Interactive-KBQA: Multi-Turn Interactions for Knowledge Base Question Answering with Large Language Models

Guanming Xiong
Peking University
Beijing, China
gm_xiong@pku.edu.cn

Junwei Bao*
Beijing, China
baojunwei001@gmail.com

Wen Zhao
Peking University
Beijing, China
zhaowen@pku.edu.cn

Abstract

This study explores the realm of knowledge base question answering (KBQA). KBQA is considered a challenging task, particularly in parsing intricate questions into executable logical forms. Traditional semantic parsing (SP)-based methods require extensive data annotations, which result in significant costs. Recently, the advent of few-shot in-context learning, powered by large language models (LLMs), has showcased promising capabilities. However, fully leveraging LLMs to parse questions into logical forms in low-resource scenarios poses a substantial challenge. To tackle these hurdles, we introduce Interactive-KBQA, a framework designed to generate logical forms through direct interaction with knowledge bases (KBs). Within this framework, we have developed three generic APIs for KB interaction. For each category of complex question, we devised exemplars to guide LLMs through the reasoning processes. Our method achieves competitive results on the WebQuestionsSP, ComplexWebQuestions, KQA Pro, and MetaQA datasets with a minimal number of examples (shots). Importantly, our approach supports manual intervention, allowing for the iterative refinement of LLM outputs. By annotating a dataset with step-wise reasoning processes, we showcase our model's adaptability and highlight its potential for contributing significant enhancements to the field.[1]

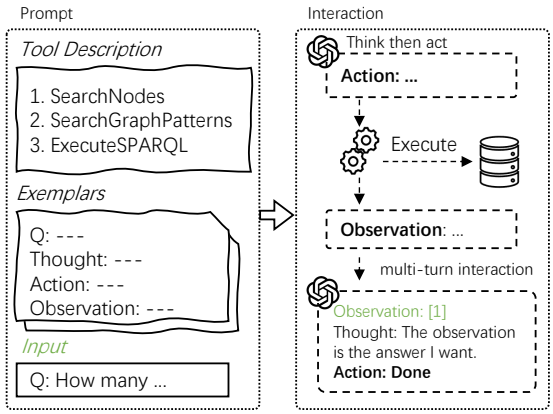

Figure 1: Overview of the interactive process.

1 Introduction

Knowledge base question answering (KBQA) is an increasingly significant research area that leverages structured knowledge bases (KBs) to provide precise answers to natural language (NL) questions. A KBQA system aims to harness the vast, structured information residing in KBs, such as Freebase (Bollacker et al., 2008), Wikidata (Vrandečić and Krötzsch, 2014), or domain-specific databases like the Movie KB (Zhang et al., 2018). Accurately interpreting and responding to user inquiries with data from knowledge bases has potent applications across numerous sectors, making it a focal point of both academic research and industrial innovation.

Recent advancements in KBQA research can generally be classified into two primary approaches: information retrieval (IR)-based methods and semantic parsing (SP)-based methods. IR-based methods focus on understanding the query, retrieving pertinent subgraphs from KBs related to the question, and extracting the answer from these subgraphs (Dong et al., 2023; Zhang et al., 2022a; Yan et al., 2021). Conversely, SP-based approaches convert NL questions into executable logical forms, leveraging pre-trained generative models to interact with the knowledge base and generate answers (Chen et al., 2021a; Ye et al., 2022; Shu et al., 2022; Yu et al., 2023).

The emergence of large language models (LLMs), such as ChatGPT (Ouyang et al., 2022) and GPT-4 (OpenAI, 2023), has opened new avenues for enhancing KBQA systems. These models have demonstrated promising results in reasoning (Wei et al., 2022) and few-shot learning ca-

*Corresponding author.
[1]Code and data are available at: https://github.com/JimXiongGM/Interactive-KBQA

pabilities (Chen, 2023), setting new benchmarks in the KBQA domain (Gu et al., 2023; Li et al., 2023a; Jiang et al., 2023b; Sun et al., 2024; Li et al., 2023b).

Despite these advancements, KBQA systems face several pressing challenges.

Problem 1. Information Retrieval (IR) 방식의 한계

Complex query handling. The primary challenge for IR-based approaches is the difficulty in processing complex queries. For instance, questions characterized solely by entity types or concepts, alongside numerical constraints (e.g., "How many basketball players are taller than 2 meters?"), pose significant challenges. Such queries demand an understanding beyond simple entity recognition, rendering them difficult to address with current methodologies.

Resource scarcity for semantic parsing. SP-based approaches require extensive annotated datasets for training, which are resource-intensive to create. This requirement severely limits the scalability of these methods and contributes to the "black box" nature of the reasoning process, which lacks transparency and interpretability.

Underutilization of large language models (LLMs). Despite the demonstrated capabilities of LLMs in reasoning and few-shot learning, existing KBQA approaches have not fully leveraged these strengths. Most current methods either use LLMs as classifiers to identify predicates (Gu et al., 2023; Sun et al., 2024) or as mimics to generate possible logical forms or questions (Li et al., 2023a,b). There remains a significant opportunity to more effectively leverage LLMs in a few-shot setting to enhance the accuracy and interpretability of KBQA systems.

To address these challenges, we introduce the Interactive-KBQA framework, which combines the reasoning power of LLMs with three tools to interact with KBs. By conceptualizing the LLM as an agent and the KB as the environment, Interactive-KBQA facilitates an iterative, dialogue-based problem-solving process. Figure 1 illustrates the overall process. The process adheres to the thought-action paradigm. Given a complex query, the LLM is required to think and provide an action to interact with the KB through a set of specific tools. These tools return the execution results as observations. Specifically, we designed a set of tools that support heterogeneous databases (i.e., Freebase, Wikidata, and a Movie KB) with a unified interaction logic. We categorized complex questions and provided only two annotated exemplars with complete interactive processes for each type, serving as in-context learning demonstrations to prompt the LLM to complete the task. Furthermore, the method introduced in this study allowed for manual intervention. Consequently, we manually annotated a small dataset with a detailed reasoning process, thereby creating a low-resource dataset. Finally, we fine-tuned open-source LLMs on this dataset. The experiments conducted have shown that this method is effective in low-resource contexts. We have released this high-quality dataset to further contribute to NLP research in the community.

The main contributions of this work are summarized as follows:

- Propose the Interactive-KBQA, a novel framework that harnesses the reasoning capabilities of LLMs for semantic parsing, enabling multi-turn interactions with KBs.
- Design a unified SPARQL-based toolset and interaction logic that efficiently address a wide array of complex queries.
- Conduct extensive experiments to demonstrate that our method achieves remarkable performance with few exemplars.
- Release a human-annotated KBQA dataset with step-wise reasoning processes, serving as a low-resource dataset.

2 Related Work

Knowledge base question answering (KBQA) methods can be broadly classified into two distinct categories: information retrieval (IR)-based methods and semantic parsing (SP)-based methods. These approaches address the challenge of effectively mapping natural language (NL) queries to the structured formats of knowledge bases (KBs).

IR-based methods aim to extract a question-specific subgraph from the KB and employ ranking algorithms to select the top entities or directly generate answers using text decoders (Lan et al., 2023). To bridge the gap between unstructured texts and structured KBs, Dong et al. (2023) proposed a Structured Knowledge-aware Pre-training method to enhance complex subgraph representation learning. Emphasizing the accuracy of subgraph retrieval, Zhang et al. (2022a) developed a trainable, decoupled subgraph retriever that boosts the performance of subgraph-oriented KBQA models. Further, Jiang et al. (2023c) introduced UniKGQA, a unified framework that integrates retrieval and

render = 만들다, 제우다, 야기하다

reasoning across architectures and learning parameters. However, IR-based methods typically rely on identifying entities within a query as an initial step for retrieving the relevant subgraph. This reliance poses a notable limitation, especially when handling complex queries.

SP-based methods parse questions into executable logical forms and perform queries against the KB to retrieve answers. Initial works translated questions into intermediate logical forms before execution. Yih et al. (2015) defined a query graph that resembles subgraphs of the KB and can be directly mapped to a logical form. Hu et al. (2018); Lan and Jiang (2020) extended query graphs to include aggregation operators, and Chen et al. (2021b) enhanced candidate query generation by leveraging predictions of query structure.

In recent years, generative models have increasingly been used to directly generate logical forms. Chen et al. (2021a) designed a retriever to fetch relevant KB items and utilized a Grammar-based Decoder based on LSTM for generating S-expressions (Gu et al., 2021). Meanwhile, Das et al. (2021) employed case-based reasoning for the KBQA task, using Big Bird (Zaheer et al., 2020) to generate logical forms by retrieving relevant cases. Notably, works such as Ye et al. (2022), Shu et al. (2022), Zhang et al. (2023), and Yu et al. (2023) involved initially retrieving basic elements from the KB, such as entities, relations, subgraphs, or texts as supplementary information, before directly generating logical forms using the T5 model (Raffel et al., 2020). However, this approach requires substantial amounts of training data and suffers from a lack of transparency in the reasoning process.

Large language models (LLMs) for KBQA leverage the potent few-shot learning capabilities inherent in LLMs. Recent studies have shown that LLMs can significantly enhance reasoning over KBs. By capitalizing on the powerful few-shot learning capabilities, these methods have demonstrated marked improvements in both the accuracy and efficiency of information retrieval from KBs.

In the context of few-shot scenarios, Gu et al. (2023) proposed that LLMs should prioritize evaluating the plausibility of agent plans over directly generating answers. Concurrently, Li et al. (2023a) advocated for the generation of logical forms as initial drafts, which are subsequently refined into executable queries using the KB. From the agent-environment perspective, Jiang et al. (2023b) developed two specialized interfaces for accessing the KB, while Gu et al. (2024) and Liu et al. (2024) designed seven tools to facilitate this process. Sun et al. (2024) introduced a novel approach that enables LLMs to iteratively employ beam search reasoning on a KB. Furthermore, Zong et al. (2024) proposed assigning three distinct roles to the agent for addressing different KBQA subtasks. Jiang et al. (2024) constructed an instruction dataset based on existing KBQA datasets. However, despite employing LLMs, these methods lack the exploration of more complex questions.

3 Approach

3.1 Problem Formulation

This study investigates a semantic parsing (SP) method for knowledge base question answering (KBQA). A knowledge base (KB) is formally represented as $K \in E \times R \times (E \cup L \cup C)$, where E denotes the set of entities, R signifies the set of relations between entities, C represents the set of classes, and L includes the literal values. Given a question Q and a knowledge base $\mathcal{K}$, our goal is to generate an executable SPARQL expression S that aligns with the question. Thus, the task can be formalized as $p(S|Q, \mathcal{K})$.

KG와 질문이 주어졌을 때, S를 생성하는 확률 (S는 SPARQL expression)

3.2 Overview

Recent advancements in large language models (LLMs) have demonstrated remarkable capabilities in few-shot learning and reasoning. However, fully leveraging LLMs to tackle complex KBQA challenges remains an elusive goal. To address this gap, we introduce Interactive-KBQA, an interactive method for KBQA that conceptualizes the LLM as an agent and the KB as an environment. This approach enables semantic parsing and SPARQL generation through dialogic interactions. Specifically, we design a unified interaction logic using three generic tools capable of supporting various complex types of questions (e.g., type constraints, count-based queries) across multiple databases (e.g., Freebase, Wikidata, Movie KB). Figure 2 presents an illustrative example of the interactive process.

3.3 Tools for Knowledge Base

The SP-based KBQA method necessitates the identification of elements and the appropriate graph patterns. Hence, it is crucial to design tools at

[2]For brevity, we omit the prefix: `PREFIX ns: <http://rdf.freebase.com/ns/>`.

elusive = 쉽게 달성하기 어려운, 피하는

Figure 2: An example of the interactive process. [2]

an atomic level to ensure their universality. In line with this principle, we introduce the following three tools.

SearchNodes(name): This function searches for nodes in the KB using the given surface name, `name`. Its primary purpose is entity linking (EL). Consequently, it not only returns the formal name of a node but also provides distinguishing features of the entity, such as its description and type. Importantly, this tool avoids dataset-specific EL techniques in favor of a generic retrieval approach.

SearchGraphPatterns(sparql, semantic): This function aims to identify and rank graph predicates within the KB that are essential, guided by the `semantic` parameter. The function requires the input `sparql`, which should consist of a SPARQL query beginning with `"SELECT ?e WHERE "`. Following this, it performs a query on the one-hop subgraph centered around the entity ?e. This includes both incoming and outgoing edges. Subsequently, it ranks the retrieved triples based on the semantic relevance to the `semantic` parameter and the predicates in triples. Ultimately, the tool returns the top K triples. This tool is designed to precisely identify subgraphs while discarding irrelevant information, thereby optimizing the context window's use in LLMs. It supports flexible operations and is specifically optimized for Compound Value Type[3] (CVT) in Freebase by flattening a CVT node to multiple single-hop relationships. For example, to find movies featuring Tom Hanks in Freebase, the usage would be: `SearchGraphPatterns('SELECT ?e WHERE {?e type.object.name "Tom Hanks"@en .}', semantic="play in film")`, and return: `[(?e, film.actor.film -> film.performance.film, "Nothing in Common"), ...]`.

ExecuteSPARQL(sparql): This tool allows for the direct execution of arbitrary SPARQL queries, ensuring unparalleled flexibility.

Implementation details are provided in Appendix A.2.1, and additional usage instructions are described in Appendix A.2.3.

3.4 Interactive Process

Given a question Q, we first construct a prompt text:

$$\text{Prompt} = \{\text{Inst}, E, Q\} \quad (1)$$

where the instruction text, Inst, consists of tool descriptions, tool usage, and the format of interaction. Inst is database-specific and pre-written. E denotes a set of exemplars, and for each type of question, we manually annotate two complete examples in an interactive format.

In each turn T, we let the LLM generate an action based on the Prompt and the history H of the

[3] A Compound Value Type is a Type within Freebase designed to represent data where each entry is composed of multiple fields.

interaction. Specifically, this procedure follows:

$$a_T = \text{LLM}(\{\text{Prompt}, H\}) \quad (2)$$

$$H = \{c_0, a_0, o_0, ..., c_{T-1}, a_{T-1}, o_{T-1}\} \quad (3)$$

where c denotes the intermediate thought process, an action a belongs to the set {SearchNodes, SearchGraphPatterns, ExecuteSPARQL, Done}, and the observation o is determined by executing an action, which is defined as $o_T = \text{Tool}(a_T)$.

We have devised a general thought-action paradigm for KBQA. **Thought**: Given Q, our initial thought c_0 is to deconstruct it into sub-queries reminiscent of triple forms; for example, the aforementioned example can be decomposed into `(Tom Hanks, act in, target movie)`. c_0 is not rigidly defined but adopts a free-form approach, leveraging the semantics of the question to facilitate understanding by both LLMs and humans. Except for the first round, the LLM is required to generate a thought c that clearly articulates its reasoning process based on observations. This approach is intended to render the decision-making process explainable. **Action**: At each turn T, the LLM must generate an action a_T that concludes the current round, using Python syntax. The tool then parses, executes, and returns the results, serving as the observation[4]. The LLM decides whether to end the dialogue based on the observation. If $a_T = \text{Done}$, we output the final observation o_T as the answer.

To minimize inference costs, we train a question type classifier on low-resource data and select E based on the predicted type, as further discussed in Appendix A.2.1.

3.5 Solutions for Complex Questions

Interactive-KBQA involves guiding LLMs through reasoning by annotating an interactive inference process. For different types of complex questions, it is crucial to identify patterns, design interaction modes, and label high-quality examples. This section outlines several representative types of complex questions to elucidate the design rationale behind this paper.

For **multi-hop** questions, our focus at each step is on specific predicates rather than concrete entities. For instance, as depicted in Figure 2, there is no need to locate specific television programs; expressing the graph pattern within SPARQL is sufficient to enable the tool to handle the ranking part of the task. In the case of **CVT structures** in Freebase, we explicitly describe the reasoning process when encountered. Moreover, we break down the star-shaped CVT structure into multiple one-hop relations and treat them accordingly. For instance, the semantics of the sentence "Tom Hanks plays the role of 'David Basner' in the film 'Nothing in Common' " can be represented by two triples: ("Tom Hanks", film.actor.film → film.performance.film, "Nothing in Common") and ("Tom Hanks", film.actor.film → film.performance.character, "David Basner"). When querying the **qualifier structure** in Wikidata, which modifies predicates, we devise a specialized SPARQL query pattern and provide a detailed thought process as well. Additional examples are provided in the Appendix A.8.

3.6 Human-Machine Collaborative Annotation

In realistic scenarios, including examples of all question types within the context is impractical for two primary reasons. First, the associated costs are substantial. Second, as discussed by Su et al. (2023); Zhu et al. (2023), LLMs experience a notable decline in performance when input tokens exceed certain limits. Additionally, Lightman et al. (2023) pointed out that process supervision can enhance a model's generalization capabilities. Consequently, we propose a human-machine collaborative data annotation method. Enabled by the Interactive-KBQA method introduced in this paper, annotating the reasoning process has become more straightforward.

Specifically, when annotators determine that the action a_T is unreasonable, it is manually corrected to a'_T. This adjustment is then incorporated into the message to generate the action a_{T+1}. Formally,

$$a_{T+1} = \text{LLM}(\{\text{Prompt}, \{c_0, a_0, o_0, ..., c_{T-1}, a'_T, o'_T\}\})$$

We set a breakpoint after each round where the LLM generates thoughts and actions, allowing human evaluators to review and decide on their acceptance. Once accepted, the process continues. If rejected, the annotator revises the generated thoughts and actions before proceeding. For each question type, we randomly sampled 50 data points from the training set and manually annotated them to provide low-resource data.

[4]In case of parsing errors, the tool will return a specific error message.

Dataset	#Type	#Anno (Train/Test)	#Raw (Train/Dev/Test)
WebQSP	2	100 / 300	3,098 / - / 1,639
CWQ	4	200 / 600	27,639 / 3,519 / 3,531
KQA Pro	9	450 / 900	94,376 / 11,797 / 11,797
MetaQA	3	150 / 900	329,282 / 39,138 / 30,903

Table 1: Statistics of the datasets.

The core principle of our methodology is to emulate the human data annotation process. Annotators intervene in specific scenarios where the model exhibits hallucinations (such as generating predicates not present in the observation), inconsistencies between Thought and Action, or deviations from the correct answer trajectory (for instance, cases where two consecutive rounds are incorrect).

4 Experiment

We examine Interactive-KBQA across a variety of complex question types and diverse databases (DBs).

4.1 Dataset & Preprocessing

WebQuestionsSP (WebQSP) (Yih et al., 2016) and **ComplexWebQuestions 1.1** (CWQ) (Talmor and Berant, 2018) are extensively used in KBQA research. These datasets comprise natural language questions paired with their corresponding SPARQL queries based on Freebase. Following Chen et al. (2019), we classify WebQSP questions 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. CWQ extends WebQSP by incorporating four types of complex questions: Conjunction (Conj), Composition (Compo), Comparative (Compa), and Superlative (Super).

KQA Pro (Cao et al., 2022) is a large-scale dataset designed for complex question answering over a dense set of Wikidata entries. It features nine types of complex questions, including Count (Ct), Query Attribute (QA), Query Attribute Qualifier (QAQ), Query Name (QN), Query Relation (QR), Query Relation Qualifier (QRQ), Select Among (SA), Select Between (SB), and Verify (Vf).

MetaQA (Zhang et al., 2018) is built upon a Movie Knowledge Base (KB) and includes three sets of question-answer pairs: 1-hop, 2-hop, and 3-hop. We converted the Movie KB into RDF triples to facilitate querying via SPARQL.

Due to the prohibitive costs associated with utilizing OpenAI, we employed a uniform sampling method across each type to construct a smaller test dataset. For each DB, we sampled 900 instances, distributing these as follows: 150 instances each for WebQSP and CWQ, 100 instances for KQA Pro, and 300 instances for MetaQA. The statistical details of the test dataset and our annotated (Anno) dataset are presented in Table 1. Our sampled dataset presents a balanced composition, enhancing the assessment of models' capabilities in handling complex questions.

4.2 Baselines

To comprehensively evaluate our approach, we have selected a range of prior state-of-the-art (SOTA) baseline models.

Fine-tuning (FT) on full data. We selected semantic parsing (SP)-based methods as baselines. DeCAF (Yu et al., 2023) generates both logical forms and direct answers to provide the final outcomes. For KQA Pro, we opted for the BART-SPARQL (Cao et al., 2022). For MetaQA, we choose Edge-aware (Zhang et al., 2022b).

Prompting methods. We chose KB-BINDER (Li et al., 2023a) which leverage large language models (LLMs) in a few-shot setting. Additionally, we compared the efficacy of Chain-of-Thought prompting (CoT prompt) (Wei et al., 2022) alongside the technique of Self-Consistency (SC) (Wang et al., 2023), further enriching our analysis. It's important to note that StructGPT (Jiang et al., 2023b) and ToG (Sun et al., 2024) operate under the assumption that golden entities have been provided. Consequently, the comparison of these methodologies is presented in Section 4.5.

FT on low-resource data. We fine-tuned open-source LLMs (open-LLMs) to directly generate SPARQL queries through supervised fine-tuning (SFT) manner. We reimplemented a prior SOTA approach for each dataset to serve as a comparison.

4.3 Evaluation Metrics

SP-based methods generate logical forms, which consequently produce answers in an unordered manner. These should be evaluated using the F1 score[5]. Additionally, we also report the Random Hits@1 (RHits@1)[6] (Shu et al., 2022), and the Exact Match (EM) metric (Talmor and Berant, 2018) for reference. For KQA Pro, we report the accuracy, defined as the condition where the two sets match exactly.

[5]We use the average of the F1 scores across all instances.

[6]Randomly selecting an answer for each question 100 times and calculating the average Hits@1.

4.4 Results

Prompting with GPT-4 Turbo Table 2 and Table 3 present comprehensive comparisons. As indicated in the table, compared to methods that utilize full data, our approach is at a natural disadvantage on the WebQSP and KQA Pro datasets due to the significant difference in the magnitude of training data (~3K/~33K vs. 4-shot/2-shot, respectively). However, on CWQ and MetaQA datasets, our approach overall outperforms baselines. Furthermore, in the cases of comparative and superlative question types, our approach achieves improvements of 29.85% and 13.96%, respectively. This can be attributed to the fact that within the original dataset, these two types of questions each constitute merely 5% (see Table 11). Unlike previous methods, our approach is not constrained by data distribution, rendering it more robust. For conjunction questions in CWQ, CoT+SC and Decaf outperform our method due to redundant constraints inherent in these queries. Such questions require multiple entity constraints (e.g., "What country in the Mediterranean has Zonguldak Province?") yet often only one is needed to pinpoint the answer ("Zonguldak Province" leads to "Turkey", rendering "country in the Mediterranean" extraneous). Our approach, which focuses on identifying each predicate, complicates the response process.

FT with open-LLMs The results show that our method outperforms all baselines in low-resource scenarios. Specifically, for evaluations involving two question types from CWQ and two types from KQA Pro, the fine-tuned model exceeded the performance of GPT-4 Turbo.

4.5 Impact of Entity Linking (EL)

The EL performance of our methods[8] (with GPT-4 Turbo) is delineated in Table 4. The table reports a comparison of the results of our method with those of the widely recognized entity linking tool, ELQ (Li et al., 2020), on the WebQSP and CWQ datasets. Moreover, we introduce the Mention Cover Rate (MCR) to quantify the difficulty of EL. MCR is defined as the rate at which the golden entity names appear within the questions. As shown in our analysis, KQA Pro and MetaQA exhibit higher MCR. It is noted that some entity constrains are redundant in KQA Pro[9]. Therefore, despite the F1 score being 75.35%, this is not a bottleneck affecting the performance of KQA Pro. Based on these observations, we conducted experiments on the WebQSP and CWQ datasets under the conditions of given golden entities. The results, as presented in Table 5, reveal that our approach significantly outperforms the previous SOTA across both datasets.

4.6 Ablation Study

We perform various ablation studies to understand the importance of different factors in Interactive-KBQA.

Impact of Exemplars This section conducts an ablation study to examine the influence of exemplars in two representative settings: CWQ, which includes four types and can barely cover all question types within the prompt text, and KQA Pro, which includes nine types and cannot cover all types due to high inference costs.

The performance of classifiers (cls) trained with our annotated dataset is shown in Table 6. Experiments were conducted in 0-shot and 4-shot scenarios on a randomly selected test subset of 100 entries. For CWQ, one exemplar per question type was used, while for KQA Pro, only four types were sampled. In the zero-shot scenario, only the instruction text was used as a prompt. As Table 7 shows, our method's performance improves with increased question type coverage in prompts. For CWQ 4-shots, we observed a 2.5 point performance increase, but the cost rose by 37.86%, from $0.5 to $0.7. KQA Pro results suggest that accurate demonstrations improve performance and reduce costs. For a detailed discussion about the interaction rounds and costs, see Appendix A.5.

Impact of Backbone Model As demonstrated in Table 8, GPT-4 significantly outperforms GPT-3.5[11] in terms of reasoning capabilities. We have also attempted to apply direct reasoning with Mistral 7B. It was observed that the untrained model significantly struggles with complex, multi-turn interactions. This finding underscores the substantial improvements that fine-tuning provides.

[7] The output of StructGPT consists of a string that contains answers, making it unsuitable for evaluation with RHits@1.

[8] We find the entity surface name rather than mid.

[9] For instance, the answer of "What is the connection between Steve Jordan (the one whose position is tight end) and Phoenix (the one that is the twinned administrative body of Chengdu)?" is equivalent to that of "What is the connection between Steve Jordan and Phoenix?".

[10] The maximum context of Llama 2, which is 4,096 tokens, is insufficient for direct inference.

[11] We use gpt-3.5-turbo-1106.

Method		WebQSP				CWQ					
		1-hop	2-hop	Overall	RHits@1	Conj	Compo	Compa	Super	Overall	EM
Prior FT SOTA	DeCAF †	74.72	76.32	75.52	80.28	69.19	53.54	18.04	28.00	42.19	50.83
Prompting SOTA	KB-BINDER♯	-		74.40	-			-			
Prompting w/GPT-4 Turbo	IO	28.54	50.05	39.29	45.51	47.54	29.71	33.66	24.67	33.89	45.67
	CoT	27.85	51.55	39.70	47.52	44.12	26.30	34.39	30.00	33.70	43.67
	CoT+SC	26.66	51.35	39.01	47.08	**50.65**	28.78	36.98	29.78	36.55	47.50
	Ours	**69.99**	**72.41**	**71.20**	**72.47**	47.44	**59.00**	47.89	41.96	**49.07**	**59.17**
Reimplement	DeCAF	24.56	27.55	27.55	39.33	32.19	10.16	11.63	6.00	15.00	19.50
Fine-tuning w/open-LLM	SFT-SPARQL (7B)	34.39	33.80	34.09	35.68	12.39	20.44	41.10	38.44	28.10	30.00
	Ours (7B)	42.02	45.03	43.57	45.09	31.90	30.70	50.98	46.03	39.90	44.00
	Ours (13B)	40.68	53.92	54.86	56.25	30.47	34.51	**55.98**	**49.06**	42.50	45.67

Method		KQA Pro									
		Ct	QA	QAQ	QN	QR	QRQ	SA	SB	Vf	Overall
Prior FT SOTA	BART-SPARQL †	89	92	87	77	95	81	96	94	86	88.56
Prompting w/GPT-4 Turbo	IO	27	23	36	40	25	50	11	69	73	39.33
	CoT	22	26	35	34	18	46	21	79	77	39.78
	CoT+SC	25	28	33	38	22	51	19	86	75	41.89
	Ours	**74**	**83**	64	**73**	73	**59**	**80**	61	**80**	**71.89**
Reimplement	BART-SPARQL	37	44	37	36	67	33	49	78	58	48.78
Fine-tuning w/open-LLM	SFT-SPARQL (7B)	52	51	52	47	69	37	60	**85**	67	57.78
	Ours (7B)	53	58	**69**	48	75	48	75	84	70	64.40
	Ours (13B)	63	65	**55**	49	**76**	52	68	75	62	62.78

Table 2: Results on WebQSP and CWQ. Results tagged with † denote data from original prediction files, but evaluated on consistent test data. Results with ♯ are reprinted from (Li et al., 2023a).

Method	MetaQA				
	1-hop	2-hop	3-hop	Overall	RHits@1
Edge-aware ♯	98.50	93.70	91.00	94.40	96.77
KB-BINDER ‡	82.15	91.26	99.66	91.02	86.52
Ours w/GPT-4	**96.75**	**98.47**	**93.55**	**96.25**	**95.97**
Ours w/SFT (7B)	93.89	85.99	95.61	91.83	91.41

Table 3: Results on MetaQA. The symbol ♯ denotes results reprinted from Zhang et al. (2022b), while ‡ indicates we reimplemented the results.

Dataset	Precision	Recall	F1	MCR
WebQSP	91.50	74.69	80.00	67.42
w/ELQ	93.67	38.69	41.30	
CWQ	87.92	70.67	76.06	76.64
w/ELQ	94.36	41.61	43.81	
KQA Pro	80.91	75.82	75.35	80.80
MetaQA	97.33	95.89	95.89	100.00

Table 4: Results of entity linking.

Methods	WebQSP		CWQ	
	Overall	RHits@1	Overall	EM
StructGPT †	44.26	-[7]	-	
ToG ‡	36.40	44.80	31.77	41.94
Ours	71.20	72.47	49.07	59.17
Ours w/golden	78.64	79.25	56.74	66.50
Gain	7.44	6.77	7.67	7.33

Table 5: Results on WebQSP and CWQ with golden entities. † indicates original predictions evaluated on a consistent dataset; ‡ denotes our reimplementation.

Dataset	Precision	Recall	F1
CWQ	92.76	92.83	92.79
KQA Pro	92.73	92.44	92.42

Table 6: The performances of question type classifiers.

Setting	CWQ		KQA Pro	
	F1	AP	Acc	AP
Ours(cls+2-shot)	54.69	$0.50	67.78	$0.38
w/4-shot α	57.19	$0.70	46.67	$0.50
w/4-shot β	-		52.22	$0.55
zero-shot	51.83	$0.37	25.25	$0.30

Table 7: The impact of exemplar number and the average price (AP). The KQA Pro configuration α includes QN, QR, QRQ, and Vf, Whereas β comprises Ct, QAQ, SA, and SB.

Model	CWQ	KQA Pro
OpenAI		
GPT-4 Turbo	49.07	71.89
GPT-3.5 Turbo	13.42	47.92
open-source LLM		
Mistral-7B FT	39.90	62.24
w/o FT	4.76	20.41
Llama 2 7B FT [10]	30.42	66.33
Llama 2 13B FT	42.50	62.78

Table 8: Performance of different backbone models

Error Type	WCWQ	KQA Pro
Entity Linking	18	7
Predicate Search	6	0
Reasoning Error	32	48
Format Compliance	17	15
Hallucination	19	21
Other	8	9

Table 9: Distribution of error types.

amalgamate = 혼합하다, 합병하다

4.7 Error Analysis

To systemically assess our method's deficiencies, we first amalgamate WebQSP+CWQ (WCWQ) and randomly select 100 error instances from each for manual inspection. The aggregated statistical findings are detailed in Table 9.

- **Entity Linking Error** refers to the failure to locate nodes using the SearchNodes tool. The primary cause of this error is the LLM's failure to extract the correct entity mention from the question. For example, in the question "Most Anticipated Tour at the Young Hollywood Awards.", the entity should have been identified as "Young Hollywood Award for Most Anticipated Tour", but instead, the LLM mistakenly searched for "Young Hollywood Awards". Besides, In CWQ and WebQSP, the entities in the returned results do not have descriptions, leading the LLM to consider the results incorrect and subsequently re-initiate the search.
- **Predicate Search Error** denotes the failure of the SearchGraphPatterns tool to return the necessary information. This error arises when vector search tools fail to return expected results, such as predicates indicating an organization's headquarters. For instance, the query "locate in" is unable to match the predicate "organization.organization.headquarters".
- **Reasoning Error** means that, given the observations, the LLM fails to generate the appropriate SPARQL query. This failure is largely due to an insufficient semantic understanding of the KG schema. Typically, the LLM often fails to understand the CVT structure (e.g. the direction of the predicate), resulting in incomplete or inaccurate SPARQL queries.
- **Format Compliance Error** implies that the LLM does not use the tool in the required format. Examples include incorrectly constructed SPARQL queries with improperly formatted time and numerical values, and the introduction of unrecognized additional parameters to the tool.
- **Hallucination** includes generating elements that are inconsistent with the observations.
- **Other Error** encompasses errors that cannot be categorized under the above types.

More details are presented in the form of case study in the Appendix A.8.

5 Conclusion

Interactive-KBQA introduces a KBQA approach which utilizes an LLM as an agent for performing semantic parsing through multi-round interactions with a KB. Initially, we developed a unified tool and an interaction methodology tailored to various DB schemas. Subsequently, by categorizing complex questions and annotating a minimal set of exemplars, we employed a few-shot learning strategy that guides the LLM to incrementally generate SPARQL queries. Moreover, we introduced a low-resource dataset that demonstrates superior performance when fine-tuned with open-source LLMs.

Limitations

The prompt learning-based approach heavily relies on the capabilities of LLMs, and in scenarios involving multiple rounds of dialogue, the cost of reasoning becomes significantly high. Additionally, it is impractical to adjust the output of LLM when invoking LLM APIs. Therefore, this paper proposes a collaborative human-machine annotation method to mitigate this issue.

Acknowledgments

This work was supported by the National Key Research and Development Program of China under Grant No. 2023YFC3304404. We would also like to thank to Prof. Zhiyuan Liu and Dr. Yujia Qin from THUNLP group for their invaluable suggestions.

References

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, page 1247–1250, New York, NY, USA. Association for Computing Machinery.

Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. 2022. KQA pro: A dataset with explicit

compositional programs for complex question answering over knowledge base. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6101–6119, 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, Online. Association for Computational Linguistics.

Wenhu Chen. 2023. Large language models are few(1)-shot table reasoners. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1120–1130, Dubrovnik, Croatia. Association for Computational Linguistics.

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 Joint Conference on Artificial Intelligence*, IJCAI'20.

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 the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 345–356, Minneapolis, Minnesota. Association for Computational Linguistics.

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, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Guanting Dong, Rumei Li, Sirui Wang, Yupeng Zhang, Yunsen Xian, and Weiran Xu. 2023. Bridging the kb-text gap: Leveraging structured knowledge-aware pre-training for kbqa. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23, page 3854–3859, New York, NY, USA. Association for Computing Machinery.

Yu Gu, Xiang Deng, and Yu Su. 2023. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4928–4949, Toronto, Canada. Association for Computational Linguistics.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond i.i.d.: Three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, WWW '21, page 3477–3488, New York, NY, USA. Association for Computing Machinery.

Yu Gu, Yiheng Shu, Hao Yu, Xiao Liu, Yuxiao Dong, Jie Tang, Jayanth Srinivasa, Hugo Latapie, and Yu Su. 2024. Middleware for llms: Tools are instrumental for language agents in complex environments.

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

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023a. Mistral 7b.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023b. 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, Singapore. Association for Computational Linguistics.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. 2024. Kg-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph.

Jinhao Jiang, Kun Zhou, Xin Zhao, and Ji-Rong Wen. 2023c. UniKGQA: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. In *The Eleventh International Conference on Learning Representations*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA. Association for Computing Machinery.

Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Complex knowledge base question answering: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(11):11196–11215.

Yunshi Lan and Jing Jiang. 2020. 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.

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)*, pages 6433–6441, Online. Association for Computational Linguistics.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023a. 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, Toronto, Canada. Association for Computational Linguistics.

Zhenyu Li, Sunqi Fan, Yu Gu, Xiuxing Li, Zhichao Duan, Bowen Dong, Ning Liu, and Jianyong Wang. 2023b. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*, abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3505–3506, New York, NY, USA. Association for Computing Machinery.

Yiheng Shu, Zhiwei Yu, Yuhan Li, Börje Karlsson, Tingting Ma, Yuzhong Qu, and Chin-Yew Lin. 2022. TIARA: Multi-grained retrieval for robust question answering over large knowledge base. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8108–8121, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. Roformer: Enhanced transformer with rotary position embedding.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, 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*.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In

The Eleventh International Conference on Learning Representations.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Yuanmeng Yan, Rumei Li, Sirui Wang, Hongzhi Zhang, Zan Daoguang, Fuzheng Zhang, Wei Wu, and Weiran Xu. 2021. Large-scale relation learning for question answering over knowledge bases with pre-trained language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3653–3660, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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, Beijing, China. Association for Computational Linguistics.

Wen-tau Yih, Matthew Richardson, Chris 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, Berlin, Germany. Association for Computational 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*.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big bird: transformers for longer sequences. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA. Curran Associates Inc.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. 2022a. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5773–5784, Dublin, Ireland. Association for Computational Linguistics.

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)*, pages 1002–1017, Toronto, Canada. Association for Computational Linguistics.

Yanan Zhang, Li Jin, Xiaoyu Li, and Honqi Wang. 2022b. Edge-aware graph neural network for multi-hop path reasoning over knowledge base. *Comput. Intell. Neurosci.*, 2022:4734179.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J. Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press.

Dawei Zhu, Nan Yang, Liang Wang, Yifan Song, Wenhao Wu, Furu Wei, and Sujian Li. 2023. Pose: Efficient context window extension of llms via positional skip-wise training.

Chang Zong, Yuchen Yan, Weiming Lu, Jian Shao, Eliot Huang, Heng Chang, and Yueting Zhuang. 2024. Triad: A framework leveraging a multi-role llm-based agent to solve knowledge base question answering.

A Appendix

This appendix provides detailed experimental results and offers further discussion.

A.1 Additional Statistics of Datasets & Databases

Database	#Node	#Rel
Freebase	22,767,149	20,815
Wikidata	17,754	1,267
Movie KG	38,131	9

Table 10: Statistics of the databases.

Question Type	Dist. (Train/Dev/Test)
conjunction	42.00 / 43.59 / 44.60
composition	47.27 / 44.76 / 43.78
comparative	5.54 / 6.22 / 6.03
superlative	5.19 / 5.43 / 5.58

Table 11: Distribution (Dist.) of question types in the original CWQ.

Table 10 presents the statistical information for Freebase, Wikidata, and Movie KG. For Freebase, we utilize a script[12] to eliminate non-english triples. In the case of Wikidata, we refer to the subset processed by KQA Pro (Cao et al., 2022). Meanwhile, Movie KG is derived by converting the knowledge base provided by MetaQA into RDF triples. Additionally, Table 11 outlines the distribution of question types in the original CWQ.

A.2 System Configurations

A.2.1 Implementation Details

In the interactive process, our study invokes the OpenAI GPT4-Turbo (`gpt-4-1106-preview`) API to serve as a LLM agent. For each type of questions, we selected and annotated two cases with complete interactive processes as exemplars for in-context learning. During the inference stage, for the WebQSP and MetaQA datasets, all annotated demonstrations are employed as few-shot learning examples, specifically, 4-shot for WebQSP and 6-shot for MetaQA. For CWQ and KQA Pro, we train a `bert-base-uncased` [13] model based on low-resource training data as a question type classifier (cls), and then select two exemplars based on the predicted question type, namely cls+2-shot for both CWQ and KQA Pro.

In the development of the **SearchNodes** tool, Elasticsearch[14] is employed to extract all node surface names from the Freebase and MetaQA databases, and vector search techniques are implemented to perform queries on nodes within Wikidata. For the ranking algorithm of the **SearchGraphPatterns** tool, vector retrieval methods are similarly employed. All processes related to vectorization utilize the OpenAI `text-embedding-ada-002` API to generate vectors and employ Chroma[15] for indexing and searching. Moreover, for the functionality of the **ExecuteSPARQL** tool, Virtuoso[16] serves as the underlying graph query engine.

In the process of fine-tuning open-source LLMs, we utilize, `Mistral-7B-v0.1` (Jiang et al., 2023a) and `Llama-2-13b` (Touvron et al., 2023). For training optimization, DeepSpeed (Rasley et al., 2020) is employed, while inference tasks are accelerated using vLLM (Kwon et al., 2023). `Llama-2-13b` is trained on four NVIDIA A100 80GB GPUs, the other experiments are conducted on two NVIDIA A100 80GB GPUs.

A.2.2 Hyper-Parameter Setting

Parameter	Value
temperature	0.7
top_p	1
n	6 # 1 when annotating data
stop	["\nObservation", "\nThought"]
max_tokens	384

Table 12: Assignments of hyper-parameters for inference.

Table 12 presents the parameter configurations for invoking the OpenAI API and fine-tuning open-source LLMs. We employ DeepSpeed (Rasley et al., 2020) to efficiently fine-tune open-source LLMs. The hyperparameter settings utilized for training are detailed in Table 13.

For both the SearchNodes and SearchGraphPatterns tools, the number of returned results is set to 10. In the interaction process, the maximum number of turns is limited to 20, corresponding to 10 rounds.

[12] https://github.com/lanyunshi/Multi-hopComplexKBQA/blob/master/code/FreebaseTool/FilterEnglishTriplets.py

[13] https://huggingface.co/bert-base-uncased

[14] https://github.com/elastic/elasticsearch

[15] https://github.com/chroma-core/chroma

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

Parameter	Value
batch size (per GPU)	1
model max length	4096
learning rate	1e-5
weight decay	0.001
epochs	10
warm up steps	0
gradient accumulation steps	4
zero stage	3

Table 13: Assignments of hyper-parameters for fine-tuning open-source LLMs.

A.2.3 Instruction Text

Instruction text consists of tool descriptions, tool usages and the format of interaction. The tools designed in this work is capable of processing a variety of complex questions through a unified approach on the different databases. Figures 3, 4, and 5 showcase comprehensive instruction texts for Freebase, Wikidata, and Movie KG, respectively. For further insight into the tool's applications, examples within these figures are highlighted in green.

A.3 Additional Results of Entity Linking

In Table 14 and Table 15, we report the impact of given golden entities on model performance by question type for WebQSP and CWQ, respectively.

A.4 Additional Results of Exemplars Selection

In Table 16 and Table 17, we report the impact of exemplar selection on model performance by question type for CWQ and KQA Pro, respectively. This examination delineates how the choice of exemplars influences the effectiveness of the model across question types, underscoring the significance of tailored exemplar selection in prompt engineering.

The observation shows that the performance of our proposed approach on KQA Pro drops significantly in zero-shot compared to few-shot scenarios as observed in Table 17. This discrepancy arises primarily due to the intricate nature of the questions within the KQA Pro dataset, which involve multi-hop reasoning, constraints, and qualifiers. Constructed using SPARQL templates and phrased naturally by human annotators, these questions are complex. In zero-shot settings, without exemplars, even human annotators may struggle with semantic parsing due to the varied and unseen KG schema. Conversely, the few-shot approach leverages a small set of carefully annotated examples, effectively guiding the parsing process and accounting for the observed performance differences.

A.5 Interaction Turns and Costs

In the process of interaction, the turn constitutes the fundamental unit, with each utterance made by any participant being recorded as a distinct turn. Table 18 provides a statistical analysis of the interaction turns. A case is deemed successful if the LLM agent explicitly generates `Action:Done`. Otherwise, the dialogue is concluded once the maximum turn number is reached. It is feasible to consider the turn count of human-annotated data as a lower bound. It is observed that, except for on CWQ, GPT-4 have approached this lower bound. Specifically, in CWQ, the overall success rate per turn for GPT-4 is 80.5/14.57, which, compared to WebQSP, shows a decrease of 14.66% and an increase of 39.96% in terms of success rate and turn count, respectively. This also delineates the difficulty level of the dataset from one perspective. In contrast, the KQA Pro dataset exhibits a stronger regularity, hence presenting a relatively lower level of difficulty. Furthermore, we discovered that on complicated question datasets, the average number of turns for fine-tuned LLMs is less than that for GPT-4, indicating that our annotated data effectively reduced the model's trial-and-error process by providing valuable information. Additionally, we report the average cost per dialogue for reference [17]. Given that we set the return sequence parameter to 6 for inference and to 1 for human annotation, the cost comparison is not strictly equivalent. Complete results categorized by question type are available in the Table 20, Table 21, and Table 22.

A.6 Impact of Interaction History Length

To investigate the impact of interaction history length on end-to-end performance, we analyzed the changes in F1 score as the number of dialogue turns increased, as presented in Figures 6. It was observed that as the number of dialogue turns increased, the model's performance generally deteriorated.

When given a question, you need to utilize specific tools to interact with a locally deployed Freebase knowledge graph and compose a SPARQL query to obtain the answer. The following document includes descriptions of the tools and some crucial graph patterns (KG schema).

1. SearchNodes(query)
Description: Searches for nodes in the knowledge graph based on the surface name.
Example 1: To find "obama": SearchNodes("obama") will return: ['"Barack Obama" | Description: president of the United States from 2009 to 2017.', ...].
Example 2: To find a village "Bush": SearchNodes("Bush") will return: ['"Bush" | Description: Bush are a British rock band formed in London in 1992. The band found their immediate success with t...| Description: Bush is a village in Williamson County, Illinois, United States. As of the 2000 census, the village ...|]. It means that there are multiple entities named "Bush" in the KG, you can use both name and description to distinguish them, for example: SELECT ?e WHERE { ?e ns:type.object.name "Bush"@en . ?e ns:common.topic.description ?desc . FILTER(REGEX(?desc, "Bush is a village in Williamson County, Illinois")) }

2. ExecuteSPARQL(sparql)
Description: Execute a SPARQL query. You can explore the KG freely using this tool.
Example 1: If you want to check the birthplace of "Barack Obama" is "Honolulu" or not, just: ExecuteSPARQL('ASK { ?e ns:type.object.name "Barack Obama"@en . ?e ns:people.person.place_of_birth ?e1 . ?e1 ns:type.object.name "Honolulu"@en . }').

3. SearchGraphPatterns(sparql, semantic)
Description: Parameter `sparql` MUST start with ["SELECT ?e WHERE"]. The tool will query the subgraphs with ?e as the head or tail entities, respectively, and return them together. The `semantic` parameter indicates the expected predicate semantics. If provided, the tool will sort the queried subgraphs by semantics. If not, the tool returns the entire subgraph. Note! This tool will return a randomly instantiated triple, you should pay attention to the semantics of the predicate, not the specific names of the head entity or the tail entity. In Freebase, due to the use of "Compound Value Type" (CVT) to represent an event, a one-hop relationship semantically requires two hops in Freebase. For example, the statement "Tom Hanks is an actor in the film Catch Me If You Can" is represented as follows: (?e0 type.object.name "Tom Hanks"@en . ?e0 film.actor.film ?c . ?c film.performance.film ?e1 . ?e1 type.object.name "Catch Me If You Can"@en .) Therefore, if you search "an actor stars in a film", this tool will return "film.actor.film -> film.performance.film" and you should consider it as one-hop relationship. You can write down more than one synonyms to the `semantic` parameter if necessary.
Example 1: If you want to find out the birthdate of "Barack Obama", you can: SearchGraphPatterns("SELECT ?e WHERE { ?e ns:type.object.name 'Barack Obama'@en . }", semantic="birthdate/natal day"), it will return: ['(?e, people.person.date_of_birth "1961-08-04"^^xsd:date), ...].
Example 2: If you want to find out the presidents of france, you need to find and confirm the two pieces of information in a cvt subgraph: the official title is "president" and the person is ?x. First, assume that you've found the node "France". Next, you are supposed to do: SearchGraphPatterns('SELECT ?e WHERE { ?e ns:type.object.name "France"@en . }', semantic="president of/governor/Chairman"). It will return: [..., (?e, government.governmental_jurisdiction.governing_officials -> government.government_position_held.office_holder, "Charles de Gaulle"@en) ...]. Attention! the tail entity "Charles de Gaulle" is instantiated randomly! Next, you are supposed to search: SearchGraphPatterns('SELECT ?e WHERE { ?e ns:type.object.name "France"@en . }', semantic="official title of government"). It will return: [..., (?e, government.governmental_jurisdiction.governing_officials -> government.government_position_held.basic_title, "President"@en), ...]. Now, you will know "the presidents of france" was expressed by a cvt subgraph and you can write the final SPARQL query: """SELECT DISTINCT ?x WHERE { ?e ns:type.object.name "France"@en . ?e ns:government.governmental_jurisdiction.governing_officials ?c . ?c ns:government.government_position_held.basic_title ?title . ?title ns:type.object.name"President"@en . ?c ns:government.government_position_held.office_holder ?x .}"""

Remember:
`Action:` must be provided in every round of conversation, and must be one of [SearchNodes, SearchGraphPatterns, ExecuteSPARQL, Done].
ALWAYS construct the complete SPARQL query as the final answer and use `Action: ExecuteSPARQL('SELECT DISTINCT ?x WHERE ...` to execute!
Do not fabricate predicates! Information must be obtained from observations.
Follow the demos' format strictly!

Note: The tool will automatically add the prefix required: "PREFIX ns: <http://rdf.freebase.com/ns/>", so you only need to use the abbreviation "ns:" in SPARQL.

Now, Think and solve the following complex questions step by step:

Figure 3: Instruction text of Freebase.

When presented with a question, you need to employ specific tools to interact with a locally deployed knowledge graph and craft a SPARQL query to retrieve the answer. The following document includes descriptions of the tools and some essential graph patterns (KG schema).

1. SearchNodes(query)
Description: Searches for nodes in the knowledge graph based on the surface name. There are two types of nodes: entities and concepts. You are suppose to use "xxx" or "xxx" to search for the nodes.
Example 1: To find "obama": SearchNodes("obama") will return: ['Barack Obama | an entity', ...]. If the entity "Barack Obama" is your target node, the SPARQL query should be: SELECT ?e WHERE { ?e <pred:name> "Barack Obama" . }.
Example 2: If you want to identify nodes associated with the concept name "visual art", initiate the search: SearchNodes("visual art") will return: ['visual artwork | a concept', ...]. If the concept "visual artwork" is your target node, the SPARQL should be: SELECT ?e WHERE { ?e <pred:instance_of> ?c . ?c <pred:name> "visual artwork" . }.

2. ExecuteSPARQL(sparql)
Description: Executes a SPARQL query. You can explore the KG freely using this tool.
Example 1: To verify whether the birthdate of "Barack Obama" is Aug 4, 1961 or not, simply use: ExecuteSPARQL('ASK { ?e <pred:name> "Barack Obama". ?e <date_of_birth> ?pv . ?pv <pred:date> "1961-08-04"^^xsd:date . }').
Example 2: Similar to Wikidata, this KG uses predicate modifiers to provide additional information. If you want to query the population of New York City in 2010, suppose you have obtained this graph pattern: "?e <pred:name> "New York City" . ?e <population> ?pv. ?pv <pred:value> ?v .", then, you need to enumerate the modifiers of <population>: ExecuteSPARQL('SELECT DISTINCT ?p (SAMPLE(?t) AS ?t) WHERE { ?e <pred:name> "New York City" . ?e <population> ?pv . [<pred:fact_h> ?e ; <pred:fact_r> <population> ; <pred:fact_t> ?pv] ?p ?t . }'), it will return: [('editor', 'United States Census Bureau'), ('point_in_time', '"1698-01-01"^^xsd:date'), ...], now, you can use the modifier <point_in_time> to craft the final SPARQL.
Example 3: If you want to identify a node with a predicate and a value, you can use the tool: ExecuteSPARQL('SELECT DISTINCT ?e WHERE { ?e <ISNI> ?pv . ?pv <pred:value> "0000 0001 0938 7748" . }')

3. SearchGraphPatterns(sparql, semantic)
Description: The parameter sparql MUST start with "SELECT ?e WHERE". The tool will query the subgraphs with ?e as the head or tail entities, respectively, and return them together. The semantic parameter indicates the expected predicate semantics. If provided, the tool will sort the queried subgraphs by semantics. If not, the tool returns the entire subgraph. Note! This tool will return a randomly instantiated triple; you should pay attention to the semantics of the predicate, not the specific names of the head entity or the tail entity. You can write down more than one synonyms to the `semantic` parameter if necessary.
Example 1: If you want to get the birthdate of "Barack Obama", just: SearchGraphPatterns("SELECT ?e WHERE { ?e <pred:name> 'Barack Obama' . }", semantic="birthdate"), it will return: [..., '?e <date_of_birth> ?pv . ?pv <pred:date> "1961-08-04"^^xsd:date .', ...].
Example 2: If you just want to find one that represents "website" among all the predicates: SearchGraphPatterns("SELECT ?e WHERE {}", semantic="website"), it will return: ['<official_website>', ...].

Here are some patterns of typed literals:
Finding a date or year:
- ?e <start_time> ?pv . ?pv <pred:date> "1992-08-15"^^xsd:date .
- ?e <start_time> ?pv . ?pv <pred:year> 1992 .
Finding a number in a certain unit of measurement:
- metre: ?e <elevation_above_sea_level> ?pv . ?pv <pred:value> "7.0"^^xsd:double . ?pv <pred:unit> "metre" .
- foot: ?e <elevation_above_sea_level> ?pv . ?pv <pred:value> "7"^^xsd:double . ?pv <pred:unit> "foot" .
And so on.

Remember:
`Action:` must be provided in every round of conversation, and must be one of [SearchNodes, SearchGraphPatterns, ExecuteSPARQL, Done].
ALWAYS construct the complete SPARQL query and use `Action: ExecuteSPARQL('SELECT DISTINCT ?x WHERE ...` to execute!
Do not fabricate predicates! Information must be obtained from observation.
The final answer should be a list with length 1, or a boolean value.
Follow the demos' format strictly!

Now, Think and solve the following complex questions step by step:

Figure 4: Instruction text of Wikidata.

When given a question, you need to utilize specific tools to interact with a locally deployed movie knowledge graph and compose a SPARQL query to obtain the answer. The following document includes descriptions of the tools and some crucial graph patterns (KG schema).

1. SearchNodes(query)
Description: Searches for nodes in the knowledge graph based on the surface name.
Example: To find "ginger rogers": SearchNodes("ginger rogers") will return: ['"ginger rogers" | A tag.', '"Ginger Rogers" | A person node.', ...], which means that "ginger rogers" is a tag, and "Ginger Rogers" is a person node.

2. ExecuteSPARQL(sparql)
Description: Executes a SPARQL query. You can explore the KG freely using this tool.
Example: To find the name of node <e0>, simply use: ExecuteSPARQL('SELECT ?n WHERE { <e0> <name> ?n . }').

3. SearchGraphPatterns(sparql, semantic)
Description: The parameter sparql MUST start with "SELECT ?e WHERE". The tool will query the subgraphs with ?e as the head or tail entities, respectively, and return them together. The semantic parameter indicates the expected predicate semantics. If provided, the tool will sort the queried subgraphs by semantics. If not, the tool returns the entire subgraph. Note! This tool will return a randomly instantiated triple; you should pay attention to the semantics of the predicate, not the specific names of the head entity or the tail entity.
Example: To find out the director of the movie "The Matrix", you can: SearchGraphPatterns('SELECT ?e WHERE { ?e <name> "The Matrix" . }', semantic="director of"), it will return: [(?e, starred_actors, "Keanu Reeves"), ...], therefore, "starred_actors" is the target predicate, and "Keanu Reeves" is one of the actors.

Critical pattern 1: tag and node are different.
If the tag is what you want, the SPARQL query is: SELECT ?e WHERE { ?e <has_tags> "ginger rogers" . }, if the node is what you want, the SPARQL query is: SELECT ?e WHERE { ?e <name> "Ginger Rogers" . }

Critical pattern 2: Pay attention to the direction of the SPARQL query.
If asked: SearchGraphPatterns('SELECT ?e WHERE { ?e <name> "aaa" . }', semantic="co-star")
If the observation is ("bbb", starred_actors, ?e), where "bbb" is the head node of ?e, so the next SPARQL query is supposed to be: SELECT ?e WHERE { ?e <starred_actors> ?e0. ?e0 <name> "aaa" . }.
Similarly, if the observation is (?e, starred_actors, "bbb"), where "bbb" is the tail node of ?e, so the next SPARQL query is supposed to be: SELECT ?e WHERE { ?e0 <starred_actors> ?e. ?e0 <name> "aaa" . }.

Critical pattern 3: Add a FILTER to the SPARQL query if it contains a ring structure.
For example, to find the co-actor of "aa", the SPARQL query should be: SELECT ?x WHERE { ?e0 <name> "aa" . ?e <starred_actors> ?e0 . ?e <starred_actors> ?x . FILTER(?x != ?e0) . }

Knowledge graph schema: This knowledge graph is relatively simple, and the schema is consists of 10 types of triples, each triple is consists of 3 elements: (head entity, predicate, tail entity), for example, (movie, directed_by, person) means that the head entity is a movie, the tail entity is a person, and the predicate is directed_by. The complete schema is as follows:
(movie, directed_by, person)
(movie, written_by, person)
(movie, starred_actors, person)
(movie, release_year, year)
(movie, in_language, string)
(movie, has_tags, string)
(movie, has_genre, string)
(movie, name, string)
(movie, has_imdb_votes, string)
(movie, has_imdb_rating, string)

Remember:
ALWAYS use ExecuteSPARQL to provide the final SPARQL query.
Finally, you MUST use "Action: Done." to stop.
Strictly adhere to the format of the demos!

Now, Think and solve the following complex questions step by step:

Figure 5: Instruction text of Movie KG for MetaQA.

WebQSP				
Methods	**1-hop**	**2-hop**	**Overall**	**RHits@1**
Ours	69.99	72.41	71.20	72.47
Ours w/golden	77.50	79.79	78.64	79.25
Gain	7.51	7.38	7.44	6.77

Table 14: The impact of given golden entities on WebQSP by question type

CWQ						
Methods	**Conj**	**Compo**	**Compa**	**Super**	**Overall**	**EM**
Ours	47.44	59.00	47.89	41.96	49.07	59.17
Ours w/golden entity	53.36	68.19	52.73	52.69	56.74	66.50
Gain	5.92	9.19	4.84	10.72	7.67	7.33

Table 15: The impact of given golden entities on CWQ by question type

A.7 Confusion Matrix of Classifiers

Table 7 illustrate the confusion matrices for question type classifiers based on `bert-base-uncased`.

A.8 Case Study

In Figure 8, 9, 10, and 11, we present four cases. Each illustrates a error and delineates the correction process with human assistance.

[17]The price of gpt-4-1106-preview: Input $0.01 / 1K tokens and Output $0.03 / 1K tokens

CWQ						
Setting	**Conj**	**Compo**	**Compa**	**Super**	**Overall**	**Ave Price**
Ours (cls+2-shot)	58.64	66.71	41.35	50.79	54.69	$0.50
w/4-shot	38.02	51.32	75.66	63.37	57.19	$0.70
zero-shot	75.86	52.92	32.86	44.00	51.83	$0.37

Table 16: The impact of exemplar selection on CWQ by question type

KQA Pro						
Setting	**Ct**	**QA**	**QAQ**	**QN**	**QR**	
Ours (cls+2-shot)	50	90	60	60	80	
w/4-shot α	20	70	40	40	50	
w/4-shot β	50	70	50	70	20	
zero-shot	18	64	9	55	0	
(continued)	**QRQ**	**SA**	**SB**	**VF**	**Overall**	**Ave Price**
Ours (cls+2-shot)	50	70	80	70	67.78	$0.38
w/4-shot α	50	50	10	90	46.67	$0.50
w/4-shot β	20	80	40	70	52.22	$0.55
zero-shot	0	45	9	27	25	$0.30

Table 17: The impact of exemplar selection on KQA Pro by question type

Method	**WebQSP**		**CWQ**	
	Overall	**Ave Price**	**Overall**	**Ave Price**
Human Anno	0.97 / 10.26	$0.16	1.34 / 11.59	
Ours w/GPT-4	94.33 / 11.01	$0.39	80.50 / 14.57	$0.49
Ours w/GPT-4 + Golden	95.33 / 9.17	$0.24	82.50 / 12.74	$0.38
Ours w/FT-Mistral	74.05 / 11.71	-	64.67 / 13.29	-
Method	**KQA Pro**		**MetaQA**	
	Overall	**Ave Price**	**Overall**	**Ave Price**
Human Anno	0.80 / 11.75	$0.23	0.08 / 7.13	$0.06
Ours w/GPT-4	93.89 / 11.91	$0.33	99.67 / 7.45	$0.14
Ours w/FT-Mistral	81.20 / 11.18	-	98.78 / 7.07	-

Table 18: Comparison of turn count and cost across human-annotated data (Human Anno) and four datasets. For Human Anno, the average number of human interventions and interaction turn are reported. Concerning model performance, the average success rate and interaction turn are reported.

Method	**WebQSP**			
	1-hop	**2-hop**	**Overall**	**Ave Price**
Human Anno	1.08 / 10.04	0.86 / 10.48	0.97 / 10.26	$0.16
Ours w/GPT-4	94.67 / 10.83	94.00 / 11.19	94.33 / 11.01	$0.39
Ours w/GPT-4 + Golden	98.00 / 8.47	92.67 / 9.87	95.33 / 9.17	$0.24
Ours w/FT Mistral-7B	77.86 / 11.36	70.47 / 12.04	74.05 / 11.71	-

Table 19: Comparison of turn count and cost across on WebQSP by question type.

Method	**CWQ**					
	Conj	**Compo**	**Compa**	**Super**	**Overall**	**Ave Price**
Human Anno	1.38 / 12.64	1.10 / 11.32	1.28 / 11.24	1.62 / 11.16	1.34 / 11.59	$0.28
Ours w/GPT-4	80.67 / 15.25	90.67 / 13.16	72.00 / 15.11	78.67 / 14.75	80.50 / 14.57	$0.49
Ours w/GPT-4 + Golden	78.00 / 13.55	91.33 / 11.40	75.33 / 13.59	85.33 / 12.44	82.50 / 12.74	$0.38
Ours w/FT Mistral-7B	49.33 / 14.71	60.00 / 13.69	68.67 / 12.71	80.67 / 12.05	64.67 / 13.29	-

Table 20: Comparison of turn count and cost across on CWQ by question type.

Method	KQA Pro					
	Ct	**QA**	**QAQ**	**QN**	**QR**	**QRQ**
Human Anno	0.86 / 11.28	0.74 / 10.76	1.16 / 14.04	0.96 / 11.12	0.30 / 10.44	0.80 / 14.04
Ours w/GPT-4	96.00 / 11.15	94.00 / 11.57	89.00 / 13.11	86.00 / 10.30	100.00 / 11.00	90.00 / 13.80
Ours w/FT Mistral-7B	94.00 / 10.36	73.00 / 10.70	75.00 / 12.63	64.00 / 10.03	85.00 / 10.20	69.00 / 13.38
(continued)	**SA**	**SB**	**Vf**	**Overall**	**Ave Price**	
Human Anno	0.28 / 10.72	0.90 / 12.64	1.20 / 10.68	0.80 / 11.75	$0.23	
Ours w/GPT-4	93.00 / 11.82	99.00 / 13.83	98.00 / 10.61	93.89 / 11.91	$0.33	
Ours w/FT Mistral-7B	83.00 / 10.78	96.97 / 12.35	91.00 / 10.36	81.20 / 11.18	-	

Table 21: Comparison of turn count and cost across on KQA Pro by question type.

Method	MetaQA				
	1-hop	**2-hop**	**3-hop**	**Overall**	**Ave Price**
Human Anno	0.02 / 6.88	0.12 / 7.22	0.10 / 7.28	0.08 / 7.13	$0.06
Ours w/GPT-4	99.67 / 7.15	100.00 / 7.14	99.33 / 8.08	99.67 / 7.45	$0.14
Ours w/FT Mistral-7B	99.00 / 7.01	99.33 / 7.11	98.00 / 7.10	98.78 / 7.07	-

Table 22: Comparison of turn count and cost across on MetaQA by question type.

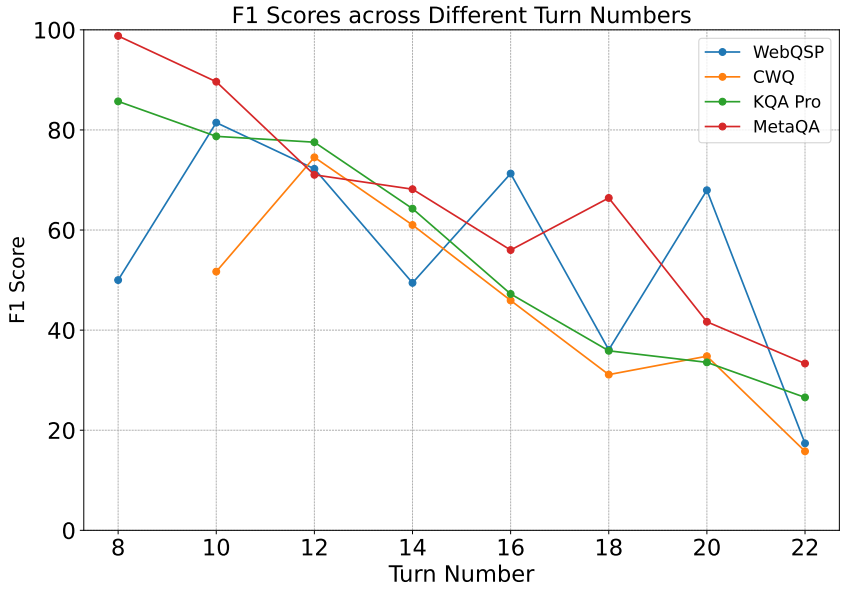

Figure 6: F1 Scores across Different Turn Numbers.

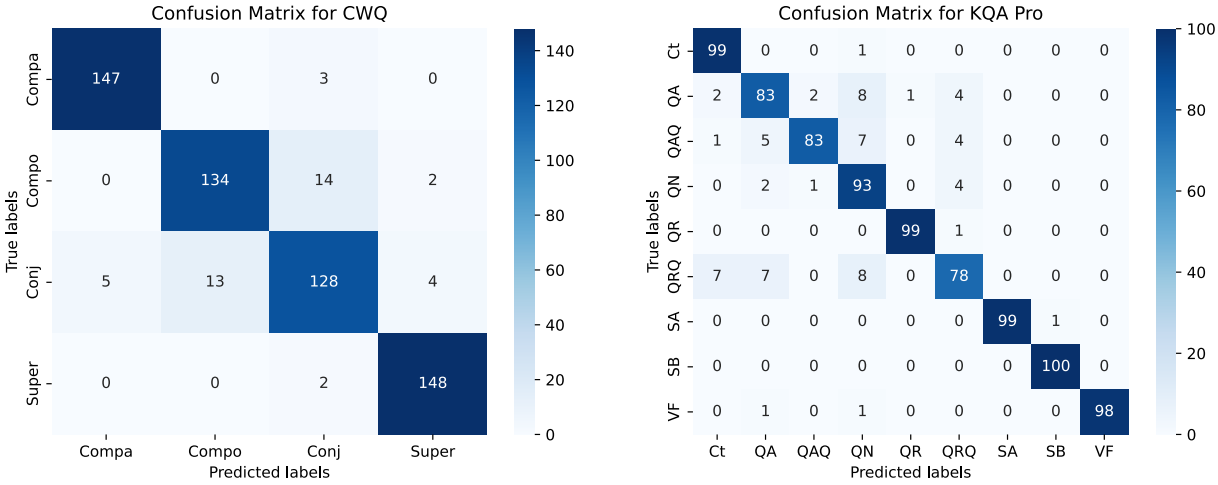

Figure 7: Confusion matrix of the classifier for CWQ (left) and KQA Pro (right).

Q: What tv programs shot in New York have an episode run time more than 30?

LLM: Thought: Let's first decouple this complex problem into simple sub-problem triples: (tv program, shot in, New York), (tv program, episode run time, ?v), (filter, ?v, > 30 minutes). According to my knowledge, the mention "New York" in Freebase may be "New York City". Next, I need to search it.
Action: SearchNodes("New York City")

Observation: [‘“New York City” | Description: New York – often called New York City or the City of New York to distinguish it from the State of Ne’, "New York city" | No description.', '"New york city" | No description.']

LLM: Thought: The correct node for “New York City” seems to be the first one. Next, I need to search for a predicate meaning "shot in" or "tv program location" in the sub-graph of the entity "New York City".
Action: SearchGraphPatterns('SELECT ?e WHERE { ?e type.object.name "New York City"@en }', semantic="tv program location/shot in")

Observation: [("The Stand", tv.tv_program.filming_locations, ?e), ("Old MacDonald Had a Curve", tv.tv_series_episode.filming_locations, ?e), (?e, tv.tv_location.tv_shows_filmed_here, "The Stand")]

❌ LLM: The predicate "tv.tv_location.tv_shows_filmed_here" seems to be the one that indicates TV shows filmed in a particular location. Now, I need to search for a predicate meaning "episode run time" in the sub-graph of TV programs.
Action: SearchGraphPatterns('SELECT ?e WHERE { ?e tv.tv_location.tv_shows_filmed_here ?e1 . ?e1 type.object.name "New York City"@en . }', semantic="episode run time")

✅ LLM: {Thought is the *same as above*}
Action: SearchGraphPatterns('SELECT ?e WHERE { ?e0 type.object.name "New York City"@en . ?e0 tv.tv_location.tv_shows_filmed_here ?e . }', semantic="episode run time")

Observation: [(?e, tv.tv_program.episode_running_time, "366"^^xsd:integer), (?e, tv.tv_program.episodes, "The Plague"), ("The Plague", tv.tv_series_episode.series, ?e)]

LLM: Thought: The predicate “tv.tv_program.episode_running_time” is the one I want. Now I can construct the SPARQL query.
Action: ExecuteSPARQL('SELECT DISTINCT ?show WHERE { ?loc type.object.name "New York City"@en . ?loc tv.tv_location.tv_shows_filmed_here ?show . ?show tv.tv_program.episode_running_time ?runtime . FILTER(xsd:integer(?runtime) > 30) }')

Observation: ['The Stand', 'Both Sides']

LLM: Thought: The answer to the question is "The Stand" and "Both Sides".
Action: Done.

Figure 8: Example of a reasoning error: Human assistance in understanding SPARQL logic.

Q: what was malcolm x trying to accomplish?

LLM: Thought: To answer the question about Malcolm X's goals, we need to find information related to his activism, beliefs, or objectives. Since Malcolm X was a prominent figure in the civil rights movement, we should look for nodes related to his advocacy, speeches, or roles in organizations. The entity name of "Malcolm X" in Freebase is likely "Malcolm X". First, we need to search for the node related to "Malcolm X".
Action: SearchNodes("Malcolm X")

Observation: ['"Malcolm X" | Description: Malcolm X, born Malcolm Little and also known as el-Hajj Malik el-Shabazz, was an American Muslim mi...', '"\'Malcolm X\'" | No description.', '"Malcolm X." | No description.', ...]

LLM: Thought: The first node with the description mentioning Malcolm X as an American Muslim minister and human rights activist is the correct node. Now, I need to search for a predicate that relates to Malcolm X's goals or what he was trying to accomplish.
Action: SearchGraphPatterns('SELECT ?e WHERE { ?e ns:type.object.name "Malcolm X"@en . }', semantic="goals or objectives/")

Observation: [(?e, book.written_work.subjects, “United States of America”), (“The Ballot or the Bullet”, event.presented_work.performances -> event.speech_or_presentation.speaker_s, ?e), (?e, film.film.subjects, “Racism”), (“Muslim Mosque, Inc.”, organization.organization.founders, ?e), (“Barrington Watson”, visual_art.visual_artist.artworks, ?e), ...]

❌ LLM: Thought: The observation includes various aspects of Malcolm X's life, including subjects he is associated with, speeches he has given, organizations he founded, and films about him. The most relevant information for his goals or objectives seems to be his speeches, such as "The Ballot or the Bullet,". These indicate his activism and the topics he was passionate about, such as fighting racism and advocating for the rights of African Americans...
Action: ExecuteSPARQL('SELECT ?goal WHERE { ?e ns:type.object.name "Malcolm X"@en . { ?e ns:book.written_work.subjects ?goal } UNION { ?e ns:film.film.subjects ?goal } UNION { ?speech ns:event.speech_topic.speeches_or_presentations_on_this_topic ?e . ?speech ns:type.object.name ?goal } UNION { ?org ns:organization.organization.founders ?e . ?org ns:type.object.name ?goal } }')

✅ LLM: {*Thought is the same as above*}. We should try a different approach, perhaps looking for his roles or influence in the civil rights movement.
Action: SearchGraphPatterns('SELECT ?e WHERE { ?e ns:type.object.name "Malcolm X"@en . }', semantic="civil rights activism")

Observation: [(?e, base.activism.activist.area_of_activism, “African Americans‘ rights”) ...]

LLM: Thought: ...
Action: ExecuteSPARQL('SELECT ?goal WHERE { ?e ns:type.object.name "Malcolm X"@en . ?e ns:base.activism.activist.area_of_activism ?goal . }')

Observation: ["African Americans' rights", 'Black Liberation']

LLM: Action: Done.

Figure 9: Example of a predicate search error: Human assistance in identifying the predicate.

Q: what is the name of justin bieber brother?

LLM: Thought: Let's first decouple this complex problem into several simple sub-problem triples: (Justin Bieber, sibling, target sibling), (target sibling, is a, brother). According to my knowledge, the entity name of "Justin Bieber" in Freebase is "Justin Bieber". So, I need to search it.
Action: SearchNodes("Justin Bieber")

Env: Observation: ['"Justin Bieber" | Description: Justin Drew Bieber is a Canadian singer and songwriter. Bieber\'s current manager, Scooter Braun, fir...', '"Baby (Justin Bieber)" | No description.', '"Justin Bieber Concerts" | No description.', ...]

LLM: Thought: "Justin Bieber" is the node I want. Next, I need to search for a predicate meaning "has a sibling".
Action: SearchGraphPatterns('SELECT ?e WHERE { ?e ns:type.object.name "Justin Bieber"@en . }', semantic="siblings of a person")

Observation: [(?e, people.person.sibling_s -> people.sibling_relationship.sibling, "Jazmyn Bieber"), ...] *# people.person.gender is not included here.*

❌ LLM: Thought: The predicate "people.person.sibling_s -> people.sibling_relationship.sibling" represents the relationship between a person and their sibling. The observation shows "Jazmyn Bieber" as a sibling, but we are looking for Justin Bieber's brother, not sister. Since the observation does not directly show a brother, I need to construct a SPARQL query to find the male sibling of Justin Bieber.
Action: ExecuteSPARQL('SELECT ?x WHERE { ?e ns:type.object.name "Justin Bieber"@en . ?e ns:people.person.sibling_s ?s . ?s ns:people.sibling_relationship.sibling ?x . ?x ns:people.person.gender ns:m.1l0bb . }')

✅ LLM: {*Thought is the same as above*}.
Action: SearchGraphPatterns('SELECT ?e WHERE { ?e0 ns:type.object.name "Justin Bieber"@en . ?e0 ns:people.person.sibling_s ?s . ?s ns:people.sibling_relationship.sibling ?e . }', semantic="gender")

Observation: [(?e, people.person.gender, "Female"), ...]

LLM: Thought: The predicate "people.person.gender" is used to indicate the gender of a person. The observation shows a gender value of "Female" for a sibling, which is not the brother we are looking for. Since we are looking for Justin Bieber's brother, we need to find a male sibling.
Action: ExecuteSPARQL('SELECT ?name WHERE { ?jb ns:type.object.name "Justin Bieber"@en . ?jb ns:people.person.sibling_s ?siblingship . ?siblingship ns:people.sibling_relationship.sibling ?brother . ?brother ns:people.person.gender ?gender . ?gender ns:type.object.name "Male"@en . ?brother ns:type.object.name ?name . }')

Observation: ['Jaxon Bieber']

LLM: Thought: The observation confirms that "Jaxon Bieber" is the name of Justin Bieber's brother, which is the answer to the question.
Action: Done.

Figure 10: Example of a hallucination error: Human intervention to rectify the hallucination.

Q: CBS Corporation owns what percentage of The CW?

LLM: Thought: Let's first decouple this complex problem into several simple sub-problem triples: (CBS Corporation, owns, The CW), (owns, percentage, target value). Next, I need to search for a node about "CBS Corporation".
Action: SearchNodes("CBS Corporation")

Env: Observation: ['CBS Corporation | entity', 'CBS News | entity', 'CBS Radio | entity', ...]

LLM: Thought: "The CW" is the entity I need. Next, I need to search for a predicate pattern that can represent the ownership percentage that CBS Corporation has in The CW.
Action: SearchGraphPatterns('SELECT ?e WHERE { ?e <pred:name> "CBS Corporation" . }', semantic="owns/ownership/percentage")

Env: Observation: ['?h <owned_by> ?e . ?h <pred:name> "UPN" .', '?e <net_profit> ?pv . ?pv <pred:value> "-1.1673e+10"^^xsd:double . ?pv <pred:unit> "United States dollar" .', '?h <operator> ?e . ?h <pred:name> "The CW" .', '?e <operating_income> ?pv . ?pv <pred:value> "-1.215e+10"^^xsd:double . ?pv <pred:unit> "United States dollar" .', '?e <subsidiary> ?t . ?t <pred:name> "CBS Radio" .', '?e <award_received> ?t . ?t <pred:name> "Peabody Award" .', '?e <follows> ?t . ?t <pred:name> "Viacom" .', '?h <parent_organization> ?e . ?h <pred:name> "UPN" .', '?e <employees> ?pv . ?pv <pred:value> "25920.0"^^xsd:double . ?pv <pred:unit> "1" .', '?e <industry> ?t . ?t <pred:name> "television" .']

❌ LLM: Thought: The pattern that represents the ownership of CBS Corporation in The CW is not directly visible in the observations. However, the pattern '?h <operator> ?e . ?h <pred:name> "The CW" .' suggests a relationship between CBS Corporation and The CW. Since the direct ownership percentage is not listed, I may need to look for a predicate that indicates the ownership share or percentage.
Action: SearchGraphPatterns('SELECT ?e WHERE { ?e <pred:name> "The CW" . }', semantic="owned by/ownership/percentage/share")

✅ LLM: Thought: <owned_by> is the predicate meaning "owned by". Since the direct ownership percentage is not listed, I think that may be a modifier of the predicate <owned_by>. Now, I need to find the correct modifier.
Action: ExecuteSPARQL('SELECT DISTINCT ?p (SAMPLE(?t) AS ?t) WHERE { ?e1 <pred:name> "The CW" . ?e2 <pred:name> "CBS Corporation" . ?e1 <owned_by> ?e2 . [<pred:fact_h> ?e1 ; <pred:fact_r> <owned_by> ; <pred:fact_t> ?e2] ?p ?t . }')

Env: Observation: [('pred:fact_r', 'owned_by'), ('pred:fact_h', 'The CW'), ('pred:fact_t', 'CBS Corporation'), ('proportion', '"0.5"^^xsd:double')]

LLM: Thought: ...
Action: ExecuteSPARQL('SELECT DISTINCT ?t WHERE { ?e1 <pred:name> "The CW" . ?e2 <pred:name> "CBS Corporation" . ?e1 <owned_by> ?e2 . [<pred:fact_h> ?e1 ; <pred:fact_r> <owned_by> ; <pred:fact_t> ?e2] <proportion> ?t . }')

Env: Observation: ['"0.5"^^xsd:double']

LLM: Thought: The observation is the answer I want, which represents the ownership percentage as a decimal.
Action: Done.

Figure 11: Example of a reasoning error: Human guidance in locating the qualifier.