Interactive Semantic Parsing with Reinforcement Learning for Knowledge Graph Reasoning

Yurun Song[1]* **Xiangqing Shen**[2]* **Jianfei Yu**[1,2,3] **Rui Xia**[2,3]†
[1]School of Computer Science and Engineering,
Nanjing University of Science and Technology, China
[2]School of Intelligence Science and Technology, Nanjing University, China
[3]University of Chinese Academy of Sciences, Nanjing, China
{yrsong, jfyu}@njust.edu.cn, {xqshen, rxia}@nju.edu.cn

Abstract

While large language models (LLMs) have achieved remarkable success, their reliability in knowledge-intensive tasks is often compromised by factual hallucinations. Integrating Knowledge Graphs (KGs) addresses this issue; however, existing approaches typically rely on simple graph traversal. This paradigm decouples topological navigation from logical operations (e.g., temporal filtering, aggregation), leading to imprecise retrieval and heavy post-processing burdens. Although semantic parsing offers a solution by grounding reasoning in logical forms, it traditionally suffers from a dependency on scarce supervised annotations. To bridge this gap, we propose Interactive Semantic Parsing, a framework that formulates reasoning as the sequential generation of executable logical clauses. This design allows logical constraints to be dynamically interleaved with graph search, while optimizing via reinforcement learning with only final answer feedback eliminates the need for gold program annotations. To tackle the sparse reward challenge in the vast symbolic space, we introduce a distance-aware process reward to evaluate intermediate steps based on their topological proximity to the answer. Experimental results on WebQSP and CWQ demonstrate that our method achieves state-of-the-art performance, particularly on complex queries, validating the effectiveness of our dense reward signal in enabling robust reasoning without supervision. Our code is available at https://github.coms/NUSTM/ISP-KGR.

1 Introduction

While large language models (LLMs) have achieved remarkable success across various tasks (Brown et al., 2020; Team, 2023; DeepSeek-AI, 2025; Wu et al., 2024), their reliability is often compromised by factual hallucinations. This problem becomes especially severe in knowledge-intensive applications (Huang et al., 2025; Ji et al., 2023). Integrating Knowledge Graphs (KGs) as structured, high-fidelity buffers has emerged as a primary solution to enhance LLM reliability (Pan et al., 2024; Zhang et al., 2024).

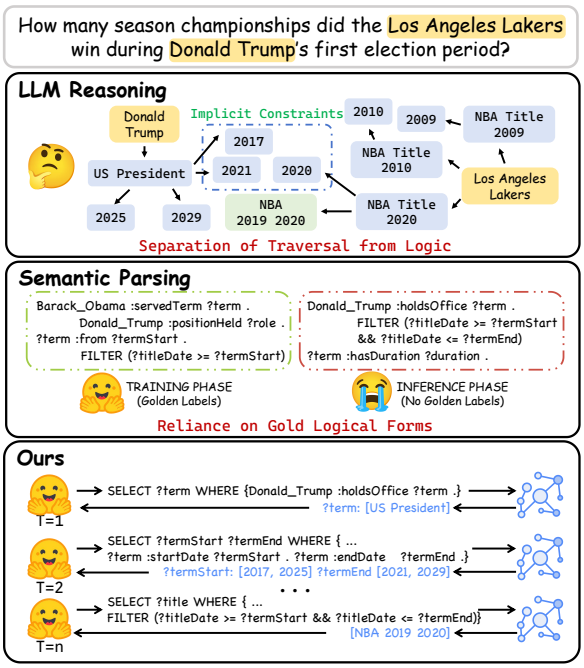


Figure 1: Comparison of three KGQA paradigms.

Existing approaches in this field largely follow an LLM reasoning paradigm over KGs, whether employing the LLM as an interactive agent to explore KGs (Jiang et al., 2023a, 2025; Xiong et al., 2025; Tan et al., 2025) or directly generating entity-relation symbol sequences as reasoning paths (Luo et al., 2024b, 2025b; Mavromatis and Karypis, 2025; Chen et al., 2024). At their core, these methods identify answers by walking along graph edges to assemble a chain of relations connecting query entities to potential answers. The critical bottleneck lies in the atomic nature of this walking process: it is restricted to simple topological navigation and cannot inject complex logical constraints into the

*Equal Contribution.
†Corresponding Author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 20464–20484

search. Yet many queries demand operations beyond mere connectivity. For example, they may require temporal filtering, attribute comparison, or aggregation. Such operations must be executed together with graph traversal to effectively prune the search space. As shown in Figure 1, consider the query "How many season championships did the Los Angeles Lakers win during Donald Trump's first election period?". A standard graph walker may retrieve the path (Lakers $\xrightarrow{won}$ NBA Title). However, it cannot enforce the temporal constraint ("during Trump's term") or the aggregation function ("count") during the walk. As a result, it returns all championship records regardless of time, leaving the heavy burden of filtering and counting to the LLM's post-processing. This separation of traversal from logic inevitably leads to imprecise retrieval and unstable reasoning.

To fundamentally resolve the detachment between graph traversal and logical operations, we turn to Semantic Parsing, which maps natural language queries into executable logical forms (LFs) (Yih et al., 2015; Yu et al., 2023; Gao et al., 2025b; Zhao et al., 2025; Fang et al., 2024). Distinct from flat path representations, LFs explicitly encode both the topological structure for subgraph matching and the logical constraints to be satisfied. This ensures that reasoning is rigorously grounded in the KG schema. This allows for the precise execution of complex reasoning tasks that LLM-based reasoning methods often fail to capture. However, the applicability of this paradigm is severely constrained by its heavy reliance on supervised training. Learning the complex mapping from natural language to LFs necessitates high-quality parallel data, yet acquiring such annotations is notoriously labor-intensive and scarce. This data scarcity creates a critical bottleneck: models fail to generalize to new domains or schemas where training samples are unavailable. This strictly limits their real-world scalability.

To navigate the trade-off between logical expressivity and the cost of supervision, we propose an Interactive Semantic Parsing framework optimized via Reinforcement Learning. Unlike graph walkers that step through nodes, our agent formulates reasoning as the sequential generation of executable logical clauses. This design allows logical constraints to be applied dynamically at each step, effectively interleaving graph search with computation. Critically, to bypass the reliance on gold programs, we formulate the training as a reward maximization problem driven by final answer feedback. Recognizing that learning from sparse binary signals in a vast symbolic space is challenging, we introduce a novel distance-aware process reward. This mechanism evaluates whether each intermediate execution step effectively reduces the topological distance to the target answer on the KG. Acting as a navigational compass, this dense feedback guides the agent to discover correct logical structures without relying on expensive step-by-step annotations.

Experimental results on WebQSP and CWQ benchmarks demonstrate that our framework achieves state-of-the-art performance. The gains are particularly pronounced on complex queries, validating the efficacy of interleaving logical generation with graph search. Extensive analyses confirm that our dense reward is the key driver of this success. Unlike sparse result-based supervision, it delivers the dense feedback needed to stabilize training and mitigate optimization difficulties in long reasoning trajectories. Notably, training process metrics reveal that our agent progressively expands its capability boundary, successfully solving questions that were initially intractable; it does not merely memorize known trajectories.

2 Related Works

LLM Reasoning On KG. Integrating LLMs with KGs combines parametric and structured knowledge for complex reasoning. Existing approaches broadly fall into path generation and agent-based methods. Path generation methods follow a retrieve-then-reason paradigm, linearizing retrieved triples or subgraphs as model input. These methods have made encouraging progress in grounding LLM reasoning on KGs: some works incorporate GNNs to improve semantic–structural alignment (Mavromatis and Karypis, 2025), while others integrate planning with retrieval (Luo et al., 2024b) or construct context-aware subgraphs (Xu et al., 2025). Building on these foundations, a promising direction is to further enhance expressiveness beyond linearized representations and to explore reasoning that goes beyond predefined logical forms, opening room for richer structural modeling. In contrast, agent-based methods model LLMs as interactive agents that explore KGs through iterative planning, reflection, or program execution to enable multi-hop reasoning (Sun et al., 2024; Chen et al., 2024; Jiang et al., 2025), with some further

improving robustness via multi-agent collaboration (Ma et al., 2025). These interactive designs have proven highly effective for multi-hop reasoning. Nevertheless, the predefined tool inventories they typically rely on can sometimes constrain how efficiently and comprehensively the graph is accessed, suggesting that deeper LLM–KG interaction through richer logical languages remains an open direction.

Semantic Parsing. Semantic parsing maps natural language questions to executable logical forms and offers full expressiveness for KGQA (Mitra et al., 2022; Sun et al., 2020; Scholak et al., 2021; Zhang et al., 2019). With the advent of LLMs, recent work revisits logical languages by combining generation with retrieval or fallback strategies, such as ChatKBQA, which generates logical skeletons with retrieved slot filling (Luo et al., 2024a), and DECAF, which jointly decodes logical forms and answers (Yu et al., 2023). Another line of research improves controllability and execution stability through structural constraints, including component composition (Zhang et al., 2023), rule-space restriction (Zhang et al., 2025), and path-grounded generation (Zhao et al., 2025). These advances have substantially improved the quality and reliability of generated logical forms. Many of these approaches leverage explicit reasoning traces or intermediate supervision to guide learning effectively (Fang et al., 2024). Complementary to this line of work, our approach explores learning interactive semantic parsing directly in the full logical language space using only final-answer supervision, without requiring gold logical forms or annotated reasoning trajectories.

3 Preliminary

Knowledge Graph is a collection of structured factual triples $\mathcal{G} = \{(e, r, e') \mid e, e' \in \mathcal{E}, r \in \mathcal{R}\}$, where $\mathcal{E}$ denotes the entity set and $\mathcal{R}$ denotes the relation set. Each triple (e, r, e') states a factual relation r from a head entity e to a tail entity e'.

Knowledge Graph Question Answering aims to answer natural language questions using structured knowledge. Given a question q, a knowledge graph $\mathcal{G}$, and a set of topic entities $\mathcal{T}_q \subseteq \mathcal{E}$ mentioned in q, the task is to identify a set of answer entities $\mathcal{A}_q \subseteq \mathcal{E}$.

Semantic Parsing transforms natural language queries $q \in \mathcal{Q}$ into logical forms $z \in \mathcal{Z}$, where $\mathcal{Z}$ denotes a space of formal representations. In the context of KGQA, the logical form z is a structured query executable on a knowledge graph $\mathcal{G}$ to retrieve the answer a.

4 Methodology

4.1 Interactive Semantic Parsing Framework

We construct an interactive semantic parsing framework to address the KGQA task. At the core of this framework is a generative agent instantiated by an LLM. Given an input question q, the agent performs interactive semantic parsing by progressively mapping the natural language query into a sequence of executable logical forms. This process operates through a multi-turn "generate-execute-feedback" loop: at each step, the agent generates a structured output based on the current history, executes it against the KG, and utilizes the feedback to guide subsequent reasoning.

We define the key components of this interactive process as follows:

- **Composite Generation** (t, c)**:** The agent generates a composite structure at each step. This consists of a reasoning thought t (serving as an intermediate semantic plan) and an executable command c (serving as the execution unit).
- **Command Formulation:** The command c is execution instructions encapsulating three potential operations: (1) `<node>` for relation expansion; (2) `<SPARQL>` for structured query;[1] and (3) `<answer>` for final answer derivation. Critically, this pipeline supports a selective activation mechanism, enabling the agent to autonomously activate specific operations while skipping others. This flexibility allows the agent to dynamically switch between exploring the knowledge graph structure and executing precise queries within a single step.
- **Environmental Feedback** (o)**:** Upon receiving the command, the environment executes the activated operations in sequence. It then returns a textual observation o based on the execution outcome. This observation is appended to the interaction history, serving as grounding context for the subsequent reasoning step.

Figure 2 illustrates a partial reasoning trajectory of this framework. Taking the k-th branch at time step 5 as an example, the agent acts based on the history where award-winning directors have been

[1]We employ SPARQL as the concrete logical form in this work.

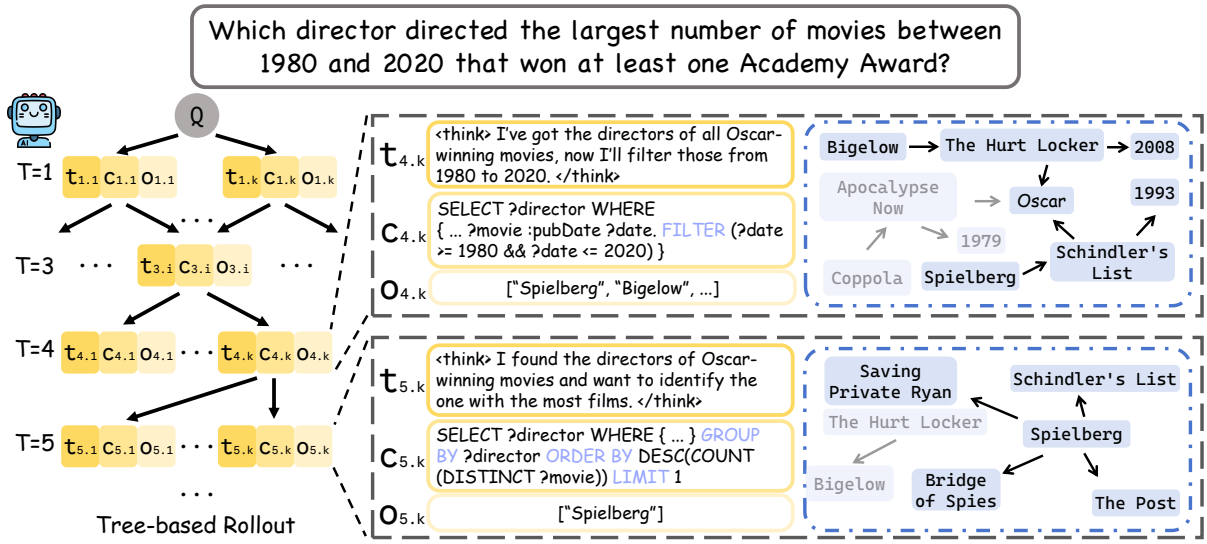


Figure 2: Tree-based rollout for our framework. At each time step, the agent generates a composite structure, executes it as query over the knowledge graph, and receives observation to guide subsequent exploration.

retrieved. First, it generates a thought $t_{5,k}$ (enclosed in <think>), analyzing the need to impose a superlative constraint. Guided by this semantic plan, it generates the command $c_{5,k}$. In this specific instance, the agent activates the <SPARQL> component to handle the superlative constraint. Finally, the environment executes this query and returns the observation $o_{5,k}$ (Spielberg), which confirms the reasoning hypothesis and drives the agent towards the final answer.

4.2 Interactive Reasoning as MDP

To rigorously model the composite generation process described in Section 4.1, we formalize the interactive reasoning as a Markov Decision Process (MDP), represented by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$.

- **State ($\mathcal{S}$):** The state $S_i \in \mathcal{S}$ is defined as the full interaction history up to time step i, where $S_i = \big(q, (A_1, o_1, \ldots, A_{i-1}, o_{i-1})\big)$. It provides the complete interaction history for the agent.
- **Action ($\mathcal{A}$):** Aligning with the interaction protocol, the action space is composite. Each action $A_i \in \mathcal{A}$ is a tuple $A_i = (t_i, c_i)$, where t_i denotes the reasoning thought (natural language plan) and c_i denotes the executable command (execution instructions). This formulation decouples semantic planning from symbolic execution.
- **Transition ($\mathcal{T}$):** The transition is deterministic and environment-driven. Given state S_i and action A_i, the environment executes the command c_i against the knowledge graph to produce an observation o_i. The next state is formed by appending this feedback to the history: $S_{i+1} = (S_i, A_i, o_i)$.
- **Reward ($\mathcal{R}$):** The reward function $\mathcal{R}(t_i, c_i)$ evaluates the quality of reasoning. To address the challenge of sparse supervision in multi-step reasoning, we design a dense reward mechanism, as detailed in Section 4.3.2.

4.3 Distance-guided Tree-rollout Policy Optimization

Interactive multi-step reasoning is characterized by long horizons and compositional complexity. In such settings, supervision is inherently sparse: relying solely on final answer rewards leads to the credit assignment problem, where the model fails to receive timely feedback for intermediate actions. Consequently, the core challenge lies in designing dense and comparable signals to evaluate intermediate steps, especially when numerous partially valid trajectories exist.

To address this, we propose the **Distance-guided Tree-rollout Policy Optimization (DTPO)** algorithm. Its core motivation is to leverage the structural information of knowledge graphs to provide distance-aware progress rewards. This mechanism offers fine-grained, state-level feedback without requiring human-annotated trajectories.

4.3.1 Tree-based Rollout and Pruning

Crucially, the quality of intermediate decisions cannot be reliably evaluated in isolation on a single trajectory. Effective evaluation requires a comparative perspective, where different composite generations stemming from a shared interaction history are assessed against each other. To achieve this,

we adopt a tree-structured rollout strategy as the concrete implementation of DTPO. By exposing multiple candidate actions at the same state, the tree structure allows the model to explicitly compare these composite generations, thereby stabilizing the distance-based reward signal. The rollout process operates iteratively through two phases: *Diversity Expansion* and *Redundancy Control*.

Phase 1: Diversity Expansion via K-Branch Sampling. To tackle the sparsity of valid reasoning trajectories, we encourage exploration by expanding the search space. At step i, given an active trajectory history S_{i-1}, we sample k independent action pairs from the current policy to form a branch set:

$$\{(t_{i,j}, c_{i,j})\}_{j=1}^{k} \sim \pi_\theta(\cdot \mid S_{i-1}).$$

Here, k is the branching factor. Unlike generating linear trajectories, this strategy allows multiple candidate actions to share the same interaction history.

Phase 2: Redundancy Control via Observation-Equivalence Pruning. While sampling enhances diversity, naive expansion often introduces substantial redundant branches. A key insight is that in KG interactions, different reasoning thoughts t may correspond to the same executable command c, resulting in equivalent environmental observations o. To maintain computational efficiency, we implement observation-equivalence pruning. We define the uniqueness key of a branch as the linearized string representation of its observation result o. For successful commands yielding the same observation, only one representative branch is retained in the active set $\mathcal{P}$; for failed commands, the corresponding branches are preserved in the completed set $\mathcal{C}$ as negative samples.

4.3.2 Distance-guided Dense Reward Design for Reasoning

Building upon the interaction history provided by the tree-structured rollout, we further construct a dense reward function $R(t, c)$ based on environmental feedback for intermediate reasoning steps to achieve fine-grained credit assignment. This reward consists of the following three components.

Format Reward (R_{format}). To ensure the format of generated actions, we impose a strict binary reward. $R_{\text{format}} = 1$ if and only if the generation adheres to the defined protocol: the reasoning thought t is correctly enclosed within <think> tags, and the command c uses valid operation tags (<node>, <SPARQL>, or <answer>) with exactly one thought-command pair. Otherwise, $R_{\text{format}} = 0$.

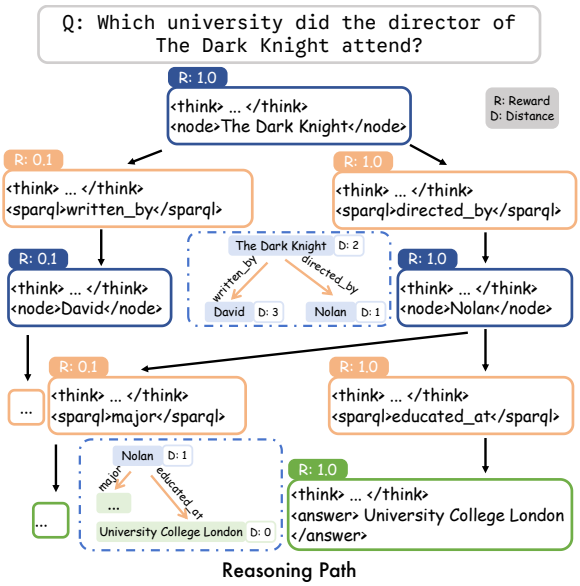


Figure 3: Illustration of the distance-guided reasoning process. The agent explores candidate actions with different rewards (R) and graph distances (D)

Distance-guided Progress Reward (R_{progress}). This component characterizes the contribution of intermediate steps toward the final goal. We define the logic of "progress" by quantifying the topological distance on the KG. In a typical interaction flow, <node> and <SPARQL> operations appear alternately: the former determines the query starting point, while the latter executes structured retrieval. Let d_{pre} be the shortest path distance from the node indicated by the most recent <node> to the answer set, and d_{post} be the distance from the observation o returned by the current <SPARQL> to the answer set. The progress reward is defined as:

$$R_{\text{progress}} = \begin{cases} +1, & \text{if } d_{post} < d_{pre} \\ 0, & \text{if } d_{post} \geq d_{pre} \\ -1, & \text{if no result or error.} \end{cases}$$

A reward of $+1$ is assigned if the query effectively shortens the distance to the answer; 0 if the distance remains unchanged; and -1 if the query leads away or causes errors. Detailed distance calculation methods and determination rules are provided in Appendix F.9.

Outcome Reward (R_{outcome}). To evaluate the correctness of the final answer, we employ the F1 score (ranging in $[0, 1]$) between the predicted answer and the ground truth. Compared to strict exact match metrics, the F1 score provides a smoother supervision signal, distinguishing partially correct predictions from completely erroneous results.

Methods	Base Model	WebQSP		CWQ	
		F1	Hit	F1	Hit
FC-KBQA (Zhang et al., 2023)	–	–	76.9	–	56.4
SR+NSM (Yih et al., 2015)	–	64.1	68.9	47.1	50.2
ReaRev (Mavromatis and Karypis, 2022)	–	70.9	76.4	–	52.9
UniKGQA (Jiang et al., 2023b)	–	70.2	75.1	48.0	50.7
DECAF (Yu et al., 2023)	–	77.1	80.7	–	67.0
StructGPT (Jiang et al., 2023a)	ChatGPT	–	72.6	–	–
ToG (Sun et al., 2024)	GPT-4	–	82.6	–	68.5
iQUEST (Wang and Yu, 2025)	GPT-4o	–	88.9	–	73.8
BYOKG-RAG (Mavromatis et al., 2025)	Claude-Sonnet	–	86.6	–	73.6
ORT (Liu et al., 2025b)	DeepSeek-V3	71.8	89.4	62.6	72.9
RoG (Luo et al., 2024b)	LLaMA-2-7B	70.8	85.7	56.2	62.6
G-Retriever (He et al., 2024)	LLaMA-2-7B	–	70.1	–	–
GNN-RAG (Mavromatis and Karypis, 2025)	LLaMA-2-7B	71.3	85.7	59.4	66.8
KG-Agent (Jiang et al., 2025)	LLaMA-2-7B	81.0	83.3	69.8	72.2
D-RAG (Gao et al., 2025a)	LLaMA-3-8B	80.5	89.1	63.8	70.3
Rule-KBQA (Zhang et al., 2025)	LLaMA-3-8B	81.9	84.1	-	73.5
SubgraphRAG (Li et al., 2025b)	LLaMA-3.1-8B	70.6	86.6	47.2	57.0
CoG (Zhao et al., 2025)	LLaMA-3.1-8B	78.0	90.5	60.8	70.3
MCTS-KBQA (Xiong et al., 2025)	LLaMA-3.1-8B	76.0	76.2	66.8	75.2
GCR (Luo et al., 2025b)	LLaMA-3.1-8B	79.1	**92.2**	61.7	**75.8**
KBQA-o1 (Luo et al., 2025a)	Qwen2.5-7B	–	57.8	–	–
KnowCoder-A1 (Chen et al., 2025)	Qwen2.5-Coder	77.2	80.1	–	68.3
Ours	Qwen2.5-3B	**84.1**	86.3	**71.8**	75.4

Table 1: Comparative performance on WebQSP and CWQ. We report F1 and Hit scores across different base models. **Bold** indicates the best performance, and underline indicates the second-best performance.

Trajectory-Aggregated Credit Assignment. Finally, we assign a scalar reward to each action (t_i, c_i) by aggregating the returns of all complete reasoning trajectories τ passing through that node. We adopt a max-reward aggregation strategy:

$$R(t_i, c_i) = w_1 \cdot R_{\text{format}}(t_i, c_i) + w_2 \cdot R_{\text{progress}}(t_i, c_i) + \max_{\tau \in \Omega(t_i, c_i)} (w_3 \cdot R_{\text{outcome}}(q, \tau))$$

where $\Omega(t_i, c_i)$ denotes the set of all complete reasoning trajectories containing this node. We use the maximum trajectory outcome reward as the outcome score for the node, implying that as long as an intermediate decision lies on at least one trajectory leading to a high-quality answer, it is considered a valuable exploration choice.

Optimization. We optimize the policy using Group Relative Policy Optimization (GRPO) (DeepSeek-AI, 2025), where the k sampled branches naturally form the comparison groups. See Appendix G.1 for the detailed objective.

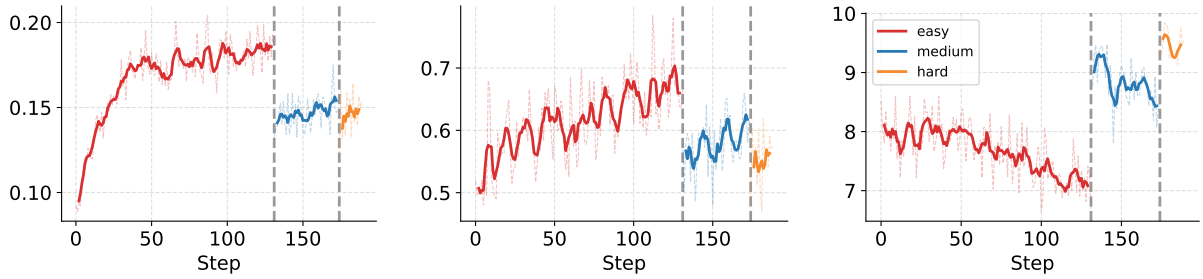
5 Experiment

5.1 Experimental Setup

Dataset. We evaluate ours on two widely used benchmarks: WebQuestionsSP (WebQSP) (Yih et al., 2016) and Complex WebQuestions (CWQ) (Talmor and Berant, 2018). Both datasets are grounded on Freebase (Bollacker et al., 2008), which contains approximately 88 million entities, 20,000 relation types, and 126 million RDF triples.

Implementation Details. We adopt Qwen2.5-3B-Instruct as the base model. During reinforcement learning, the sampling factor is set to $k = 16$. The reward weights are set to $w_1 = 0.1$, $w_2 = 0.6$, and $w_3 = 0.3$. We further employ a curriculum learning strategy based on question difficulty, where the training data are divided into three subsets (easy, medium, and hard) and introduced sequentially during training.

Evaluation Metrics. Following prior work, we report Hit and F1 as the evaluation metrics. Hit measures whether at least one predicted answer appears in the gold answer set, while F1 evaluates the overlap between the predicted answers and the gold answers. Compared with Hit, F1 provides a more stringent assessment of both answer accuracy and completeness.



(a) Average Reward (b) Success Group Rate (c) Average Nodes

Figure 4: Training process metrics. Fig. 4(a) shows the mean reward of interaction trajectories. Fig. 4(b) reports the success rate per batch (samples with at least one correct trajectory). Fig. 4(c) displays the average branch factor (number of child nodes) after filtering.

Baselines. To demonstrate the effectiveness of our framework, we compare our method with several representative baselines. All baseline results are taken directly from the corresponding original papers. Detailed descriptions of these baselines are provided in the appendix B.

5.2 Main Results

Table 1 reports performance on WebQSP and CWQ. Our method achieves the best F1 score on both benchmarks. Meanwhile, it consistently maintains competitive Hit scores. These results indicate that our model not only covers correct entities but also predicts more complete answer sets.

Notably, our approach outperforms recent reinforcement learning based KGQA methods, such as KnowCoder-A1. We attribute this improvement to our fine-grained process reward design. Unlike trajectory-level rewards, our approach provides denser and more stable supervision for multi-step reasoning. This design mitigates low signal-to-noise ratios and optimization difficulty in long reasoning chains. As a result, the model more precisely evaluates the quality of each decision.

In contrast to methods that assemble retrieved knowledge paths and rely on free-form language model reasoning, our framework interacts with the knowledge graph through explicit logical queries. This design avoids reasoning over redundant and noisy paths. Consequently, it yields more stable and substantial F1 improvements. The gains are particularly pronounced on CWQ, which requires more complex reasoning.

To further analyze the source of performance gains, we monitor key training process metrics (Figure 4). Figure 4(a) shows a steady increase in average reward, indicating efficient RL convergence. Figure 4(c) shows a decreasing number of

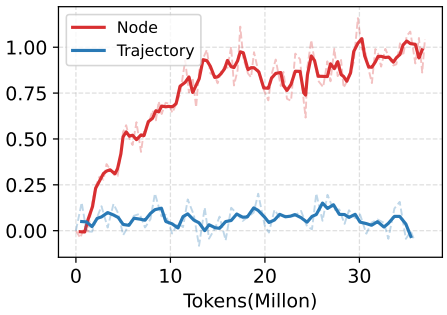


Figure 5: Comparison of reward dynamics between state-level and trajectory-level optimization strategies during training.

child nodes, suggesting fewer erroneous actions over time. This trend reflects a more efficient reasoning process.

To distinguish improvement on already solvable questions from genuine generalization, we introduce a question-level success rate (Figure 4(b)). In our setting, each question samples multiple trajectories. Thus, higher rewards may only reflect reduced errors on known solvable questions. However, Figure 4(c) tracks the proportion of questions that contain at least one correct trajectory. The consistent increase of this metric has clear statistical significance. It indicates an expanding capability boundary. Specifically, the model begins to solve questions that were initially unsolvable. These results demonstrate that fine-grained process rewards promote effective exploration, rather than merely stabilizing existing policies.

5.3 Ablation Studies

In this section, we further analyze the effects of tree-based expansion and reward design. **RQ1**: whether state-level optimization yields better reward signals than trajectory-level optimization. **RQ2**: whether deduplicating successful actions

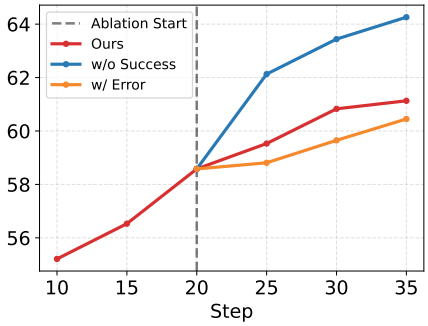


Figure 6: F1-score variations on CWQ test dataset across three pruning strategies.

while retaining all failed actions affects training. **RQ3**: whether process rewards improve training compared with result-only rewards.

Effectiveness of Node-Level Optimization (RQ1). We compare trajectory-level and state-level optimization using identical hyperparameters. For the former, we average state-level rewards across nodes in a trajectory to assign a single score, sampling 16 independent trajectories per example. Figure 5 shows that trajectory-level optimization suffers from high variance and lacks a clear upward trend, whereas state-level optimization yields stable, continuous reward growth. This suggests that trajectory-level aggregation dilutes learning signals (the credit assignment problem), making it unsuitable for long-horizon reasoning. In contrast, state-level optimization provides finer-grained feedback, stabilizing training dynamics.

Impact of Environment-Guided Pruning (RQ2). To balance diversity and efficiency, our method deduplicates successful actions (identical executable commands) while retaining all failed actions. We evaluate this by: (a) disabling successful action deduplication, and (b) partially filtering failed actions. Figure 6 and Table 2 indicate that while retaining all successful duplicates slightly improves F1 (+4.2%), it drastically increases computational costs (13.9 × rollout time). Conversely, filtering failed actions offers minimal savings but hinders learning, as failures provide valuable negative feedback without extending search depth. Our strategy achieves the optimal trade-off: efficient computation by pruning redundant successes and robust learning by preserving full failure signals.

Influence of Distance-Aware Progress Rewards (RQ3). We examine the necessity of process rewards by adjusting the weights of process (w_2) and outcome (w_1) rewards on the medium subset.

Methods	Time Spent	
	Rollout	Train
StepKG	1.0×	1.0×
w/o success filter	13.9×	12.6×
w error filter	1.0×	0.86×

Table 2: Comparison of Rollout and Training Time Overheads for Three Filtering Strategies

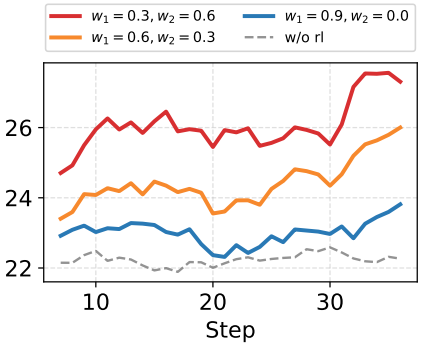


Figure 7: Average F1 scores of all trajectories collected during training on the "medium" dataset, using various reward weight configurations.

For $R(t, c)$, we test: (1) no process reward ($w_1 = 0.9, w_2 = 0.0$); (2) outcome-dominant rewards ($w_1 = 0.6, w_2 = 0.3$); (3) process-dominant rewards ($w_1 = 0.3, w_2 = 0.6$). As shown in Figure 7, the process-dominant setting achieves a 16.5% relative improvement, significantly outperforming the outcome-only setting (+3.6%). This confirms that sparse result rewards are insufficient for complex reasoning. Our distance-aware progress mechanism provides dense intermediate feedback, effectively mitigating the credit assignment problem and guiding the model through large search spaces.

6 Conclusion

In this paper, we presented an Interactive Semantic Parsing framework to bridge the gap between graph traversal and logical reasoning in KG-augmented LLMs. By formulating reasoning as the sequential generation of executable logical clauses, our approach effectively interleaves topological search with dynamic constraint execution, addressing the limitations of atomic graph walkers on complex queries. Crucially, to mitigate the reliance on scarce logical form annotations, we introduced a novel distance-guide process reward. This mechanism provides dense, navigational feedback based on topological distance, enabling the agent to learn

robust reasoning policies solely from weak answer supervision. Experimental results on WebQSP and CWQ benchmarks demonstrate that our framework achieves state-of-the-art performance, with significant gains in complex multi-hop reasoning. Our analysis confirms that replacing expensive gold logical forms with graph-grounded dense process rewards effectively stabilizes training, offering a scalable and interaction-driven paradigm for high-fidelity neuro-symbolic reasoning.

Limitations

Despite these results, our work has several limitations. First, although the framework supports logical language interaction, node-level operations still rely on predefined tools. Thus, the model does not fully leverage SPARQL-style query generation to explore relational structures. Second, we evaluate our method only on Freebase-based benchmarks. We do not assess it on Wikidata, whose schema and scale differ substantially. Finally, our reward design assumes access to a relatively complete and reliable knowledge graph. When the graph contains missing or noisy edges, distance-aware process rewards may become less effective. We leave these issues for future work.

Ethics Statement

This work does not pose any ethical issues. All the data and models used in this paper are publicly available and are used under following licenses: Creative Commons BY 4.0 License, MIT License, Apache license 2.0. All existing artifacts (the open-source model and benchmark datasets) are used strictly for research purposes, which is consistent with their intended use as specified in the original releases.

Acknowledgments

This work was supported by the Natural Science Foundation of China under Grant No.62476134 and No. 62476132.

References

Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250. ACM.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

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. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Zhuo Chen, Fei Wang, Zixuan Li, Zhao Zhang, Weiwei Ding, Chuanguang Yang, YongJun Xu, Xiaolong Jin, and Jiafeng Guo. 2025. Knowcoder-a1: Incentivizing agentic reasoning capability with outcome supervision for KBQA. *CoRR*, abs/2510.25101.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948.

Haishuo Fang, Xiaodan Zhu, and Iryna Gurevych. 2024. DARA: decomposition-alignment-reasoning autonomous language agent for question answering over knowledge graphs. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 3406–3432. Association for Computational Linguistics.

Guangze Gao, Zixuan Li, Chunfeng Yuan, Jiawei Li, Wu Jianzhuo, Yuehao Zhang, Xiaolong Jin, Bing Li, and Weiming Hu. 2025a. 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 35386–35405, Suzhou, China. Association for Computational Linguistics.

Jianqi Gao, Jian Cao, Ranran Bu, Nengjun Zhu, Wei Guan, and Hang Yu. 2025b. Promoting knowledge base question answering by directing llms to generate task-relevant logical forms. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 23914–23922. AAAI Press.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2):42:1–42:55.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12):248:1–248:38.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023a. Structgpt: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 9237–9251. Association for Computational Linguistics.

Jinhao Jiang, Kun Zhou, 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), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 9505–9523. Association for Computational Linguistics.

Jinhao Jiang, Kun Zhou, Xin Zhao, and Ji-Rong Wen. 2023b. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, Kuan Li, Liangcai Su, Litu Ou, Liwen Zhang, Pengjun Xie, Rui Ye, Wenbiao Yin, Xinmiao Yu, Xinyu Wang, Xixi Wu, and 36 others. 2025a. Tongyi deepresearch technical report. *CoRR*, abs/2510.24701.

Mufei Li, Siqi Miao, and Pan Li. 2025b. Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Guangyi Liu, Yongqi Zhang, Yong Li, and Quanming Yao. 2025a. Dual reasoning: A GNN-LLM collaborative framework for knowledge graph question answering. In *Conference on Parsimony and Learning, Stanford University, USA, 24-27 March 2025*, volume 280 of *Proceedings of Machine Learning Research*, pages 351–372. PMLR.

Runxuan Liu, Bei Luo, Jiaqi Li, Baoxin Wang, Ming Liu, Dayong Wu, Shijin Wang, and Bing Qin. 2025b. Ontology-guided reverse thinking makes large language models stronger on knowledge graph question answering. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15269–15284, Vienna, Austria. Association for Computational Linguistics.

Haoran Luo, Haihong E, Yikai Guo, Qika Lin, Xiaobao Wu, Xinyu Mu, Wenhao Liu, Meina Song, Yifan Zhu, and Anh Tuan Luu. 2025a. Kbqa-o1: Agentic knowledge base question answering with monte carlo tree search. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*. OpenReview.net.

Haoran Luo, Haihong E, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, Yifan Zhu, and Anh Tuan Luu. 2024a. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 2039–2056. Association for Computational Linguistics.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024b. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Yuan-Fang Li, Chen Gong, and Shirui Pan. 2025b. Graph-constrained reasoning: Faithful reasoning on knowledge graphs with large language models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*. OpenReview.net.

Jie Ma, Zhitao Gao, Qi Chai, Wangchun Sun, Pinghui Wang, Hongbin Pei, Jing Tao, Lingyun Song, Jun Liu, Chen Zhang, and Lizhen Cui. 2025. Debate on graph: A flexible and reliable reasoning framework for large language models. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 24768–24776. AAAI Press.

Costas Mavromatis, Soji Adeshina, Vassilis N. Ioannidis, Zhen Han, Qi Zhu, Ian Robinson, Bryan Thompson, Huzefa Rangwala, and George Karypis. 2025. BYOKG-RAG: multi-strategy graph retrieval for knowledge graph question answering. *CoRR*, abs/2507.04127.

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, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 2447–2458. Association for Computational Linguistics.

Costas Mavromatis and George Karypis. 2025. GNN-RAG: graph neural retrieval for efficient large language model reasoning on knowledge graphs. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 16682–16699. Association for Computational Linguistics.

Sayantan Mitra, Roshni R. Ramnani, and Shubhashis Sengupta. 2022. Constraint-based multi-hop question answering with knowledge graph. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track, NAACL 2022, Hybrid: Seattle, Washington, USA + Online, July 10-15, 2022*, pages 280–288. Association for Computational Linguistics.

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

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 9895–9901. Association for Computational Linguistics.

Jinyeop Song, Song Wang, Julian Shun, and Yada Zhu. 2025. Efficient and transferable agentic knowledge graph RAG via reinforcement learning. *CoRR*, abs/2509.26383.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Yawei Sun, Lingling Zhang, Gong Cheng, and Yuzhong Qu. 2020. SPARQA: skeleton-based semantic parsing for complex questions over knowledge bases. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8952–8959. AAAI Press.

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, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 641–651. Association for Computational Linguistics.

Xingyu Tan, Xiaoyang Wang, Qing Liu, Xiwei Xu, Xin Yuan, and Wenjie Zhang. 2025. Paths-over-graph: Knowledge graph empowered large language model reasoning. In *Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025- 2 May 2025*, pages 3505–3522. ACM.

Gemini Team. 2023. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805.

Shuai Wang and Yinan Yu. 2025. iquest: An iterative question-guided framework for knowledge base question answering. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 15616–15628. Association for Computational Linguistics.

Siwei Wu, Zhongyuan Peng, Xinrun Du, Tuney Zheng, Minghao Liu, Jialong Wu, Jiachen Ma, Yizhi Li, Jian Yang, Wangchunshu Zhou, Qunshu Lin, Junbo Zhao, Zhaoxiang Zhang, Wenhao Huang, Ge Zhang, Chenghua Lin, and Jiaheng Liu. 2024. A comparative study on reasoning patterns of openai's o1 model. *CoRR*, abs/2410.13639.

Guanming Xiong, Haochen Li, and Wen Zhao. 2025. MCTS-KBQA: monte carlo tree search for knowledge base question answering. *CoRR*, abs/2502.13428.

Xiaotong Xu, Yizhao Wang, Yunfei Liu, and Shengyang Li. 2025. SKRAG: A retrieval-augmented generation framework guided by reasoning skeletons over knowledge graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 13983–13994, Suzhou, China. Association for Computational Linguistics.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1321–1331. The Association for Computer Linguistics.

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, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*. The Association for Computer Linguistics.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Yang Wang, Zhiguo Wang, and Bing Xiang. 2023. Decaf: Joint decoding of answers and logical forms for question answering over knowledge bases.

In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Haoyu Zhang, Jingjing Cai, Jianjun Xu, and Ji Wang. 2019. Complex question decomposition for semantic parsing. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4477–4486. Association for Computational Linguistics.

Lingxi Zhang, Jing Zhang, Yanling Wang, Shulin Cao, Xinmei Huang, Cuiping Li, Hong Chen, and Juanzi Li. 2023. FC-KBQA: A fine-to-coarse composition framework for knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 1002–1017. Association for Computational Linguistics.

Qinggang Zhang, Junnan Dong, Hao Chen, Daochen Zha, Zailiang Yu, and Xiao Huang. 2024. Knowgpt: Knowledge graph based prompting for large language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Zhiqiang Zhang, Liqiang Wen, and Wen Zhao. 2025. Rule-kbqa: Rule-guided reasoning for complex knowledge base question answering with large language models. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 8399–8417. Association for Computational Linguistics.

Ruilin Zhao, Feng Zhao, and Hong Zhang. 2025. Correcting on graph: Faithful semantic parsing over knowledge graphs with large language models. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 5364–5376. Association for Computational Linguistics.

A Datasets

We utilize two standard datasets, WebQSP and CWQ. To ensure fairness, we adopt the same training and test splits as in previous studies. The detailed statistics of the datasets are presented in Table 3. Both WebQSP and CWQ are based on Freebase. In our approach, we employ the complete Freebase as the knowledge graph for information retrieval.

Datasets	#Train	#Test	Max #hop
WebQSP	2,826	1,628	2
CWQ	27,639	3,531	4

Table 3: Statistics of datasets.

B BASELINES

- FC-KBQA (Zhang et al., 2023) proposes a coarse-to-fine architecture that decomposes the generation process into three steps: extracting fine-grained components, reorganizing mid-grained pairs, and synthesizing coarse-grained expressions, utilizing KB constraints to resolve the generation of illegal expressions.
- SR+NSM (Yih et al., 2015) adopts an architecture that decouples subgraph retrieval from reasoning, dynamically expanding subgraphs via a dual-encoder retriever (SR) and achieving deep synergy between retrieval and global reasoning through a three-stage training strategy.
- REAREV (Mavromatis and Karypis, 2022) addresses instruction decoding and execution order errors by utilizing a BFS execution module to dynamically filter reasoning steps and an adaptive decoding module to feedback KG information into the initial instructions for iterative optimization.
- UniKGQA (Jiang et al., 2023b) unifies the model architecture and parameter space for retrieval and reasoning, introducing "abstract subgraphs" to alleviate search space pressure and utilizing contrastive learning to unify pretraining tasks.
- StructGPT (Jiang et al., 2023a) proposes an iterative "Interface-Read-Reason" (IRR) framework that treats structured data as a black box, linearizing evidence through specific interfaces to guide the LLM in multi-round logical planning and reasoning.
- DECAF (Yu et al., 2023) combines semantic parsing with direct answering by using a T5 model to simultaneously decode logical forms and answers, enhancing robustness against parsing errors through the weighted fusion of both results.
- Think-on-Graph (ToG) (Sun et al., 2024) treats the LLM as an agent performing iterative beam search on a KG, where the LLM evaluates and prunes search paths in real-time to achieve interpretable reasoning exploration.
- RoG (Luo et al., 2024b) employs a "Plan-Retrieve-Reason" workflow, where the LLM first generates a relational path plan, then extracts specific subgraphs from the KG, and finally guides the LLM to generate faithful answers based on the retrieved paths.
- G-Retriever (He et al., 2024) models subgraph retrieval as a PCST optimization problem for textual graph understanding, combining graph embeddings (soft prompts) generated by a GAT with linearized subgraphs (hard prompts) for LLM reasoning.
- Dual-Reasoning (DualR) (Liu et al., 2025a) simulates a cognitive dual-system by utilizing the LLM's semantic representation to drive a lightweight GNN for explicit reasoning, after which the LLM makes final decisions based on the reasoning chains.
- GNN-RAG (Mavromatis and Karypis, 2025) combines the topological reasoning of GNNs with the understanding capabilities of LLMs, using a GNN to calculate candidate answer probabilities and converting the retrieved shortest reasoning paths into natural language for LLM processing.
- SubgraphRAG (Li et al., 2025b) adopts Directed Distance Encoding (DDE) to capture long-distance structural relationships and utilizes a lightweight MLP to quickly filter subgraphs with adjustable density, achieving efficient contextual reasoning.

- D-RAG (Gao et al., 2025a) introduces differentiable retrieval by transforming discrete subgraph selection into an optimizable form via Gumbel-Softmax and neural prompt modules, enabling end-to-end collaborative optimization of the retriever and generator.

- MCTS-KBQA (Xiong et al., 2025) models semantic parsing as a Monte Carlo Tree Search (MCTS) task, using a step-wise reward mechanism and the UCT algorithm to balance path exploration, thereby overcoming the linear decision-making limitations of LLMs in complex logical queries.

- KBQA-o1 (Luo et al., 2025a) serves as a heuristic agent framework based on MCTS that simulates human reasoning through atomic query tools and incorporates an incremental fine-tuning mechanism to achieve self-evolution in low-resource environments.

- KG-Agent (Jiang et al., 2025) is an autonomous agent framework designed for small models that constructs a multi-functional toolbox and employs programmatic instruction fine-tuning, enabling the LLM to act as a planner that autonomously schedules tools and updates knowledge memory.

- Rule-KBQA (Zhang et al., 2025) is a framework for complex KBQA consisting of an induction phase and a deduction phase. In induction, it constructs a comprehensive rule library by extracting rules from data and generating new ones using a fine-tuned LLM. In deduction, it employs a symbolic agent to retrieve relevant rules and incrementally build logical forms, utilizing the LLM as a discriminator to ensure reasoning accuracy and faithfulness to the knowledge base.

- GCR (Luo et al., 2025b) restricts the LLM's output space during the decoding phase using a prefix tree (KG-Trie), ensuring that generated reasoning paths strictly conform to the KG structure and eliminating hallucinations.

- COG (Zhao et al., 2025) is designed to enhance the faithfulness and accuracy of LLMs in KGQA. This method first employs structured knowledge decoding to enable LLMs to utilize their internal parametric knowledge for filling in intermediate entities when generating logical queries. Subsequently, a knowledge path correction mechanism leverages real triples from the KG to correct hallucinated entities produced by the LLMs and complete missing retrieval paths.

- BYOKG-RAG (Mavromatis et al., 2025) adopts a two-stage collaborative architecture where the LLM generates query primitives (Artifacts), which are then iteratively optimized in conjunction with various specialized retrieval tools such as paths or OpenCypher.

- iQUEST (Wang and Yu, 2025) is an iterative question-guided framework that dynamically generates answerable sub-questions and maintains a clear reasoning trajectory in heterogeneous KGs through a GNN-based forward-looking search mechanism.

- ORT (Liu et al., 2025b) simulates human reverse thinking by initiating a reverse search from target labels at the ontology layer to construct abstract paths, which then guides forward entity retrieval to generate answers.

- KG-R1 (Song et al., 2025) draws inspiration from the O1 model by employing a single agent interacting with a schema-agnostic server and utilizes the GRPO reinforcement learning algorithm for end-to-end optimization, significantly reducing reasoning costs and enhancing cross-graph generalization.

- KnowCoder-A1 (Chen et al., 2025) adopts a multi-stage curriculum reinforcement learning approach, transitioning from cold-start fine-tuning to outcome-supervised GRPO optimization, fostering the model's error recovery and path discovery capabilities through an "easy-to-hard" reward curriculum.

C Experiments

All experiments were conducted on four NVIDIA A6000 GPUs, each with 48GB memory. Due to the high computational cost, each experiment was run only once, and all results reported in this paper correspond to a single run without aggregation or error bars.

Supervised fine-tuning During the SFT stage, we trained the model for one epoch to learn the interaction protocol with the knowledge graph (KG)

system. We set the learning rate to 5×10^{-5} and used 10 warm-up steps.

Reinforcement Learning Phase We utilize the cold-start model as the initial model for training. The branching factor is set to $K = 16$. The batch size is configured at 64, while the mini-batch size is set to 128. The learning rate is fixed at 1×10^{-6} without a warm-up phase. We employ a modified GRPO (Group Relative Policy Optimization) as the optimization objective, ensuring that the rewards for samples within each group are distinct. Additionally, we do not utilize KL divergence and have set the clipping coefficient to 0.2. During inference, we used vLLM as the decoding engine. We set the sampling temperature to 1.0, with top-$p = 1$ and top-$k = -1$, to encourage output diversity. The RL training consisted of 168 update steps. The total training cost was 84 GPU-hours.

Knowledge graph interaction system We implemented an interaction system based on Flask to facilitate communication between the model and the KG. For relationship retrieval at each node, we use Qwen3-Embedding-0.6B to retrieve the top 10 relations most relevant to the current query. This embedding model is used out-of-the-box without further fine-tuning. The Knowledge Graph is deployed on a cluster of four NVIDIA RTX 3090 GPUs (24GB VRAM each), utilizing Virtuoso for database management and 128GB of CPU memory.

D Templates and Prompts

D.1 Prompt

For the trained Qwen2.5 model, we adopted the prompt template shown in Figure 8.

D.2 Retrieval Prompt

For relation retrieval with Qwen3-Embedding, we employed the prompt shown in Figure 9 to achieve strong retrieval performance.

D.3 SPARQL Templates

We list the SPARQL templates used for relation enumeration. Given a topic entity <mid>, these templates retrieve candidate relations under different structural patterns.

Template 1: One-hop outgoing relations

```
SELECT DISTINCT ?relation
WHERE {
  <mid> ?relation ?a .
}
```

Template 2: One-hop incoming relations

```
SELECT DISTINCT ?relation
WHERE {
  ?b ?relation <mid> .
}
```

Template 3: One-hop outgoing value relations

```
SELECT DISTINCT ?relation
WHERE {
  <mid> ?relation ?value .
}
```

Template 4: Two-hop outgoing relation chains

```
SELECT DISTINCT ?relation1 ?relation2
WHERE {
  <mid> ?relation1 ?a1 .
  ?a1 ?relation2 ?a2 .
}
```

Template 5: Two-hop incoming relation chains

```
SELECT DISTINCT ?relation1 ?relation2
WHERE {
  ?b1 ?relation1 <mid> .
  ?b2 ?relation2 ?b1 .
}
```

Template 6: Two-hop relation–value chains

```
SELECT DISTINCT ?relation1 ?relation2
WHERE {
  <mid> ?relation1 ?a .
  ?a ?relation2 ?value .
}
```

E Cold-Start Stage

Although large language models pretrained on large corpora exhibit strong reasoning and tool-use abilities, directly deploying them to interact with structured KG query systems remains challenging. In particular, unconstrained generation often violates the expected interaction protocol and produces non-executable queries.

To address this issue, we introduce a cold-start stage. In this stage, we perform minimal fine-tuning to enable reliable interaction with our KG system. This stage provides a stable foundation for subsequent reinforcement learning.

The cold-start data are constructed from standard logical forms. This choice conflicts with our goal of reducing reliance on supervised logical annotations. Therefore, we deliberately restrict the data scale. We use only a small subset of the training data, approximately $1/30$ of the full set.

In the following, we describe the data preprocessing procedure, the difficulty-aware organization strategy, and the construction of cold-start interaction data.

Prompt template for Reasoning Model

```
You are an intelligent Q&A assistant capable of interacting with a knowledge graph (KG).
By continuously interacting with the KG, you obtain the necessary information to answer
questions and provide users with accurate and effective responses. The KG service is
running normally; if no valid information is returned, it means that your interaction
request contains an error.

For every question asked by the user, you must include your reasoning inside <think>
and </think> tags, and summarize your reasoning process and provide the final answer
inside <answer> and </answer> tags.

When interacting with the KG, you may use <node> and <SPARQL> to communicate with the
KG. The KG system will execute your queries and return corresponding information within
<relation> and <information> tags.

You can output a key node inside <node> </node> tags to request exploration of that
node. The KG will return all relations associated with that node. A key node may be
either the ID of a concrete entity, or an intermediate variable from your SPARQL query.
Returned relations are wrapped in <relation> tags. Within these relations, <node>
indicates the position of the node you queried, serving as either the subject or object
of the relation. ?a or ?b represent connected entities. You may only explore
information through relations returned inside <relation> tags.

You may write a SPARQL query wrapped in <sparql> </sparql> tags to query information
from the knowledge graph. The KG system will execute it and return results within
<information> </information> tags. If your SPARQL query returns no results, you may
attempt a different SPARQL query.

If you believe you have obtained sufficient information to support your answer,
summarize your reasoning and provide the final answer within <answer> </answer> tags,
and enclose the answer in \\boxed{{}}.

Question: {}
```

Figure 8: Prompt template for ours

Prompt template for retrieval

```
Instruct: Given a question and the relationships in the knowledge graph, select the
relationships that are most relevant to the current question.

Query:{}
```

Figure 9: Prompt template for Retrieval

E.1 Data Preprocessing and Difficulty Stratification

Prior work shows that large language models struggle with compositional generalization in complex logical reasoning tasks. In contrast, humans typically learn logical systems in stages, from simple cases to complex ones. Motivated by this observation, we organize training data by increasing reasoning difficulty.

In KGQA, a natural proxy for reasoning complexity is the number of reasoning hops required to answer a question. Although standard logical forms provide hop information, prior studies report that these hops do not always reflect actual execution steps. This discrepancy arises from structural differences in the knowledge graph.

Following this insight, we re-segment the original logical forms. We ensure that each hop corresponds to a concrete reasoning step. Based on the revised hop counts, we partition the dataset into multiple difficulty levels.

In addition, we observe that many samples share identical reasoning patterns. Training on redundant samples yields limited learning signals. To mitigate this issue, we represent each reasoning process as a relation path. We further abstract it into a path composition pattern.

We then cluster and deduplicate samples based on these patterns. We retain only representative samples with unique compositional structures.

Finally, we adopt an incremental data introduction strategy. The model first learns low-complexity

reasoning patterns. It then progressively incorporates more complex compositions. This design helps the model acquire basic reasoning skills before generalizing to challenging multi-hop scenarios. It also significantly reduces the overall training scale.

E.2 Cold-Start Data Construction

To minimize dependence on standard logical forms while ensuring correct KG interaction, we construct a cold-start dataset with only 800 samples. Each sample takes the form of a multi-turn interaction trajectory. Standard reasoning paths guide the construction.

Following prior categorizations of KGQA reasoning paradigms, we treat reasoning as a combination of two basic modes: *progressive* reasoning and *compositional* reasoning.

For progressive reasoning, we decompose a multi-hop logical form into a sequence of gradually expanded sub-forms. For example, a logical form involving relations $a \rightarrow b$ is decomposed into two steps. The first step contains only relation a. The second step contains both a and b. This process simulates step-by-step human reasoning.

For compositional reasoning, we adopt a "separate-then-merge" strategy. Given a logical form involving relations a and b, we first construct two independent logical forms. Each form contains only one relation. We then construct a final logical form that integrates both relations.

By applying these decomposition rules, we derive an ordered reasoning process from each standard logical form. This process serves as the structural backbone of the cold-start data. The construction relies primarily on rule-based procedures. We provide additional details in the appendix.

After determining the logical skeleton of each interaction, we employ a strong closed-source language model to generate natural language reasoning text for each step. To prevent information leakage, we provide only the preceding $n-1$ steps when generating the reasoning text for step n. We describe the prompt design in the appendix.

Each interaction step consists of five components: `<think>`, `<node>`, `<relation>`, `<sparql>`, and `<information>`. The language model generates the `<think>` component. We deterministically construct the remaining components using rule-based SPARQL decomposition.

This approach yields a cold-start dataset that combines structured supervision with natural language reasoning. It avoids exposing the model to complete standard logical forms.

E.3 Training Details

During cold-start supervised fine-tuning, we employ a dedicated loss masking strategy inspired by prior work. A complete KG interaction trajectory includes both model-generated reasoning and KG-returned information. If we compute loss on both parts, the model tends to memorize KG facts.

To avoid this behavior, we mask the loss on KG-returned information. We compute training loss only on tokens generated by the model. This design encourages the model to learn how to construct queries and integrate retrieved evidence. It prevents the model from fitting the KG itself.

As a result, the cold-start stage primarily equips the model with a structured and reliable interaction capability with the KG system.

E.4 Cold-Start Data Scale Ablation

To verify that the primary performance gains come from RL training rather than cold-start supervision, we ablate the cold-start dataset size on CWQ. Table 4 reports CWQ F1 before any RL training (i.e., the cold-start model alone), and the number of RL update steps required to recover the performance of the full 800-sample baseline.

Cold-start Size	F1	% of baseline	Steps
50	20.4	40.6%	$\sim$60
100	24.6	48.9%	—
200	30.7	61.0%	—
400	38.2	75.9%	$\sim$10
800	50.3	100%	—

Table 4: Effect of cold-start data scale on CWQ F1 before RL training, and RL steps required to reach 800-sample performance. "—" indicates convergence within standard training.

Even with only 50 cold-start samples ($\approx 1/240$ of the full training set), the RL phase converges to the 800-sample baseline level within approximately 60 update steps. This rapid recovery confirms that cold-start fine-tuning merely instills interaction protocol knowledge, while the actual reasoning capability emerges from RL exploration.

F Rollout Stage

F.1 Tree-based Rollout

The pseudo-code for the tree expansion process is detailed in Algorithm 1.

Algorithm 1 K-sampling Tree Expansion with KG Deduplication

Require: Dataset $\mathcal{D}$, Policy Model π_θ, KG interface `KGExecute`
Ensure: Search Tree $\mathcal{T}$, Leaf trajectories $\mathcal{L}$

Initialize: Active set $A_1 \leftarrow$ initial questions from $\mathcal{D}$; Search tree $\mathcal{T}$ with A_1; Query cache $Q_{\text{cache}} \leftarrow \emptyset$; State cache $S_{\text{cache}} \leftarrow \emptyset$.
for $t = 1$ **to** T **do**
  break if $A_t = \emptyset$
  // Step 1: Batch Candidate Generation
  $X \leftarrow \{p_i \mid i \in A_t\}$ ▷ Collect path prefixes
  $Y \leftarrow \textsc{GenerateBatch}(\pi_\theta, X, K)$ ▷ Sample K actions per parent
  $C \leftarrow \bigcup_{i \in A_t} \textsc{ParseAction}(y_{i,1:K})$ ▷ Grouped candidate actions
  // Step 2: Knowledge Graph Query Deduplication
  $\mathcal{Q} \leftarrow \{q \mid q \in$ queries in C and $\textsc{Key}(q) \notin Q_{\text{cache}}\}$
  for each unique query $q \in \mathcal{Q}$ **do**
    $r \leftarrow \texttt{KGExecute}(q)$
    $Q_{\text{cache}}[\textsc{Key}(q)] \leftarrow r$ ▷ Cache new results
  end for
  // Step 3: State Transition and Pruning
  $A_{t+1} \leftarrow \emptyset$
  for each candidate $c \in C$ **do**
    $r \leftarrow Q_{\text{cache}}[\textsc{Key}(c)]$
    $s_{child} \leftarrow \textsc{Transition}(s_{parent}, c, r)$
    if $\textsc{StateKey}(s_{child}) \in S_{\text{cache}}$ **then**
      continue ▷ Prune redundant search paths
    end if
    $S_{\text{cache}} \leftarrow S_{\text{cache}} \cup \{\textsc{StateKey}(s_{child})\}$
    Add s_{child} to $\mathcal{T}$
    if s_{child}.*complete* **or** $|p_{child}| \geq L_{\max}$ **then**
      Mark s_{child} as leaf and add to $\mathcal{L}$
    else
      $A_{t+1} \leftarrow A_{t+1} \cup \{s_{child}\}$
    end if
  end for
end for
return $\mathcal{T}, \mathcal{L}$

F.2 Failure Modes

During the rollout stage, we observe several types of failure modes. These failures mainly fall into four categories. (1) The generated action a_t cannot be parsed into an executable command. (2) The node query or SPARQL query is not executable. (3) The query process exceeds the time limit. (4) The query returns an empty result.

Although these failures belong to different error categories, we uniformly retain all failed behaviors during the pruning stage. This design allows the model to observe diverse negative samples during training. As a result, the model can better identify and avoid erroneous behaviors.

It is worth noting that all failure modes ultimately lead to invalid query results. The underlying causes often include incorrect relations in the generated SPARQL query or syntactic errors in the query formulation.

F.3 Filtering Successful Queries

We serialize the query result returned by the knowledge graph, denoted as o_t, into a string representation e_t. In our query system, identical queries always return exactly the same list of entities or relations. Therefore, semantically equivalent queries produce identical result strings.

Based on this property, we apply deduplication to successful query results. Specifically, for SPARQL queries, o_t represents the returned entity list. For node queries, o_t represents the retrieved relation list. By comparing the serialized results o_t, we can identify and filter semantically duplicate successful queries. This process prevents redundant samples from interfering with training.

F.4 Formal Definition of the Reasoning Tree

We model the reasoning process as a rooted tree $\mathcal{T} = (V, E)$, where V denotes the set of nodes and E denotes the set of edges. Each node $v \in V$ corresponds to a reasoning triplet (t, c, o), where t represents the reasoning thought, c is the execution command, and o is the observation returned by the environment.

Given an input question q and a knowledge graph $\mathcal{G}$, we define the construction process of the reasoning tree as follows:

Root Node The root node v_0 corresponds to the initial question q and does not contain any reasoning triplets.

Node Expansion For a node v at depth $t-1$, the interaction history from the root to this node is:

$$H_{t-1} = \big(q, (t_1, c_1, o_1), \ldots, (t_{t-1}, c_{t-1}, o_{t-1})\big).$$

Conditioned on this history, the policy model π_θ samples K candidate state-action pairs:

$$\{(t_{t,i}, c_{t,i})\}_{i=1}^{K} \sim \pi_\theta(\cdot \mid H_{t-1}).$$

Child Node Generation For each action $c_{t,i}$, the environment executes the corresponding query and returns an observation:

$$o_{t,i} = \mathcal{E}(c_{t,i}, \mathcal{G}).$$

This forms a complete reasoning triplet $(t_{t,i}, c_{t,i}, o_{t,i})$. Subsequently, we create a child node $v_{t,i}$ and add an edge $(v, v_{t,i})$ between node v and $v_{t,i}$.

Paths and Trajectories A path from the root node v_0 to any leaf node v_{leaf} defines a complete reasoning trajectory:

$$\tau = \big((t_1, c_1, o_1), \ldots, (t_T, c_T, o_T)\big).$$

The branching factor k controls the exploration width, while the maximum depth T limits the reasoning length. Unlike independent trajectory sampling, the tree structure explicitly captures decision branches. Shared prefixes allow different trajectories to reuse early reasoning steps. This design improves sampling efficiency and supports more granular node-level reward allocation.

During the expansion process at step t, let V_{t-1} denote the set of currently active parent nodes. For each parent node $v \in V_{t-1}$, the model samples k candidate branches based on its history $S_{t-1}(v)$:

$$\{(t_{t,i}, c_{t,i})\}_{i=1}^{k} \sim \pi_\theta(\cdot \mid S_{t-1}(v)).$$

F.5 Environment-Feedback-Based Pruning

The environment executes each action and returns an observation $o_{t,i} = \mathcal{E}(c_{t,i}, \mathcal{G})$, generating K reasoning triplets:

$$\mathcal{T}_v = \{(t_{t,i}, c_{t,i}, o_{t,i})\}_{i=1}^{K}.$$

We serialize each observation into a unique result key:

$$e_{t,i} = \text{String}(o_{t,i}),$$

where lists of entities and relations are normalized and sorted. We then define the set of result keys:

$$E_v = \{e_{t,i}\}_{i=1}^{K}.$$

Deduplication of Non-Empty Observations For branches yielding non-empty observations, we retain only one representative branch for each unique result. We define:

$$E_v^{\text{non-empty}} = \{e_{t,i} \mid e_{t,i} \neq \emptyset\},$$

and the corresponding set of triplets as:

$$\mathcal{T}_v^{\text{non-empty}} = \{(t_{t,i}, c_{t,i}, o_{t,i}) \mid e_{t,i} \neq \emptyset\}.$$

We apply a selection function $\text{Select}(\cdot)$ to choose a representative triplet for each unique result:

$$\mathcal{T}_v^{\text{unique}} = \text{Select}(\mathcal{T}_v^{\text{non-empty}}),$$

such that:

$$\forall \tau_1 \neq \tau_2 \in \mathcal{T}_v^{\text{unique}} : e(\tau_1) \neq e(\tau_2),$$

$$\forall e \in E_v^{\text{non-empty}},\ \exists \tau \in \mathcal{T}_v^{\text{unique}}$$

such that $e(\tau) = e$.

The corresponding child nodes are added to the set of active nodes for the next step, V_t^{active}.

Handling Empty Observations For branches yielding empty observations, we define the set of failed triplets:

$$\mathcal{T}_v^{\text{fail}} = \{(t_{t,i}, c_{t,i}, o_{t,i}) \mid e_{t,i} = \emptyset\}.$$

Paths corresponding to these triplets are treated as terminated failure trajectories and are added to the set of completed trajectories C.

F.6 Necessity of Format Rewards

Recent studies (Li et al., 2025a) suggest that once a model learns output format constraints through SFT in the cold-start stage, format rewards may be unnecessary. However, this assumption does not fully hold in our setting.

Specifically, we fine-tune the model using a relatively small cold-start dataset. In addition, our interaction process involves complex multi-step reasoning and tool usage. Under these conditions, the model struggles to consistently maintain correct output formats throughout training.

We observe that as multi-round training proceeds, the model gradually forgets predefined format constraints. This behavior degrades system stability. Based on this observation, we retain the format reward as a component of the final reward function. This reward continuously reinforces adherence to the required output format.

This decision is grounded in empirical observations. For example, the model may insert arbitrary numbers of newline tokens at the end of each SPARQL line. Although such queries remain executable, they significantly increase the number of generated tokens. The format reward suppresses this behavior and encourages more compact and standardized outputs.

F.7 Erroneous Behaviors in Distance Rewards

In the design of distance rewards, we categorize model behaviors into three types. First, erroneous behaviors include syntax errors, hallucinated variables, and repeated queries. These behaviors provide no useful information gain. We therefore treat them as negative behaviors and assign penalties.

Second, no-progress behaviors are syntactically valid but do not reduce the distance to the target. We treat these behaviors as neutral.

Third, progress behaviors effectively reduce the distance between the current state and the target. These behaviors align with our objectives and receive positive rewards. This categorization allows the distance reward to clearly differentiate feedback types during training. As a result, the model learns more effective query strategies.

F.8 Distance Reward for Aggregation Queries

Aggregation queries require special handling in our distance-guided progress reward. They include counting queries (e.g., "How many films did Spielberg direct?") and superlative-constrained queries (e.g., "Which film had the highest box office?").

For **counting queries**, the final answer is a scalar rather than an entity. We decompose the SPARQL query by removing the `COUNT` operator, exposing the underlying entity retrieval sub-problem. The progress reward then evaluates whether the query successfully retrieves the relevant entity set, while the outcome reward measures whether the count derived from this set matches the ground truth.

For **constrained queries** (e.g., superlatives, ordinals), the answer entity exists as a node in the KG. We compute topological distance normally from the current query result to the answer entity. A query that retrieves the full candidate set without the constraint is assigned $d_{\text{post}} = 1$ (one step from the answer), while a query that retrieves candidates and applies the correct constraint (e.g., `ORDER BY ... LIMIT 1`) is assigned $d_{\text{post}} = 0$. This staged design ensures the agent explicitly optimizes for global constraint satisfaction rather than stopping after reaching any single answer entity.

F.9 Reward Function

The specific reward mechanism is implemented as shown in Algorithm 2.

Algorithm 2 Trajectory Step Reward Computation

Require: Path $\mathcal{P} = \{n_1, \ldots, n_L\}$, Search Tree $\mathcal{T}$, Ground Truth $\mathcal{G}$, Terminal score F_1^{final}
Ensure: Step-wise rewards $\mathcal{R} = \{r_1, \ldots, r_L\}$

Initialize: $\mathcal{R} \leftarrow \emptyset$; $\mathcal{V}_q \leftarrow \emptyset$ ▷ Visited queries; $s_{prev} \leftarrow 0.0$ ▷ Previous best score
$\mathcal{S}_{map} \leftarrow \{\text{MentionEntity} : \textsc{Dist}(\text{MentionEntity}, \mathcal{G})\}$
for $i = 1$ **to** L **do**
  $n_i \leftarrow \mathcal{T}[n_i]$; $o_i, type \leftarrow \textsc{ParseNode}(n_i)$
  // 1) Format Reward
  $R_{fmt} \leftarrow 1$ **if** $\textsc{IsValid}(o_i)$ **else** -1
  // 2) Progress Reward
  if *type* $=$ NODE **then**
    $e \leftarrow \textsc{ExtractEntity}(o_i)$
    if $e \in \mathcal{S}_{map}$ **then** $\{R_{prog} \leftarrow 0;\ s_{curr} \leftarrow \mathcal{S}_{map}[e]\}$ **else** $\{R_{prog} \leftarrow -1;\ s_{curr} \leftarrow 0\}$
  else ▷ *type* $=$ SPARQL
    $q \leftarrow \textsc{ExtractQuery}(o_i)$
    if $\textsc{Invalid}(q)$ **or** $q \in \mathcal{V}_q$ **then**
      $R_{prog} \leftarrow -1;\ s_{curr} \leftarrow 0$
    else
      $\mathcal{V}_q \leftarrow \mathcal{V}_q \cup \{q\}$; $res \leftarrow \textsc{Execute}(q)$
      $s_{curr} \leftarrow \textsc{Dist}(res, \mathcal{G})$; $\textsc{Update}(\mathcal{S}_{map}, q, s_{curr})$
      $R_{prog} \leftarrow 1$ **if** $s_{curr} > s_{prev}$ **else** 0
    end if
  end if
  // 3) Outcome & Total Reward
  $R_{out} \leftarrow F_1^{final}$ **if** $(R_{fmt} \geq 0 \wedge R_{prog} \geq 0)$ **else** 0
  $r_i \leftarrow w_1 \cdot R_{fmt} + w_2 \cdot R_{prog} + w_3 \cdot R_{out}$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{r_i\}$
  // 4) State Update
  if $R_{prog} \geq 0$ **and** *type* $\in \{\text{NODE}, \text{SPARQL}\}$ **then**
    $s_{prev} \leftarrow s_{curr}$
  end if
end for
return $\mathcal{R}$

G Training Stage

G.1 Policy Optimization

To optimize the agent's decision-making process within the multi-turn interaction framework, we employ Group Relative Policy Optimization (GRPO). For each state S_k, we sample G candidate action branches $\{A_k^{(i)}\}_{i=1}^{G}$ from the behavior policy $\pi_{\theta_{\text{old}}}$. The optimization objective is formulated as follows:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\Big(\mathcal{L}_{\text{CLIP}}^{(i)}(\theta) - \beta\mathbb{D}_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})\Big)\Bigg]$$

where $\mathcal{L}_{\text{CLIP}}^{(i)}(\theta)$ represents the step-level surrogate loss for the i-th action:

$$\mathcal{L}_{\text{CLIP}}^{(i)}(\theta) = \frac{1}{|A_k^{(i)}|}\sum_{m=1}^{|A_k^{(i)}|}\min\Big(\rho_{i,m}\hat{A}_{k,i}, \text{clip}(\rho_{i,m}, 1 \pm \epsilon)\hat{A}_{k,i}\Big)$$

In the above expressions, $\rho_{i,m}$ denotes the token-level importance ratio $\frac{\pi_\theta(a_{i,m}|S_k, a_{i,<m})}{\pi_{\theta_{\text{old}}}(a_{i,m}|S_k, a_{i,<m})}$. The advantage $\hat{A}_{k,i}$ is computed by normalizing the rewards within the group to provide a stable training signal:

$$\hat{A}_{k,i} = \frac{r_k^{(i)} - \text{mean}(\{r_k^{(j)}\}_{j=1}^{G})}{\text{std}(\{r_k^{(j)}\}_{j=1}^{G}) + \delta}$$

where $r_k^{(i)}$ is the step-level reward assigned to action $A_k^{(i)}$, and δ is a small constant used for numerical stability. The KL divergence term $\mathbb{D}_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})$ ensures that the optimized policy does not deviate excessively from the reference model π_{ref}.

H Additional Experiments

H.1 Zero-Shot Transfer to Wikidata

To evaluate the cross-KG generalizability of our framework, we directly apply the Freebase-trained model to Wikidata without any retraining or fine-tuning. We compare against Think-on-Graph (ToG) (Sun et al., 2024), which supports inference on arbitrary KGs through iterative beam search. Both methods are given equivalent access to the target KG at inference time.

Method	WebQSP (F1)	CWQ (F1)
ToG w/ Wikidata	68.6	54.9
Ours w/ Wikidata	**75.7**	**65.3**
ToG w/ Freebase	76.2	58.8
Ours w/ Freebase	**84.1**	**71.8**

Table 5: Zero-shot transfer results to Wikidata. All models are trained on Freebase only. The Wikidata rows show direct transfer without any retraining or fine-tuning.

Our Freebase-trained model achieves +7.1 F1 on WebQSP and +10.4 F1 on CWQ over ToG under this zero-shot setting. The margin widens on CWQ, which is consistent with the advantage of logical-form interaction over graph traversal for multi-hop queries. These results suggest that the interaction protocol and reasoning strategy learned through RL on Freebase transfer effectively to a structurally different KG, even without schema adaptation.

H.2 Reward Weight Sensitivity Analysis

The reward function $R = w_1 \cdot R_{\text{format}} + w_2 \cdot R_{\text{progress}} + w_3 \cdot R_{\text{outcome}}$ combines three components. We fix $w_1 = 0.1$ and perform a grid search over w_2 and w_3 on the CWQ medium subset.

The configuration $w_2 = 0.6$, $w_3 = 0.3$ (i.e., $w_2 : w_3 \approx 2 : 1$) achieves the highest F1. Performance is robust within $w_2 \in [0.45, 0.60]$, and removing process rewards entirely (Outcome Dominated, $w_2 = 0$) causes the largest drop (-3.8 F1), corroborating the RQ3 findings in Section 5.3.

Configuration	w_1	w_2	w_3	CWQ F1
Outcome Dominated	0.1	0.0	0.9	66.4
High Outcome	0.1	0.3	0.6	68.9
Equal	0.1	0.45	0.45	69.1
High Process	0.1	0.6	0.3	**70.2**
Process Dominated	0.1	0.9	0.0	69.5

Table 6: Reward weight sensitivity on the CWQ medium subset. $w_1 = 0.1$ is fixed throughout.

H.3 Branching Factor K Ablation

The branching factor K controls exploration width in tree-based rollout. We ablate $K \in \{4, 8, 16, 32\}$ on CWQ, reporting F1, effective branch count after observation-equivalence pruning, average pruning rate, and normalized rollout time.

K	Eff. Branches	Pruning Rate	F1	Rel. Rollout
4	∼3.8	∼6%	60.7	0.39×
8	∼5.0	∼38%	64.2	0.61×
16	∼7.7	∼52%	**68.5**	1.00×
32	∼9.0	∼72%	68.4	2.34×

Table 7: Branching factor ablation on CWQ. "Eff. Branches" is the average number of branches retained after pruning. "Rel. Rollout" is normalized to $K = 16$.

$K = 16$ achieves the best F1 at $1.00\times$ rollout cost. Increasing to $K = 32$ raises rollout cost by 83% while gaining only marginal effective branches ($7.7 \rightarrow 9.0$), confirming diminishing returns. At $K \leq 8$, the search space is too narrow to leverage tree-based exploration, and F1 drops by 4.3–7.8 points.