Beyond Local Evidence: LLM-Guided Knowledge Graph Reasoning for Grounded Question Answering

Anonymous ACL submission

Abstract

Large language models (LLMs) are increasingly combined with knowledge graphs (KGs) for grounded question answering. Existing LLM-KG methods mainly follow two paradigms: KG-RAG based generation uses KG evidence as auxiliary context, while LLM-guided KG reasoning constrains reasoning to traceable symbolic KG paths. However, current LLM-guided KG reasoning methods typically search within local neighborhoods of given topic entities, and may miss crucial evidence involving semantically relevant but non-adjacent entities, especially in incomplete KGs. To address this limitation, we propose LM-KGQA, an LLM-guided Monte Carlo Tree Search (MCTS) framework for expanding KG evidence search beyond local topic-entity neighborhoods. We further introduce self-rewarded node evaluation and aggregate high-reward traces for evidence-grounded answer derivation. Experiments on three KGQA benchmarks show that LM-KGQA outperforms nine SOTA baselines across three categories, with nearly 30% average relative improvement on CWQ across backbone LLMs. Further analyses validate the self-reward mechanism and show robust performance across test-time reasoning budgets.

1 Introduction

Large language models (LLMs) have demonstrated strong language understanding and reasoning abilities across a wide range of natural language processing tasks. However, for complex knowledge-intensive questions, LLMs still suffer from hallucinations, outdated knowledge, and opaque decision-making (Peng et al., 2023). In contrast, knowledge graphs (KGs) organize entities and relations in an explicit and editable symbolic structure, providing faithful and traceable factual knowledge for complex reasoning (Luo et al., 2024). Combining LLMs with KGs has therefore become a promising direction for grounded question answering (Pan et al., 2024).

Existing LLM-KG methods mainly follow two paradigms. The first is **KG-RAG based LLM generation**, which retrieves sub-KG structures as external context for answer generation (Jiang et al., 2023; Lin et al., 2025; Wan et al., 2025; Cui et al., 2026; Gao et al., 2025). In this paradigm, KG evidence mainly serves as auxiliary context, while the reasoning process is still carried out by the LLM over the provided prompt. Therefore, reasoning performance largely depends on the relevance and completeness of the constructed KG context, and missing or noisy evidence can mislead the LLM. The second paradigm is **LLM-guided KG reasoning**, where LLMs interactively explore entities and relations on KGs and derive answers from the explored evidence (Sun et al., 2024; Chen et al., 2024). Unlike KG-RAG based generation, this paradigm treats the KG not merely as an auxiliary context source, but as a symbolic reasoning space that constrains the reasoning process. The LLM guides which entities or relations to explore, while each reasoning step is grounded in valid KG evidence. This makes the reasoning trajectory more faithful and traceable, reducing unsupported hallucination.

However, existing LLM-guided KG reasoning methods still suffer from limited exploration scope. They typically initialize reasoning from given topic entities and expand evidence within their connected local KG neighborhoods. Such local traversal works when key evidence lies near the starting entities, but may fail when crucial evidence involves semantically relevant yet non-adjacent entities. This issue is especially common in real-world KGs, which are inevitably incomplete and may lack edges between question-related entities. Therefore, a more comprehensive LLM-guided KG reasoning framework is needed to discover useful non-local entities beyond initial topic-entity neighborhoods.

To address this gap, we build upon the LLM-

guided KG reasoning paradigm and propose **LM-KGQA**, an LLM-guided MCTS-based framework for KGQA that enables non-local KG evidence exploration. Rather than restricting exploration to local neighborhoods of initial topic entities, LM-KGQA formulates grounded question answering as reasoning-space search over KG-grounded evidence states. Each search-tree node represents a partial evidence state, and each root-to-terminal path forms a complete reasoning trace for answer derivation. During node expansion, the LLM proposes complementary candidate entities based on the accumulated evidence and the semantic requirements of the question. These entities are then grounded back to the KG to extract reasoning paths, enabling LM-KGQA to discover useful non-local evidence inaccessible to previous local exploration methods. We further design an LLM-judged self-reward mechanism to estimate node rewards from evidence usefulness, answer groundedness, and answer correctness. After multiple MCTS rollouts, LM-KGQA aggregates high-reward reasoning traces to derive the final answer. Finally, extensive experiments on three KGQA benchmarks validate the effectiveness of our method. The main contributions of this work are summarized as follows:

- We propose **LM-KGQA**, an LLM-guided MCTS framework for expanding KG evidence search beyond local topic-entity neighborhoods.
- We introduce a self-reward mechanism to evaluate evidence usefulness, answer groundedness, and answer correctness for MCTS node estimation.
- Experiments on three KGQA benchmarks demonstrate the effectiveness of LM-KGQA, with further analyses on self-rewarding and robustness across reasoning budgets.

2 Related Work

2.1 LLM Reasoning

Large language models (LLMs) have shown remarkable success across many natural language tasks due to their powerful pre-training on massive text corpora. Techniques such as chain-of-thought prompting (Wei et al., 2022) and self-consistency sampling (Wang et al., 2022) further improve reasoning by generating intermediate reasoning steps. Despite these advances, LLMs still suffer from hallucinations, outdated knowledge, and opaque decision-making when tackling complex reasoning and knowledge-intensive questions (Peng et al., 2023). These limitations arise because LLMs rely solely on their parametric memory, without a grounded mechanism to retrieve and verify external facts. To mitigate these issues, a line of work integrates external sources via RAG to improve factual grounding and reasoning robustness of LLM (Gao et al., 2023; Mansurova et al., 2024; Sharma et al., 2025).

2.2 LLM-KG Reasoning

Knowledge graphs (KGs) provide explicit entities and relations for grounding LLM reasoning. Existing LLM-KG methods mainly follow two lines. The first retrieves KG triples, subgraphs, or paths as auxiliary context for LLM generation (Jiang et al., 2023; Lin et al., 2025). Although this retrieve-then-reason paradigm improves factual grounding, its performance depends on the coverage and quality of the retrieved evidence. The second line uses LLMs to guide KG exploration by selecting entities, relations, or paths during reasoning (Sun et al., 2024; Chen et al., 2024). These methods make reasoning more traceable by grounding each step in KG evidence, but they are often restricted to local neighborhoods of given topic entities and may miss useful non-local evidence. In contrast, our work formulates KGQA as reasoning-space search over KG-grounded evidence states and introduces an LLM-guided MCTS framework to expand evidence search beyond local topic-entity neighborhoods.

3 Our Method

3.1 Preliminaries

3.1.1 Problem Formulation

Given a question q, an optional topic entity set E_0, and a knowledge graph G, our goal is to derive an answer a grounded in factual evidence from G. We formulate KGQA as a reasoning evidence search problem over G. At step t, the system maintains a reasoning state $s_t = (q, E_t, P_t)$, where $E_t = \{e_i\}_{i=1}^{m}$ and $P_t = \{p_i\}_{i=1}^{m}$ denote the accumulated entities and reasoning paths, respectively. The initial state is constructed from q, E_0, and the initial KG paths P_0 retrieved for E_0. The system then iteratively expands the state by identifying candidate entities and retrieving their associated KG paths. Importantly, newly introduced entities are

not required to be directly connected to the original topic entities, as long as they provide useful evidence for answering the question. The objective is to find an evidence set $\mathcal{Z}^* = \{p_i\}_{i=1}^k$ from which the final answer a can be derived. To search the large combinatorial space of evidence trajectories, we cast this process as a Monte Carlo Tree Search (MCTS) problem (Browne et al., 2012).

3.1.2 Reasoning Node Representation

To support the MCTS-based reasoning evidence search defined above, we represent each node in the search tree as a reasoning state associated with the question. Specifically, each node maintains three components: an accumulated entity set, an accumulated reasoning path set, and a scalar reward. Formally, the node content is written as (E_t, P_t, r_t), where E_t records the entities encountered along the current reasoning trajectory, P_t stores the reasoning paths collected from the knowledge graph for E_t, and r_t denotes the utility of the node for the overall question answering process.

Different node types correspond to different stages of the reasoning process. The root node is initialized from the question q, the optional topic entity set E_0, and the corresponding initial reasoning paths P_0. A non-terminal child node extends its parent state with newly introduced reasoning evidence, while a terminal node represents a completed state from which a final answer can be derived. The reward r_t is used to evaluate the quality of each node during tree search, and its detailed definition is given in the following method sections.

3.2 Reasoning MCTS Framework

The whole pipeline of our framework is shown in figure 1. We'll introduce our framework in details consisting of selection, expansion, self-rewarding based simulation and backpropogation.

3.2.1 Selection

Starting from the root node, we recursively select a child node according to the Upper Confidence Bound applied to Trees (UCT) criterion (Kocsis and Szepesvári, 2006), which balances exploitation of high-reward nodes and exploration of less-visited nodes. The UCT score of a child node v is defined as:

$$UCT(v) = R(v) + c\sqrt{\frac{\ln N(p(v))}{N(v)}}, \qquad (1)$$

where $R(v)$ denotes the estimated reward of node v, $N(v)$ is the visit count of v, $p(v)$ is the parent node of v, and c is a hyperparameter controlling the exploration–exploitation trade-off. The selection process continues until a leaf node or a terminal node is reached.

3.2.2 Expansion

Given the accumulated entity set E_{t-1} and reasoning path set P_{t-1} along the current reasoning trace, the expansion step constructs the next node. Depending on the current state, the next node can be either a terminal node or a non-terminal child node. In our framework, a terminal node can be generated under two cases. First, if the next expanded node reaches the predefined maximum search depth $D_{\max}$, the expansion process is forced to terminate. Second, given the current reasoning trace, the LLM may determine that the answer can already be derived from E_{t-1} and P_{t-1}. In this case, the model directly outputs a response starting with "Now we can answer the question:", and the corresponding child node is treated as a terminal node. All remaining cases are regarded as non-terminal expansion.

For a non-terminal expansion step, the LLM is first prompted to generate one candidate topic entity name that may provide useful complementary evidence for answering the question. Formally, given the current reasoning trace $\mathcal{T}_{t-1} = (q, E_{t-1}, P_{t-1})$, we construct the entity-generation prompt as $\Pi_{t-1}^{\text{ent}} = \Phi_{\text{ent}}(q, E_{t-1}, P_{t-1})$, where Φ_{ent} denotes the prompt template for candidate topic entity generation. Based on this prompt, the LLM generates a candidate topic entity name $\hat{e}_t$.

Since $\hat{e}_t$ is a textual entity name rather than a grounded KG node, we link it to the knowledge graph in two steps. We first retrieve a candidate KG entity set by lexical matching, and then refine these candidates using embedding-based semantic similarity between the recalled entities and the query formed by the question and $\hat{e}_t$. The top-w linked entities are retained for subsequent child expansion:

$$\begin{aligned} \tilde{E}_t^{(1)} &= \text{Retrieve}(\hat{e}_t, G), \\ \tilde{E}_t^{(2)} &= \{\tilde{e}_{t,1}, \tilde{e}_{t,2}, \ldots, \tilde{e}_{t,w}\}, \end{aligned} \qquad (2)$$

where w denotes the expansion width, i.e., the number of child nodes expanded at the current step.

For each linked KG entity $\tilde{e}_{t,j}$, we further extract a corresponding reasoning path $\hat{p}_{t,j}$ from G as grounded evidence followed the 2-step path exploration process introduced in paper (Chen et al.,

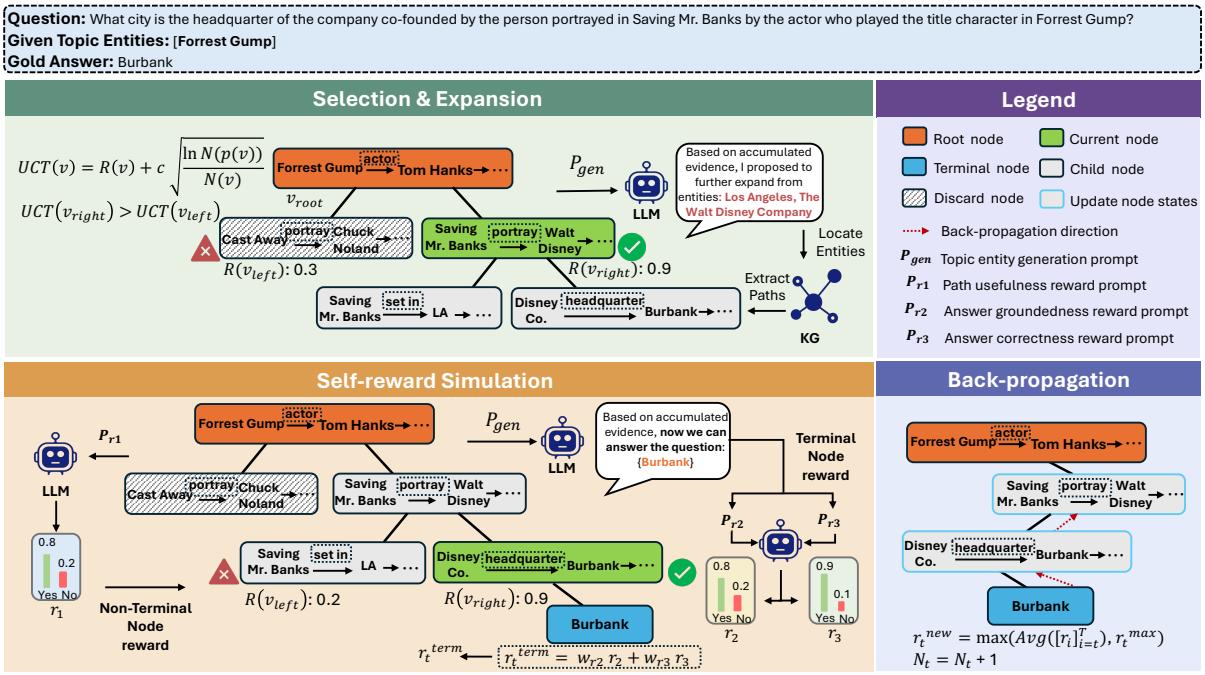

Figure 1: Overall framework of our proposed MCTS-based reasoning method for grounded knowledge graph question answering.

2024). Based on the linked entity and its associated reasoning path, we construct a non-terminal child node as:

$$v_{t,j}^{\text{non-term}} = \left(E_{t-1} \cup \{\tilde{e}_{t,j}\},\ P_{t-1} \cup \{\hat{p}_{t,j}\},\ r_{t,j}\right), \tag{3}$$

where $r_{t,j}$ denotes the estimated reward of the child node.

When the next expanded node is terminal, we no longer generate additional candidate entities or extract new reasoning paths. Instead, we directly prompt the LLM to generate a candidate final answer $\hat{a}_t$ based on the current reasoning trace. The corresponding terminal node is then constructed as:

$$v_t^{\text{term}} = \left(E_{t-1},\ P_{t-1},\ \hat{a}_t,\ r_t^{\text{term}}\right), \tag{4}$$

where r_t^{term} is the terminal-node reward, which evaluates the generated answer from both correctness and groundedness perspectives.

Details of reward estimation for non-terminal and terminal nodes are introduced in Section 3.2.3. The prompts used in expansion can be found in appendix A.

3.2.3 Self-Reward based Simulation

In conventional MCTS, the simulation step estimates the value of the current node by performing future rollouts from that node and using the rollout outcomes as reward signals. In our grounded KGQA setting, however, such random simulation is not well suited to structured evidence search over the knowledge graph. Therefore, we replace the standard simulation step with a self-reward mechanism based on LLM judgment, which directly evaluates the quality of each expanded node.

Specifically, for each node, we construct a reward prompt and use the probability that the LLM generates "Yes" as the first token as the reward score. Formally, for a reward prompt Π^{rew}, the corresponding reward is defined as

$$r = P(\text{Yes} \mid \Pi^{\text{rew}}). \tag{5}$$

We design different reward functions for non-terminal and terminal nodes.

Non-terminal node reward. For a non-terminal child node $v_{t,j}^{\text{non-term}}$, the reward evaluates whether the newly introduced reasoning path $\hat{p}_{t,j}$ provides useful new evidence for answering the question, conditioned on the accumulated evidence history (E_{t-1}, P_{t-1}). Intuitively, a newly added path should receive a higher reward if it introduces information that moves the reasoning closer to the final answer, and a lower reward if it introduces irrelevant information or information that does not contribute to answering the question. Formally, we define the non-terminal reward prompt as

$$\Pi_{t,j}^{\text{path}} = \Phi_{\text{path}}(q, E_{t-1}, P_{t-1}, \hat{p}_{t,j}), \tag{6}$$

and compute the reward of the non-terminal child node as follows, where r_0 is a node value prior that helps enhance stability.

$$r_{t,j}^{path} = P\Big(\text{Yes} \mid \Pi_{t,j}^{\text{path}}\Big), \\ r_{t,j} = w_{r1}\, r_{t,j}^{\text{path}} + (1 - w_{r1})\, r_0 \tag{7}$$

Terminal node reward. For a terminal node v_t^{term}, the reward should evaluate the generated final answer $\hat{a}_t$ from two complementary perspectives: **answer groundedness** and **answer correctness**. Answer groundedness measures whether the final answer is sufficiently supported by the accumulated reasoning evidence, while answer correctness measures whether the final answer correctly answers the question.

We first define the groundedness reward prompt as

$$\Pi_t^{\text{ground}} = \Phi_{\text{ground}}(q, P_{t-1}, \hat{a}_t), \tag{8}$$

and compute the groundedness reward as

$$r_t^{\text{ground}} = P\Big(\text{Yes} \mid \Pi_t^{\text{ground}}\Big). \tag{9}$$

Similarly, we define the correctness reward prompt as

$$\Pi_t^{\text{corr}} = \Phi_{\text{corr}}(q, \hat{a}_t), \tag{10}$$

and compute the correctness reward as

$$r_t^{\text{corr}} = P(\text{Yes} \mid \Pi_t^{\text{corr}}). \tag{11}$$

The final terminal-node reward is then defined as the weighted sum of the two components:

$$r_t^{\text{term}} = w_{r2}\, r_t^{\text{ground}} + w_{r3}\, r_t^{\text{corr}}. \tag{12}$$

where w_{r2} and w_{r3} are hyperparameters controlling the relative importance of answer groundedness and answer correctness, respectively.

In this way, our self-reward mechanism serves as a task-specific replacement for the simulation step in conventional MCTS: instead of estimating node values through random future rollouts, we directly evaluate each node based on its contribution to grounded question answering. The prompts used in reward computation can be found in Appendix A.

3.2.4 Backpropagation

When a terminal node v_t^{term} is reached as shown in figure 1, we obtain a complete reasoning trajectory from the root node to the terminal node which delivers the final answer. We then start to back-propogate the rewards and update each node's reward and visit count along the current reasoning trace. Prior MCTS-based planning framework (Hao et al., 2023) updates each node's reward by aggregating the rewards of all its subsequent nodes on the selected trace, which is a way to estimate the long-term value of the partial reasoning state represented by each non-terminal node. Based on this, we adopt a max-backup strategy for node-level reward update. We further maintain the best reward value observed for each node v_t from previous rollouts and we update each node's reward by selecting the maximum value between the best maintained reward $R_{max}(v_t)$ and the current estimated long-term value $\bar{R}(v_t)$. Our max-backup design for node-level reward update prevents a former promising reasoning node from being downgraded by a later low-quality rollout, which is important for noisy LLM-guided KG exploration.

Formally, given a selected trajectory $\tau = (v_0, v_1, \ldots, v_T)$, for each node v_t, we update the node reward with the max-backup rule as:

$$\bar{R}(v_t) = \text{Avg}\left(\sum_{i=t}^{T} R(v_i)\right), \\ R(v_t) \leftarrow \max\big(R_{\max}(v_t), \bar{R}(v_t)\big). \tag{13}$$

Meanwhile, the visit count of each node is updated as

$$N(v_t) \leftarrow N(v_t) + 1. \tag{14}$$

This update encourages the search to retain high-potential partial evidence states while still allowing newly discovered high-quality trajectories to improve their value estimates.

3.3 Final Reasoning Trace Selection

After completing all MCTS rollouts, we collect the terminal nodes reached during the search, where each terminal node corresponds to a complete reasoning trace and a candidate final answer. We calculate the reward value for each trace by averaging the rewards of all nodes along the trace. We adopt a trace-vote strategy to improve robustness (Zhang et al., 2025). Specifically, we first select the top-k reasoning traces according to their rewards. We

then extract the candidate final answers from these traces and apply majority voting over the answers to determine the final answer. If the same final answer is produced by multiple reasoning traces, we select the trace with the highest reward among them as the final supporting reasoning trace.

4 Experiments

4.1 Experimental Setups

4.1.1 Datasets & Evaluation Metrics

To evaluate the effectiveness of our LM-KGQA framework on complex reasoning over KGs, we conduct experiments on three widely used multi-hop KGQA benchmarks: CWQ (Talmor and Berant, 2018), WebQSP (Yih et al., 2016), and GrailQA (Gu et al., 2021) following their original licenses and terms of use, and use them solely for research and evaluation purposes consistent with their intended use. All three datasets are grounded in the Freebase KG (Bollacker et al., 2008). For the large-scale GrailQA dataset, we utilize the same testing samples as those in ToG (Sun et al., 2024). In our evaluation, we use 3,531 test samples from CWQ, 1,639 test samples from WebQSP, and 1,000 test samples from GrailQA. Consistent with prior work (Jiang et al., 2023; Sun et al., 2024; Chen et al., 2024), we adopt exact match accuracy (Hits@1) as the evaluation metric.

4.1.2 Comparison Methods

We select prior state-of-the-art approaches as baselines for each dataset, which can be categorized into three groups: (1) *KG-augmented LLM Reasoning Methods*: including ToG (Sun et al., 2024), PoG (Chen et al., 2024), StructGPT (Jiang et al., 2023) and ReKnos (Wang et al., 2025); (2) *LLM-only methods*: including CoT (Wei et al., 2022), IO prompt (Brown et al., 2020) and SC (Wang et al., 2022); (3) *Non-LLM KGQA methods*: including NSM (He et al., 2021) and ReaRev (Mavromatis and Karypis, 2022). For each LLM-involved method, we compare the performance of two backbone LLMs: DeepSeek v3.2 (Liu et al., 2025) and Qwen3.6-plus (Qwen Team, 2026). It is worth noting that NSM and ReaRev only release trained checkpoints for CWQ and WebQSP, and no official training or evaluation splits are available for GrailQA; therefore, we only report their results on CWQ and WebQSP.

Methods	CWQ	WebQSP	GrailQA
Non-LLM KGQA Methods			
NSM	43.78	71.45	-
ReaRev	47.18	75.72	-
Backbone LLM: DeepSeek v3.2			
ToG	50.49	72.21	65.56
PoG	59.93	84.62	77.68
StructGPT	43.67	74.08	66.10
ReKnoS	63.45	80.26	74.35
CoT	56.61	77.26	35.79
IO	51.70	77.80	39.77
SC	54.58	76.04	36.84
LM-KGQA	**67.73**	**85.60**	**80.98**
Backbone LLM: Qwen3.6-plus			
ToG	45.49	70.05	70.16
PoG	57.19	76.45	79.28
StructGPT	45.06	70.66	63.00
ReKnos	60.03	77.16	75.95
CoT	65.33	73.15	42.41
IO	51.19	74.92	34.20
SC	62.58	72.52	42.24
LM-KGQA	**69.03**	**83.46**	**81.28**

Table 1: Performance comparison across different baselines and backbone LLMs (values multiplied by 100).

4.2 Performance Comparison

Table 1 compares LM-KGQA with different baselines across three datasets and two backbone LLMs. LM-KGQA consistently achieves the best performance on all datasets and backbones, demonstrating the effectiveness of our LLM-guided KG evidence search framework. The improvement is especially notable on CWQ, which requires more complex multi-hop reasoning: LM-KGQA outperforms the strongest baseline ReKnoS under DeepSeek v3.2 (0.6773 vs. 0.6345), and shows similar gains under Qwen3.6-plus. It also obtains the best results on WebQSP and GrailQA, indicating good generalization across KGQA scenarios. Overall, these results show that expanding KG evidence search beyond local topic-entity neighborhoods improves grounded question answering.

4.3 Reward Weight Analysis

We analyze the sensitivity of the reward weights in the self-reward mechanism. As shown in Table 2, varying w_{r1} leads to only small performance changes across the three datasets, suggesting that LM-KGQA is relatively robust to the weight of the non-terminal evidence usefulness reward. The

$\mathbf{w}_{r1}$	CWQ	WebQSP	GrailQA
0.5	0.6724	0.8578	0.8080
0.7	**0.6773**	0.8560	**0.8098**
0.9	0.6686	**0.8584**	0.8080

Table 2: Parameter experiments on non-terminal node reward weight w_{r1}.

$(\mathbf{w}_{r2}, \mathbf{w}_{r3})$	CWQ	WebQSP	GrailQA
(0.3, 0.7)	**0.6773**	**0.8560**	0.8098
(0.5, 0.5)	0.6740	0.8542	**0.8130**
(0.7, 0.3)	0.6703	0.8536	**0.8130**

Table 3: Parameter experiments on terminal node reward weight w_{r2} and w_{r3}.

best results are obtained at $w_{r1} = 0.7$ on CWQ and GrailQA, while WebQSP slightly favors $w_{r1} = 0.9$, indicating that a stronger emphasis on newly introduced evidence can be helpful but may also introduce noise.

Table 3 further examines the terminal reward weights for answer groundedness and correctness. The setting $(w_{r2}, w_{r3}) = (0.3, 0.7)$ performs best on CWQ and WebQSP, while GrailQA obtains slightly better results with larger groundedness weights. This shows that correctness is important for selecting reliable terminal answers, while groundedness helps ensure that the answer is supported by KG evidence. Overall, the results demonstrate that our self-reward mechanism remains stable under different weight configurations.

4.4 Test-time Reasoning Budget Analysis

We compare three methods under varying test-time reasoning budgets. For SC, the budget is the number of sampled reasoning paths; for ToG, it is the number of candidate reasoning paths retained at each depth; and for LM-KGQA, it is the number of MCTS rollouts. As shown in Figure 2, LM-KGQA consistently outperforms SC and ToG across all budget levels on CWQ, WebQSP, and GrailQA. This indicates that the proposed MCTS-based evidence search is effective even under limited inference-time budgets.Compared with SC, which only increases the number of sampled textual reasoning paths, LM-KGQA explicitly expands and evaluates KG-grounded evidence states. Compared with ToG, which relies on beam-style path retention at each depth, LM-KGQA further uses self-rewarded MCTS rollouts to balance exploration and exploitation over partial evidence states. As

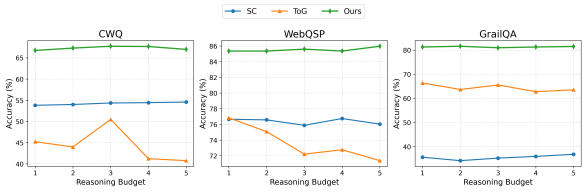

Figure 2: Performance under different test-time reasoning budgets. Our method consistently outperforms SC and ToG across all budget levels on CWQ, WebQSP, and GrailQA, showing the robustness of the proposed MCTS-based KG evidence search framework.

Method	CWQ	WebQSP	GrailQA
Ours	**0.6773**	**0.8560**	**0.8130**
w/o non-terminal reward	0.5547	0.8387	0.8090
w/o terminal reward	0.5593	0.8381	0.7990
w/o self-reward	0.5487	0.8373	0.8100

Table 4: Ablation study of the proposed self-reward mechanism.

a result, our method maintains stronger and more stable performance across different reasoning budgets, demonstrating the robustness of the proposed KG evidence search framework.

4.5 Ablation Study: Effects of self-reward mechanism

Table 4 reports the ablation results of the proposed self-reward mechanism. Removing any reward component leads to performance degradation, demonstrating that both non-terminal and terminal reward signals contribute to the effectiveness of LM-KGQA. Specifically, removing the non-terminal reward causes a clear drop on CWQ, from 0.6773 to 0.5547, indicating that evaluating the usefulness of newly introduced evidence is important for guiding MCTS toward promising reasoning trajectories. Similarly, removing the terminal reward also significantly hurts performance, especially on CWQ and GrailQA, showing that answer-level evaluation based on groundedness and correctness is necessary for selecting reliable terminal nodes. The obvious degradation is generally observed when the entire self-reward mechanism is removed, where rewards are no longer guided by LLM judgment and all set to 0.5. This confirms that the self-reward mechanism plays a central role in replacing the standard simulation step of MCTS in our KGQA setting. Overall, these results show that process-level evidence evaluation and answer-level terminal evaluation are complementary: the former helps

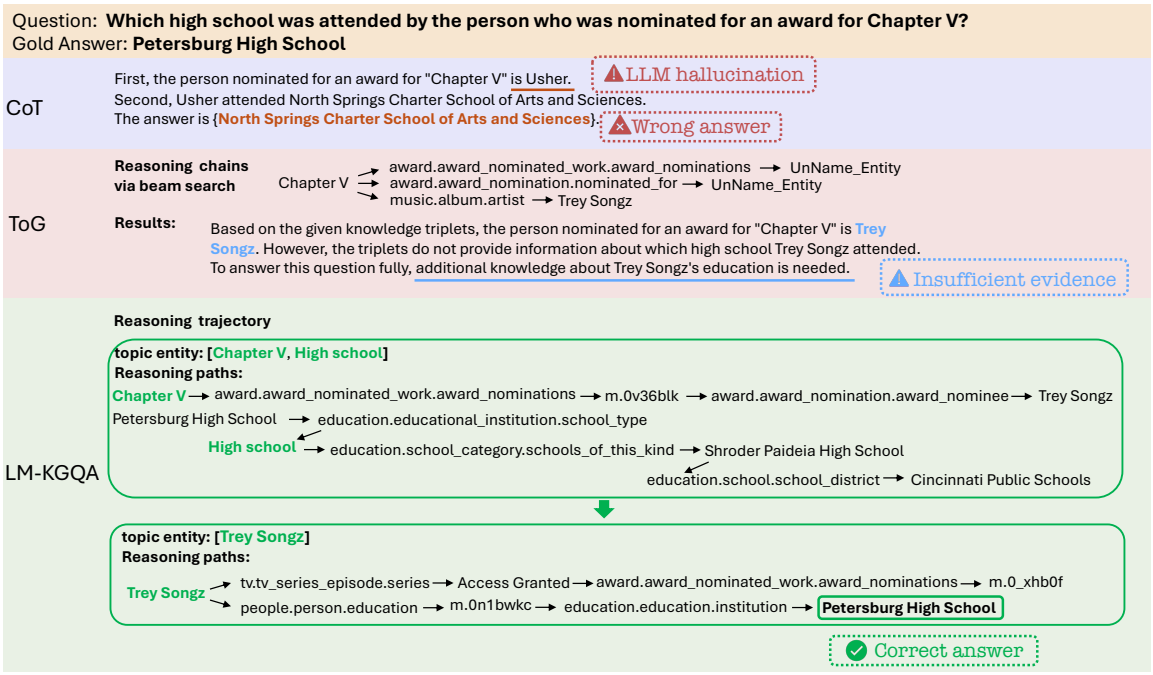

Figure 3: A case study comparing different methods to answer one complex question from CWQ.

expand useful KG evidence during search, while the latter ensures that the final answer is both correct and supported by the accumulated reasoning evidence.

4.6 Effects of different trace selection methods.

After several MCTS rollouts, we compare three reasoning trace selection strategies and examine how they affect final answer accuracy: *trace vote*, *best trace*, and *greedy trace*. Details of the implementation of these strategies can be found in Appendix D.

Table 5 in appendix D shows that Trace Vote generally yields the best or competitive performance by aggregating answers from multiple top-reward traces. Greedy Trace performs comparably and excels on GrailQA.

4.7 Case Study

Figure 3 represents a typical case from the CWQ dataset. We compare the results of CoT, ToG and LM-KGQA in answering the question "Which high school was attended by the person who was nominated for an award for Chapter V?". The backbone LLM utilized for all methods is DeepSeek v3.2. CoT fails to generate the correct person nominated for Chapter V as the first thinking step because of the hallucination caused by unsing only the LLM inherent knowledge, which leads to the final wrong answer. ToG successfully extracts the reasoning paths deriving the correct nominee "Trey Songz" from the topic entity "Chapter V", but constrained by their pre-defined search depth, it stops to further explore education information of "Trey Songz" and refuse to answer the question due to the insufficient evidence. Different from the above methods,

5 Conclusion

We introduce LM-KGQA, an LLM-guided Monte Carlo Tree Search framework for knowledge graph question answering. By modeling evidence retrieval as a reasoning-space search and incorporating a self-reward mechanism for node evaluation, our method effectively balances exploration and exploitation to comprehensively collect relevant evidence for complex questions. Experiments on CWQ, WebQSP, and GrailQA show that LM-KGQA consistently outperforms LLM-only, agent-based KG exploration, and non-LLM baselines across multiple backbone LLMs, with particularly large gains on multi-hop reasoning tasks. Analyses further validate the effectiveness of the self-reward mechanism and demonstrate robust performance across different test-time reasoning budgets.

Limitations

Despite its effectiveness, LM-KGQA has several limitations. First, the framework relies on multiple

LLM rollouts, which can incur significant computation and API costs, especially for large-scale datasets or larger backbone LLMs. Second, the self-reward mechanism depends on LLM-based evaluation, making it susceptible to inherent biases or inaccuracies in LLM judgment. In addition, when applied to open-domain or high-stakes question answering, incomplete KG evidence or inaccurate LLM judgments may lead to unsupported or incorrect answers, so the system should be carefully validated before deployment. Future work could explore more efficient search strategies and alternative reward estimation methods to further improve scalability and robustness.

References

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.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. 2024. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. *Advances in Neural Information Processing Systems*, 37:37665–37691.

Shuting Cui, Ying Sun, Yuting Zhang, Qingxin Meng, and Hengshu Zhu. 2026. Llm-enhanced career knowledge graph understanding for job mobility prediction. *ACM Trans. Manage. Inf. Syst.* Just Accepted.

Guangze Gao, Zixuan Li, Chunfeng Yuan, Jiawei Li, Wu Jianzhuo, Yuehao Zhang, Xiaolong Jin, Bing Li, and Weiming Hu. 2025. D-RAG: Differentiable retrieval-augmented generation for knowledge graph question answering. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 35398–35417, Suzhou, China. Association for Computational Linguistics.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, Haofen Wang, and 1 others. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1):32.

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.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.

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. In *Proceedings of the 14th ACM international conference on web search and data mining*, pages 553–561.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. 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.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Can Lin, Zhengwang Jiang, Ling Zheng, Qi Zhao, Yuhang Zhang, Qi Song, and Wangqiu Zhou. 2025. Rje: A retrieval-judgment-exploration framework for efficient knowledge graph question answering with llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17288–17305.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Linhao Luo, Yuan-Fang Li, Reza Haffari, and Shirui Pan. 2024. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *International Conference on Learning Representations*, volume 2024, pages 14400–14423.

Aigerim Mansurova, Aiganym Mansurova, and Aliya Nugumanova. 2024. Qa-rag: Exploring llm reliance on external knowledge. *Big Data and Cognitive Computing*, 8(9):115.

Costas Mavromatis and George Karypis. 2022. Rearev: Adaptive reasoning for question answering over knowledge graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2447–2458.

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.

Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and 1 others. 2023. Check your facts and try again: Improving large language models with external knowledge and automated feedback. *arXiv preprint arXiv:2302.12813*.

Qwen Team. 2026. Qwen3.6-Plus: Towards real world agents.

Kartik Sharma, Peeyush Kumar, and Yunqing Li. 2025. Og-rag: ontology-grounded retrieval-augmented generation for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 32950–32969.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *International Conference on Learning Representations*, volume 2024, pages 3868–3898.

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.

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.

Song Wang, Junhong Lin, Xiaojie Guo, Julian Shun, Jundong Li, and Yada Zhu. 2025. Reasoning of large language models over knowledge graphs with super-relations. *arXiv preprint arXiv:2503.22166*.

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*.

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.

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.

Congzhi Zhang, Jiawei Peng, Zhenglin Wang, Yilong Lai, Haowen Sun, Heng Chang, Fei Ma, and Weijiang Yu. 2025. Vrest: Enhancing reasoning in large vision-language models through tree search and self-reward mechanism. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3922–3941.

A Prompts

This section provides the prompt templates used in LM-KGQA. We include four types of prompts: the candidate entity generation prompt for LLM-guided node expansion, three self-reward prompts for evaluating path usefulness, answer groundedness, and answer correctness. Placeholders such as `{question}`, `{his_entities}`, and `{his_reason_paths}` are filled with the corresponding question-specific reasoning state during inference.

A.1 Candidate Entity Generation Prompt

```
You are given a question, a list of entities that have
      appeared in the current reasoning trajectory, and the
      reasoning paths that have already been explored. Your
      main task is to generate additional concrete entity
      names that may help continue reasoning on a knowledge
      graph (such as Freebase or Wikidata) to answer the
      question. Before generating entities, check whether the
       existing entities and reasoning paths already allow
      the answer to be directly inferred. Only output "Now we
       can answer the question: {{final_answer}}" if the
      answer can be clearly determined from the provided
      reasoning paths and entities. Otherwise, continue the
      reasoning process by generating candidate entity.

Output format:
- If the answer can already be directly inferred from the
      reasoning paths:
Now we can answer the question: {{final_answer}}
- Otherwise output exactly one candidate entity enclosed in
      curly braces:
{{entity}}

Now analyze the following reasoning state.
Question:
{question}
History Entities:
{his_entities}
History Reasoning Paths:
{his_reason_paths}
Output:
```

A.2 Path Usefulness Reward Prompt

```
You are given a question, historical entities, historical
      reasoning paths, and newly added reasoning paths. Your
      task is to determine whether the new reasoning paths
      are helpful for answering the question. A reasoning
      path is considered helpful if it introduces information
       that moves the reasoning closer to the final answer.
A reasoning path is NOT helpful if it introduces irrelevant
      information or information that does not contribute to
      answering the question. Output 'Yes' or 'No', and a
      reason.
```

```
Now determine whether the new reasoning path is helpful for
      answering the question.
Question:
{question}
History Entities:
{his_entities}
History Reasoning Paths:
{his_reason_paths}
New Reasoning Path:
{new_reason_path}
Is the new reasoning path helpful?
```

A.3 Answer Groundedness Reward Prompt

```
Given the question, accumulated reasoning paths, and a
      candidate final answer, determine whether the final
      answer is sufficiently supported by the provided
      reasoning paths. Output 'Yes' or 'No', and a reason.

Now determine whether the final answer is sufficiently
      supported by the provided reasoning paths.
Question:
{question}
Accumulated Reasoning Paths:
{his_reason_paths}
Candidate Final Answer:
{candidate_final_answer}
Is the final answer sufficiently supported?
```

A.4 Answer Correctness Reward Prompt

```
You are given a question and a candidate final answer.
      Determine whether the candidate final answer correctly
      answers the question. Use both the question and your
      internal knowledge to judge whether the answer is
      correct. Output only one word: Yes or No.

Question:
{question}
Candidate Final Answer:
{candidate_final_answer}
Is the answer correct?
```

B Baselines

This section briefly introduces the baselines used in our experiments. We group them into three categories: KG-augmented LLM reasoning methods, LLM-only methods, and non-LLM KGQA methods.

KG-augmented LLM Reasoning Methods

- ToG (Sun et al., 2024): ToG treats the LLM as an agent that interactively explores entities and relations on KGs through beam-search-style reasoning paths.
- PoG (Chen et al., 2024): PoG decomposes a question into sub-objectives and performs adaptive KG path exploration with guidance, memory, and reflection.
- StructGPT (Jiang et al., 2023): StructGPT uses an iterative reading-then-reasoning framework, where LLMs collect evidence from structured data through external interfaces and reason over the collected information.

Methods	CWQ	WebQSP	GrailQA
Trace-Vote	**0.6773**	**0.8560**	0.8098
Best-Trace	0.6684	0.8505	0.8070
Greedy-Trace	0.6748	0.8523	**0.8140**

Table 5: Analysis on different final answer aggregation methods.

- ReKnos (Wang et al., 2025): ReKnoS enhances LLM reasoning over KGs by introducing super-relations to organize KG evidence and support more effective reasoning.

LLM-only Methods

- CoT (Wei et al., 2022): CoT prompts the LLM to generate intermediate reasoning steps before producing the final answer.
- IO prompt (Brown et al., 2020): IO directly prompts the LLM with the input question and few-shot examples and asks it to output the final answer without explicit reasoning-path generation.
- SC (Wang et al., 2022): SC samples multiple reasoning chains and selects the final answer by majority voting over generated outputs.

Non-LLM KGQA methods

- NSM (He et al., 2021): NSM is a neural-symbolic KGQA method that learns to reason over intermediate entities for multi-hop question answering.
- ReaRev (Mavromatis and Karypis, 2022): ReaRev performs adaptive KG reasoning by updating question instructions with KG-aware information and emulating breadth-first search with graph neural networks.

C Implementation Details

All experiments are conducted in an inference-only setting without training or fine-tuning additional model parameters. We use DeepSeek v3.2 and Qwen3.6-plus as backbone LLMs through API-based inference. Since these are proprietary API models, their exact parameter sizes are not publicly disclosed. For LM-KGQA, the main computational cost comes from LLM calls during node expansion, self-reward evaluation, and final trace aggregation. We evaluate 3,531 samples from CWQ,

1,639 samples from WebQSP, and 1,000 samples from GrailQA. Unless otherwise specified, we use 3 MCTS rollouts with a maximum child-node depth of 3 and trace-vote size of 3. We generate 5 candidate entities for node expansion, link each candidate to 3 KG entities, extract 2-hop reasoning paths, and retain at most 3 paths per entity. We set the entity generation temperature to 0.3 and the maximum generation length to 2048 tokens. Experiments are performed on NVIDIA RTX A6000. All reported results are from a single run under the same experimental setting. For parameter and ablation studies, we report the corresponding single-run accuracy under each configuration.

We use all-MiniLM-L6-v2 pre-trained LM as the sentence embedding model to support semantic matching during graph retrieval and reasoning.

D Details of Trace Selection Methods

Trace Vote We first compute the reward of each reasoning trace by averaging the rewards of all nodes along the trace. Then, we select the top-k traces with highest trace rewards and extract the final answer by performing a majority vote over the answers from these top-k traces.

Best Trace Similar to Trace Vote, we compute the reward of each reasoning trace by averaging the rewards of all nodes along the trace. We then select the single trace with the highest trace reward as the final reasoning trace, and take the answer from its terminal node as the final answer.

Greedy Trace Starting from the root, we greedily select at each level the child node with the highest reward and follow this path until a terminal node is reached. The answer from that terminal node is taken as the final answer.

E Use of AI Assistants

We used AI assistants during the preparation of this paper. Specifically, ChatGPT was used to assist with language polishing, grammar correction, and improving the clarity of writing. Codex was used to assist with code debugging and implementation troubleshooting during experiments. All technical ideas, method design, experimental decisions, result interpretation, and final manuscript content were reviewed and verified by the authors. The AI assistants were not used to generate original research claims, conduct independent scientific analysis, or make final decisions about the paper.