Thought-Action Graph Reasoning: Faithful and Efficient Reasoning of Large Language Models via Reusing Past Experience

Zhixiao Qi[1], Feng Huang[2,3], Yunqi Zhang[3], Shijie Zhang[4], Qingqing Sun[4], Yongfeng Huang[1], Minghu Jiang[1], Shuai Chen[4]*, Tianyi Zhang[4]*
[1]Tsinghua University
[2]Beijing University of Posts and Telecommunications
[3]Zhongguancun Laboratory [4]Ant International, Ant Group
qizx24@mails.tsinghua.edu.cn
shuai.cs@antgroup.com, zhangtianyi718@gmail.com

Abstract

Large language models (LLMs) often hallucinate in question answering (QA) tasks due to a lack of factual knowledge. While integrating knowledge graphs (KGs) with LLMs has alleviated this issue, existing methods suffer from poor generalization or low reasoning efficiency, and critically, they overlook the learning and reuse of reasoning paths from past experiences. To address these challenges, we introduce **T**hought-**A**ction **G**raph (TAG), a structured repository of reasoning experiences. TAG decomposes successful LLM-KG interaction trajectories into fine-grained semantic operators, which are stored in TAG constructed by the thought layer and action layer. Building upon TAG, we propose a novel KGQA paradigm —TAG-Reasoning (TAGR). TAGR first retrieves and assembles reasoning blueprints from TAG, and then guides LLMs to efficiently execute on KG according to them. Through this approach, TAGR transforms the computationally expensive online exploration process of LLMs into an offline process of TAG retrieval and assembly. Experimental results on multiple KGQA benchmarks demonstrate that TAGR significantly outperforms state-of-the-art methods across key metrics, while drastically reducing the number of LLM calls and generated tokens. This work[†] opens new avenues for building continual learning, efficient, and faithful KGQA systems.

1 Introduction

Large language models (LLMs) have demonstrated remarkable performance across a wide range of natural language processing (NLP) tasks (Brown et al., 2020; Zhao et al., 2023; Qiao et al., 2023). However, when confronted with complex questions, LLMs often suffer from hallucinations and a significant drop in accuracy due to their lack of factual knowledge (Hu et al., 2023; Huang et al., 2023). To address this issue, a growing body of research integrates knowledge graphs (KGs) with LLMs (Sun et al.; Luo et al., 2024a; Pan et al., 2024). In this paradigm, KGs provide factual information to support the LLM's reasoning process, aiming to achieve faithful reasoning in knowledge graph question answering (KGQA) (Peng et al., 2024; Han et al., 2025).

KGs organize world knowledge in a structured form, enhancing the reasoning capabilities of LLMs through various methods (Bollacker et al., 2008). Current methods for KG-augmented LLM reasoning can be broadly categorized into two types: 1) Retrieval-based methods: These approaches retrieve relevant information from KGs and inject it into the LLM's reasoning process as part of the prompt (as illustrated in Figure 1(a)) (Li et al., 2023; Dehghan et al., 2024; Yang et al., 2024b); and 2) Agent-based methods: These methods treat LLMs as agents that iteratively interact with the KG to identify a suitable reasoning path (as depicted in Figure 1(b)) (Sun et al.; Luo et al., 2024a; Dong et al., 2025; Jiang et al., 2025).

However, both methods mentioned above have limitations. Retrieval-based methods rely too heavily on the quality of the retriever, making it difficult to generalize to unseen questions or concepts. Agent-based methods require multiple calls to LLMs to interact with the KG, resulting in low efficiency. Whether retrieval-based or agent-based, neither of these approaches fully considers a key research aspect in KGQA: learning reusable reasoning paths from historical tasks and achieving sustained adaptation and evolution based on past successful experiences. Existing approaches typically treat each question as an isolated query, initiating their reasoning process from scratch. This results in a lack of capability to identify and reuse solution patterns from similar problems. Therefore, these approaches are difficult to accumulate

*Corresponding author.
†Code is available at https://github.com/THUqizx/Thought-Action_Graph_Reasoning

Findings of the Association for Computational Linguistics: ACL 2026, pages 31433–31449
July 2-7, 2026 ©2026 Association for Computational Linguistics

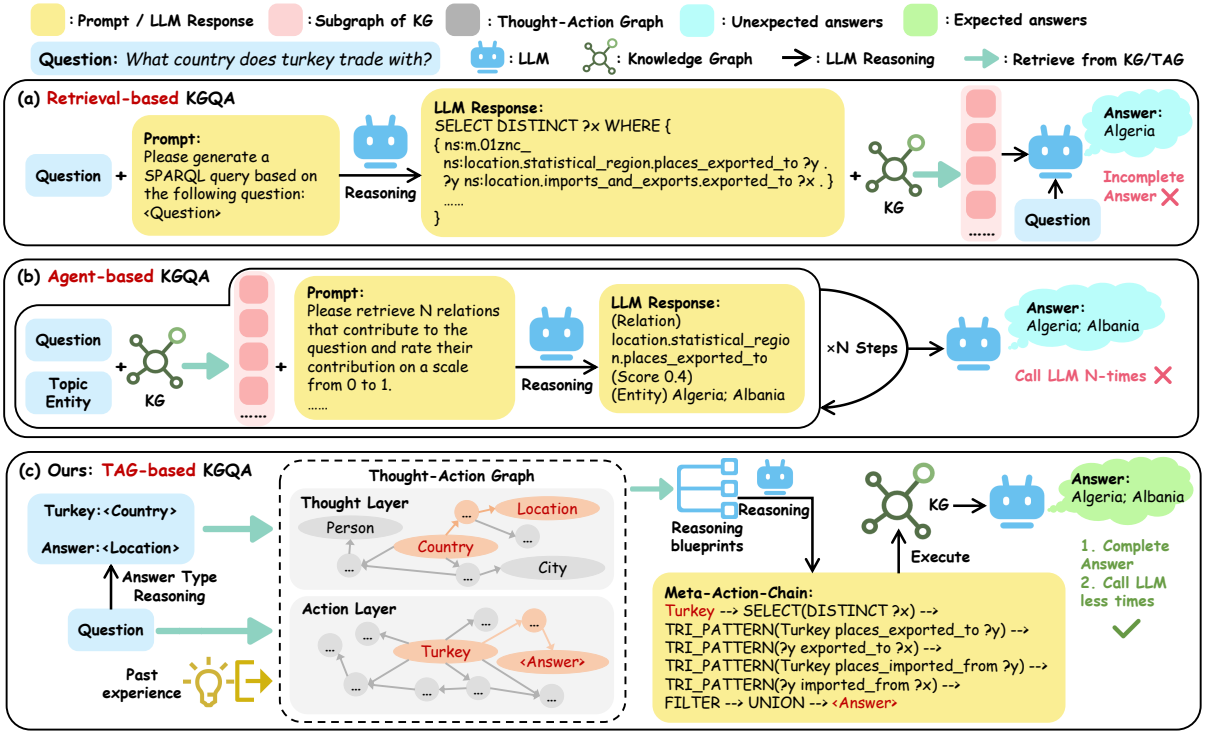

Figure 1: Three frameworks for implementing KGQA via LLMs: (a) Retrieval-based methods; (b) Agent-based methods; (c) TAG-based methods.

effective reasoning strategies and cannot quickly call and adjust previously validated path templates when facing new questions or complex multi-hop queries. This limits the potential for overall performance generalization and efficiency improvement (Shen et al., 2025; Jiang et al., 2025).

Drawing inspiration from past experiences to guide LLM reasoning research, we attempt to implement similar capabilities in KGQA. We introduce the Thought-Action Graph (TAG), which is a framework aimed at achieving this goal. At its heart, TAG distills and codifies reusable reasoning logic from successful LLM-KG interaction traces. Specifically, TAG deconstructs complex interaction sequences into a set of fine-grained, semantically explicit fundamental operators. These operators are systematically organized within a structured graph, which in turn forms a dynamic and searchable repository of reasoning experience. As a result, when encountering new queries, TAG will guide LLM to explore KG and arrive at the final answer.

As illustrated in Figure 1(c), the TAG-based KGQA process comprises two key stages. First, for a given question, the approach constructs meta-action-chains (MACs) from the TAG to guide the LLM's exploration of KG. Subsequently, the LLM follows this structured sequence of actions to perform reasoning efficiently over the KG. Unlike retrieval-based approaches that are constrained by static training data and exhibit limited generalization, TAG decomposes unseen natural language questions into a sequence of known, empirically validated basic operations. This enables LLMs to maintain reliable and faithful reasoning when faced with novel concepts or complex queries. In addition, compared with agent-based approaches that require multiple trial and error interactions, TAG-based KGQA fundamentally eliminates the extensive trial-and-error costs in LLMs exploration by proactively providing reasoning blueprints for LLMs, significantly improving computational efficiency and the stability of answer generation. The main contributions of this work are summarized as follows:

1. We propose the novel concept of a Thought-Action Graph (TAG) and design dedicated methods for its construction and retrieval. The TAG decomposes past interaction trajectories between LLMs and KGs into a series of fine-grained, semantically explicit fundamental reasoning operators, stored in a structured graph format.

2. We introduce TAG-Reasoning (TAGR), a

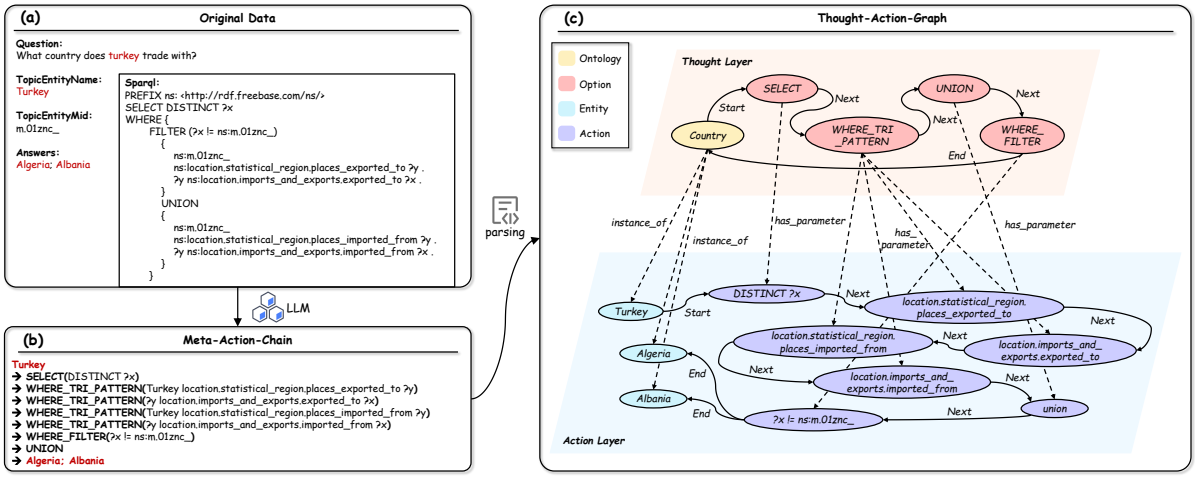

Figure 2: An example of TAG construction. (a) First, we locate the question, topic entity, answer, and SPARQL from the data. (b) Then, we use GPT-4o-mini to convert SPARQL into MAC. A complete MAC starts from the topic entity and ends at answers after multiple steps. (c) Finally, we post-process the MAC by decomposing it into basic nodes and edges in TAG and linking them into the TAG.

new TAG-based paradigm for KGQA. TAGR proactively provides structured reasoning blueprints for LLMs from TAG, named meta-action-chains (MACs). It transforms the costly trial-and-error interaction between LLMs and KGs into an efficient process of TAG retrieval and combination, which significantly reduces the number of LLM calls and ensures the efficiency of KGQA.

3. We conduct experiments on multiple KGQA reasoning benchmarks. The results demonstrate that our proposed TAGR outperform traditional methods on several key metrics. This validates the effectiveness of TAG and the superiority of TAGR.

2 Related Work

LLM Reasoning. Extensive research has explored the reasoning methods of LLMs. Post-training is a targeted approach to enhance LLMs reasoning ability (Tie et al., 2025). Reinforcement learning is currently a popular research topic, encouraging LLMs to engage in deeper thinking during reasoning (Schulman et al., 2017; Rafailov et al., 2023; Shao et al., 2024). Notably, recent advances have focused on utilizing process-based rewards to guide models through multi-step logic, ensuring that each intermediate step aligns with factual constraints (Lightman et al., 2023). In addition to training LLMs, Chain-of-Thought indicates the importance of prompts in LLMs reasoning (Wei et al., 2022). Furthermore, Tree-of-Thought and Graph-of-Thought explicitly transform the reasoning process of LLMs into tree and graph structures, enabling the model to navigate complex search spaces through backtracking and look-ahead strategies (Yao et al., 2023; Besta et al., 2024). Self-consistency guides LLMs to generate multiple reasoning trajectories and select the optimal answer among them (Wang et al., 2022). Moreover, some approaches leverage external feedback loops to iteratively refine these trajectories, further bridging the gap between neural generation and symbolic execution.

KG-enhanced LLM Reasoning. Although LLMs can reason independently, they still suffer from knowledge gaps and hallucinations. KG-enhanced LLM reasoning is a way to alleviate the above problems. KD-CoT provides additional information for CoT in LLMs reasoning by retrieving factual knowledge from KG (Wang et al., 2023). RoG constructs the entire process as a pipeline of planning-retrieval-reasoning, retrieves reasoning paths from KG to guide LLMs reasoning (Luo et al., 2024a). GFM-RAG and GNN-RAG achieve effective retrieval of KG through graph neural networks (Luo et al., 2025b; Mavromatis and Karypis, 2024). StructGPT and ToG consider LLMs as agents, guiding them to continuously interact with KG during LLMs reasoning process to obtain answers (Jiang et al., 2023; Sun et al.). EoG introduces reinforcement learning into the process of exploring KG in LLMs, achieving autonomous exploration in LLMs inference process (Yan et al., 2026).

Reasoning from Experiences. So far, there

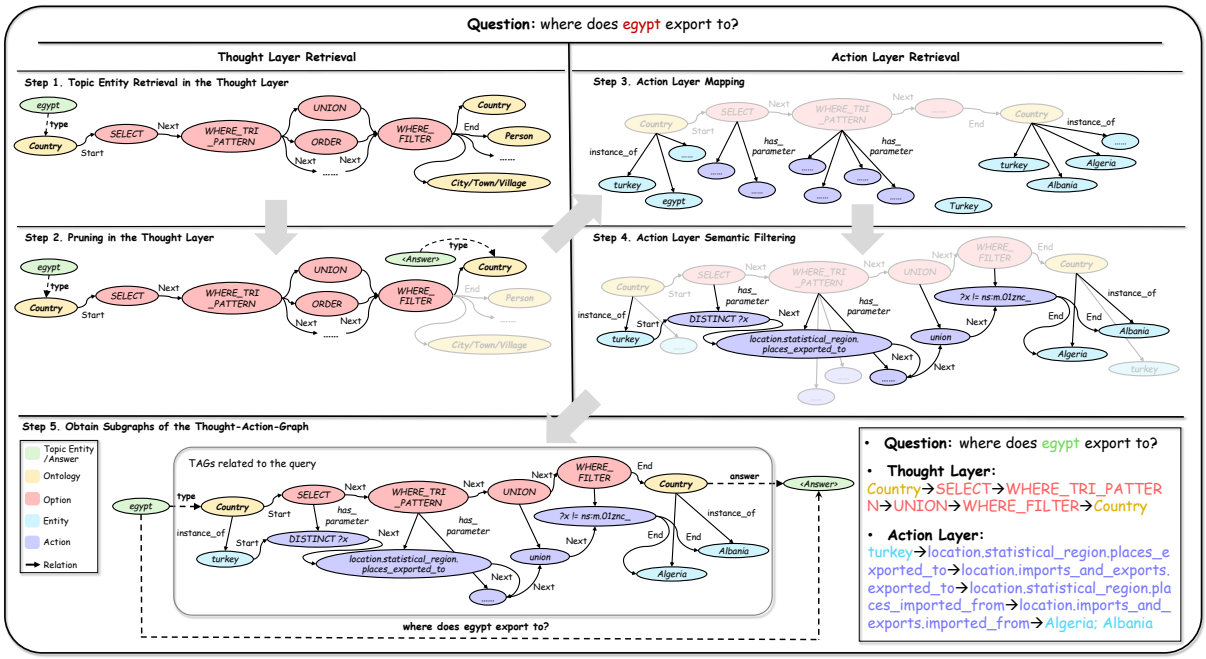

Figure 3: The TAG Retrieval Strategy. **Step 1-2** retrieves the abstract operational blueprint of the query from the thought layer. **Step 3-4** retrieves the specific parameters needed to implement abstract operational blueprint from the action layer. **Step 5** composes the thought layer and the action layer to obtain a complete thought-action subgraph.

have been attempts to abstract problem-solving processes from past experience and use them to guide future reasoning. AWM generates workflows by extracting reusable processes from the agent's past reasoning, and then integrates these workflows into a knowledge base to guide future task solving processes (Wang et al., 2024).

3 Preliminaries

Knowledge Graph. A KG $\mathcal{G}$ is composed of a large number of triples, which are formalized as $\mathcal{G} = \{(e, r, e') \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}\}$. $\mathcal{E}$ and $\mathcal{R}$ represent sets of entities and relationships, where extracted from texts or knowledge bases. The triple (e, r, e') is the most basic unit of KGs, representing the relationship between entity e and e' with r. Multiple connected triples in $\mathcal{G}$ form a path, multiple paths form a subgraph, and multiple subgraphs form a complete KG.

Knowledge Graph Question Answering. KGQA aims to retrieve precise answers to natural language questions by querying KGs. KGQA has gradually shifted from traditional pipeline to LLM-based question answering paradigm. Given a question Q, the naive method retrieves relevant subgraph $\mathcal{G}_s$ from $\mathcal{G}$. The LLM then interprets Q and $\mathcal{G}_s$ to produce a structured query (e.g., SPARQL) or directly generates answers $\mathcal{A}$. The LLM effectively acts as a semantic parser and reasoner, bridging the gap between natural language and structured knowledge.

4 Approach

In this section, we introduce the proposed TAG, including its construction, retrieval strategy, and TAG-based reasoning.

4.1 TAG Construction

Schema. TAG requires a specialized structure to explicitly reveal the reasoning process of LLMs on KGs. We design the TAG, denoted as $G_{T,A}$ with a two-layer architecture: a **thought layer** and an **action layer**, as shown in Figure 2(c).

The thought layer serves as the upper layer, storing basic operation patterns for querying KGs, such as `SELECT`, `WHERE`, `FILTER` and `UNION`. This layer contains two types of nodes: *Ontology* and *Option*. Ontology is a formal definition of various concepts in the real world, consistent with KGs. An option node is a fundamental action unit for querying KG (e.g., `SELECT`, `WHERE`, `UNION`). A complete path in the thought layer should start from an ontology node, traverse multiple option nodes, and end at another ontology node. This path represents how to query a target ontology from a source ontology through a sequence of operations.

Algorithm 1: The TAG retrieval strategy

Input: User query Q; Start ontology node $t_{\text{on}}^{\text{start}}$; Pre-constructed TAG $G_{T,A}$; Embedding model $\mathcal{E}$; LLM π
Output: Thought-Action Subgraph $\mathcal{G}$
1. Locating;
for $t_{on}^{i} \in \{all\ ontology\ nodes\}$ **do**
  Find the path $T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{i}, t_{\text{op}}^{j})$;
2. Pruning;
$t_{\text{on}}^{\text{end}} \leftarrow \pi\left(t \mid t_{\text{on}}^{\text{start}}, G_{T,A}\right)$;
for $t_{on} \in T_p(t_{on}^{start}, t_{on}^{i}, t_{op}^{j})$ **do**
  Find the pruned path $T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{\text{end}}, t_{\text{op}}^{k})$;
3. Mapping;
$a_{\text{en}}^{i}, a_{\text{ac}}^{j} \leftarrow f\left(T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{\text{end}}, t_{\text{op}}^{k})\right)$;
4. Filtering;
$a_{\text{en}}^{m}, a_{\text{ac}}^{n} \leftarrow \text{Top-}k\left(\mathcal{E}(Q, a_{\text{en}}^{i}, a_{\text{ac}}^{j})\right)$;
$A_p(a_{\text{en}}^{m}, a_{\text{ac}}^{n}) \leftarrow a_{\text{en}}^{m} \oplus a_{\text{ac}}^{n}$;
5. Composition;
$\mathcal{G} \leftarrow T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{\text{end}}, t_{\text{op}}^{k}) \oplus A_p(a_{\text{en}}^{m}, a_{\text{ac}}^{n})$;
return $\mathcal{G}$;

The action layer, as the lower layer, stores entity nodes corresponding to ontology nodes in the upper layer, along with the action parameters required for these operations. This layer also contains two types of nodes: *Entity* and *Action*. The concepts in entity and KGs are the same, representing specific things in the real world. An entity node is linked to ontology nodes in the thought layer via the "`instance_of`" relationship. An action node corresponds to an option node in the thought layer and specifies the concrete parameters for that option node during querying KGs. Option nodes and action nodes are linked through the "`has_parameter`" relationship. A complete path in the action layer should start from an entity node, traverse multiple action nodes, and end at another entity node. Furthermore, all entity nodes and action nodes must correspond to ontology nodes and option nodes in the thought layer, respectively.

Construction. Although constructing TAG from scratch is challenging, SPARQL from KGs provide comprehensive information for querying KGs. Therefore, we build TAG based on SPARQL. First, we employ GPT-4o-mini to extract MACs from given SPARQL queries, as illustrated in Figure 2(a) and (b) (OpenAI et al., 2024). The MAC can be viewed as a sequence starting from a query, which decomposes SPARQL operators into multi-step action sequences and executes them step by step until the answer is found. We perform post-processing on MACs to obtain nodes and edges that conform to the TAG schema, and then store them in the TAG. The overall construction process is illustrated in Figure 2, with detailed steps provided in Appendix A.

4.2 TAG Retrieval Strategy

After constructing $G_{T,A}$, we design a dedicated retrieval strategy for TAG. This strategy retrieves nodes and edges from TAG that correspond to the user's query on KGs, and assembles them into a complete thought-action subgraph. The retrieval process is detailed in Figure 3.

Thought Layer Retrieval. 1) Locating. We first locate the starting ontology node $t_{\text{on}}^{\text{start}}$ in the thought layer based on the ontology mentioned in the user query. Then, we retrieve all paths originating from this node in thought layer, denoted as $T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{i}, t_{\text{op}}^{j})$. 2) Pruning. We use LLMs to predict the target answer ontology $t_{\text{on}}^{\text{end}}$. Then, we prune the path set $T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{i}, t_{op}^{j})$ to retain only those paths that terminate at $t_{\text{on}}^{\text{end}}$, resulting in the pruned path set $T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{\text{end}}, t_{op}^{k})$.

Action Layer Retrieval. 3) Mapping. The pruned paths $T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{\text{end}}, t_{\text{op}}^{k})$ are mapped to the action layer to obtain their corresponding entity nodes a_{en}^{i} and action nodes a_{ac}^{j}. 4) Filtering. We employ embedding models to compute the semantic similarity between the current query and the source questions associated with the candidate nodes a_{en}^{i} and a_{ac}^{j}. We then select the top-k most similar source questions and retain their corresponding nodes, denoted as a_{en}^{m} and a_{ac}^{n}. Finally, we connect these filtered nodes to form the paths in the action layer $A_p(a_{\text{en}}^{m}, a_{\text{ac}}^{n})$.

Composition. We combine the pruned paths $T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{\text{end}}, t_{\text{op}}^{k})$ and the filtered paths $A_p(a_{\text{en}}^{m}, a_{\text{ac}}^{n})$ to form the final thought-action subgraph for implementing the query on KGs. In this subgraph, $T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{\text{end}}, t_{\text{op}}^{k})$ provides the abstract operational blueprint for querying, while $A_p(a_{\text{en}}^{m}, a_{\text{ac}}^{n})$ supplies the concrete parameters required to instantiate these operations.

The algorithmic formulation of the TAG retrieval strategy is presented in Algorithm 1.

4.3 TAG-based Reasoning

We propose a TAG-based reasoning mechanism, called TAGR. TAGR guides LLMs to first generate

Table 1: Performance and efficiency comparison of different baselines on three KGQA datasets. The block where * is located represents the LLM/Graph Reasoning method, and the block where † is located represents the KG-enhanced LLM Reasoning method. For a fair comparison, all reported results are based on models after supervised fine-tuning.

Methods	WebQSP		CWQ		GrailQA		Avg. # LLM Cost	
	Hits@1	F1	Hits@1	F1	Hits@1	F1	Calls	Tokens
*Qwen2-7B	50.8	35.5	25.3	21.6	-	-	-	-
Llama-3.1-8B	55.5	34.8	28.1	22.4	-	-	-	-
ReaRev	76.4	70.9	52.9	47.8	-	-	-	-
UniKGQA	77.2	72.2	51.2	49.1	-	-	-	-
†KD-CoT (Llama-2-7B)	68.6	52.5	55.7	-	-	-	-	-
EWEK-QA	71.3	-	52.5	-	60.4	-	-	-
ToG (GPT-4)	82.6	-	68.5	-	81.4	-	11.6	7,069
RoG (Llama-2-7B)	85.7	70.8	62.6	56.2	-	-	2	521
EffiQA (GPT-4)	82.9	-	69.5	-	78.4	-	7.3	-
GNN-RAG (Llama-2-7B)	85.7	71.3	66.8	59.4	-	-	1	414
KG-Agent (Llama-2-7B)	83.3	81.0	72.2	69.8	-	-	-	-
DoG (Qwen2.5-7B)	92.7	78.6	74.1	60.3	85.6	82.3	-	-
GCR (GPT-4o-mini)	92.2	74.1	75.8	61.7	-	-	2	231
KBQA-o1 (Qwen2.5-7B)	-	57.8	-	-	70.8	77.9	-	-
EoG_{SFT} (Llama-3.1-8B)	86.3	74.5	70.5	62.1	91.4	88.2	-	-
TAGR (GPT-4o-mini)	85.3	65.0	66.9	44.9	88.7	78.3	3	-
TAGR (Qwen2.5-7B)	87.3	60.7	67.2	39.5	**95.6**	84.0	3	**71**
TAGR (Llama-3.1-8B)	**93.2**	65.7	**77.4**	46.3	**96.1**	83.8	3	**65**

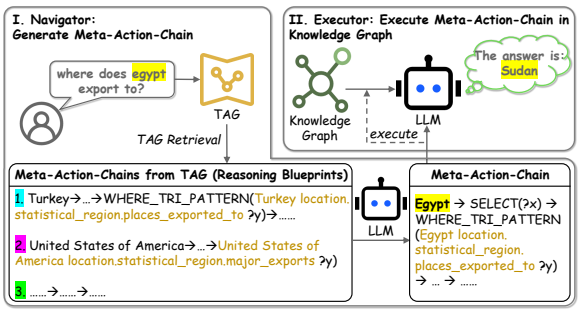

Figure 4: The process of TAGR.

MACs based on TAG and then execute them in KGs to obtain answers. The detailed process is illustrated in Figure 4.

The TAGR consists of two phases: I) Navigator: Given a user query Q, we first retrieve operationally and semantically similar paths $T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{\text{end}}, t_{\text{op}}^{k})$ and $A_p(a_{\text{en}}^{m}, a_{\text{ac}}^{n})$ from $G_{T,A}$. These paths are then composed into MACs $\mathcal{M}$. The query Q and $\mathcal{M}$ are then provided as prompts to the LLM π, which generates a query-specific MAC $\mathcal{M}^*$:

$$\mathcal{M} = T_p(t_{\text{on}}^{\text{start}}, t_{\text{on}}^{\text{end}}, t_{\text{op}}^{k}) \oplus A_p(a_{\text{en}}^{m}, a_{\text{ac}}^{n}) \quad (1)$$

$$\mathcal{M}^* \sim \pi(m \mid Q, \mathcal{M}) \quad (2)$$

$\mathcal{M}^*$ outlines the sequence of operations to performed on KGs to derive answers for Q. II) Executor: In this phase, guided by $\mathcal{M}^*$ and $\mathcal{K}^*$, the LLM executes $\mathcal{M}^*$ on $\mathcal{K}^*$ to reason final answers a^* for Q:

$$a^* \sim \pi(a \mid Q, \mathcal{M}^*, \mathcal{K}^*) \quad (3)$$

Specifically, $\mathcal{K}^*$ denotes the subgraph obtained by filtering the original KG. The detailed filtering process of $\mathcal{K}^*$ is provided in Section 4.4.

4.4 KG Filtering

KG Filtering is an important part of TAGR. In the second stage of TAGR, due to the large number of triples in the original KG, LLMs will encounter a lot of noise when executing MACs in KG. Based on the above issues, before the second stage of TAGR, we filter the original KG based on each step of MACs, removing a large number of triples. The filtered subgraph is input to TAGR. KG filtering is divided into three steps: 1) Disassemble MACs to obtain semantic information of actions in each step. 2) At each step, the embedding model[‡] is used to

[‡]To ensure computational efficiency, we use Fasttext as the embedding model (Joulin et al., 2017).

Algorithm 2: The specific process of KG filtering

Input: Original KG $\mathcal{K}$; MACs $\mathcal{M}$; Embedding model $\mathcal{E}$
Output: Filtered subgraph $\mathcal{K}^*$
$\mathcal{K}^* \leftarrow \emptyset$;
$actions \leftarrow \mathcal{M}$.split;
for $i \in actions$ **do**
  triples$_i \leftarrow$ top-k$(\mathcal{E}(i, \mathcal{K}))$;
  $\mathcal{K}^* \leftarrow \mathcal{K}^* \cup$ triples$_i$
return $\mathcal{K}^*$;

calculate the similarity between the action semantic information and each triple in KG, while retaining top-k triples. 3) Combine the reserved top-k triples into a subgraph as the KG required for the second stage of TAGR. Algorithm 2 demonstrates the specific process of KG filtering.

5 Experiments

5.1 Experimental Setup

Datasets. Following prior research, we evaluate our method on three datasets: WebQSP, CWQ, and GrailQA (Yih et al., 2016; Talmor and Berant, 2018; Gu et al., 2021). All datasets are based on the Freebase KG (Bollacker et al., 2008). We use training sets from these datasets to construct the TAG, treating them as past experience, and their test sets to evaluate the performance of TAGR. Specific details are provided in Appendix B.1. Due to the large scale of GrailQA, we adopt the same evaluation scale as ToG to save computational costs (Sun et al.).

Baselines. We compare TAGR with 15 baselines, which can be categorized into two groups: 1) LLM/Graph Reasoning: Methods that rely solely on LLMs or Graph Neural Networks (GNNs) to reason on KGs. 2) KG-enhanced LLM Reasoning: Methods that augment LLM's reasoning capabilities by retrieving relevant information from KGs to answer the query.

Evaluation Metrics. We adopt Hits@1 and F1 as our evaluation metrics. Hits@1 measures whether the top-1 generated answer matches any of the ground-truth answers. F1 comprehensively assesses the completeness of the generated answers by considering both precision and recall. Furthermore, since efficiency is a key consideration, we also evaluate reasoning efficiency using the average number of LLM calls and the average number of generated tokens.

Implementations. To evaluate the generalizability of TAGR, we employ one closed-source LLM and two open-source LLMs for the LLM component within TAGR: GPT-4o-mini, Llama-3.1-8B, and Qwen2.5-7B (Dubey et al., 2024; Team, 2024; Yang et al., 2024a). To enhance their generation and execution capabilities, we perform additional fine-tuning on two open-source LLMs using 6K samples.

5.2 Main Results

The experimental results for TAGR are presented in Table 1. The results show that TAGR outperforms state-of-the-art methods across all datasets, achieving relative improvements in Hits@1 of 0.5%, 1.6%, and 4.7%, respectively. Furthermore, compared to other methods, TAGR requires a significantly lower average number of generated tokens from LLMs. These significant improvements demonstrate the effectiveness and efficiency of TAGR. Appendix B provides more experimental details.

5.3 Ablation Study

We conduct ablation studies on three key components to understand their respective contributions:

1. **KG Filtering.** We filter the original KG based on each step of MACs to remove a large number of irrelevant triples. This ablation is designed to validate the effectiveness of our proposed TAG and MACs.

2. **Complete TAG.** As TAG is fundamental to TAGR, we cannot remove it entirely. Therefore, we replace it with an incomplete version to demonstrate the contribution of the complete TAG to the overall performance.

3. **Fine-tuning.** The reasoning task in TAGR is highly specialized, requiring fine-tuning LLMs to adapt. We conduct an experiment to investigate the impact of unfine-tuned LLMs on TAGR.

The results of our ablation studies are presented in Table 2. Notably, for all experiments, we adopt the best hyperparameters for each component. A detailed analysis of them is provided in Section 5.4.

5.4 Further Analysis

Impact of TAG Retrieval Breadth. A larger retrieval breadth for TAG results in the retrieval of

Table 2: Ablation studies of TAGR on WebQSP datasets.

Method	WebQSP		CWQ	
	Hits@1	F1	Hits@1	F1
TAGR	93.2	65.7	77.4	46.3
w/o				
KG Filtering	72.4	61.7	60.1	43.5
Full TAG	86.6	60.2	70.9	42.4
SFT	82.2	56.1	66.3	39.5

more MACs, which expands the execution space for LLMs on the KG. Figure 5 shows the performance on the three datasets as we vary the TAG's retrieval breadth.

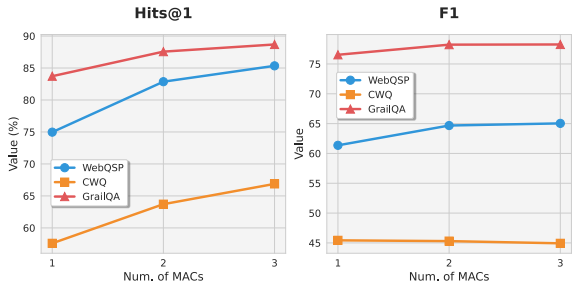

Figure 5: TAGR performance with varying numbers of MACs.

We represent the retrieval breadth of TAG by the number of MACs. A larger number increases the probability of TAGR generating correct answers, leading to a continuous rise in Hits@1. Conversely, a larger number also increases the likelihood of hallucination during execution on the KG, which can potentially cause the F1 to decrease.

Impact of KG Filtering. In the second stage of TAGR, when LLMs execute MACs on the original KG (containing >1K triples), performance degrades due to the abundance of irrelevant nodes. To mitigate this, we filter the original KG based on each step of MACs, retaining only the top-k most relevant triples to form a subgraph. We evaluate TAGR under the original subgraph size (>1K triples) and various reduced subgraph sizes (triples=200/100/50/25). For this experiment, the TAG is constructed from WebQSP training set. The results are shown in Figure 6.

Impact of TAG Scale. A larger TAG scale enables LLMs to generate more comprehensive MACs. We design a three-stage experiment where we progressively increase the TAG's scale to investigate whether a larger TAG leads to better performance. The results are presented in Figure 7.

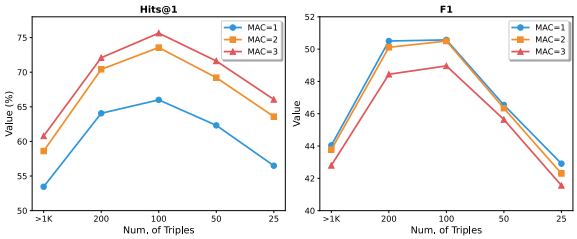

Figure 6: TAGR performance on WebQSP with KGs of varying scales.

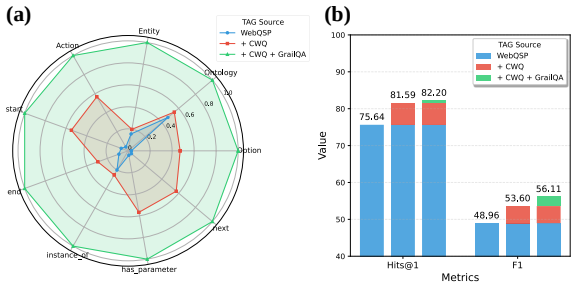

Figure 7: Performance of TAGR on WebQSP as the TAG scale increases. The blue bar denotes the first stage, where the TAG is constructed solely from WebQSP training set. The red bar represents the second stage, where CWQ training set is added. The green bar shows the third stage, which further incorporates GrailQA training set.

Figure 7a shows that the number of entities and relations within TAG continuously increases as training sets expand. The growth rate of ontologies is relatively small, as the number ontologies in Freebase is finite, limiting the head and tail of paths explored by TAGR. In contrast, the growth rate of actions is much larger, indicating that TAGR explores a wider action space. As the TAG scale increases, Figure 7b demonstrates that the performance of TAGR also improves.

5.5 Out-of-Distribution Performance

To evaluate TAGR's question-answering capability on unseen datasets, we conduct cross-dataset evaluations. Specifically, we individually construct TAGs solely from the training sets of each dataset (WebQSP, CWQ, and GrailQA) and evaluate their performance across all three test sets. The results are shown in Figure 8.

As shown in Figure 8, no significant "diagonal advantage" is observed for any metric, which demonstrates the strong generalization capability of TAGR. Moreover, the similar color patterns

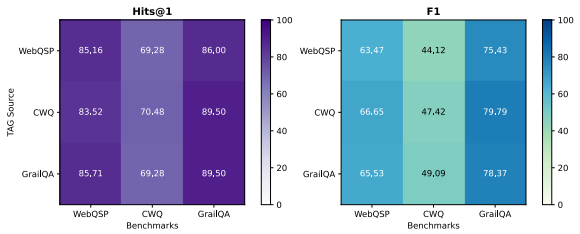

Figure 8: Cross-dataset evaluation of TAGR.

within each column of heatmaps further indicate that TAGR's performance is primarily correlated with the inherent difficulty of test sets, rather than specific datasets used for TAG construction.

6 Conclusion

In this work, we propose TAG, which abstracts fundamental operators from past LLM-KG interactions and models them as a graph structure. By introducing a new method for constructing, retrieving, and reasoning with TAGs, TAGR shifts the cost of LLMs' iterative exploration of KGs from traditional approaches to TAGs. This addresses the performance-efficiency trade-off inherent in traditional KG-enhanced LLM reasoning methods. Extensive experimental results demonstrate that TAGR not only achieves state-of-the-art performance but also significantly enhances KGQA efficiency. Furthermore, TAGR's performance is expected to improve as the TAG scale increases. All of the above demonstrates the effectiveness of both TAG and TAGR.

Limitations

One limitation of our approach is that the LLM's exploration on KGs relies on a single-round, serialized execution mechanism of MACs. This linear traversal strategy constrains the retrieval space, potentially preventing the approach from exploring all relevant reasoning paths and thus impacting the completeness of final answers. However, our experimental results indicate that this single-round exploration mechanism can efficiently and accurately locate the correct answer. This confirms the approach's effectiveness and reliability in specific scenarios, providing empirical support for its practical application value. Another limitation is the cold-start problem associated with TAG. The performance of our approach relies on the past experience accumulated within TAG, and significant cold-start challenges arise during system initialization or when dealing with entirely new domains. When past experience is sparse or unavailable, the system struggles to provide effective guidance, leading to a substantial drop in the LLM's exploration accuracy. Therefore, designing a cold-start strategy for TAG presents a promising direction for future research.

Acknowledgments

This work was supported by Ant Group Research Fund. The proposed method has been implemented and validated on Ant Group's business datasets.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Mohammad Dehghan, Mohammad Alomrani, Sunyam Bagga, David Alfonso-Hermelo, Khalil Bibi, Abbas Ghaddar, Yingxue Zhang, Xiaoguang Li, Jianye Hao, Qun Liu, and 1 others. 2024. Ewek-qa: Enhanced web and efficient knowledge graph retrieval for citation-based question answering systems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14169–14187.

Zixuan Dong, Baoyun Peng, Yufei Wang, Jia Fu, Xiaodong Wang, Xin Zhou, Yongxue Shan, Kangchen Zhu, and Weiguo Chen. 2025. Effiqa: Efficient question-answering with strategic multi-model collaboration on knowledge graphs. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 7180–7194.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond iid: three levels of generalization for question answering on knowledge bases. In *Proceedings of the web conference 2021*, pages 3477–3488.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, and 1 others. 2025. Retrieval-augmented generation with graphs (graphrag). *CoRR*.

Xuming Hu, Junzhe Chen, Xiaochuan Li, Yufei Guo, Lijie Wen, Philip S Yu, and Zhijiang Guo. 2023. Do large language models know about facts? *arXiv preprint arXiv:2310.05177*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*.

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.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. *arXiv preprint arXiv:2212.00959*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomáš Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th conference of the European chapter of the association for computational linguistics: volume 2, short papers*, pages 427–431.

Kun Li, Tianhua Zhang, Xixin Wu, Hongyin Luo, James Glass, and Helen 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.

Shiyang Li, Yifan Gao, Haoming Jiang, Qingyu Yin, Zheng Li, Xifeng Yan, Chao Zhang, and Bing Yin. 2023. Graph reasoning for question answering with triplet retrieval. In *ACL (Findings)*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The twelfth international conference on learning representations*.

Haoran Luo, Yikai Guo, Qika Lin, Xiaobao Wu, Xinyu Mu, Wenhao Liu, Meina Song, Yifan Zhu, Luu Anh Tuan, and 1 others. 2025a. Kbqa-o1: Agentic knowledge base question answering with monte carlo tree search. *arXiv preprint arXiv:2501.18922*.

L Luo, YF Li, G Haffari, and S Pan. 2024a. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *ICLR 2024: The Twelfth International Conference on Learning Representations*. ICLR.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Yuan-Fang Li, Chen Gong, and Shirui Pan. 2024b. Graph-constrained reasoning: Faithful reasoning on knowledge graphs with large language models. *arXiv preprint arXiv:2410.13080*.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Dinh Phung, Chen Gong, and Shirui Pan. 2025b. Gfm-rag: graph foundation model for retrieval augmented generation. *arXiv preprint arXiv:2502.01113*.

Costas Mavromatis and George Karypis. 2022. Rearev: Adaptive reasoning for question answering over knowledge graphs. *arXiv preprint arXiv:2210.13650*.

Costas Mavromatis and George Karypis. 2024. Gnn-rag: Graph neural retrieval for large language model reasoning. *arXiv preprint arXiv:2405.20139*.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, and 401 others. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

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.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *ACM Transactions on Information Systems*.

Shuofei Qiao, Yixin Ou, Ningyu Zhang, Xiang Chen, Yunzhi Yao, Shumin Deng, Chuanqi Tan, Fei Huang, and Huajun Chen. 2023. Reasoning with language model prompting: A survey. In *Proceedings of the 61st annual meeting of the Association for Computational Linguistics (volume 1: long papers)*, pages 5368–5393.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

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, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Xiangqing Shen, Fanfan Wang, and Rui Xia. 2025. Reason-align-respond: Aligning llm reasoning with knowledge graphs for kgqa. *arXiv preprint arXiv:2505.20971*.

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*.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Guiyao Tie, Zeli Zhao, Dingjie Song, Fuyang Wei, Rong Zhou, Yurou Dai, Wen Yin, Zhejian Yang, Jiangyue Yan, Yao Su, and 1 others. 2025. A survey on post-training of large language models. *arXiv e-prints*, pages arXiv–2503.

Keheng Wang, Feiyu Duan, Sirui Wang, Peiguang Li, Yunsen Xian, Chuantao Yin, Wenge Rong, and Zhang Xiong. 2023. Knowledge-driven cot: Exploring faithful reasoning in llms for knowledge-intensive question answering. *arXiv preprint arXiv:2308.13259*.

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

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

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.

Shiqi Yan, Yubo Chen, Ruiqi Zhou, Zhengxi Yao, Shuai Chen, Tianyi Zhang, Shijie Zhang, Wei Qiang Zhang, Yongfeng Huang, Haixin Duan, and 1 others. 2026. Explore-on-graph: Incentivizing autonomous exploration of large language models on knowledge graphs with path-refined reward modeling. *arXiv preprint arXiv:2602.21728*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Rui Yang, Haoran Liu, Edison Marrese-Taylor, Qingcheng Zeng, Yuhe Ke, Wanxin Li, Lechao Cheng, Qingyu Chen, James Caverlee, Yutaka Matsuo, and 1 others. 2024b. Kg-rank: Enhancing large language models for medical qa with knowledge graphs and ranking techniques. *CoRR*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

A TAG Construction

A.1 Steps

We use GPT-4o-mini to generate MACs from SPARQL, and then post-processing the MACs to decompose it into the basic elements that make up TAGs. Finally, we combine these elements into TAG according to predefined schemas. The entire process is illustrated in Algorithm 3.

Algorithm 3: Constructing TAG from SPARQL

Input: SPARQL query S; GPT-4o-mini π
Output: TAG $\mathcal{T}$
MACs $\leftarrow \pi(o \mid S)$;
$operators \leftarrow$ MACs.split;
for $o_i \in operators$ **do**
  Remove PREFIX;
  $\mathcal{T} \leftarrow \mathcal{T} \oplus o_i$
return $\mathcal{T}$;

A.2 Case

Table 3 shows a case of constructing a TAG. We first use GPT-4o-mini to extract MACs from SPARQL queries, and then use rule-based post-processing to extract basic elements of TAG from MACs. Finally, we combine these elements into TAG according to predefined schemas.

A.3 Results

We construct TAG from training sets of WebQSP, CWQ, and GrailQA, and remove samples whose answers are not entities. The results are shown in Table 4.

B Experiments

B.1 Datasets

WebQSP is a fundamental KGQA benchmark. It evaluates the model's ability to answer fact-based questions, which typically require retrieving facts from KGs. This dataset provides complete SPARQL queries for its questions, which will be executed on Freebase.

CWQ extends complexity beyond WebQSP by introducing questions that require combinatorial reasoning. These issues typically involve multiple constraints, conjunctions, and superlatives, requiring multi-hop reasoning paths on Freebase. This dataset is annotated using complex SPARQL queries to reflect these reasoning structures.

GrailQA is a large-scale high-quality dataset built on Freebase, aimed at evaluating the ability of KGQA models at three different levels: previously seen patterns (i.i.d.), new combinations of seen patterns (combinatorial), and completely new domains and relations (zero-shot).

The statistics of datasets are shown in Table 5. It is worth mentioning that due to the incompleteness of the source KG, the gold answers in some instances may not be present in the KG. Following previous research, we remove these instances from the evaluation (Li et al., 2025).

B.2 Baselines

We compare TAGR with multiple baselines:

1. **ReaRev** decomposes natural language problems into a set of instructions that serve as guiding sequences for KG traversal, and performs reasoning through graph neural network (GNN) simulation of breadth first search (BFS) strategy (Mavromatis and Karypis, 2022).

2. **UniKGQA** unifies the retrieval and reasoning stages within the same framework to better share semantic matching and information dissemination capabilities. It effectively alleviates the problem of large scale differences between two-stage search spaces and enhances capability sharing between stages through unified architecture and abstract subgraph design (Jiang et al., 2022).

3. **KD-CoT** explicitly integrates factual knowledge into the reasoning process of CoT to reduce the illusion caused by insufficient knowledge or erroneous memory in models (Wang et al., 2023).

4. **EWEK-QA** includes adaptive web search and KG fusion. Adaptive web search avoids semantic fragmentation caused by fixed length segmentation by dynamically adjusting text truncation points, thereby improving the integrity of search content; KG fusion enhances the system's performance in complex reasoning tasks by efficiently encoding structured knowledge (Dehghan et al., 2024).

5. **ToG** transforms LLM into an agent that interacts with KGs for collaborative reasoning. It is not just about transforming problems into queries, but also enabling the model to actively search on the KG to gradually explore relevant entities and relationships (Sun et al.).

6. **RoG** fine-tunes LLM to generate reliable inference plans. Then these plans are used to retrieve specific, factual reasoning path instances from KG. Finally, the actual retrieved path is used for reasoning of the final answer (Luo et al., 2024a).

7. **EffiQA** first utilizes LLM's common sense ability to explore potential reasoning paths. Subsequently, it offloads the semantic pruning task to a small plugin model to reduce computational overhead. Finally, the exploration results of KG are sent back to LLM for evaluation to optimize the decisions of the first two stages (Dong et al., 2025).

8. **GNN-RAG** is a method that combines GNNs with retrieval-augmented generation (RAG) frameworks, aimed at enhancing the reasoning ability of LLMs in KGQA tasks. This method utilizes GNN to process the complex

Table 3: A case of constructing TAG from SPARQL query.

> Step-1: Extract MACs from SPARQL queries through GPT-4o-mini.

Input_Question:
What country does turkey trade with?
Input_SPARQL:
SELECT DISTINCT ?x WHERE
{ FILTER (?x != ns:m.01znc_)
{ ns:m.01znc_ ns:location.statistical_region.places_exported_to ?y .
?y ns:location.imports_and_exports.exported_to ?x . }
UNION
{ ns:m.01znc_ ns:location.statistical_region.places_imported_from ?y .
?y ns:location.imports_and_exports.imported_from ?x . }}
Output_MACs:
Turkey
–>SELECT(DISTINCT ?x)
–>WHERE_TRI_PATTERN(Turkey location.statistical_region.places_exported_to ?y)
–>WHERE_TRI_PATTERN(?y location.imports_and_exports.exported_to ?x)
–>WHERE_TRI_PATTERN(Turkey location.statistical_region.places_imported_from ?y)
–>WHERE_TRI_PATTERN(?y location.imports_and_exports.imported_from ?x)
–>WHERE_FILTER(?x != ns:m.01znc_)
–>UNION
–>Algeria; Albania

> Step-2: Extract basic elements of TAG through rule-based post-processing

Output_Elements:
[Country, SELECT, WHERE_TRI_PATTERN, WHERE_FILTER, UNION]
[Turkey, DISTINCT ?x,
location.statistical_region.places_exported_to,
location.imports_and_exports.exported_to,
location.statistical_region.places_imported_from,
location.imports_and_exports.imported_from,
?x != ns:m.01znc_,
Algeria, Albania]

> Step-3: Combine elements into TAG according to predefined schemas

Output_Nodes_and_Edges:
- *Ontology_Nodes:* [Country]
- *Option_Nodes:* [SELECT, WHERE_TRI_PATTERN, WHERE_FILTER, UNION]
- *Entity_Nodes:* [Turkey, Algeria, Albania]
- *Action_Nodes:*

[DISTINCT ?x,
location.statistical_region.places_exported_to,
location.imports_and_exports.exported_to,
location.statistical_region.places_imported_from,
location.imports_and_exports.imported_from,
?x != ns:m.01znc_]

- *Edges:* Automatically complete according to the schema of TAG.

Table 4: The result of TAG construction. Number of SPARQL queries = 69,293.

Nodes/Edges	**Type and Quantity**				
Nodes	Ontology 4,608	Option 372	Entity 145,052	Action 373,098	- -
Edges	start 64,914	instance_of 960,092	next 312,489	has_parameter 373,113	end 350,341

Table 5: Statistics of datasets.

Dataset	#Train	#Test	#Ans=1	2≤#Ans≤4	5≤#Ans≤9	#Ans≥10
WebQSP	2,826	1,628	51.2%	27.4%	8.3%	12.1%
CWQ	27,639	3,531	70.6%	19.4%	6%	4%
GrailQA	44,337	13,231	68.5%	-	-	-

structural information of KG, while taking advantage of LLM's language understanding advantages to achieve efficient and accurate question answering (Mavromatis and Karypis, 2024).

9. **KG-Agent** first extracts reasoning chains from the KGQA dataset, generates a query path from the question entity to the answer entity through rule matching and BFS, and then converts reasoning chains into a program consisting of a series of function tool calls. Meanwhile, the framework utilizes inference programs to construct instruction datasets for fine-tuning lightweight LLMs (such as LLaMA2-7B) to enhance their reasoning capabilities. During the reasoning process, KG-Agent adopts a knowledge memory mechanism to maintain contextual information, including the problem, tool definition, current KG state, and historical reasoning steps. It interacts with the KG through the tool selection planner, gradually updating its memory until answers are obtained (Jiang et al., 2025).

10. **DoG** allows LLMs to perform step-by-step inference directly on KGs, rather than relying on external retrievers or complex iterative instructions. The key insight of the study is that LLMs excel at generating coherent reasoning steps, while KGs provide structured factual connections, which can be deeply fused through constraint decoding mechanisms to achieve faithful and sound reasoning processes (Li et al., 2025).

11. **GCR** integrates LLMs with KGs by building a KG-Trie index. KG-Trie encodes the reasoning path of KG and constrains the generation of faithful reasoning paths and answers. This method combines specific KGs with powerful general LLM to improve the reasoning performance of LLMs (Luo et al., 2024b).

12. **KBQA-o1** introduces an intelligent agent workflow based on the ReAct framework, gradually generating logical forms through knowledge base environment exploration. Additionally, by adopting the MCTS heuristic search method driven by policy and reward models, it balances agent exploration performance with search space. With the heuristic exploration mechanism, KBQA-o1 can generate high-quality annotated data and continuously optimize the model through incremental fine-tuning (Luo et al., 2025a).

13. **EoG** is a reinforcement learning framework that incentivizes LLMs to autonomously explore diverse reasoning paths on KGs via path-refined reward modeling. It enhances generalizability by reducing the model's reliance on fixed demonstrations (Yan et al., 2026).

C Prompts

The prompts used for TAGR consist of three components. The first is the prompt for predicting answer types using LLMs. The second prompt is located in the first step of TAGR, which generates the corresponding MAC for user questions based on MACs in the TAG. The third one is located in

the second step of TAGR, which is to execute MAC in KG to obtain answers. All prompts are shown in Figure 9-11.

D Case Study

Figure 12 shows two cases in TAGR. The first question is a common sense question that requires real-world knowledge to answer. For these questions, TAGR can output accurate and complete answers. The second question is a highly specialized one, which significantly challenges the LLM's reasoning capability on KG. Although the answer obtained by TAGR is incomplete, it still accurately arrives at the correct answer.

Please read the example first, and then predict the type name of the answer based on the given question. Please note that you only need to output the type name of the predicted answer, and do not need to output specific answers or other content.

[examples]
Question: what is the name of justin bieber brother?
The type name of the answer:
Person

Question: what are the 2 conferences in the nfl?
The type name of the answer: American football conference

Question: what are the 5 biggest cities in the usa?
The type name of the answer:
City/Town/Village

Please write the type name of the answer corresponding to the following question based on the above example:
<Question>

Figure 9: Prompts for predicting answer types.

The meta-action in SPARQL refer to single step operations on the database, such as SELECT, WHERE, GROUP, ORDER, FILTER, etc. Meta-action-chain refers to the process of obtaining an answer entity that can solve a question through multiple meta-action from the topic entity of the question. Specifically, the meta-action-chain needs to start from the topic entity, with intermediate nodes being meta-actions. Each meta-action contains a specific operation description in parentheses, such as attributes, triplet patterns, etc. The final meta-action-chain ends with the answer entity corresponding to the question. Please read examples of generating a meta-action-chain first, and then generate its corresponding meta-action-chain based on the given question and topic entity. Because you do not yet know the answer entity corresponding to the given question, you can use the special tag <Answer_Entity>to represent the final step of the meta-action-chain. Please note that you only need to write the meta-action-chain, no other content is required.

[examples]
Question:
<TAG Question>
Topic entity:
<TAG Topic entity>
Meta-action-chain:
<TAG Meta-action-chain>

Please write a meta-action-chain based on the given question and topic entity according to the above examples:
Question:
<User Question>
Topic entity:
<User Topic entity>
Meta-action-chain:
<User Meta-action-chain>

Figure 10: Prompts for generating MAC.

The meta-action in SPARQL refer to single step operations on the database, such as SELECT, WHERE, GROUP, ORDER, FILTER, etc. Meta-action-chain refers to the process of obtaining an answer entity that can solve a question through multiple meta-action from the topic entity of the question. Specifically, the meta-action-chain needs to start from the topic entity, with intermediate nodes being meta-actions. Each meta-action contains a specific operation description in parentheses, such as attributes, triplet patterns, etc. The final meta-action-chain ends with the answer entity corresponding to the question. Please refer to the provided knowledge graph and search for the corresponding answer to the given question in the knowledge graph based on the given question, the topic entity in the question, and the meta-action-chain of the question. Please note that you only need to:
1. Output the corresponding answer to the given question, without any additional content.
2. If there are multiple answers, please output all answers in a [List], separated by commas. Each element in the [List] is an answer. For example: [Lightning rod, Bifocals, Glass harmonic, Franklin stone].
3. Please write your answer between <ANSWER> and </ANSWER>, do not output any other text.

Knowledge Graph:
<Knowledge Graph>
Question:
<Question>
Topic Entity:
<Topic Entity>
Meta-action-chain:
<Meta-action-chain>

The corresponding answer to the question is:

Figure 11: Prompts for executing MAC in KG.

Common sense question:

Question: What did james k polk do before he was president?

MAC:
James K. Polk
→SELECT(DISTINCT?x)
→ WHERE_TRI_PATTERN(James K. Polk government.politician.government_positions_held?y)
→ WHERE_TRI_PATTERN(?y government.government_position_held.office_position_or_title?x)
→ WHERE_TRI_PATTERN(?y government.government_position_held.from?from)
→ WHERE_FILTER(xsd:dateTime(?pFrom) - xsd:dateTime(?from) > 0)
→ <Answer_Entity>

TAGR Answers: [United States Representative, Speaker of the United States House of Representatives, Governor of Tennessee]

Ground Truth Answers: [United States Representative, Governor of Tennessee, Speaker of the United States House of Representatives]

Highly specialized question:

Question: Where can i find the genomic locus in chromosome 5?

MAC:
Chromosome 5
→SELECT(?x0 AS?value)
→WHERE_TRI_PATTERN(?x0 type.object.type biology.genomic_locus)
→WHERE_TRI_PATTERN(?y0 type.object.type biology.genomic_locus)
→WHERE_TRI_PATTERN(?y0 biology.genomic_locus.end_base?y1)
→WHERE_TRI_PATTERN(?y0 biology.genomic_locus.chromosome?y2)
→WHERE_FILTER(?y0!=?y1 &&?y0!=?y2 &&?y1!=?y2)
→VALUES(?y2 { :m.099p5k })
→WHERE_TRI_PATTERN(?x0 biology.genomic_locus.end_base?x1)
→WHERE_TRI_PATTERN(?x0 biology.genomic_locus.chromosome?x2)
→WHERE_FILTER(?x0!=?x1 &&?x0!=?x2 &&?x1!=?x2)
→<Answer_Entity>

TAGR Answers: [5 - [10732405,10814338]]

Ground Truth Answers: [5 - [10732405,10814338], 5 - [178973785,178983275], Locus for Human Cytogenetic Band 5p15.32, Locus for Human Cytogenetic Band 5q12.1, …]

Figure 12: Two cases in TAGR.