Rule-KBQA: Rule-Guided Reasoning for Complex Knowledge Base Question Answering with Large Language Models

Zhiqiang Zhang[1], **Liqiang Wen**[1], **Wen Zhao**[1],
[1]Peking University, Beijing, China
zhangzhiqiang@stu.pku.edu.cn, {wenlq,zhaowen}@pku.edu.cn

Abstract

Knowledge base question answering (KBQA) is recognized as a challenging task, especially when parsing complex questions into executable logical forms. Traditional semantic parsing (SP)-based approaches exhibit inconsistent performance in handling various complex questions. As large language models (LLMs) have exhibited exceptional reasoning ability and language comprehension, recent studies have employed LLMs for semantic parsing to directly generate logical forms that can be executed on knowledge bases (KBs) to achieve the desired results. However, these methods of relying exclusively on LLMs to ensure grammaticality, faithfulness, and controllability may diminish their effectiveness due to hallucinations in the reasoning process. In this paper, we introduce Rule-KBQA, a framework that employs learned rules to guide the generation of logical forms. The proposed method contains two phases, an induction phase and a deduction phase. In the induction phase, we initially extract rules from the existing data and then employ the Rule-Following Fine-Tuned (RFFT) LLM to generate additional rules, ultimately constructing a comprehensive rule library. In the deduction phase, a symbolic agent, guided by learned rules, explores the environment KB to incrementally construct executable logical forms. Meanwhile, we leverage the discriminative capability of LLMs to evaluate the plausibility of candidate decisions. Extensive experiments indicate that our method achieves competitive results on standard KBQA datasets, clearly demonstrating its effectiveness.

1 Introduction

Knowledge base question answering (KBQA) is a crucial task aimed at accurately interpreting and answering user inquiries using knowledge bases (KBs) with extensive structured information, such as Freebase (Bollacker et al., 2008) and Wikidata (Tanon et al., 2016). Its wide-ranging application potential across multiple fields has made it a focal point of both academic research and industrial innovation. An important direction in KBQA is parsing natural language questions as logical forms (e.g., SPARQL), which can be directly executed on the KG to yield answers.

Answering complex questions requires more information and even functional operations, such as comparison, aggregation, and sorting. Semantic parsing (SP) is regarded as an effective strategy for understanding the compositional semantics of complex questions, which converts an unstructured question into its structured representation. Some methods (Reddy et al., 2014; Hu et al., 2018a,b) convert questions into semantic graphs using a parser, which are subsequently transformed into query graphs based on predefined rules. Free from intermediate semantic graphs, in (Das et al., 2021; Ye et al., 2022; Yu et al., 2023), the executable logical forms can be directly generated through pre-trained generative models to enable efficient querying of structured knowledge bases.

The emergence of large language models (LLMs), such as GPT-4 (Achiam et al., 2023) and Llama (Touvron et al., 2023), has led to significant advancements in various natural language processing (NLP) tasks, driven by their exceptional reasoning capability and language comprehension. Therefore, leveraging LLMs to enhance KBQA systems is an intuitive and promising approach. These methods (Li et al., 2023; Luo et al., 2023; Gu et al., 2023; Sun et al., 2024; Xiong et al., 2024) integrate LLMs with KBs, where KBs provide accurate and comprehensive factual information in a triple format while LLMs offer advanced natural language processing capabilities, thereby setting new benchmarks in the KBQA domain.

Despite these advancements, there are still several challenges to be addressed.

Complex questions handling. Complex question answering often involves performing various reasoning operations, such as logical, comparative, and quantitative reasoning, to extract the relevant answers from KBs. Consequently, the generated logical forms must integrate more complicated syntax, often involving constructs like `UNION`, `FILTER`, aggregation (e.g., `MAX`, `COUNT`) functions, and even nested queries. The complexity of logical forms results in huge search space, making it challenging to identify the correct logical form from a vast number of candidates.

Inconsistent performance across various questions. The traditional SP-based approaches are typically to train a single model to handle all complex questions. However, complex questions may exhibit diversity due to their inherent characteristics. For instance, in the ComplexWebQuestions dataset (Talmor and Berant, 2018), samples are categorized into four distinct types, with an uneven distribution of data across these types. Furthermore, the difficulty levels of different questions may vary. Therefore, such models often struggle to find a global optimum in the parameter space that fits all samples, leading to inconsistent performance in answering different complex questions.

Overburdened large language models. While the advent of LLMs has opened new avenues for enhancing KBQA systems, most existing approaches overly shift the burden of performance improvement onto the LLMs. Directly using LLMs to generate logical forms may be suboptimal (Gu et al., 2023), as hallucination can undermine both grammaticality (i.e., compliance with SPARQL query syntax) and faithfulness (i.e., executability within the knowledge base). Additionally, autoregressive generation with LLMs lacks fine-grained control over reasoning.

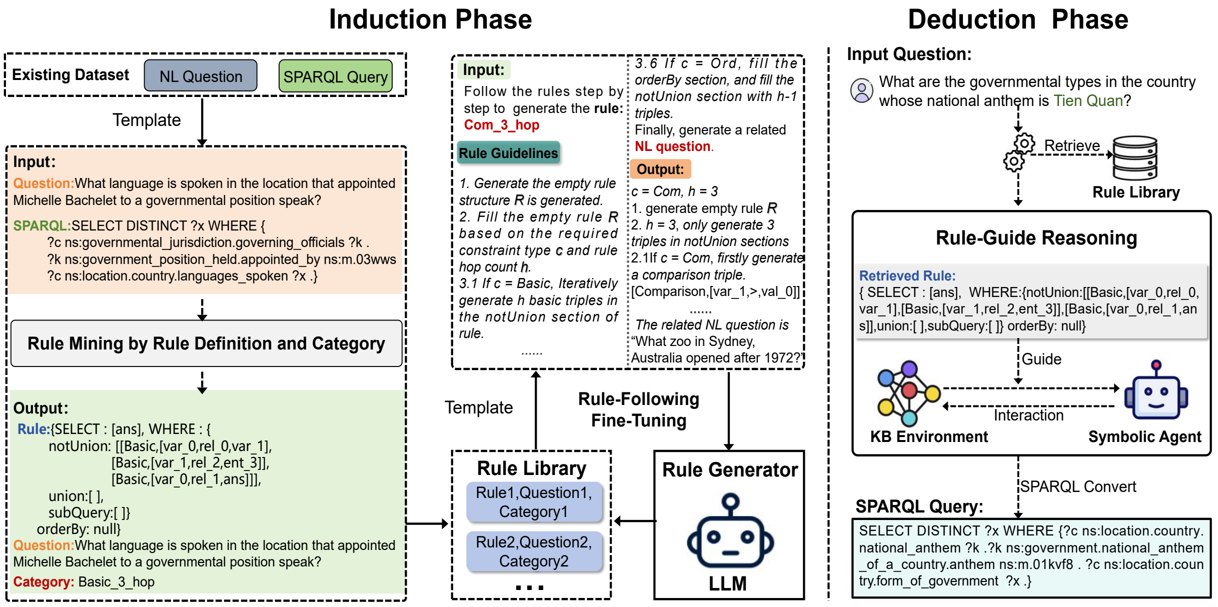
To tackle these challenges, we propose the Rule-KBQA framework, which consists of an induction phase and a deduction phase. Our approach leverages the rules acquired during the induction phase to accurately guide the generation of logical forms in the deduction phase. Specifically, in the induction phase, we formulate rules to describe diverse logical forms, as regardless of the complexity of their syntax, they can be decomposed into entities, variables, values, relationships, and other built-in properties. Subsequently, the rules can be extracted from extant question-SPARQL paired datasets. Furthermore, to enable LLMs to accurately generate new rules, we employ the Rule-Following Fine-Tuning (RFFT) technique (Hu et al., 2024) that explicitly incorporates rule generation guidelines into the input, forcing LLMs to reference and follow the necessary rules step by step during the reasoning process. Ultimately, these rules constitute a comprehensive rule library that can guide the parsing process for various questions, thereby ensuring performance consistency. In the deductive phase, a symbolic agent, guided by the learned rules, interacts with the KB environment to propose candidate decisions for the generation of logical forms. This process guarantees both grammaticality and faithfulness of the generated logical forms. Due to the huge search space, the agent is designed to search in the environment and incrementally generate the logical forms. Meanwhile, we utilize the discriminative capability of LLMs to assess the plausibility of each candidate decision, rather than relying on their generative capability. This reduces the burden on the LLMs and enhances the controllability of the generative process.

We validate the efficacy of the proposed framework through comprehensive experiments on widely-used benchmark datasets. Experimental results demonstrate that our method achieves competitive performance compared to state-of-the-art approaches for complex KBQA.

2 Related Work

Our KBQA method is closely related to the studies on Semantic Parsing and Large Language Models.

Semantic Parsing (SP) for KBQA. SP-based methods initially convert questions into logical forms, subsequent to which they execute queries against the knowledge base to extract the corresponding answers. Early works (Yih et al., 2015; Hu et al., 2018a,b) typically engage in semantic parsing by generating query graphs, which are analogous to subgraphs within the knowledge base and can be directly mapped to logical forms. Lan and Jiang (2020a) integrate constraints within the query graph, thereby efficiently reducing the search space and enabling a more adaptable generation of query graphs. To avoid producing noisy query graphs, Chen et al. (2021b) optimize the process of candidate query generation by utilizing predictions of query structure. As generative models gain prominence, the executable logical forms can be directly generated via them, thereby facilitating efficient querying within knowledge bases. Several studies (Das et al., 2021; Ye et al., 2022; Yu et al., 2023) initially extract key components like entities, relations, subgraphs, or text from the knowledge



Input:

Question:What language is spoken in the location that appointed Michelle Bachelet to a governmental position speak?

```
SPARQL:SELECT DISTINCT ?x WHERE {
        ?c ns:governmental_jurisdiction.governing_officials ?k .
        ?k ns:government_position_held.appointed_by ns:m.03wws
        ?c ns:location.country.languages_spoken ?x .}
```

Output:

```
Rule:{SELECT : [ans], WHERE : {
        notUnion: [[Basic,[var_0,rel_0,var_1],
                    [Basic,[var_1,rel_2,ent_3]],
                    [Basic,[var_0,rel_1,ans]]],
        union:[ ],
        subQuery:[ ]}
      orderBy: null}
```

Question:What language is spoken in the location that appointed Michelle Bachelet to a governmental position speak?

Category: Basic_3_hop

Input:

Follow the rules step by step to generate the **rule:** Com_3_hop

Rule Guidelines

1. Generate the empty rule structure R is generated.
2. Fill the empty rule R based on the required constraint type c and rule hop count h.
3.1 If c = Basic, Iteratively generate h basic triples in the notUnion section of rule.
......
3.6 If c = Ord, fill the orderBy section, and fill the notUnion section with h-1 triples.
Finally, generate a related **NL question**.

Output:

c = Com, h = 3
1. generate empty rule *R*
2. *h = 3, only generate 3 triples in notUnion sections*
2.1If *c = Com, firstly generate a comparison triple.*
[Comparison,[var_1,>,val_0]]
......
The related NL question is "What zoo in Sydney, Australia opened after 1972?"

Input Question:

What are the governmental types in the country whose national anthem is Tien Quan?

Retrieved Rule:

```
{ SELECT : [ans],  WHERE:{notUnion:[[Basic,[var_0,rel_0,var_1],[Basic,[var_1,rel_2,ent_3]],[Basic,[var_0,rel_1,ans]],union:[ ],subQuery:[ ]} orderBy: null}
```

SPARQL Query:

```
SELECT DISTINCT ?x WHERE {?c ns:location.country.national_anthem ?k .?k ns:government.national_anthem_of_a_country.anthem ns:m.01kvf8 . ?c ns:location.country.form_of_government  ?x .}
```

Figure 1: Overall architecture of our proposed method.

base, using them as supplementary data, before generating executable logical forms directly via pre-trained language models such as T5. Nevertheless, this method does not guarantee uniform efficacy of the model across various questions, and it exhibits a deficiency in the transparency of the reasoning process.

Large Language Models (LLMs) for KBQA. Various methods have been proposed to unify KBs and LLMs to solve the KBQA task, where KBs provide accurate and comprehensive factual information in a triple format while LLMs offer advanced natural language processing capabilities. Some methods (Li et al., 2023; Luo et al., 2023) directly treat LLMs as semantic parsers that generate an initial logical form corresponding to a given question, followed by retrieving and replacing entities and relations from the KB. In addition, recent approaches conceptualize LLMs as agents that engage in multi-turn interactions with a KB environment to derive answers. Within this framework, Sun et al. (2024) interactively explore relational paths in KBs step by step and performs reasoning based on the retrieved paths. Xiong et al. (2024) propose an iterative, dialogue-based problem-solving process, which enables semantic parsing and SPARQL generation by developing three generic APIs for KB interaction. Nevertheless, relying solely on LLMs to ensure grammaticality, faithfulness, and controllability may affect models' effectiveness due to hallucinations in the reasoning process. Therefore, Gu et al. (2023) employ a symbolic agent to explore the KB environment to propose valid plans and the LLM only focuses on evaluating the plausibility of the proposed plans. However, the agent struggles to formulate effective plans when confronted with complex questions that require various reasoning operations.

3 Preliminaries

Logical Form. A logical form is a structured representation of a natural language question. For example, SPARQL usually serves as a logical form to represent complex questions, which expresses relationships between entities in a knowledge base and utilizes constructs like `SELECT`, `FILTER`, and `ORDER BY` to formalize the query structure.

Rule Definition. Since the logical forms (eg., SPARQL) can be represented as query graphs, the rules describing them can similarly be formulated as directed acyclic graphs, denoted by $G_r = (V_r, E_r)$. where V_r is the set of labeled vertices $v \in \{\text{"Ans"}, \text{"Var"}, \text{"Ent"}, \text{"Val"}\}$; $E_r = \{e \mid e = \langle v, v' \rangle, v, v' \in V_r\}$ represents the set of directed edges $e \in \{\text{"Rel"}, \text{"Ord"}, \text{"Cmp"}, \text{"Agg"}\}$. The more details are presented in Appendix A.

Task Formulation. This study explores a semantic parsing (SP) approach for KBQA. A KB can be formally represented as $K = (E, R)$, where E is the set of entities and R is the set of relations. Given a question Q and a KB K, our goal is to generate an executable SPARQL query S corresponding to the question Q under the guidance of the learned rule R. Based on the above definition, this task can be formalized as $p(S|Q, R, K)$.

4 Methodology

4.1 Framework Overview

Figure 1 presents the overall architecture of our approach, comprising two key phases: the induction phase and the deduction phase. In the induction phase, we initially mine rules from the extant datasets based on rule definitions and categorize them accordingly. Subsequently, we employ the RFFT technique to enable LLMs to generate new rules precisely. Ultimately, a comprehensive rule library can be constructed to provide strong support for the deduction phase. In the deduction phase, a symbolic agent explores the KB environment under the guidance of learned rules to incrementally generate executable logical forms. Throughout this process, LLMs are employed to assess the validity of candidate decisions at each step.

4.2 Induction Phase

To establish a comprehensive rule library that guides semantic parsing for various complex questions, we initially mine rules from extant datasets and subsequently utilize an LLM-based rule generator to produce additional rules.

4.2.1 Rule Mining

Based on the rule definition, we can directly extract rules from existing question-SPARQL pairs datasets. As shown in Figure 1, the rule can reflect the topology of the SPARQL query while its vertices and edges are slots of categories without instances. Moreover, the rule retains the syntax construction of the SPARQL query. Each SPARQL query corresponds to a unique rule, while a single rule can map to multiple SPARQL queries. Therefore, the rules can efficiently guide the generation of various complex logical forms.

In addition, we conduct a detailed classification of these rules to construct a systematic and hierarchical rule library. The classification criteria are primarily based on the difficulty level of the rule and the type of constraints involved. We define the difficulty level as the count of edges in the rule. Drawing on the structure of the rules, along with the classes of their vertices and edges, we define six types of constraints involved in rules, including "Basic", "Bridge", "Union", "Comparison", "Aggregation", and "Ordinal". Further details are listed in Appendix B. Therefore, the category of a rule can be collectively represented by its difficulty level and type of constraint. In Figure 1, a rule example is presented, with its category labeled as "Basic_3_hop", indicating that it contains three triplets and the constraint type is "Basic".

4.2.2 LLM-based Rule Generator

Relying exclusively on existing data to extract rules is inadequate for constructing a comprehensive and diverse rule library. To address this limitation, we develop an LLM-based rule generator to produce new rules. We utilize the RFFT technology (Hu et al., 2024) to fine-tune LLMs, enhancing the reasonableness and accuracy of the generated rules. As shown in Figure 1, we construct the training data using the rules extracted from existing datasets. Based on the definitions of rule categories mentioned above, we provide a detailed description of the rule generation guidelines in the input. These guidelines can be used to direct the generation of various category rules. We then fine-tune the LLMs incrementally to adhere to the rule generation guidelines. Specifically, the LLMs must explicitly recite the guidelines used at each step and update the intermediate variables after each application. More details of the fine-tuning process are described in Appendix C. By employing the rule generator, we can generate a diverse set of rules to cover various complex logical forms.

4.3 Deduction Phase

To accurately parse complex questions into executable logical forms (eg., SPARQL query), we employ a symbolic agent to perform step-by-step reasoning in the KB environment guided by the retrieved relevant rules. Throughout the rule-guided reasoning process, we also utilize the discriminative capability of LLMs to evaluate the plausibility of each candidate decision at every step.

4.3.1 Rule Retrieval

After constructing a rule library encompassing various categories of rules and their corresponding natural language questions, it becomes crucial to accurately retrieve the relevant rules. Drawing inspiration from (Gao et al., 2024), we achieve this by utilizing information from both the questions and the rules. For the question similarity retrieval, we first mitigate the impact of domain-specific information by replacing topic entities and values in all questions with a mask token. Subsequently, we use a pre-trained language model to embed the

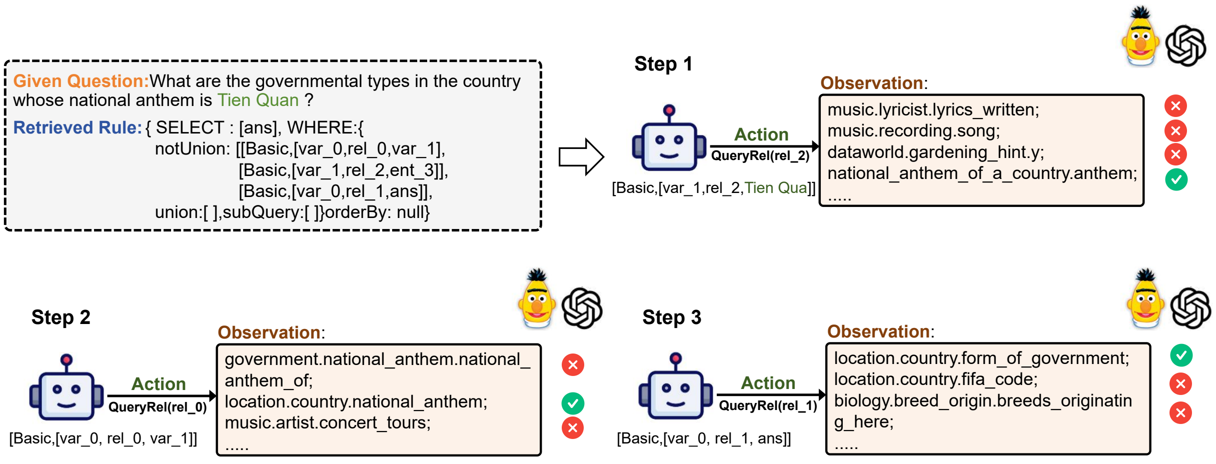

Given Question: What are the governmental types in the country whose national anthem is Tien Quan ?
Retrieved Rule: { SELECT : [ans], WHERE:{ notUnion: [[Basic,[var_0,rel_0,var_1], [Basic,[var_1,rel_2,ent_3]], [Basic,[var_0,rel_1,ans]], union:[ ],subQuery:[ ]}orderBy: null}
Step 1
Action
QueryRel(rel_2)
[Basic,[var_1,rel_2,Tien Qua]]
Observation:
music.lyricist.lyrics_written;
music.recording.song;
dataworld.gardening_hint.y;
national_anthem_of_a_country.anthem;
.....
Step 2
Action
QueryRel(rel_0)
[Basic,[var_0, rel_0, var_1]]
Observation:
government.national_anthem.national_anthem_of;
location.country.national_anthem;
music.artist.concert_tours;
.....
Step 3
Action
QueryRel(rel_1)
[Basic,[var_0, rel_1, ans]]
Observation:
location.country.form_of_government;
location.country.fifa_code;
biology.breed_origin.breeds_originating_here;
.....

Figure 2: An example of the rule-guided reasoning.

masked example questions in the rule library and the masked target question. After that, we rank the candidates by calculating the Euclidean distance between the embeddings of the example questions and the target question. For the rule similarity retrieval, we employ a preliminary model to generate an approximate rule corresponding to the target question. Then we calculate the Jaccard similarity between candidate rules and the predicted rule as their rule similarity. Finally, the retrieval criterion prioritizes the ranked candidates based on question similarity, provided that the rule similarity exceeds a predefined threshold θ. In this way, we enable a more precise retrieval of the rule corresponding to the target question.

4.3.2 Rule-Guided Reasoning

During the rule-guided reasoning process, the symbolic agent explores the KB environment to make decisions that generate valid logical forms, while LLMs are used to assess the validity of each candidate decision at each step. Figure 2 presents an illustrative example of the reasoning process.

To handle the large search space, the agent casts the logical form generation as a step-wise decision-making problem. Specifically, the agent usually initiates reasoning from the topic entity of the given question and selects relevant rule components containing entity-type vertices (e.g., "[Basic, [var_1, rel_2, Tien Quan]]") for guidance. The entity can be obtained using off-the-shelf entity linkers (Li et al., 2020). At step t, the agent, following the relevant rule, takes actions to interact with the KB environment, generating a series of observations that serve as candidate decisions. These actions refer to retrieving relevant relations by executing SPARQL queries on the KB. Consider the example in Figure 2 at step 1, guided by the rule "[Basic, [var_1, rel_2, Tien Quan]]", the agent executes a SPARQL query on the KB, specifically "*SELECT ?rel_2 WHERE {var_1, rel_2, ns:m.01kvf8.}*", to find all relevant relations as candidate decisions. Subsequently, the LLM distinguishes the most suitable relation for the current step (ie., "national_anthem_of_a_country.anthem"). At step 2, based on the previous reasoning result and the corresponding rule "[Basic, [var_0, rel_0, var_1]]", the agent takes an action to execute a SPARQL query "*SELECT ?rel_0 WHERE {var_1, ns:national_anthem_of_a_country.anthem, ns:m.01kvf8. var_0, rel_0, var_1.}*", and the KB environment returns the relevant relations. The executable logical form can be incrementally generated throughout the reasoning process, guided by the relevant rule.

Inspired by (Gu et al., 2023), we leverage the discriminative capability of LLMs to evaluate the plausibility of candidate decisions proposed by the agent, thereby assisting its reasoning process. The evaluation process can be understood as semantic matching performed by LLMs in combination with contextual information. Specifically, we first construct a prompt text.

$$\text{Prompt} = \{\text{Inst}, R, Q\} \tag{1}$$

where Inst is the evaluation instruction description; R denotes the relevant rule; Q is the given question. At step t, we let the LLM evaluate candidate decisions C_t based on the Prompt and the interaction history H_t to select the most suitable relation r_t. This procedure is scheduled as follows:

$$r_t = \text{LLM}(\{\text{Prompt}, H_t, C_t\}) \tag{2}$$

The details are described in Appendix D. In this way, we safeguard the LLM from having to handle a large search space in order to generate valid logical forms.

5 Experiment

5.1 Dataset

To evaluate the effectiveness of the proposed method, we conducted experiments on three public datasets, including WebQuestionsSP (WebQSP) (Yih et al., 2016), ComplexWebQuestions (CWQ)(Talmor and Berant, 2018), and KQA Pro (Cao et al., 2022a). Table 1 shows the statistics of the three datasets. We give a detailed description of each dataset in Appendix E.

Table 1: Statistics of the experiment datasets.

Dataset	Train	Dev	Test	Program Type
WebQSP	3,098	-	1,639	SPARQL
CWQ	27,639	3,519	3,531	SPARQL
KQA Pro	94,376	11,797	900	SPARQL

5.2 Baselines

Our baselines encompass a wide coverage of related models, which can comprehensively evaluate our approach. These methods can be divided into three categories: (1) SP-based methods, including QGG (Lan and Jiang, 2020b), ReTraCk (Chen et al., 2021a), BART-SPARQL(Cao et al., 2022a), RnG-KBQA (Ye et al., 2022), Program Trans (Cao et al., 2022b), DecAF (Yu et al., 2023). (2) Prompting methods with GPT-4, including IO (Brown et al., 2020), CoT (Wei et al., 2022), SC (Wang et al., 2023b). (3) LLMs+KBs methods, including KB-BINDER (Li et al., 2023), StructGPT (Jiang et al., 2023), Pangu (Gu et al., 2023), KD-CoT (Wang et al., 2023a), ToG (Sun et al., 2024), Inter-KBQA (Xiong et al., 2024). The details of each baseline are described in Appendix F.

5.3 Evaluation Metrics

Following previous works, we use F1 and Hits@1 as the evaluation metrics. F1 considers the coverage of all answers, which balances the precision and recall of the predicted answers. Hits@1 typically indicates the accuracy of the top one among predicted answer entities. It's worth noting that our method returns unordered answers. Therefore, we randomly select one answer per question as the top-ranked answer and then calculate the average Hits@1 result by repeating this process 100 times.

5.4 Implementation Details

In the induction phase, we use the open-source Llama-3-8B as the LLM-based generator. By extracting rules from existing datasets and generating additional ones using a rule generator, we have built a rule library comprising 1056 rules, which comprehensively covers the various questions in the current test dataset. In the deduction phase, our study invokes the OpenAI GPT4-Turbo API to evaluate candidate decisions at each step. For rule retrieval, we first take the n-gram matching to replace entities with "<e>" and values with "<v>". Then, we embed the masked questions using all-mpnet-base-v2 (Song et al., 2020) to calculate their similarities. Simultaneously, for rule similarity, we utilize HGNet (Chen et al., 2023) as the preliminary model to predict the approximate rule. Moreover, the rule similarity threshold θ is set as 0.85. For executing SPARQL queries during the reasoning process, Virtuoso[1] serves as the underlying graph query engine.

Table 2: Experimental results on WebQSP and CWQ. "-" indicating that no results are reported in the original papers. Bold font denotes the best performance.

Model		WebQSP		CWQ
		F1	Hits@1	Hits@1
SP-based	QGG	72.2	71.9	44.1
	ReTraCk	74.7	74.6	-
	RnG-KBQA	75.6	-	-
	Program Trans	76.5	74.6	58.1
	DeCAF	78.8	82.1	-
Pompting	CoT w/GPT-4	-	67.3	46.0
LLMs+KGs	KB-BINDER	74.4	-	-
	StructGPT	63.7	72.6	54.3
	Pangu w/Codex	68.3	-	-
	KD-CoT	52.5	68.6	55.7
	ToG w/GPT-4	-	82.6	69.5
	Ours	**81.9**	**84.1**	**73.5**

5.5 Main Results

We present the experimental results for the WebQSP, CWQ, and KQA Pro datasets in Table 2 and Table 3, respectively. The results demonstrate

[1]https://github.com/openlink/virtuoso-opensource

Table 3: Experimental results (F1) on the sampled test set of KQA Pro. We copy the results of baselines from (Xiong et al., 2024)

Model		KQA Pro									
		CT	QA	QAQ	QN	QR	QRQ	SA	SB	VF	Overall
SP-based	BART-SPARQL	37	44	37	36	67	33	49	78	58	48.78
Pompting	IO w/GPT-4	27	23	36	40	25	50	11	69	73	39.33
	CoT w/GPT-4	22	26	35	34	18	46	21	79	77	39.78
	SC w/GPT-4	25	28	33	38	22	51	19	86	75	41.89
LLMs+KGs	Inter-KBQA	74	83	64	73	73	59	80	61	80	71.89
	Ours	**82**	**87**	**79**	**82**	**84**	**75**	**87**	**88**	**86**	**83.33**

the significant superiority of our method. Table 2 presents the performance of various models on WebQSP and CWQ datasets. First, SP-based methods perform well on the WebQSP dataset, while their performance declines significantly on the complex CWQ dataset. This is because the SPARQL queries corresponding to complex questions involve more intricate syntax (e.g., `MAX`, `Count`), making it challenging for trained models to directly generate accurate SPARQL queries. However, our approach can iteratively generate accurate SPARQL queries for complex questions under the guidance of learned rules, achieving a 15.4% improvement on the more complex CWQ dataset compared to the best SP-based method. Secondly, although LLMs are highly powerful, the overall performance remains suboptimal even when utilizing GPT-4 directly, particularly manifesting a significant performance gap on the CWQ dataset. This indicates that relying solely on LLMs to address complex questions remains difficult. Therefore, some methods integrate external KBs to enhance LLMs in addressing complex questions. Although these methods demonstrate promising results, our approach achieves competitive advancements, yielding an average improvement of 2.8% across two datasets compared to the best-performing method, ToG.

As shown in Table 3, we compare the performance of our proposed model with other existing approaches on the sampled test set of KQA Pro. As we can see, our approach achieves a significant improvement of an average of 11.4 percentage points over the best-performing method, Inter-KBQA. In addition, the experimental results demonstrate that our method can consistently achieve superior performance for nine types of complex questions in the KQA Pro dataset. Unlike previous methods, our approach is not constrained by question type, thereby making it more robust.

Table 4: Ablation study results on WebQSP and CWQ.

Model	WebQSP		CWQ	
	F1	Hits@1	F1	Hits@1
Ours	81.9	84.1	70.2	73.5
w/o RFFT	80.2	82.9	68.6	72.3
w/o Ques_sim	79.4	82.2	67.8	71.7
w/o Rule_sim	78.1	80.4	66.9	70.2
w/o Rule Guide	67.8	70.2	52.9	57.5

5.6 Further Analysis

5.6.1 Ablation Study.

To evaluate the contribution of each component in our proposed framework, we conduct a series of ablation experiments on the WebQSP and CWQ datasets. The results are presented in Table 4.

w/o RFFT. Instead of using RFFT, we utilize the PEFT method (Xu et al., 2023) to fine-tune the LLM for directly generating rules. However, this does not enable the LLM to learn how to generate a variety of rules step-by-step. As a result, the rule generator may produce incorrect rules, failing to accurately guide the reasoning process. The performance of the ablation models declines consistently across both datasets. Additionally, we conduct a case comparison of the fine-tuning methods in Appendix G.

w/o Ques_sim and **w/o Rule_sim**. For rule retrieval, we achieve this by measuring question similarity and rule similarity. Experimental results clearly show that the absence of either component leads to a significant decrease in model performance. Therefore, it is essential to simultaneously consider both question and query information for

Table 5: Case Study. Two typical questions from the CWQ dataset.

Question I	What country speaks Germanic languages with a capital called Brussels?	Right
Inter-KBQA	*SELECT ?x WHERE {m.04306rv language.countries_spoken_in ?x .?x country.official_language m.04306rv .?x country.capital m.0177z. }*	Yes
Ours	*SELECT ?x WHERE {m.04306rv language.countries_spoken_in ?x .?x country.official_language m.04306rv .?x country.capital m.0177z. }*	Yes
Question II	What country is located in the Greenwich Mean Time Zone that is the main trading partner of China?	Right
Inter-KBQA	*SELECT ?x WHERE{{ m.0d05w3 statistical_region. places_exported_to ?x} UNION{ m.0d05w3 statistical_region.places_imported_from ?x}}*	No
Ours	*SELECT ?x WHERE{{ m.0d05w3 statistical_region.places_exported_to ?y . ?y imports_and_exports.exported_to ?x . } UNION {m.0d05w3 statistical_region.places_imported_from ?y . ?y imports_and_exports. imported_from ?x .}?x location.time_zones m.03bdv . }*	Yes

the accurate retrieval of relevant rules.

w/o Rule Guide. Rather than the rule-guided incremental reasoning approach, we feed the given question and the relevant rule into the LLM (i.e., GPT-4) to directly generate the SPARQL query. The experimental results significantly illustrate the significant contribution of rule-guided reasoning to the model's performance, resulting in a notable performance gap between the ablation model and the original model.

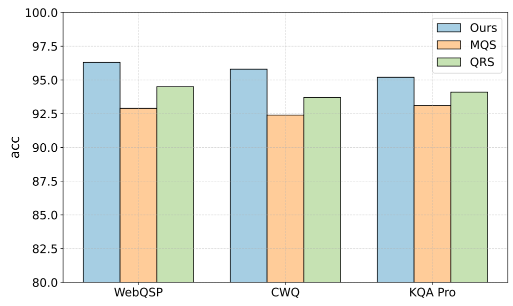


Figure 3: The accuracy of rule retrieval .

5.6.2 Rule Retrieval Effectiveness Study

In this experiment, we explored the effectiveness of our rule retrieval. Figure 3 presents a comparison of rule retrieval accuracy between our approach and two other methods, MQS (Guo et al., 2023) and QRS (Nan et al., 2023), evaluated across the WebQSP, CWQ, and KQA Pros datasets. The experimental results demonstrate that our approach consistently outperforms the other two methods across all datasets. This superiority is attributed to the fact that MQS considers only question information, while QRS focuses solely on rule information. In contrast, our approach incorporates both, enabling more accurate retrieval of relevant rules.

5.6.3 Case Study

To better showcase the advantages of our method, we conducted a case comparison of our method with Inter-KBQA in Table 5. As observed, the Inter-KBQA method successfully generates the correct SPARQL query for Question I but fails to do so for the more complex Question II. This failure arises because the SPARQL query for Question II involves a more intricate syntactic structure, such as `UNION`, which semantic parsing by LLMs alone cannot ensure in terms of grammaticality, faithfulness, and controllability. In contrast, our approach, guided by learned rules, is capable of accurately generating the corresponding SPARQL queries for both questions.

6 Conclusion

In this paper, we introduce Rule-KBQA, a novel framework that utilizes rules learned during the induction phase to accurately guide logical form generation in the deduction phase. In the induction phase, We establish rules to describe diverse logical forms and use REFT to fine-tune LLMs for generating new rules, building a comprehensive rule library. In the deduction phase, we employ a symbolic agent to engage in interacting with the KB environment, guided by the learned rules, to incrementally generate executable logical forms. Meanwhile, we utilize the LLM to evaluate the plausibility of each candidate decision at every step. Extensive experiments conducted on three benchmark datasets clearly demonstrate the effectiveness of the proposed model, which surpasses state-of-the-art approaches.

Limitations

In our work, we construct a comprehensive rule library by mining rules from existing data and generating new ones using large language models. We then retrieve relevant rules from this library to guide the semantic parsing of various questions. Although this approach yields satisfactory results, the entire process is relatively redundant and complex. Therefore, in future work, we plan to explore the development of a rule generator that can precisely generate corresponding rules given a question, thereby simplifying our framework and making it more straightforward.

Acknowledgments

This work was supported by the National Key Research and Development Program of China under Grant No.2023YFC3304404.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

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, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 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*.

Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. 2022a. KQA pro: A dataset with explicit compositional programs for complex question answering over knowledge base. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 6101–6119. Association for Computational Linguistics.

Shulin Cao, Jiaxin Shi, Zijun Yao, Xin Lv, Jifan Yu, Lei Hou, Juanzi Li, Zhiyuan Liu, and Jinghui Xiao. 2022b. Program transfer for answering complex questions over knowledge bases. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8128–8140, Dublin, Ireland. Association for Computational Linguistics.

Shuang Chen, Qian Liu, Zhiwei Yu, Chin-Yew Lin, Jian-Guang Lou, and Feng Jiang. 2021a. Retrack: A flexible and efficient framework for knowledge base question answering. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing: system demonstrations*, pages 325–336.

Yongrui Chen, Huiying Li, Yuncheng Hua, and Guilin Qi. 2021b. Formal query building with query structure prediction for complex question answering over knowledge base. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 3751–3758.

Yongrui Chen, Huiying Li, Guilin Qi, Tianxing Wu, and Tenggou Wang. 2023. Outlining and filling: Hierarchical query graph generation for answering complex questions over knowledge graphs. *IEEE Trans. Knowl. Data Eng.*, 35(8):8343–8357.

Zi-Yuan Chen, Chih-Hung Chang, Yi-Pei Chen, Jijnasa Nayak, and Lun-Wei Ku. 2019. Uhop: An unrestricted-hop relation extraction framework for knowledge-based question answering. In *Proceedings of NAACL-HLT*, pages 345–356.

Rajarshi Das, Manzil Zaheer, Dung Thai, Ameya Godbole, Ethan Perez, Jay Yoon Lee, Lizhen Tan, Lazaros Polymenakos, and Andrew Mccallum. 2021. Case-based reasoning for natural language queries over knowledge bases. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9594–9611.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

Yu Gu, Xiang Deng, and Yu Su. 2023. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

Chunxi Guo, Zhiliang Tian, Jintao Tang, Pancheng Wang, Zhihua Wen, Kang Yang, and Ting Wang.

2023. A case-based reasoning framework for adaptive prompting in cross-domain text-to-sql. *CoRR*, abs/2304.13301.

Sen Hu, Lei Zou, Jeffrey Xu Yu, Haixun Wang, and Dongyan Zhao. 2018a. Answering natural language questions by subgraph matching over knowledge graphs. *IEEE Trans. Knowl. Data Eng.*, 30(5):824–837.

Sen Hu, Lei Zou, and Xinbo Zhang. 2018b. A state-transition framework to answer complex questions over knowledge base. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2098–2108. Association for Computational Linguistics.

Yi Hu, Xiaojuan Tang, Haotong Yang, and Muhan Zhang. 2024. Case-based or rule-based: How do transformers do the math? *Preprint*, arXiv:2402.17709.

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

Yunshi Lan and Jing Jiang. 2020a. Query graph generation for answering multi-hop complex questions from knowledge bases. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 969–974, Online. Association for Computational Linguistics.

Yunshi Lan and Jing Jiang. 2020b. Query graph generation for answering multi-hop complex questions from knowledge bases. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 969–974. Association for Computational Linguistics.

Belinda Z. Li, Sewon Min, Srinivasan Iyer, Yashar Mehdad, and Wen-tau Yih. 2020. Efficient one-pass end-to-end entity linking for questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6433–6441. Association for Computational Linguistics.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6966–6980.

Haoran Luo, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, et al. 2023. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. *arXiv preprint arXiv:2310.08975*.

Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023. Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies. *CoRR*, abs/2305.12586.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Trans. Assoc. Comput. Linguistics*, 2:377–392.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

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

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *NAACL-HLT*, pages 641–651.

Thomas Pellissier Tanon, Denny Vrandecic, Sebastian Schaffert, Thomas Steiner, and Lydia Pintscher. 2016. From freebase to wikidata: The great migration. In *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 1419–1428. ACM.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

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

Guanming Xiong, Junwei Bao, and Wen Zhao. 2024. Interactive-kbqa: Multi-turn interactions for knowledge base question answering with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 10561–10582. Association for Computational Linguistics.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. 2023. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *Preprint*, arXiv:2312.12148.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. 2022. RNG-KBQA: Generation augmented iterative ranking for knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6032–6043, Dublin, Ireland. 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 (Volume 1: Long Papers)*, pages 1321–1331.

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.

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.

Appendix

A Rule Definition

Following the work given in (Chen et al., 2023), the vertex class denotes {"Ans", "Var", "Ent", "Val"}, and the edge class denotes {"Rel", "Ord", "Cmp", "Agg"}. The specific meanings of these classes are as follows: "Ans" represents the only answer node in the query graph. "Var" represents variables, which are placeholders for unknown vertices other than the answer. "Ent" represents the entity node in the query graph, such as $m.03wws\_$. "Val" represents the values, such as 1 and $1996-01-01$. "Rel" denotes the relations in KB, such as $location.country.languages\_spoken$. "Ord" denotes whether the ranking order is ascending (i.e., ASC) or descending (i.e., DSC), in ORDER BY clauses. "Cmp" denotes $\{<, \leq, >, \geq, =, \neq\}$ that indicate numerical or temporal comparison. In addition, we also add $During$ and $Overlap$ to handle temporal intervals. "Agg" denotes aggregation functions over the variables or answers, including $Count$, Max, Min, and Ask. The rule can reflect the topology of the query graph while its vertices and edges are slots of categories without instances.

```
select: [ans], where: {
   notUnion: [],
   union: [],
   subQueries: []},
orderBy: null
```

Table 6: An Empty Rule Structure

In addition, the rule retains the syntax construction of SPARQL queries. Table 6 shows an empty rule structure, where the "union" component corresponds to the `UNION` clause in SPARQL queries, the "subQueries" component reflects the subquery clause and the "orderBY" component aligns with the `ORDER BY` clause. All other components are categorized under the "notUnion" section. The content populated within the rule is represented in the form of triples. Furthermore, we classify the types of triples based on edge labels: the "Rel" class corresponds to "Basic," "Agg" to "Aggregation," "Cmp" to "Comparison," and "Ord" to "Ordinal."

B Rule Constraint Type

We define six types of constraints involved in rules, including "Basic", "Bridge", "Union", "Comparison", "Aggregation", and "Ordinal". Based on the rule definition mentioned above, the classification basis of each constraint type is described as follows:

"Basic" indicates that the vertex classes in the rule include "Ent", "Ans", and "Var", with only one vertex being of class "Ent". Additionally, all edge classes are designated as "Rel." In the rule structure, all parts except for the "notUnion" section are empty.

"Bridge" indicates that the vertex classes in the rule include "Ent", "Ans", and "Var", with at least two vertices being of class "Ent". Additionally, all edge classes are designated as "Rel". In the rule structure, all parts except for the "notUnion" section are empty.

"Union" indicates that the vertex classes in the rule include "Ent", "Ans", and "Var", with at least two vertices being of class "Ent".Additionally, all edge classes are designated as "Rel". In the rule structure, the "notUnion" and "union" sections are not empty, while the rest of the sections are empty.

"Comparison" indicates that the edge classes in the rule include "Cmp", while all other edge classes are designated as "Rel".

"Aggregation" indicates that the edge classes in the rule include "Agg", while all other edge classes do not include "Ord".

"Ordinal" indicates that the edge classes in the rule contain "Ord".

Table 8 presents relevant rule examples for the six types of constraints.

C Rule-Following Fine-Tuning Process

We fine-tune the LLM-based rule generator using the set of rules mined from existing data. Additionally, based on the definitions of rule categories mentioned above, we provide a detailed description of the guideline for generating various category rules in the input, teaching the LLM to progressively refer to and follow the necessary rule generation guidelines during its inference process. We list an example of the input for rule-following in Table 9 and an example of the output in Table 10. Therefore, the fine-tuned rule generator can accurately produce various categories of rules along with corresponding natural language questions.

D LLMs Evaluation Prompt

Table 7 shows the detail of LLMs evaluation prompt.

Instruction Description: Based on the given Question, the corresponding Rule, and the interaction history, please retrieve relations (separated by semicolon) to select the most suitable relation to replace the rel in the rule triple. Please state one relation, No need for an explanation. *In-Context Few-shot* **Question:** {The given Question} **Rule:** {The corresponding Rule} **History:** {The previous interaction history} **Candidates:** {The current candidate relations} **Selected Relation:**

Table 7: LLMs Evaluation Prompt

E Dataset

To evaluate the effectiveness of the proposed method, we conducted experiments on three public datasets, including WebQuestionsSP (WebQSP) (Yih et al., 2016), ComplexWebQuestions (CWQ)(Talmor and Berant, 2018), and KQA Pro (Cao et al., 2022a).

WebQuestionsSP (WebQSP) (Yih et al., 2016) is extensively utilized in KBQA research and comprises 4,937 natural language questions alongside their corresponding SPARQL queries based on Freebase. Following (Chen et al., 2019), the questions can be classified into 1-hop and 2-hop categories based on the length of the inferential relation chain, i.e., the path from the topic entity to the answer node.

ComplexWebQuestions(CWQ) (Talmor and Berant, 2018) is a question answering dataset over Freebase. Building upon WebQuestionsSP, it expands question entities or introduces answer constraints to formulate complex questions, which comprise four types and necessitate up to four hops of reasoning. These question types encompass composition, conjunction, comparison, and superlative. CWQ also offers SPARQL annotations that can be directly executed on the knowledge base.

KQA Pro (Cao et al., 2022a) is a dataset for Complex KBQA, including 117,970 diverse natural language questions. For each question, it provides the corresponding SPARQL query executed on the Wikidata. It features nine types of complex questions, which include Count (Ct), QueryAttr (QA), QueryAttrQualifier (QAQ), QueryName(QN), QueryRelation (QR), QueryRelationQualifier (QRQ), SelectAmong (SA), SelectBetween(SB), and Verify (VF).

Notably, for the KQA Pro dataset, we use the sampled test dataset from (Xiong et al., 2024), which can enhance the assessment of the model's capability in addressing various complex questions. This dataset employs a unified sampling method for each type of question, ensuring a balanced composition across different question categories. Specifically, the study samples 100 questions from each category in KQA Pro.

F Baselines

Our baselines encompass a wide coverage of related models, which can comprehensively evaluate our approach. These methods can be divided into three categories: (1) SP-based methods, including QGG (Lan and Jiang, 2020b), ReTraCk (Chen et al., 2021a), BART-SPARQL(Cao et al., 2022a), RnG-KBQA (Ye et al., 2022), Program Trans (Cao et al., 2022b), DecAF (Yu et al., 2023). (2) Prompting methods with GPT-4, including IO (Brown et al., 2020), CoT (Wei et al., 2022), SC (Wang et al., 2023b). (3) LLMs+KBs methods, including KB-BINDER (Li et al., 2023), StructGPT (Jiang et al., 2023), Pangu (Gu et al., 2023), KD-CoT (Wang et al., 2023a), ToG (Sun et al., 2024), Inter-KBQA (Xiong et al., 2024). The details of each baseline are described as follows.

QGG (Lan and Jiang, 2020b) propose a modified staged query graph generation method with more flexible ways to generate query graphs and ranks candidate query graphs using different features.

ReTraCk (Chen et al., 2021a) introduces a versatile and efficient KGQA framework, which can generate well-formed but unnecessarily faithful programs by grammar-level constraints.

BART-SPARQL (Cao et al., 2022a) directly learn a parsers using supervised learning by regarding semantic parsing as a sequence-to-sequence task.

RnG-KBQA (Ye et al., 2022) ranks candidate logical forms using a contrastive ranker, then composes the final logical form with a generation model based on the question and top candidates.

Program Trans (Cao et al., 2022b) leverages the valuable program annotations on the rich-resourced

KBs as external supervision signals to aid program induction for the low-resourced KBs.

DeCAF (Yu et al., 2023) jointly generates both logical forms and direct answers and then combines the merits of them to get final answers.

Prompting methods w/GPT-4. We compare the efficacy of standard prompting (IO prompt) (Brown et al., 2020), Chain-of-Thought prompting (CoT prompt) (Wei et al., 2022), and the technique of Self-Consistency (SC)(Wang et al., 2023b).

KB-BINDER (Li et al., 2023) uses LLMs to create preliminary logical forms through demonstration imitation and then binds the draft to an executable version through knowledge base integration.

KD-CoT (Wang et al., 2023a) proposes an interactive framework that utilizes a QA system to access external knowledge and provide high-quality answers to LLMs for solving knowledge-intensive KBQA tasks.

StructGPT (Jiang et al., 2023) gathers relevant evidence from structured data, allowing LLMs to focus on the reasoning task using the acquired information.

Pangu (Gu et al., 2023) proposes using LLMs for discrimination rather than generation in grounded language understanding, combining symbolic search with neural scoring in a novel way.

ToG (Sun et al., 2024) enables the LLM agent to iteratively execute beam search on KG, discover the most promising reasoning paths, and return the most likely reasoning results.

Inter-KBQA (Xiong et al., 2024) proposes an iterative, dialogue-based problem-solving process, which enables semantic parsing and SPARQL generation by developing three generic APIs for KB interaction.

G Case Comparison of Fine-Tuning Methods

In Table 11, we present a case comparison between our RFFT method and the traditional fine-tuning approach (PEFT) to better demonstrate the advantages of our method. As shown, for generating rules of the Comparison_4_hop category, our RFFT method produces a more reasonable rule compared to the PEFT method. This is because our approach trains the LLM to incrementally generate various rules based on a rule generation guide, whereas the PEFT method directly generates rules without such guidance. An input-output example for the PEFT method is provided in Table 12.

```
Rule1(Basic_3_hop): {
    select: [ans],
    where: {
        notUnion: [
            [Basic,[var_0,rel_1,var_1]],
            [Basic,[var_1,rel_2,ent_3]],
            [Basic,[var_0,rel_0,ans]]
        ],
        union: [],
        subQueries: []
    },
    orderBy: null
}
Rule2(Bridge_3_hop): {
    select: [ans],
    where: {
        notUnion: [
            [Basic,[ent_3,rel_2,var_1]],
            [Basic,[var_1,rel_1,ans]],
            [Basic,[ans,rel_0,ent_2]]
        ],
        union: [],
        subQueries: []
    },
    orderBy: null
}
Rule3(Union_3_hop]): {
    select: [ans],
    where: {
        notUnion: [
            [Basic,[ans,rel_0,ent_1]]
        ],
        union: [
            [[Basic,[ ent_2, rel_1, ans]]],
            [[Basic,[ ent_2, rel_2, ans]]]
        ],
        subQueries: []
    },
    orderBy: null
}

Rule4(Comparison_3_hop): {
    select: [ans],
    where: {
        notUnion: [
            [Basic,[ent_3,rel_2,ans]],
            [Basic,[ans,rel_1,var_1]],
            [Comparison,[var_1,>,val_0]]
        ],
        union: [],
        subQueries: []
    },
    orderBy: null
}
Rule5(Aggregation_9_hop): {
    select: [ans],
    where: {
        notUnion: [
            [Basic,[ent_7,rel_5,var_5]],
            [Basic,[var_5,rel_4,ans]],
            [Aggregation,[ans,COUNT,var_0]],
            [Comparison,[var_2,=,var_0]],
            [Basic,[ans,rel_3,ent_9]]
        ],
        union: [],
        subQueries: [
            {
                select: [var_2],
                where: {
                  notUnion: [
                  [Basic,[ent_7,rel_5,var_5]],
                  [Basic,[var_5,rel_4,ans]],
                  [Aggregation,[ans,COUNT,var_0]],
                  [Aggregation,[var_0,MAX,var_2]]],
                union: [],
                subQueries: []
                },
                orderBy: null
            }
        ]
    },
    orderBy: null
}
Rule6(Ordinal_3_hop): {
    select: [ans],
    where: {
        notUnion: [
            [Basic,[ent_3,rel_2,ans]],
            [Basic,[ans,rel_1,var_1]]
        ],
        union: [],
        subQueries: []
    },
    orderBy: [Ordinal,[var_1,DESC,val_0]]
}
```

Table 8: The relevant rule examples for the six types of constraints.

Input:

Follow the rule generation guidelines step by step to generate the rule: Union_3_hop, and also create a natural language question corresponding to the rule.

You need to generate an X-type rule with h hops. The rule should be in JSON format, with each part consisting of a key and value pair.

1.The keys are select, where, orderBy, and question.

2.The value for select is fixed as ["ans"].

3.The value for where is also a JSON object containing three keys: notUnion, union, and subQueries.

3.1. The value of notUnion is a JSON list containing triples and their corresponding types.

3.2. Each triple is a list containing two nodes and an edge. Nodes can be either ent, var, ans, or val. In addition, edges are represented by rel,cmp,agg, or ord.

3.3. Each triple has a type, which can be Basic, Comparison, Aggregation, or Ordinal.

3.4. Basic denotes a regular type with a relation of only rel. Comparison indicates a situation describing a comparative relationship between variables and values, which may include relations such as $<$, $\leq$, $>$, $\geq$, $=$, $\neq$, DURING, and OVERLAP. DURING and OVERLAP are used to handle time intervals. Aggregation describes aggregation functions on variables or answers, such as COUNT, MAX, MIN, or ASK. Ordinal indicates that the type of the edge in the triple is ord.

3.5. Triples in where must be connected nodes and cannot be isolated.

3.6. Each rule must contain at least one entity node and one ans node.

4.The structure of union is the same as notUnion. When there are multiple relations between two nodes, these are placed in union.

5.The structure of subQueries is the same as notUnion.

6.The value for orderBy is a JSON list that includes the sorting method, the entity or variable being sorted, and the type of the entity or variable. Sorting methods can be DESC or ASC, and the types of entities or variables can be datetime, integer, or float.

7.Triples which can be Basic, Bridge, Union, Comparison, Aggregation, or Ordinal.

7.1. If X-type is Ordinal, it indicates that the rule graph contains edges with orderBy. Additionally, h-1 triples should be generated where the remaining edges do not include orderBy.

7.2. If X-type is Aggregation, it indicates that the rule graph contains edges with Aggregation, and the remaining edges do not include orderBy. The triples include Basic, Comparison, and Aggregation types.

7.3. If X-type is Comparison, it indicates that the rule graph contains edges with Comparison. Comparison type triples should be in the notUnion section, while the rest are Basic type triples added to the notUnion section, excluding union, orderby, and subquery sections.

7.4. If X-type is Bridge, it indicates that the problem rule graph contains only edges of type Rel. The triple types are only Basic. The graph has two or more vertices of entity type, and all other vertices, except for ans, are of variable type. These should be added to the notUnion section, excluding union, orderby, and subquery sections.

7.5. If X-type is Basic, it indicates that the problem rule graph contains only edges of type rel, and the triple types are only Basic. These should be added to the notUnion section, excluding union, orderby, and subquery sections.

7.6. If X-type is Union, it indicates that there are multiple edges connecting entity vertices with variable nodes in the problem rule graph. Generate triples in the union section for entities and variables with multiple edges, with the triple type being only Basic. The remaining triples should be added to the notUnion section, excluding orderby and subquery sections.

8.Based on the generated rule, create a corresponding natural language question.

Table 9: Input for rule-following.

Output:

1. According to the requirements, a basic rule structure can be generated.

```
{
  select: [ans],
  where: {
    notUnion: [
    ]
    union: [
    ]
    subQueries: []
  },
  orderBy: null
}
```

2. To generate a 3-hop rule, you need to add 3 triples.
2.1 To generate a Union-type rule, first generate a triple for Basic:
  [Basic, [ans, rel_0, ent_1]]
2.2 For the 2nd triple, select a variable node to serve as the intermediate node and generate 1 Basic type triple:
  [Basic, [ent_2, rel_1, ans]]
2.3 For the 3rd triple, select a variable node to serve as the intermediate node and generate 1 Basic type triple:
  [Basic, [ent_2, rel_2, ans]]
3. Based on the above rule, the generated natural language question is:
  What sports team owned by Jerry Buss that played in the 2010 NBA Finals?
4. Add the generated triples to the notUnion and union fields respectively to obtain the final rule.

```
{
  select: [ans],
  where: {
    notUnion: [
      [Basic,[ans, rel_0, ent_1 ] ],
    ]
    union: [
      [Basic,[ent_2, rel_1, ans] ],
      [Basic,[ent_2, rel_2, ans] ],
    ]
    subQueries: []
  },
  orderBy: null
  question: What sports team owned by Jerry Buss played in the 2010 NBA Finals?
}
```

Table 10: Output for rule-following.

Method	**Generated Rule**

Traditional fine-tuning	{
	  select: [ans],
	  where: {
	    notUnion: [
	      [Bridge,[ent_4,var_3,rel_3]],
	      [Bridge,[var_3,var_2,rel_2]],
	      [Comparison,[var_2,overlap,ans]],
	      [Bridge,[ans,var_1,rel_1]]
	    ]
	    union: []
	    subQueries: []
	  },
	  orderBy: null
	}

Rule-Following fine-tuning	{
	  select: [ans],
	  where: {
	    notUnion: [
	      [Basic, [ent_4,rel_3, ans]],
	      [Basic, [ans,rel_2, var_1]],
	      [Basic, [var_1,rel_0, var_2]],
	      [Comparison, [var_2, <, val_0]]],
	    ]
	    subQueries: []
	  },
	  orderBy: null
	}

Table 11: Comparison_4_hop rule generated by two fine-tuning methods.

Prompt:
You need to generate an X-type rule with h hops. The rule should be in JSON format, with each part consisting of a key and value pair.
1.The keys are select, where, orderBy, and question.
2.The value for select is fixed as ["ans"].
3.The value for where is also a JSON object containing three keys: notUnion, union, and subQueries.
3.1. The value of notUnion is a JSON list containing triples and their corresponding types.
3.2. Each triple is a list containing two nodes and an edge. Nodes can be either ent, var, ans, or val. In addition, edges are represented by rel,cmp,agg, or ord.
3.3. Each triple has a type, which can be Basic, Comparison, Aggregation, or Ordinal.
3.4. Basic denotes a regular type with a relation of only rel. Comparison indicates a situation describing a comparative relationship between variables and values, which may include relations such as <, $\leq$, >, $\geq$, =, $\neq$, DURING, and OVERLAP. DURING and OVERLAP are used to handle time intervals. Aggregation describes aggregation functions on variables or answers, such as COUNT, MAX, MIN, or ASK. Ordinal indicates that the type of the edge in the triple is ord.
3.5. Triples in where must be connected nodes and cannot be isolated.
3.6. Each rule must contain at least one entity node and one ans node.
4.The structure of union is the same as notUnion. When there are multiple relations between two nodes, these are placed in union.
5.The structure of subQueries is the same as notUnion.
6.The value for orderBy is a JSON list that includes the sorting method, the entity or variable being sorted, and the type of the entity or variable. Sorting methods can be DESC 7.Triples which can be Basic, Bridge, Union, Comparison, Aggregation, or Ordinal.
8.Based on the generated rule, create a corresponding natural language question as the value for question.

Input:
Please generate a Union_3_hop type rule according to the prompt above, and also create a natural language question corresponding to that rule.

Output:
```
{
  select: [ans],
  where: {
    notUnion: [
      [Basic,[ans, rel_0, ent_1 ] ],
    ]
    union: [
      [Basic,[ent_2, rel_1, ans] ],
      [Basic,[ent_2, rel_2, ans] ],
    ]
    subQueries: []
  },
  orderBy: null
  question: What team owned by Comcast Spectacor played in the 2010 Stanley Cup?
}
```

Table 12: An example of input-output for PEFT.