EM), Working Memory (WM), and Workflow (WF) sequentially.

5.3 Analysis and Discussion

5.3.1 Impact of Query Complexity

Figure 3 shows how performance varies with different reasoning hops. As query complexity increases (from 1 to 5 hops), baseline methods exhibit a steady performance decline, particularly beyond 3 hops. In contrast, EvoMemKG maintains consistently high accuracy across all hop counts. This robustness stems from our dual-memory design: WM prevents context overflow during long trajectories, while EM provides high-level strategies that guide the agent through complex multi-hop searches. Furthermore, as the number of hops increases, the performance gap between EvoMemKG and its variants widens, demonstrating the effectiveness of our memory mechanism specifically for multi-hop reasoning.

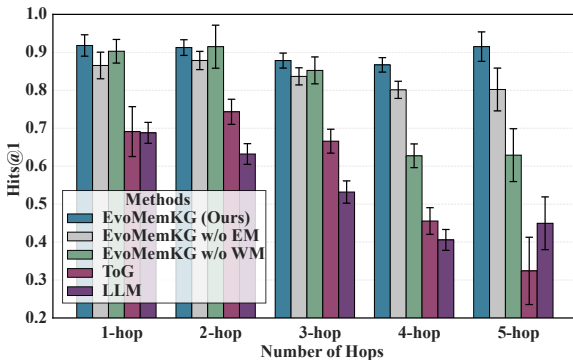


Figure 3: The average performance across various methods under different hop queries.

5.3.2 Exploration Efficiency

Figure 4 compares the average number of exploration steps required to find correct answers. EvoMemKG find answers with approximately 40% fewer steps than the w/o EM variant. This suggests that EM enables the agent to bypass most of the search space by retrieving and instantiating relevant strategies, thereby allowing it to proceed directly to promising regions.

5.3.3 Compression Efficiency

Table 3 quantifies WM’s compression effectiveness. As hop count increases, raw triplet count grows exponentially (from 6,678 at 1-hop to 115,585 at 5-hop), while WM’s compressed representation remains stable and compact (105 to 240 tokens). Compression ratios exceed 98% at all hop levels, demonstrating WM’s ability to transform combinatorial information growth into fixed-size representations that fit within LLM context windows.

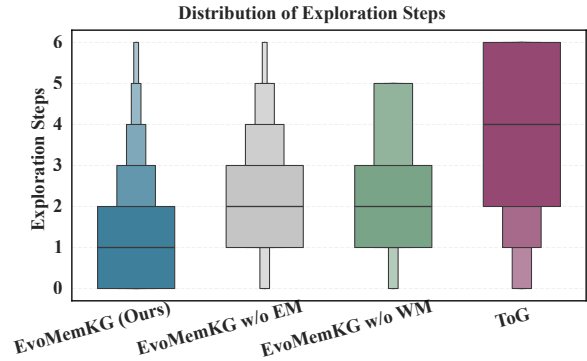


Figure 4: The distribution of exploration steps across different methods.

Hop	w/o WM	w/ WM	Comp. Ratio
1	6,678	105	98.43%
2	15,200	183	98.80%
3	22,037	252	98.86%
4	84,462	253	99.70%
5	115,585	240	99.79%

Table 3: The average token usage per sample with or without working memory.

5.3.4 Memory Reuse and Evolution

Figure 5 shows that the top 20% of strategies in EM are utilized by 41.7% of query samples. This high reuse rate indicates that our strategy distillation process effectively captures transferable

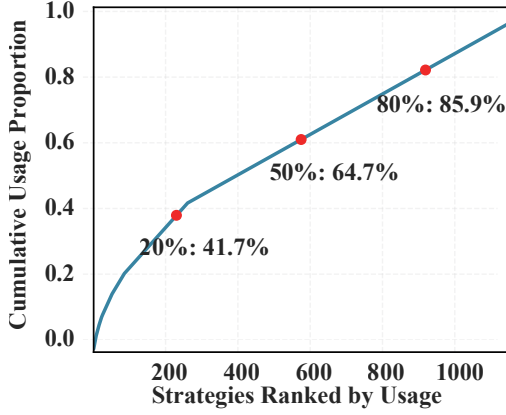


Figure 5: Cumulative contribution of strategies, showing the reuse rate of different strategies.

reasoning patterns rather than simply memorizing specific answers.

Figure 6 illustrates performance improvement as EM accumulates more strategies. It exhibits an upward trend during the first 300 strategy accumulations, then gradually stabilizes. This indicates that EM endows the agent evolution capability; as the scope and quality of its accumulated strategies expand, the agent’s capabilities continue to improve.

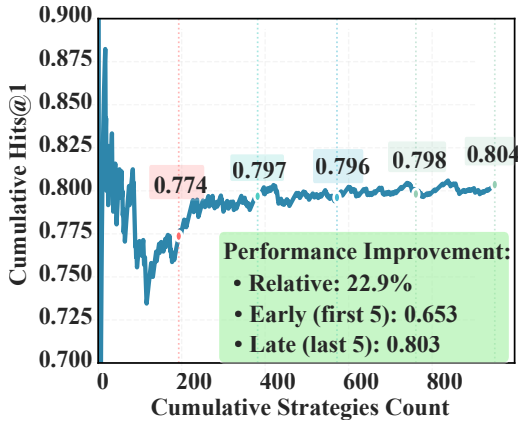


Figure 6: Learning Curve: the relationship between cumulative Hits@1 and cumulative strategies count.

5.4 Case Study

In Table 4, we present an example of WM compressed index. The WM compresses over 5,000 triplets of the form (xxx, instance of, human) into a single index entry, which significantly reduces context usage and thus preserves more context space for storing key information. Furthermore, the index provides CoT-like procedural knowledge, such as James Wong Jim is the person who died on 2004-

Method	Context (part)	Tokens Usage
Query	Which profession is the work of the person whose death date is 2004-11-24?	
w/ WM	(ent_set_0, instance of, Q28640: profession); (ent_set_1, instance of, Q5: human); (ent_set_1-0, date of death, ent_set_2); (ent_set_1-1, date of birth, ent_set_4); (ent_set_1-2: James Wong Jim, date of death = 2004-11-24)	~50
w/o WM	(Q311317: Isaac Stern, instance of, Q5: human); (Q875556: Israelis, instance of, Q5: human); (Q935: Isaac Newton, instance of, Q5: human); (Q181819: Fredric March, instance of, Q5: human). and other 5434 triplets about instance of human	~54340

Table 4: Case Study: Compressed indexes from WM reduce context usage and serve as a CoT-like prompt.

Type	Content
Query	Who plays Ken Barlow in Coronation Street ?
Top-1 Retrieved Strategy	k: "Who voiced Meg on Family Guy?", v: (Person , in-tv-program, TV Program , actor, Person), s: 0.9
Instantiated Strategy (state I_0)	Compressed Index: (Ken Barlow, in-tv-program, Coronation Street) (Coronation Street, actor, ent_set_0) Entity Set: ent_set_0: {William Roache, Barbara Knox,...}

Table 5: Case Study: Using retrieved strategy to instantiate the state of new query.

11-24, and it construct a reasoning path from "profession" to "James Wong Jim", which enhances the reasoning capabilities of LLM. In contrast, without using WM, the system must store a large number of irrelevant triplets in the context, which quickly exhaust the context space, leading to critical information loss.

Table 5 demonstrates a simplified case of using the retrieved EM strategy to instantiate the state of a new query, by filling the query entities into the triplets of the strategy that match its type and search in the KG.

6 Conclusion

We propose EvoMemKG, an agent framework with dynamic memory tailored for multi-hop KG reasoning, which addresses two critical limitations in previous research: the state space explosion problem in multi-hop exploration and the inability to learn from past experiences. The core of the framework is a dual-layer memory: the working memory compresses and manages intermediate states using a relation-based compression algorithm; experience memory extracts reusable strategies from historical experiences via a strategy distillation algorithm; and a dual-loop workflow coordinates the operation of these components. Experiments on three datasets demonstrate that EvoMemKG achieves state-of-the-art performance without task-specific training, particularly on multi-hop queries.

Limitations

- Our strategy abstraction mechanism relies on replacing specific entities with their corresponding types to form generalizable rules. We currently evaluate our method on KGs with well-defined ontologies. Applying EvoMemKG to KGs with sparse, noisy, or missing type hierarchies may degrade the quality of distilled strategies, and robust adaptation to such environments remains a subject for future study.
- The speed and cost of our method depend on the LLM API (e.g., OpenAI API) and the network environment. Sometimes, calls to the API may time out or give errors, which will negatively affect our approach.
- As demonstrated in our learning curve analysis, the system’s performance improves as valid strategies accumulate in the Experience Memory. Consequently, EvoMemKG may exhibit lower efficiency in cold-start phases where the memory is empty. Developing warm-up mechanisms or transferring strategies from other tasks to mitigate this initial performance gap is a key direction for future research.

Acknowledgments

We would like to thank anonymous reviewers for their suggestions and comments sincerely. The work was partially supported by the Beijing Natural Science Foundation (L247010).

References

- Perna Agarwal, Nishant Kumar, and Srikanta Bedathur. 2024. Symkgqa: Few-shot knowledge graph question answering via symbolic program generation and execution. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10119–10140.
- Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250.
- Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. 2022. Kqa pro: A dataset with explicit compositional programs for complex question answering over knowledge base. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6101–6119.
- Ronán Conroy. 2018. *The rcsi sample size handbook*.
- Zixuan Dong, Baoyun Peng, Yufei Wang, Jia Fu, Xiaodong Wang, Yongxue Shan, and Xin Zhou. 2024. Effiqa: Efficient question-answering with strategic multi-model collaboration on knowledge graphs. In *International Conference on Computational Linguistics*.
- Prayushi Faldu, Indrajit Bhattacharya, and Mausam . 2024. RetinaQA: A robust knowledge base question answering model for both answerable and unanswerable questions. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6643–6656, Bangkok, Thailand. Association for Computational Linguistics.
- Tiezheng Guo, Qingwen Yang, Chen Wang, Yanyi Liu, Pan Li, Jiawei Tang, Dapeng Li, and Yingyou Wen. 2023. Knowledgenavigator: Leveraging large language models for enhanced reasoning over knowledge graph. *Complex Intelligent Systems*.
- Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji rong Wen. 2021. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. *Web Search and Data Mining*.
- Linmei Hu, Zeyi Liu, Ziwang Zhao, Lei Hou, Liqiang Nie, and Juanzi Li. 2024a. A survey of knowledge enhanced pre-trained language models. *IEEE Transactions on Knowledge and Data Engineering*, 36(4):1413–1430.
- Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. 2024b. Grag: Graph retrieval-augmented generation. *arXiv preprint arXiv: 2405.16506*.
- Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023a. Structgpt: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251.
- Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. 2025. KG-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9505–9523, Vienna, Austria. Association for Computational Linguistics.
- Jinhao Jiang, Kun Zhou, Xin Zhao, Yaliang Li, and Ji-Rong Wen. 2023b. ReasoningLM: Enabling structural subgraph reasoning in pre-trained language models for question answering over knowledge graph. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages

- 3721–3735, Singapore. Association for Computational Linguistics.
- Jinhao Jiang, Kun Zhou, Xin Zhao, and Ji-Rong Wen. 2023c. [UniKGQA: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph](#). In *The Eleventh International Conference on Learning Representations*.
- Kun Li, Tianhua Zhang, Xixin Wu, Hongyin Luo, James R. Glass, and Helen M. Meng. 2025. [Decoding on graphs: Faithful and sound reasoning on knowledge graphs through generation of well-formed chains](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24349–24364, Vienna, Austria. Association for Computational Linguistics.
- Runxuan Liu, Bei Luo, Jiaqi Li, Baoxin Wang, Ming Liu, Dayong Wu, Shijin Wang, and Bing Qin. 2025. [Ontology-guided reverse thinking makes large language models stronger on knowledge graph question answering](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15269–15284, Vienna, Austria. Association for Computational Linguistics.
- Linhao Luo, Yuan-Fang Li, Reza Haf, and Shirui Pan. 2024. [Reasoning on graphs: Faithful and interpretable large language model reasoning](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Elan Markowitz, Anil Ramakrishna, Jwala Dhamala, Ninareh Mehrabi, Charith Peris, Rahul Gupta, Kai-Wei Chang, and Aram Galstyan. 2024. [Tree-of-traversals: A zero-shot reasoning algorithm for augmenting black-box language models with knowledge graphs](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12302–12319, Bangkok, Thailand. Association for Computational Linguistics.
- OpenAI. 2023. <https://chat.openai.com>.
- Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jipu Wang, and Xindong Wu. 2024. [Unifying large language models and knowledge graphs: A roadmap](#). *IEEE Transactions on Knowledge and Data Engineering*.
- Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025. [Tool learning with large language models: A survey](#). *Frontiers of Computer Science*, 19(8):198343.
- Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. [Zep: A temporal knowledge graph architecture for agent memory](#). *arXiv preprint arXiv: 2501.13956*.
- Vipula Rawte, Amit Sheth, and Amitava Das. 2023. [A survey of hallucination in large foundation models](#). *arXiv preprint arXiv:2309.05922*.
- Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. [Improving multi-hop question answering over knowledge graphs using knowledge base embeddings](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4498–4507, Online. Association for Computational Linguistics.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language models can teach themselves to use tools](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc.
- Lilian Some, Wenli Yang, Michael Bain, and Byeong Kang. 2025. [A comprehensive survey on integrating large language models with knowledge-based methods](#). *arXiv preprint arXiv: 2501.13947*.
- Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018. [Open domain question answering using early fusion of knowledge bases and text](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4231–4242, Brussels, Belgium. Association for Computational Linguistics.
- Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. [Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph](#). In *The Twelfth International Conference on Learning Representations*.
- Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. 2024. [Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Alon Talmor and Jonathan Berant. 2018. [The web as a knowledge-base for answering complex questions](#). In *NAACL*, pages 641–651, New Orleans, Louisiana. Association for Computational Linguistics.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shrutu Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizen-

- stein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.
- Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.
- Junhong Wan, Tao Yu, Kunyu Jiang, Yao Fu, Weihao Jiang, and Jiang Zhu. 2025. [Digest the knowledge: Large language models empowered message passing for knowledge graph question answering](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15426–15442, Vienna, Austria. Association for Computational Linguistics.
- Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.
- Xiaojun Wu, Cehao Yang, Xueyuan Lin, Chengjin Xu, Xuhui Jiang, Yuanliang Sun, Hui Xiong, Jia Li, and Jian Guo. 2025a. Think-on-graph 3.0: Efficient and adaptive llm reasoning on heterogeneous graphs via multi-agent dual-evolving context retrieval. *arXiv preprint arXiv: 2509.21710*.
- Yaxiong Wu, Yongyue Zhang, Sheng Liang, and Yong Liu. 2025b. Sgmem: Sentence graph memory for long-term conversational agents. *arXiv preprint arXiv: 2509.21212*.
- Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.
- Guanming Xiong, Junwei Bao, and Wen Zhao. 2024. [Interactive-kbqa: Multi-turn interactions for knowledge base question answering with large language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL 2024, Bangkok, Thailand, August 11-16, 2024, pages 10561–10582. Association for Computational Linguistics.
- Yao Xu, Shizhu He, Jiabei Chen, Zihao Wang, Yangqiu Song, Hanghang Tong, Guang Liu, Jun Zhao, and Kang Liu. 2024. [Generate-on-graph: Treat LLM as both agent and KG for incomplete knowledge graph question answering](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18410–18430, Miami, Florida, USA. Association for Computational Linguistics.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chuji Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Wen-tau Yih, Matthew Richardson, Chris Meek, Mingwei Chang, and Jina Suh. 2016. [The value of semantic parse labeling for knowledge base question answering](#). In *ACL*, pages 201–206, Berlin, Germany. Association for Computational Linguistics.
- Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. 2022. [Subgraph retrieval enhanced model for multi-hop knowledge base question answering](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5773–5784, Dublin, Ireland. Association for Computational Linguistics.
- Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2024. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv: 2404.13501*.
- Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. [Expel: Llm agents are experiential learners](#). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.
- Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A survey of large language models. *arXiv preprint arXiv: 2303.18223*.
- Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2023. [Synapse: Trajectory-as-exemplar prompting with memory for computer control](#). In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.
- Chulun Zhou, Chunkang Zhang, Guoxin Yu, Fandong Meng, Jie Zhou, Wai Lam, and Mo Yu. 2026. [Improving multi-step rag with hypergraph-based memory for long-context complex relational modeling](#). *Preprint, arXiv:2512.23959*.

A Details of the Method

A.1 Algorithms

We provide a formal description of how to recall top-k strategies from EM and initialize the state for the current query in Algorithm 1, and the write and refine process in Algorithm 2.

A.2 Actions

Below, we list detailed descriptions of all actions through which the agent interacts with the knowledge graph and memory. The example and prompt of each action are shown in G.

Within the internal loop, the agent can autonomously choose one of the Explore, Extract, and Answer actions to execute at each step:

Explore is used to explore 1-hop of KG to get more information, which contains two sub-actions executed sequentially:

- $\text{select_entities}(es \in \mathcal{I}_t) \rightarrow \mathcal{R}_t$: Let the agent select an entity set (or entity) $es \in \mathcal{I}_t$ from the index of WM for further exploration. The KG environment returns all relations connected to these entities. See Table 13.
- $\text{select_relation}(r \in \mathcal{R}_t) \rightarrow \mathcal{T}_t$: Let the agent select one relation $r \in \mathcal{R}_t$ from the returned neighboring edges. The KG environment returns all triplets with $(e \in es, r, ?)$.

After executing these two sub-actions, the system retrieves all triplets from the KG that satisfy the condition (i.e., head entity in e and relation is r), completing one-hop exploration. The newly obtained triplets are auto-compressed and written into Working Memory.

Extract is used to read and reconstruct triplets from Working Memory. It includes six optional functions, from which the agent selects one based on current needs:

- $\text{FilterbyStr}(es, str) \rightarrow \mathcal{T}'$: Takes an entity set es and a string str , retrieves all one-hop neighboring triplets of entities in es , and returns those triplets whose entity or relation names contain str .
- $\text{FilterbyConstrain}(es, k, op, v) \rightarrow \mathcal{T}'$: Takes an entity set es , a relation key k , an operator op , and a value v , retrieves all one-hop neighboring triplets of entities in es , and returns those triplets that satisfy the constraint (i.e., the value of relation k satisfies $op v$).

Algorithm 1 Retrieve and Initialize from Experience Memory (EM)

Input: current query q , query entity $e_q \in \mathcal{E}$, knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$, experience memory $M_{EM} = \{(k_i, v_i, s_i)\}$, retrieval threshold K

Output: initial exploration state I_0 (compressed index)

1. Retrieve Top-K Similar Strategies from EM

$\mathbf{q} \leftarrow \text{Emb}(q)$ {Get query embedding}

$S \leftarrow \emptyset$ {Initialize strategy set}

for all $(k_i, v_i, s_i) \in M_{EM}$ **do**

$s_i \leftarrow \text{sim}(\mathbf{q}, k_i)$ {Compute semantic similarity}

$S \leftarrow S \cup \{(s_i, v_i)\}$

end for

Sort S by s_i in descending order

$S_{\text{top}} \leftarrow \text{top-}K$ strategies from S

2. Build & Retrieve Query Graphs

$E \leftarrow \{e_q\}$ {Initialize entity set with query entities}

$QG \leftarrow \emptyset$ {Initialize query graph set}

$T \leftarrow \emptyset$ {Initialize triple set}

$step \leftarrow 3$ {Max search step}

while $step > 0$ **do**

2.1 Build Query Graphs

for all $(\tau(e_i), r_i, \tau(e_{i+1})) \in v = (\tau(e_0), r_0, \tau(e_1), \dots, r_n, \tau(e_{n+1})) \in S_{\text{top}}$ **do**

for all $e \in E$ **do**

if $\tau(e) = \tau(e_i)$ **then**

$QG \leftarrow QG \cup \{(e, r_i, ?)\}$

else if $\tau(e) = \tau(e_{i+1})$ **then**

$QG \leftarrow QG \cup \{(? , r_i, e)\}$

end if

end for

end for

2.2. Retrieve triplets from KG

for all $qg \in QG$ **do**

if $qg = (e, r, ?)$ **then**

$T_{\text{new}} \leftarrow \{(e, r, t) | (e, r, t) \in \mathcal{G}\}$

$E \leftarrow E \cup \{t | (e, r, t) \in T_{\text{new}}\}$

end if

if $qg = (?, r, e)$ **then**

$T_{\text{new}} \leftarrow \{(h, r, e) | (h, r, e) \in \mathcal{G}\}$

$E \leftarrow E \cup \{h | (h, r, e) \in T_{\text{new}}\}$

end if

$T \leftarrow T \cup T_{\text{new}}$

end for

$step \leftarrow step - 1$

end while

3. Initialize Working Memory

$I_0 \leftarrow \mathcal{C}(T)$ {Apply Working Memory compression}

return I_0

Algorithm 2 Write/Refine Experience Memory (EM)

Input: successful reasoning path $\mathcal{P}_c = (e_q, r_1, e_1, \dots, r_n, e_a)$, query q , knowledge graph \mathcal{G} , experience memory M_E , similarity threshold θ

Output: updated M_E

1. Strategy Distillation

$(k_{\text{new}}, v_{\text{new}}, s_{\text{new}}) \leftarrow D(\mathcal{P}_c, q, \mathcal{G})$

$k_{\text{new}} \leftarrow \text{Emb}(q)$

$v_{\text{new}} \leftarrow (\tau(e_q), r_1, \tau(e_1), \dots, r_n, \tau(e_a))$

$s_{\text{new}} \leftarrow \text{LLM}(q, \mathcal{P}_c, e_a)$

2. Check for Similar Strategies

found \leftarrow False

for all existing strategy $(k_i, v_i, s_i) \in M_E$ **do**

$\text{sim}_i \leftarrow \text{cosine_sim}(k_{\text{new}}, k_i)$

if $\text{sim}_i \geq \theta$ **then**

 found \leftarrow True

if $s_{\text{new}} > s_i$ **then**

$(k_i, v_i, s_i) \leftarrow (k_{\text{new}}, v_{\text{new}}, s_{\text{new}})$

end if

break

end if

end for

3. Add New Strategy if No Similar Found

if not found **then**

$M_E = M_E \cup (k_{\text{new}}, v_{\text{new}}, s_{\text{new}})$

end if

return M_E

- $\text{GetRelation}(e, e') \rightarrow \mathcal{T}'$: Takes two entities e and e' , and returns all triplets in the KG that connect these two entities (regardless of direction).
- $\text{LogicOperation}([\text{es}_1, \text{es}_2, \dots], \text{op}) \rightarrow \text{es}'$: Takes a list of entity sets and a logical operator $\text{op} \in \{\text{intersection, union}\}$, and returns the intersection or union of these entity sets.
- $\text{Count}(\text{es}, k, \text{op}, v) \rightarrow \text{num}$: Similar to FilterbyConstrain , but returns only the number of triplets satisfying the constraint.
- $\text{Verify}(\text{es}, k, \text{op}, v) \rightarrow \text{bool}$: Similar to FilterbyConstrain , but returns a Boolean value indicating whether any triplets satisfy the constraint.

When executing the Extract action, the agent must generate input parameters that meet the formatting requirements of the selected function.

Answer is executed when the agent believes that the current information is sufficient to produce an answer or reach the max step limit. It consists of three sequentially executed sub-actions:

- $\text{select_index}(\mathcal{I}_t) \rightarrow \mathcal{T}'$: The agent selects an index from the current state which may contain the correct answer, and reconstructs the raw triplets by Read WM (see Table 16).
- $\text{generate_answer}(s, \mathcal{T}') \rightarrow \text{ans}$: Generates the final answer based on the current state and the reconstructed triplets (see Table 17).
- $\text{evaluate_answer}(s, \text{ans}) \rightarrow \text{score}$: Evaluates the correctness of the generated answer, checks whether it follows the reasoning path, and assigns a confidence score between 0 and 1 (see Table 18).

B Datasets

B.1 Dataset Statistics

The dataset statistics are shown in Table 6.

B.2 Sampling Strategy

Unlike previous methods (Xiong et al., 2024) that arbitrarily choose the number of random samples, we calculate the number of sampled data on different datasets based on RCSI theory (Conroy, 2018) to ensure that the confidence level and confidence interval are $95\% \pm 3\%$. The sampled data number of each dataset are in Table 6.

Dataset	Split	#Sample	#Raw	KG	ent_num/rel_num/trip_num
WQSP	test	646	1639	FreeBase	2721501/13439/6829392
CWQ	test	819	3531	FreeBase	2721501/13439/6829392
KGAPro	dev	978	11797	WikiData	17754/993/607198

Table 6: Dataset Statistics: #Sample is the number of data we randomly sampled and used; #Raw is the total number of data in different splits of each dataset; ent_num/rel_num/trip_num is the number of entities, the number of relations, and the number of triplets, in different KGs.

C Evaluation

We directly use the evaluation script¹ from LMP (Wan et al., 2025) to calculate Hits@1. It compute Hits@1 by evaluating whether the text or ID of the correct answers appears in the generated answer. Due to the LLM API’s randomized nature, the same inputs can give different outputs. Therefore, we ran each sample 3 times and took the best results.

D Baselines

We compare our EvmMemKG with 20 methods from four categories:

- Full fine-tuned language models: GraftNet (Sun et al., 2018), EmbedKGQA (Saxena et al., 2020), NSM (He et al., 2021), Subgraph Retrieval (Zhang et al., 2022), ReasoningLM (Jiang et al., 2023b), UniKGQA (Jiang et al., 2023c).
- LLM Baselines: Llama-2 (Touvron et al., 2023), GPT-3.5, GPT-4 (OpenAI, 2023), Qwen3 (Yang et al., 2025). The results of these models are copied from Jiang et al. (2025) and Agarwal et al. (2024). The results of GPT-4.1-mini and GPT-5-mini are tested by us.
- Indirect interaction agent: StructGPT (Jiang et al., 2023a), SymKGQA (Agarwal et al., 2024), Interactive-KBQA (Xiong et al., 2024), DoG (Li et al., 2025).
- Direct interaction agents: ROG (Luo et al., 2024), EffiQA (Dong et al., 2024), KG-Agent (Jiang et al., 2025), ToG (Sun et al.), GoG (Xu et al., 2024), ORT (Liu et al., 2025), LMP (Wan et al., 2025).

¹<https://github.com/wanjunhong0/LMP/blob/main/eval.py>

E Experiment Settings

E.1 Main Experiment Settings

Memory Configuration. For Working Memory, we compress triplets by clustering entities under the same relation, displaying up to 5 example entities per set. For Experience Memory, we use the *stella_en_400M_v5*² sentence encoder for similarity computation and set the similarity threshold θ to 0.8 for policy merging. The max step for initialize EM strategy is 3, as shown in Algorithm 1.

Exploration Limits. We set maximum steps per query as: WQSP=10, CWQ=15, KQA=20, based on dataset complexity. The confidence threshold for whether to return the final answer or continue exploration is 0.9.

Reproducibility. We run each query three times and report the best result to account for LLM randomness. All open-sourced LLMs are run on a single A100 GPU.

E.2 Analyse Experiment Settings

All average performance metrics in the analysis experiments are based on the CWQ and KGA datasets and the gpt-4.1-mini and gpt-5-mini models.

The hop is the maximum path length between any two entities in the labeled subgraph (i.e., the diameter of the subgraph) from the datasets.

The exploration steps Figure 4 is the number of *Explore* actions performed by the agent in the inner loop.

F Additional Experiments

F.1 More Ablation Study

Method	WQSP	CWQ	BaseModel
EvoMemKG	75.2	48.4	Llama-3-3B Instruct
w/o EM	72.1	40	
w/o EM+WM	38	12.1	
EvoMemKG	84.8	63.6	GPT-3.5
w/o EM	79.4	62	
w/o EM+WM	71.8	54.4	
EvoMemKG	86.6	73.3	GPT-4o
w/o EM	84.7	70	
w/o EM+WM	81.3	66.6	

Table 7: More Ablation Studies.

²https://huggingface.co/dunzhang/stella_en_400M_v5

In Table 7, we show the ablation study results on more base models. Both our EM and WM can improve the model performance, proving the effectiveness of the two memory mechanisms.

Moreover, we also test our memory mechanism on a small LLM, llama-3b. We find that our method remains effective on smaller LLM, even though it has only 3B parameters. Thus, we believe the size of the model has little impact on our approach, while the capability of the model is more important.

F.2 Effects of Experience Memory

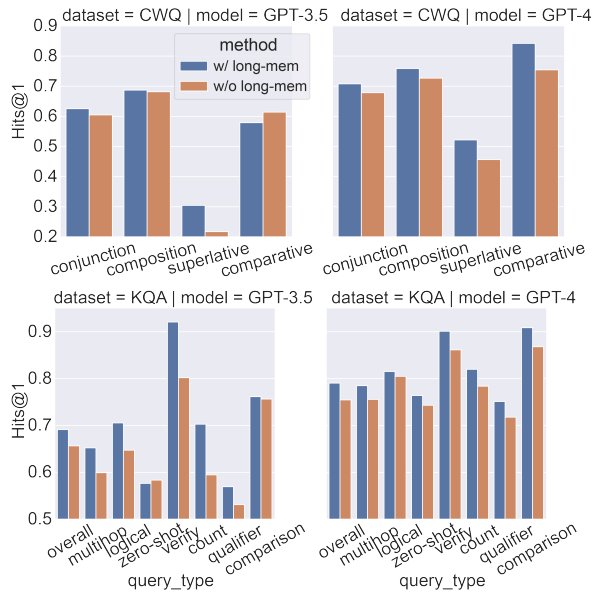


Figure 7: Performance with or without experience memory on different types of questions.

In Figure 7, we show the impact of EM on different types of queries. In general, using EM improves almost all types of queries. Improvements are more significant in certain types of queries (such as superlative and count), which are often complex questions. That suggests using long-term memory can evolve the agent’s ability to deal with complex queries that were previously unsolvable.

F.3 Efficiency Analysis

In Figure 8, we show the performance and efficiency trade-off between our EvoMemKG and other baselines. It demonstrates that EvoMemKG achieves a favorable balance between cost and effectiveness, yielding superior results at a lower cost compared to the ToG method.

In Tables 8 and 9, we conducted step-level and sample-level efficiency analyses to show the runtime, token usage, and cost of the inner loop and outer loop.

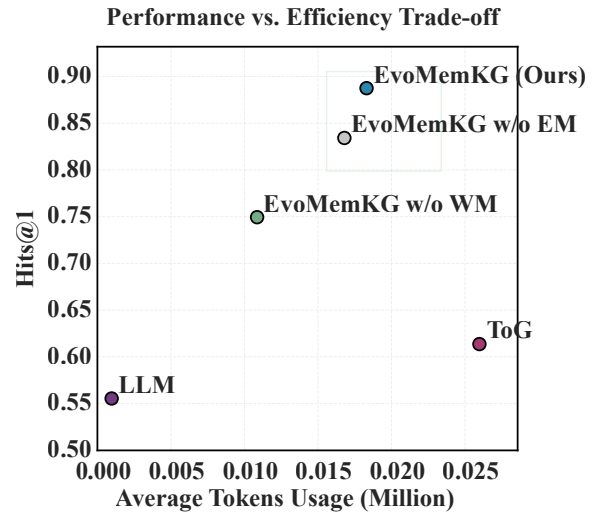


Figure 8: Performance and average token usage trade-off.

Table 8: Step-level (inner loop) Efficiency Analysis

Dataset	Base Model	Runtime (s)	Input Tokens	Output Tokens	Cost (\$)
WQSP	ChatGPT	1.16	1055.93	62.19	0.00062
	GPT-4o	3.82	935.10	113.38	0.0034
KQA	ChatGPT	1.23	1218.59	65.84	0.0007
	GPT-4o	2.89	1031.07	126.18	0.0038

Table 9: Sample-level (outer loop) Efficiency Analysis

Dataset	Base Model	Runtime (s)	Input Tokens	Output Tokens	Cost (\$)	Call Count
WQSP	ChatGPT	18.041	17505.27	950.55	0.0101	15.68
	GPT-4o	50.21	14081.25	1506.31	0.0502	13.765
KQA	ChatGPT	31.27	33341.17	1643.04	0.0191	25.65
	GPT-4o	52.34	21740.92	2293.17	0.0772	18.64

F.4 Hyperparameter Sensitivity Analysis

Table 10 illustrates the impact of the number of examples k in the compressed index on the final performance and compression ratio. It can be observed that when $k=5$, the LLM is sufficient to understand the content contained in the entity set and achieves a good balance between performance and compression ratio.

k	CWQ (Hits@1)	Comp. Ratio
1	88.2%	99.1%
3	89.1%	98.7%
5	89.5%	98.2%
10	89.4%	97.1%

Table 10: The impact of the number of examples k in the compressed index.

Random Seed	CWQ Hits@1	KQA Hits@1
42 (original)	89.47%	93.62%
123	89.21%	93.38%
2048	89.53%	93.71%
Mean \pm Std	89.40% \pm 0.17%	93.57% \pm 0.17%

Table 11: The impact of random seeds (query order).

Table 11 shows the performance of EvoMemKG under different random seeds (query orders). As shown in the Table, the standard deviation across orders is 0.17%, which demonstrates that EvoMemKG’s performance is stable regardless of query order.

G Prompts

We give some cases in this section to show the prompt we used for each action and the complete working process.

Table 12 shows how the prompt is used for the agent to select the next action to perform based on the current state.

If the agent selects EXPAND_KG, the *Explore* actions will be executed, first selecting entities (as Table 13) and then selecting relations to explore (as Table 14). After that, the state will be updated and moved to the next step.

If the agent selects FILTER, the *Extract* actions will be executed. As shown in Table 15, the agent needs to select a function and give the required input parameters. After that, the states will be updated and moved to the next step.

If the agent selects ANSWER, the *Answer* actions will be executed. It first lets the agent select and index from the state to reconstruct (as Table 16). Then, the agent generates a final answer based on the final state (as Table 17). Finally, the agent gives a score for the final answers to check if it is correct (as Table 18).

Input (Prompt)	<p>Original Query: what was sir isaac newton 's inventions</p> <p>Knowledge Graph Entities: topic entities: m.03s9v: Isaac Newton ent_set_0: contain 2 entities m.01nhc9: Reflecting telescope m.079vqm: Pet door</p> <p>Knowledge Graph Edges: (m.03s9v: Isaac Newton, law.inventor.inventions, ent_set_0)</p> <p>Previous actions: EXPAND_KG: SELECT ENTITIES: ['m.03s9v'] SELECT RELATION: ['law.inventor.inventions'] Observation: after EXPAND_KG 3 new entities, 2 new edges added.</p> <p>You are a superintelligent AI equipped with the ability to search a knowledge graph for definitive, up-to-date answers. Your task is to explore the knowledge graph to answer the above questions. You can choose from the following actions to explore or process KG to find the correct answer. Think in detail before acting.</p> <p>Available actions: 'EXPAND_KG' - Explore the external Knowledge Graph to get more information. This is a helpful action for getting to the correct answer. 'FILTER' - Extract the exact information from the current ent_sets or perform logical operations on them by using six functions: [FilterbyCondition, FilterbyStr, LogicOperation, FindRelation, Count, Verify]. This is a useful action for obtaining valid information from ent_sets. 'ANSWER' - Use only if the answer of Original Query appears explicitly in the current Knowledge Graph Entities or Edges.</p> <p>First think carefully about how the current knowledge graph relates to the original question (or sub-question). Then choose an action from Available actions.</p> <p>E.g. THINK:... SELECT ACTION ['ANSWER', 'FILTER', 'EXPAND_KG']:...</p>
Output	<p>THINK: The original query asks about the inventions of Sir Isaac Newton. The current knowledge graph entities already include two inventions attributed to him: the reflecting telescope and the pet door. However, there might be more inventions or contributions by Newton that are not yet captured in the current knowledge graph. To ensure a comprehensive answer, it would be beneficial to explore the knowledge graph further to see if there are additional inventions associated with Isaac Newton. SELECT ACTION ['EXPAND_KG']: Explore the knowledge graph further to find more inventions attributed to Isaac Newton.</p>

Table 12: The example and prompt of Observation. The red text in Input is the variable part, determined by historical trajectories, while the black text is the fixed prompt part.

Input (Prompt)	<p>Original Query: what was sir isaac newton 's inventions</p> <p>Knowledge Graph Entities: topic entities: m.03s9v: Isaac Newton ent_set_0: contain 2 entities m.01nhc9: Reflecting telescope m.079vqm: Pet door</p> <p>Knowledge Graph Edges: (m.03s9v: Isaac Newton, law.inventor.inventions, ent_set_0)</p> <p>Previous actions: EXPAND_KG: SELECT ENTITIES: ['m.03s9v'] SELECT RELATION: ['law.inventor.inventions'] Observation: after EXPAND_KG 3 new entities, 2 new edges added. EXPAND_KG:</p> <p>Objective: The original query asks about the inventions of Sir Isaac Newton. The current knowledge graph entities already include two inventions attributed to him: the reflecting telescope and the pet door. However, there might be more inventions or contributions by Newton that are not yet captured in the current knowledge graph. To ensure a comprehensive answer, it would be beneficial to explore the knowledge graph further to see if there are additional inventions associated with Isaac Newton.</p> <p>Your current task is to select an entity_set or entities from the KG to expand to get more information about.</p> <p>If you want to explore a specific entity, select exactly one entity from the following entity_ids: ['m.03s9v', 'm.079vqm', 'm.01nhc9'] If you want to explore more than one entity or a set of entities, select from the following entity sets: ['ent_set_0']</p> <p>You MUST select exactly one ent_set or entity_id from those listed above.</p> <p>E.g. SELECT ENTITIES:</p>
	Output

Table 13: The example and prompt of *select_entities*. The red text in Input is the variable part, determined by historical trajectories, while the black text is the fixed prompt part.

Input (Prompt)	<p>Original Query: what was sir isaac newton 's inventions</p> <p>Knowledge Graph Entities: topic entities: m.03s9v: Isaac Newton ent_set_0: contain 2 entities m.01nhc9: Reflecting telescope m.079vqm: Pet door</p> <p>Knowledge Graph Edges: (m.03s9v: Isaac Newton, law.inventor.inventions, ent_set_0)</p> <p>Previous actions: EXPAND_KG: SELECT ENTITIES: ['m.03s9v'] SELECT RELATION: ['law.inventor.inventions'] Observation: after EXPAND_KG 3 new entities, 2 new edges added. EXPAND_KG:</p> <p>Objective: The original query asks about the inventions of Sir Isaac Newton. The current knowledge graph entities already include two inventions attributed to him: the reflecting telescope and the pet door...</p> <p>Your current task is to select a relation to expand along for the selected entities. The selected entities are: ['ent_set_0']</p> <p>The options of relations to choose from are: ['astronomy.telescope_principle_type.telescopic_classifications_of_this_technique', 'law.inventor.inventions', 'law.invention.inventor', 'type.type.instance', 'type.object.key', ...]</p> <p>First think carefully about how to choose an appropriate relation to get close to the correct answer of original question / sub-questions / Objective.</p> <p>Then, select exactly one relation from the options listed above.</p> <p>E.g. THINK:... SELECT RELATION:...</p>
	Output

Table 14: The example and prompt of *select_relations*. The red text in Input is the variable part, determined by historical trajectories, while the black text is the fixed prompt part.

Input (Prompt)	<p>Original Query: who does jeremy shockey play for in 2012</p> <p>Knowledge Graph Entities: topic entities: m.076ltd: Jeremy Shockey ent_set_0: contain 4 entities m.0jb8dd7: sports.sports_team_roster(m.0jb8dd7) m.0hqff5d: sports.sports_team_roster(m.0hqff5d) m.0hqff53: sports.sports_team_roster(m.0hqff53) m.0hqff4t: sports.sports_team_roster(m.0hqff4t) ent_set_1:...</p> <p>Knowledge Graph Edges: (m.076ltd: Jeremy Shockey, sports.pro_athlete.teams, ent_set_0)</p> <p>Previous actions: EXPAND_KG: SELECT ENTITIES: ['m.076ltd'] SELECT RELATION: ['sports.pro_athlete.teams'] Observation: after EXPAND_KG 28 new entities, 27 new edges added. FILTER:</p> <p>Objective: The original query is asking about the team Jeremy Shockey played for in 2012... Your current task is to filter out valid information from the current knowledge graph by using a list of functions.</p> <p>List of Functions:</p> <ul style="list-style-type: none"> - FilterbyStr Description - Filter out entities related to the given string. i.e., one-hop neighborhoods containing the string. Wide range filtering when exact conditions are uncertain. Inputs - (string) - FilterbyCondition Description - Filtering entities that have specific edge and its corresponding value satisfy a specific condition, i.e. find the <entities>whose <key / relation>is <operator><value>. Input - (key, operator, value); the operator should be one of [=, !=, <, <=, >, >=, argmax, argmin], The value should be a string or "ANY" when there is no value. - FindRelation Description - Find the relation / connection of two specific entities, i.e. get the <relation>of <entity_1>and <entity_2>. Input - (entity_1, entity_2) - LogicOperation Description - Processing a list of ent_sets with a specific logical operation, i.e. conduct <operator>on entity set list [<entity_set_id>]. Input - (operator, [ent_set_xx, ent_set_xx, ...]), the operator should be one of [intersect, union] - Count Description - Similar to FilterbyCondition, but only returns the number of entities that satisfy the condition, i.e. get the number of <entities>whose <key>is <operator><value>. Input - (key, operator, value), the operator should be one of [=, !=, <, <=, >, >=] - Verify Description - Similar to FilterbyCondition, but only returns a boolean value indicating whether any entity meets the given conditions, i.e. For <entities>, is his/her/its <key><operator><value>. Input - (key, operator, value), the operator should be one of [=, !=, <, <=, >, >=] <p>First think carefully about what information has been filtered out and how choosing the right function helps to answer the original question/sub-question. Then select ent_set(s) to process on from: ['ent_set_4', 'ent_set_2', 'ent_set_1', 'ent_set_0', 'ent_set_3', 'ent_set_5', 'ent_set_6'] Finally, select a function and give an input that meets the formatting requirements of that function.</p> <p>Do not use the same functions and inputs that have appeared in Previous actions!</p> <p>e.g. THINK: ... Select Entity Set(s): ... Select Function: ... Function Input: (...)</p>
Output	<p>THINK: To determine which team Jeremy Shockey played for in 2012, I need to identify the team associated with the time period that includes 2012. The relevant entity sets are ent_set_0 (sports team roster). I need to filter the sports team roster entities by the time period that includes 2012.</p> <p>Select Entity Set(s): ent_set_0 Select Function: FilterbyCondition Function Input: ('sports.sports_team_roster.to', '>=', '2012-01-01')</p>

Table 15: The example and prompt of *Extract*. The red text in Input is the variable part, determined by historical trajectories, while the black text is the fixed prompt part.

<p>Input (Prompt)</p>	<p>Original Query: what was sir isaac newton 's inventions</p> <p>Knowledge Graph Entities: topic entities: m.03s9v: Isaac Newton ent_set_0: contain 2 entities m.01nhc9: Reflecting telescope m.079vqm: Pet door</p> <p>Knowledge Graph Edges: (m.03s9v: Isaac Newton, law.inventor.inventions, ent_set_0) (ent_set_0, law.invention.inventor, ent_set_1: m.03s9v: Isaac Newton)</p> <p>Objective: The original query is asking about the inventions of Sir Isaac Newton...</p> <p>Your current task is to find the answer to the original query from the above knowledge graph.</p> <p>If the answer appears directly in Knowledge Graph Edges, select the corresponding edges. If the answer appears in Knowledge Graph Entities, select the corresponding ent_sets or entities. If the question cannot be answered based on the current KG information, select "NO ANS". Note the direction of the triplets, if you are not sure which is the correct direction, select more than one.</p> <p>First think carefully about how the current knowledge graph relates to the original question and where to extract the valid information. Then, double-check if each constraint in the original query is satisfied by the corresponding ent_set or edge in above KG. Finally, if all constraints are satisfied, select the ent_sets or edges that contain the final answer.</p> <p>e.g. THINK: ... CHECK: constraint 1: which ent_set or edge satisfies it ... constraint 2: which ent_set or edge satisfies it ... constraint 3: SELECT: ent_sets or edges or NO ANS</p>
<p>Output</p>	<p>THINK: The original query is asking about the inventions of Sir Isaac Newton. The knowledge graph provides information about two inventions associated with Isaac Newton: the reflecting telescope and the pet door. The relevant information is captured in the knowledge graph edges, which show the relationship between Isaac Newton and his inventions.</p> <p>CHECK: constraint 1: The query asks for inventions by Isaac Newton. The edge (m.03s9v: Isaac Newton, law.inventor.inventions, ent_set_0) satisfies this by linking Isaac Newton to his inventions. constraint 2: The inventions themselves are specified in ent_set_0, which includes the reflecting telescope (m.01nhc9) and the pet door (m.079vqm). SELECT: (m.03s9v: Isaac Newton, law.inventor.inventions, ent_set_0)</p>

Table 16: The example and prompt of *select_triplets/entities*. The red text is the variable part of the prompt, while the black is fixed.

Input (Prompt)	<p>Original Query: what was sir isaac newton 's inventions</p> <p>Knowledge Graph Entities: topic entities: m.03s9v: Isaac Newton ent_set_0: contain 2 entities m.01nhc9: Reflecting telescope m.079vqm: Pet door</p> <p>Knowledge Graph Edges: (m.03s9v: Isaac Newton, law.inventor.inventions, ent_set_0) (ent_set_0, law.invention.inventor, ent_set_1: m.03s9v: Isaac Newton)</p> <p>Objective: The original query is asking about the inventions of Sir Isaac Newton...</p> <p>This information extracted from the current KG may be helpful: (m.03s9v: Isaac Newton, law.inventor.inventions, ent_set_0)</p> <p>Give your best answer based on the knowledge graph. If the knowledge graph do not contain the correct answer, try to answer based on your own knowledge. First think carefully about what the original question is asking and which triplets or entities contain the answer. The correct answer will usually appear explicitly in a Knowledge Graph Edge or ent_set containing no more than 5 entities. Pay attention to the direction of the triplets to make sure they are consistent with the question.</p> <p>If the question is a judgment or comparison, answer "Yes" or "No". If the question asks for a specific number or count, answer with a specific number. If the answer appears directly in the Knowledge Graph please give its IDs, which starts with m.xxx or Q...</p> <p>e.g. THINK: ... ANSWER: ... ANSWER_IDS: ...</p>
Output	<p>THINK: The original query is asking about the inventions attributed to Sir Isaac Newton. The knowledge graph indicates that Isaac Newton is associated with two inventions: the reflecting telescope and the pet door. These are the inventions listed in the knowledge graph.</p> <p>ANSWER: Sir Isaac Newton is credited with inventing the reflecting telescope and the pet door. ANSWER_IDS: m.01nhc9 (Reflecting telescope), m.079vqm (Pet door)</p>

Table 17: The example and prompt of *generate_answer*. The red text is the variable part of the prompt, while the black is fixed.

Input (Prompt)	<p>Original Query: what was sir isaac newton 's inventions</p> <p>Knowledge Graph Entities: topic entities: m.03s9v: Isaac Newton ent_set_0: contain 2 entities m.01nhc9: Reflecting telescope m.079vqm: Pet door</p> <p>Knowledge Graph Edges: (m.03s9v: Isaac Newton, law.inventor.inventions, ent_set_0) (ent_set_0, law.invention.inventor, ent_set_1: m.03s9v: Isaac Newton)</p> <p>Provided answer: Sir Isaac Newton is credited with inventing the reflecting telescope and the pet door. ANSWER_IDS: m.01nhc9 (Reflecting telescope), m.079vqm (Pet door)</p> <p>Your task is to score the correctness of the provided answer based on the original query, and the knowledge graph. Double-check that each constraint in the original query is satisfied by the edges or ent_sets in KG. Give a pessimistic score from 0.0 to 1.0 on how likely the answer is to be correct.</p> <p>0.0 if definitely wrong or avoids answering the question 0.0 if unable to answer based on the knowledge graph 0.3 if not all constraints (sub-questions) in the original query are satisfied or only some of them are satisfied 0.5 if unsure 0.7 for probably correct but not confirmed in knowledge graph 1.0 for definitely correct and confirmed in knowledge graph</p> <p>First think carefully about what the original question is asking and whether the current knowledge graph fulfills all of the requirements (each sub-question) in the original query. Then give a score about how likely the provided answer is to be correct.</p> <p>E.g. THINK: ... SCORE: ...</p>
Output	<p>THINK: The original query asks for the inventions of Sir Isaac Newton. The knowledge graph provides two entities related to Isaac Newton's inventions: the reflecting telescope and the pet door. The provided answer lists both of these inventions, which are confirmed by the knowledge graph. Therefore, the answer satisfies the constraints of the original query by listing the inventions associated with Isaac Newton as per the knowledge graph. SCORE: 1.0</p>

Table 18: The example and prompt of *evaluate_answer*. The red text in Input is the variable part, determined by historical trajectories, while the black text is the fixed prompt part.