Learning to Plan for Retrieval-Augmented Large Language Models from Knowledge Graphs

Junjie Wang[1,2,5*], Mingyang Chen[3*], Binbin Hu[2,5], Dan Yang[2,5], Ziqi Liu[2,5], Yue Shen[2,5], Peng Wei[2,5], Zhiqiang Zhang[2,5], Jinjie Gu[2,5], Jun Zhou[2,5], Jeff Z. Pan[4], Wen Zhang[1,5†], Huajun Chen[1,5†]

[1]Zhejiang University, [2]Ant Group, [3]Baichuan Inc., [4]The University of Edinburgh
[5]Zhejiang University - Ant Group Joint Laboratory of Knowledge Graph
{wangjj2018,zhang.wen,huajunsir}@zju.edu.cn, chenmingyang@baichuan-inc.com
http://knowledge-representation.org/j.z.pan/
https://github.com/zjukg/LPKG

Abstract

Improving the performance of large language models (LLMs) in complex question-answering (QA) scenarios has always been a research focal point. Recent studies have attempted to enhance LLMs' performance by combining stepwise planning with external retrieval. While effective for advanced models like GPT-3.5, smaller LLMs face challenges in decomposing complex questions, necessitating supervised fine-tuning. Previous work has relied on manual annotation and knowledge distillation from teacher LLMs, which are time-consuming and not accurate enough. In this paper, we introduce a novel framework for enhancing LLMs' planning capabilities by using planning data derived from knowledge graphs (KGs). LLMs fine-tuned with this data have improved planning capabilities, better equipping them to handle complex QA tasks that involve retrieval. Evaluations on multiple datasets, including our newly proposed benchmark, highlight the effectiveness of our framework and the benefits of KG-derived planning data.

1 Introduction

The past few years have witnessed significant innovations in LLMs (Ouyang et al., 2022; Touvron et al., 2023; Chowdhery et al., 2023; AI@Meta, 2024). While LLMs excel in many natural language processing tasks, they still face challenges, particularly the smaller models, in handling complex question-answering (QA) tasks (Press et al., 2023; Shao et al., 2023; Yao et al., 2022; Xiong et al., 2024a; Huang et al., 2024).

To improve the performance of LLMs on complex QA tasks, past research has tried various methods: (1) Employing carefully designed prompt strategies to guide the model in reasoning, such as Chain of Thought (CoT) (Kojima et al., 2022; Wei et al., 2022) and Tree of Thought (ToT) (Yao et al., 2024) methods; (2) Utilizing retrieval techniques to obtain supplemental information from external knowledge source (Lewis et al., 2020; Guu et al., 2020); (3) Combining prompt strategies with retrieval enhancements, as exemplified by methods like ReAct (Yao et al., 2022) and Self-Ask (Press et al., 2023). The third approach has garnered widespread research interest due to its integration of the advantages of the first two methods. The fundamental idea of this class of methods is to guide LLMs in breaking down a complex question into multiple simpler sub-questions and then use a retrieval-augmented generation (RAG) (Huang et al., 2023, 2024) method to answer each sub-question, thereby deducing the answer to the original complex question. However, planning for complex questions is non-trivial, especially for smaller LLMs (with fewer than 10 billion parameters), which often require supervised fine-tuning (Aksitov et al., 2023; Chen et al., 2023a; Qin et al., 2023).

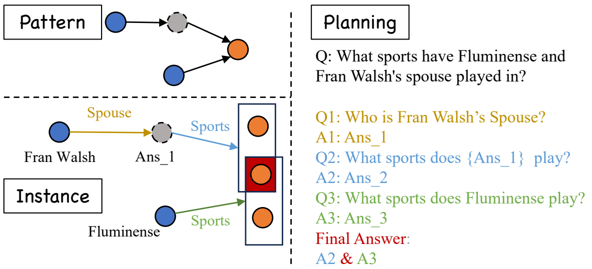

Pattern
Planning
Q: What sports have Fluminense and Fran Walsh's spouse played in?
Spouse
Sports
Fran Walsh
Ans_1
Q1: Who is Fran Walsh's Spouse?
A1: Ans_1
Q2: What sports does {Ans_1} play?
A2: Ans_2
Q3: What sports does Fluminense play?
A3: Ans_3
Final Answer:
A2 & A3
Instance
Sports
Fluminense

Figure 1: An example of a KG pattern, its grounded instance, and verbalized planning process.

This raises a widely concerning issue: how to obtain supervised data for learning the planning ability on complex questions. Manual annotation is time-consuming and labor-intensive, making it difficult to scale. Most existing methods attempt to distill knowledge from teacher LLMs (Yao et al., 2022; Aksitov et al., 2023), which places excessive

[*] Equal contribution.
[†] Corresponding authors.

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 7813–7835
November 12-16, 2024 ©2024 Association for Computational Linguistics

trust in the teacher LLMs and, in reality, cannot guarantee the accuracy of the distilled knowledge. These challenges inspire us to explore new ways of obtaining supervised planning data.

Knowledge Graphs (KGs) (Pan et al., 2017b,a) usually store accurate knowledge in a structured way. We find that a KG pattern can be viewed as the abstract of a complex question, as shown in Figure 1, which reveals the connection between question planning and patterns. This opens up the possibility of constructing training data to enhance the planning capabilities of LLMs using KGs. Specifically, we start by grounding predefined patterns in an open-domain KG to extract numerous instances, which we then verbalize into complex questions and corresponding sub-questions in natural language. In this way, we effectively create a large number of accurate planning data for fine-tuning. Being fine-tuned with these planning data, LLMs' capability of generating plans for complex questions is enhanced, resulting in better final answers by parsing and executing these plans. We refer to this innovative framework as **L**earning to **P**lan from **K**nowledge **G**raphs (LPKG).

Additionally, we construct a **C**omprehensive **L**ogical **QA** benchmark, CLQA-Wiki, from a subset of Wikidata (Vrandecic and Krötzsch, 2014) via grounding rich patterns as aforementioned. Existing complex QA benchmarks (Yang et al., 2018; Ho et al., 2020; Press et al., 2023; Trivedi et al., 2022) primarily focus on multi-hop and comparison-type questions and lack logical operations. Furthermore, most questions are labeled with only one answer, whereas in reality, they often have multiple correct answers. The CLQA-Wiki benchmark evenly covers multi-hop, comparison, intersection, and union types of questions, which is more comprehensive and challenging for complex QA evaluation.

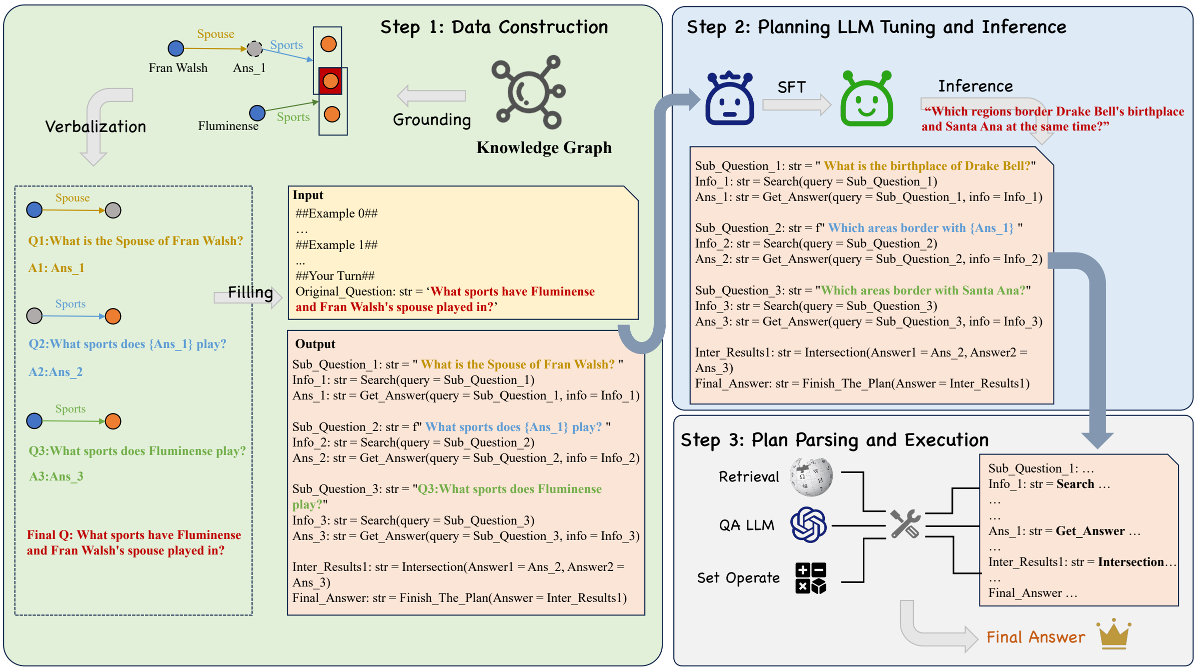
Our contributions can be summarized as follows: (1) We introduce a novel framework LPKG that enhances the planning ability of LLMs using data constructed from KG patterns; (2) We develop a comprehensive and challenging evaluation benchmark, named CLQA-Wiki, to more effectively assess the performance of LLMs on complex QA tasks; (3) Our proposed framework LPKG achieves better results than popular baselines on multiple conventional complex QA benchmarks, and we verify the effectiveness of the introduction of KG-sourced planning data.

2 Related Works

Reasoning and Planning with LLMs In the context of LLMs, reasoning typically involves decomposing complex questions into sub-questions (Mialon et al., 2023; Hao et al., 2023). Prominent techniques include Chain-of-Thought (CoT) prompting (Wei et al., 2022) which elicits rationales that lead to the final answers, and its extension, using self-consistency (Wang et al., 2023) or automated demonstration selection (Zhang et al., 2023). Other methods, such as ReAct (Yao et al., 2022), generate reasoning steps sequentially by integrating planning, with additional strategies like Tree of Thoughts (ToT) (Yao et al., 2024), Reasoning via Planning (RAP) (Hao et al., 2023), and other methods (Khot et al., 2023; Zhou et al., 2023) facilitating complex question decomposition through varied planning approaches. Unlike most methods that rely on in-context learning through prompt engineering, our approach generates planning data from KGs to fine-tune LLM, thereby enhancing their planning capabilities.

Retrieval-Augmented Generation Retrieval-Augmented Generation (RAG) can enhance LLMs by incorporating external data, allowing models to access up-to-date information and factual knowledge to mitigate hallucinations (Gao et al., 2023; Guu et al., 2020; Lewis et al., 2020). Each module in the RAG pipeline can be optimized, for instance, through retriever tuning (Shi et al., 2023; Lin et al., 2023), self-reflection during retrieval (Asai et al., 2023; Yan et al., 2024), or query refinement (Chan et al., 2024). To address multi-hop questions, iterative RAG models (Shao et al., 2023; Feng et al., 2023; Press et al., 2023) have been developed, which iteratively conduct retrieval-enhanced generation and generation-enhanced retrieval. However, the multiple RAG steps in existing methods are not optimized and rely heavily on in-context learning. Our approach uses planning data from KGs to facilitate more efficient RAG.

LLMs with KGs In the existing realm of LLMs, KGs are primarily utilized as sources of structured factual knowledge (Pan et al., 2023). For example, Think-on-Graph (Sun et al., 2023) extracts relevant triples from KGs to assist in QA. Reasoning on Graph (RoG) (Luo et al., 2023) generates relation-based plans and retrieves corresponding paths from these graphs. While aiding in KGQA tasks where answers are directly sourced from


Step 1: Data Construction
Spouse
Sports
Fran Walsh
Ans_1
Fluminense
Sports
Grounding
Knowledge Graph
Verbalization
Spouse
Q1:What is the Spouse of Fran Walsh?
A1: Ans_1
Sports
Q2:What sports does {Ans_1} play?
A2:Ans_2
Sports
Q3:What sports does Fluminense play?
A3:Ans_3
Final Q: What sports have Fluminense and Fran Walsh's spouse played in?
Filling
Step 2: Planning LLM Tuning and Inference
SFT
Inference
"Which regions border Drake Bell's birthplace and Santa Ana at the same time?"
Step 3: Plan Parsing and Execution
Retrieval
QA LLM
Set Operate
Final Answer

```
Input
##Example 0##
…
##Example 1##
...
##Your Turn##
Original_Question: str = 'What sports have Fluminense
and Fran Walsh's spouse played in?'
```

```
Output
Sub_Question_1: str = " What is the Spouse of Fran Walsh? "
Info_1: str = Search(query = Sub_Question_1)
Ans_1: str = Get_Answer(query = Sub_Question_1, info = Info_1)

Sub_Question_2: str = f" What sports does {Ans_1} play? "
Info_2: str = Search(query = Sub_Question_2)
Ans_2: str = Get_Answer(query = Sub_Question_2, info = Info_2)

Sub_Question_3: str = "Q3:What sports does Fluminense
play?"
Info_3: str = Search(query = Sub_Question_3)
Ans_3: str = Get_Answer(query = Sub_Question_3, info = Info_3)

Inter_Results1: str = Intersection(Answer1 = Ans_2, Answer2 =
Ans_3)
Final_Answer: str = Finish_The_Plan(Answer = Inter_Results1)
```

```
Sub_Question_1: str = " What is the birthplace of Drake Bell?"
Info_1: str = Search(query = Sub_Question_1)
Ans_1: str = Get_Answer(query = Sub_Question_1, info = Info_1)

Sub_Question_2: str = f" Which areas border with {Ans_1} "
Info_2: str = Search(query = Sub_Question_2)
Ans_2: str = Get_Answer(query = Sub_Question_2, info = Info_2)

Sub_Question_3: str = "Which areas border with Santa Ana?"
Info_3: str = Search(query = Sub_Question_3)
Ans_3: str = Get_Answer(query = Sub_Question_3, info = Info_3)

Inter_Results1: str = Intersection(Answer1 = Ans_2, Answer2 =
Ans_3)
Final_Answer: str = Finish_The_Plan(Answer = Inter_Results1)
```

```
Sub_Question_1: …
Info_1: str = Search …
…
…
Ans_1: str = Get_Answer …
…
Inter_Results1: str = Intersection…
…
Final_Answer …
```

Figure 2: Overview of our Learning to Plan from Knowlege Graph (LPKG) framework.

KGs, these graphs also support rationale generation. Chain-of-Knowledge (CoK) (Li et al., 2024) further leverages KGs along with other heterogeneous sources to generate faithful rationales. Unlike previous studies, our approach constructs planning data for complex questions from KGs, recognizing that patterns within KGs inherently represent multi-step plans. This data is utilized to enhance the planning capabilities of LLMs.

Complex Logical Query in KGs Recent research on complex logic queries in KGs primarily focuses on first-order logical (FOL) queries that incorporate operations like conjunctions, disjunctions, negation, and existential quantifiers within incomplete KGs (Hamilton et al., 2018; Ren et al., 2020; Ren and Leskovec, 2020; Arakelyan et al., 2021; Chen et al., 2022; Xu et al., 2022; Xiong et al., 2024b; Wu et al., 2024). These works define diverse patterns to assess the capability of logical operations in vector spaces, specifically targeting logical forms rather than natural language. Nonetheless, their methodologies for pattern definition and extraction inspire our approach to deriving complex questions from KGs.

3 Method

3.1 Overview

As shown in Figure 2, there are 3 steps in our **L**earning to **P**lan from **K**nowledge **G**raphs (LPKG) framework. (1) In the data construction step, we construct planning data from KGs. Specifically, we defined some basic KG patterns as shown in Figure 3. We ground patterns in an existing KG to extract instances. For each extracted instance, we sequentially verbalize the sub-queries within the instance into natural language sub-questions according to their order in the instance, eventually assembling them into a complex question. Afterward, we build input and output templates for planning data, where complex questions are concatenated to the input prompt, and sub-questions are filled into the corresponding positions in the output text according to the type of patterns. (2) In the planning LLM tuning and inference step, we fine-tune LLMs based on such planning data to enable the LLMs to follow instructions to infer the plan for each question in the downstream test sets. (3) In the third step, such a plan will be parsed and executed, thereby obtaining the final answer to each question.

3.2 Construction of Planning Data

Basic KG Patterns. Inspired by previous work on complex logic queries within KGs (Ren and Leskovec, 2020), we define the basic KG patterns as shown in Figure 3. The set of KG patterns is denoted as $\mathcal{P} = \{1p, 2p, 3p, 2i, 3i, 2u, ip, pi, compare\}$. Specifically, p, i, u respectively indicate projection, intersection, and union. $1p$, $2p$, and $3p$ represent

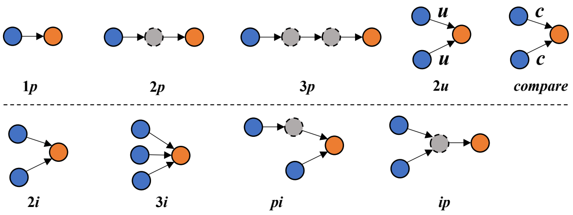


Figure 3: Basic KG patterns.

queries that span from one to three hops, $2i$ and $3i$ respectively represent the intersection of two sub-queries and three sub-queries, $2u$ represents the union of two sub-queries, and ip and pi represent complex queries that combine two-hop with intersection logic. In addition, we also combine pairs of triples that have numeric tail entities and the same relations to construct comparison patterns, denoted as $compare$.

Grounding. Given a KG, we first ground these patterns in it to extract instances:

$$\mathcal{I}_{pat} = f_{pat}(\mathcal{KG}), pat \in \mathcal{P} \tag{1}$$

where $\mathcal{I}_{pat}$ are the instances grounded by knowledge graph $\mathcal{KG}$ of pattern pat, f_{pat} is the corresponding extraction function. For example, an instance of the $2p$ pattern can be "(Inkheart, (cast member, educated at))". To best meet the needs of open-domain QA, we use Wikidata15k (Chen et al., 2023b), a subset of the open-domain KG Wikidata, as $\mathcal{KG}$.

Verbalization. Subsequently, based on the grounded instances, we need to verbalize them bottom-up into sub-questions and assemble them into complex questions. There are several methods for this step, such as a templates-based method, manual annotation, or utilizing an LLM. Since the template-based approach often lacks fluency in language expression, and the manual method is time-consuming and labor-intensive, we opt for an LLM-based method. Specifically, we write a small number of verbalization examples for each pattern type. These examples are used as demonstrations De_1 to fill in the prompt. Finally, we concatenate a grounded instance $i \in \mathcal{I}_{pat}$ to the prompt, asking an LLM to verbalize it to a natural language question:

$$\{\{Q_{s_n}\}_{n=1}^{k}, Q_c\} = llm(concat(De_1, i)) \tag{2}$$

where $\{Q_{s_n}\}_{n=1}^{k}$ and Q_c represent the resulting sub-questions and complex question respectively, $concat$ is string level concatenation. We use GPT-4 as llm here. It is important to note that here the llm's role is merely to transform the data format; the sub-questions and complex question still originate from the structure of the KG itself, without introducing any knowledge from the llm in the task of question planning. The prompt we use can be found in Appendix C.1.

Filling. We then extract sub-questions and complex questions from the output of the llm. Subsequently, we built a set of planning templates $\mathcal{T}_{pat}$ for the planning process of questions corresponding to each pattern. The $\{Q_{s_n}\}_{n=1}^{k}$ obtained in the previous step will be filled into fixed positions in $\mathcal{T}_{pat}$ corresponding to their pattern type, thereby obtaining the output for training. The Q_c obtained in the previous step is concatenated to the end of a fixed instruction Ins and some planning demonstrations De_2 (also constructed from KGs), thus obtaining the input for training data:

$$x = concat(Ins, De_2, Q_c) \tag{3}$$

$$y = \mathcal{T}_{pat}.fill(\{Q_s\}_{n=1}^{k}), pat \in \mathcal{P} \tag{4}$$

where $.fill$ is a filling function of templates $\mathcal{T}_{pat}$. Inspired by (Aksitov et al., 2023), we use a code-formatted input x and output y here (shown in "Input" and "Output" in Figure 2) to facilitate formatting and subsequent parsing and execution of the output plan (more details in Appendix C.2). In the end, we obtain 9000 training data entries $\mathcal{D}_{train} = \{x_n, y_n\}_{n=1}^{9000}$, with 1000 entries for each pattern. We randomly select 100 items from the training sets for manual verification, with an accuracy rate of over 95%.

3.3 Fine-tuning and Inference of Planning LLMs

We use the obtained training data $\mathcal{D}_{train}$ to fine-tune the planning LLMs $\mathcal{M}_p$ directly with the standard next token training objective:

$$\underset{\mathcal{M}_p}{max}\, \mathbb{E}_{(x,y)\in\mathcal{D}_{train}} \operatorname{Log} p_{\mathcal{M}_p}(y|x) \tag{5}$$

The fine-tuned planning LLM $\mathcal{M}_p$ can be used to infer the plan P for each question Q_{test} in the downstream test set:

$$P = \mathcal{M}_p(concat(Ins, De_2, Q_{test})) \tag{6}$$

where Ins and De_2 are the same as the contents in the Equation (3). It should be noted that in the

Type	Count	Type	Count
2p question	200	3p question	200
2i question	200	3i question	200
ip question	50	pi question	50
2u question	200	compare question	100

Table 1: Distribution of CLQA-Wiki.

multi-hop questions, the specific sub-questions in the second and third hops need to be constructed based on the answers to the previous hop's sub-questions. Since our P outputs all processes at once, the $\mathcal{M}_p$ cannot know the answers to the previous hop's sub-questions when outputting the plans. Therefore, we will use a placeholder to replace the answer to the previous hop sub-questions, allowing the planning to proceed smoothly (as shown in Table 9, 10, 13, 14 in Appendix C.1). These placeholders will then be filled in during the subsequent parsing and execution process.

3.4 Plan Parsing and Execution

The obtained plan P needs to be parsed and executed to obtain the final answer of the Q_{test}. Due to our adoption of code-formatted input and output for fine-tuning the $\mathcal{M}_p$, the P here is also highly formatted code, which facilitates our parsing of each step of the plan and executing them. In particular:

• When a step includes a "Search" function, we will call an external retrieval tool.

• When a step includes a "Get Answer" function, we'll invoke an external QA LLM $\mathcal{M}_{QA}$ to get answers for a sub-question based on the retrieved information. The possible placeholders in sub-questions will be filled with previous answers. We ask QA LLM to organize answers in the form of a list (prompt is shown in Table 7 in Appendix C.3).

• When "Intersection" or "Union" appears in the step, we will run actual intersection or union functions. This can be easily completed due to list format answers in the previous step.

It is important to note that the planning LLM $\mathcal{M}_p$ and the QA LLM $\mathcal{M}_{QA}$ are completely decoupled in our framework. Here we can use any LLM off-the-shelf to handle the task of QA. Ultimately, we can obtain the answer to Q_{test}.

4 New BenchMark: CLQA-Wiki

The conventional complex QA datasets include HotPotQA (Yang et al., 2018), 2WikiMultiopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023). Despite their widespread use in evaluating the QA performance of language models, we identify some problems with these datasets:

(1) All these datasets are primarily focused on multi-hop and comparison-type questions. The types of questions are not balanced and comprehensive enough, and less attention is paid to questions involving intersection and union logic, which are also very common in reality.

(2) Except for MuSiQue, the questions on the rest of the other three datasets only have one answer, whereas many questions in reality often have multiple answers. For example, the answer to an intersection question "Which country borders with Russia and China at the same time?" is a set [Mongolia, Kazakhstan, North Korea].

In light of this, we aim to construct a new testing benchmark that embodies more comprehensive logic and allows for an unrestricted number of answers to more thoroughly evaluate the performance of language models on various logical questions. Considering the detailed pattern structures and unrestricted number of answer entities in KGs, we construct a test set based on Wikidata15k.

Similar to the method used to construct the planning data, we extract instances from Wikidata15k (which do not appear in the training data) and use GPT-4 to do verbalization. Moreover, for each instance, we can obtain all the answer entities from Wikidata15k, which we then designate as the answers to the questions. After manual quality checks, we obtain a test set called CLQA-Wiki, which contains 1,200 pieces of data featuring a variety of **C**omprehensive **L**ogical **QA** pairs. The question types and their distribution are listed in Table 1. It is worth noting that we have constructed 9 types of testing questions until now, and for newly defined patterns, we can also quickly construct corresponding questions using the above method, showing the better scalability of our dataset.

5 Experiment

We aim to answer the following research questions in our experiments:

• **RQ1**: Can LPKG outperform baseline methods on conventional complex QA datasets?

• **RQ2**: Can planning data derived from KGs help improve the planning ability of the LLMs?

• **RQ3**: Can planning data derived from KGs be more helpful in improving the LLMs' planning ability compared to normal distillation methods?

• **RQ4**: Can LPKG outperform baseline methods on the new benchmark CLQA-Wiki?

5.1 Experimental Settings

Datasets We first conduct experiments on four conventional complex QA datasets: HotPotQA (Yang et al., 2018), 2WikiMultiHopQA(2WikiMQA) (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023). Among them, HotPotQA, 2WikiMQA, and MuSiQue contain completed train sets, development sets, and test sets, while Bamboogle is a small dataset that only contains 125 test data. Similar to the previous method (Shao et al., 2023; Aksitov et al., 2023), we respectively extract the first 500 entries from the development set of HotPotQA, 2WikiMQA. For MuSiQue, we follow Press et al. (2023) to use only 2-hop questions in the development set. And for Bamboogle, we use all of its data as test data. Finally, we conduct testing on our benchmark CLQA-Wiki.

Baselines We compare our framework to various baselines: • **Direct**: Directly input the original question into LLM. • **CoT**: Follow Kojima et al. (2022), we instruct LLM firstly "Think step by step" and then give the final answers. • **Direct RAG**: The prompt sent to LLM contains the original question and retrieved information related to the original question. • **ReAct** (Yao et al., 2022): Answering questions through iterative planning, action, and observation. The action here is the retrieval tool and observation is the retrieved information. The planning and QA are conducted on a single LLM. • **Self-Ask** (Press et al., 2023): Similar to ReAct, it first instructs LLM to judge whether sub-questions are needed. If so, it will request LLM to generate the sub-questions, then conduct external retrieval based on the sub-questions, and allow LLM to provide answers based on the retrieved information. • **ICLPKG** A variant of LPKG framework. Planning LLMs are not fine-tuned, while just using **I**n-**C**ontext **L**earning to do **P**lanning with some **KG**-sourced planning demonstrations.

Evaluation Metrics Exact Match (EM) is set as an evaluation metric in HotPotQA, 2WikiMQA, Bamboogle, and MuSiQue. While in CLQA-Wiki, we use Recall and Precision.

Implementation Details All baselines are conducted with `gpt-3.5-turbo-1106`[1] (GPT-3.5). The prompts of "Direct", "CoT", and "Direct RAG" are written by ourselves. The ReAct and Self-Ask are replicated based on their source code with the GPT-3.5 API. To facilitate assessment, we will ask the model to only output concise answer phrases.

In our framework: (1) For pattern grounding, we use Wikidata15k as $\mathcal{KG}$, which contains about 15k entities and 263 relations. The extraction tool in grounding is modified from existing works (Ren and Leskovec, 2020). (2) For the planning LLM $\mathcal{M}_p$, we choose CodeQwen1.5-7B-Chat and Llama3-8B-Instruct, one excels at coding while the other excels at common sense reasoning. We fine-tune them with Lora tuning, running on 4x80G A100 GPUs for about 3 hours. The fine-tuning is conducted for 2 epochs, with a learning rate of $5e$-5 and a cosine learning rate scheduler. (3) For retrieval, following previous works (Shao et al., 2023; Asai et al., 2023), we employ Wikipedia as the corpus for document retrieval and use the off-the-shelf Contriever-MS as the retriever. We select the Top 5 documents as the retrieved information. (4) For QA LLM, since we only care about the ability of the planning LLMs, in order to eliminate the impact of differences in the ability of QA LLMs, we use GPT-3.5 to align with baselines.

5.2 Results on Conventional Complex QA

Main Results (RQ1,RQ2) Table 2 shows results on conventional complex QA datasets. Since the QA LLM remains unchanged in our framework, we use "LPKG(Llama3)" and "LPKG(CodeQwen)" to represent LPKG frameworks using different planning LLMs, respectively. They are fine-tuned on Llama3-8B-Instruct and CodeQwen1.5-7B-Chat with KG-sourced planning data. It can be found that our framework outperforms the baseline methods on the majority of datasets. Particularly, compared to ReAct and Self-Ask, our approach shows significant improvement. It is worth noting that both ReAct and Self-Ask iterative planning and RAG in their workflows, whereas our approach decouples planning and RAG into two separate models. This allows each model to focus more intensively on its individual task. Moreover, we specifically enhance the planning part by fine-tuning $\mathcal{M}_p$ with planning data sourced from KG. These two changes have brought significant improvements to

[1] https://platform.openai.com/docs/models/gpt-3-5-turbo

	Planning	RAG	HotPotQA	2WikiMQA	Bamboogle	MuSiQue
Direct	✗	✗	0.268	0.284	0.128	0.090
CoT	✔	✗	0.288	0.286	0.280	0.090
Direct RAG	✗	✔	0.292	0.230	0.080	0.088
ReAct	✔	✔	0.211	0.216	0.168	0.060
Self-Ask	✔	✔	0.176	0.194	0.136	0.116
ICLPKG(GPT-3.5)	✔	✔	0.352	0.344	0.296	0.254
LPKG(CodeQwen)	✔	✔	0.338	0.356	0.280	0.266
LPKG(Llama3)	✔	✔	**0.376**	**0.372**	**0.304**	**0.296**

Table 2: Exact match results on conventional complex QA datasets. The best results are in bold, and the second best is underlined. All baseline methods are conducted on GPT-3.5. LPKG(CodeQwen), and LPKG(Llama3) respectively represent using our framework with fine-tuned CodeQwen1.5-7B-Chat and fine-tuned Llama3-8B-Instruct (fine-tuning is conducted on KG-sourced planning data).

	HotPotQA	2WikiMQA	Bamboogle	MuSiQue
LPKG(CodeQwen)	**0.338**	**0.356**	**0.256**	**0.266**
ICLPKG(CodeQwen)	0.110	0.286	0.176	0.176
LPKG(Llama3)	**0.376**	**0.372**	**0.272**	**0.296**
ICLPKG(Llama3)	0.369	0.353	0.256	0.290

Table 3: Ablation study on the KG-sourced planning data. ICLPKG(CodeQwen) and ICLPKG(Llama3) represent using the raw CodeQwen1.5-7B-Chat and Llama3-8B-Instruct to conduct planning, respectively.

the overall accuracy.

At the same time, we also attempt to replace the fine-tuned $\mathcal{M}_p$ with GPT-3.5 while keeping other parts unchanged, denoted as "ICLPKG(GPT-3.5)" in Table 2. Results show that even though fine-tuned $\mathcal{M}_p$ CodeQwen (7B) and Llama3 (8B) have significantly fewer parameters than GPT-3.5 (more than 175B), they can maintain or even surpass GPT-3.5 in terms of planning ability. Next, we replace the $\mathcal{M}_p$ fine-tuned on KG-sourced data with their raw models, and the experimental results are shown in Table 3. It can be observed that after fine-tuning with planning data derived from the KG, both CodeQwen1.5-7B-Chat and Llama3-8B-Instruct show significant improvements in planning ability. In particular, CodeQwen1.5-7B-Chat, which is significantly inferior to GPT-3.5 across all datasets in planning ability before fine-tuning, exhibits a notable enhancement after fine-tuning on KG-based planning data, especially achieving better results than GPT-3.5 on 2WikiMQA and MuSiQue. All these experimental phenomena fully demonstrate the efficacy of using KG-sourced planning data in improving the planning ability of the LLMs.

	Bamboogle
LPKG(CodeQwen)	**0.272**
DLPKG(CodeQwen)	0.216
ICLPKG(CodeQwen)	0.176

Table 4: Comparison with normal distillation methods. QA LLM is GPT-3.5.

Compare to Normal Distillation (RQ3) To further validate the effectiveness of using planning data constructed from KG, we compare it with the normal distillation method. Specifically, we extracted 3000 questions each from the training sets of HotPotQA, 2WikiMQA, and MuSiQue (9000 questions in total). Using the same input prompt with Equation (6), we obtain the planning process of these questions by invoking GPT-3.5. These planning data are then used to fine-tune CodeQwen1.5-7B-Chat which has relatively weaker planning capabilities, resulting in DLPKG(CodeQwen).

To ensure the fairness of the comparison, we conduct testing on the unseen dataset Bamboogle, and the experimental results are shown in Table 4. The results demonstrate that, under the same amount of training data and without using in-domain ques-

Planning Error	Retrieval Error	QA LLM Error
13	17	10

Table 5: Error analysis of LPKG(Llama3).

	CLQA-Wiki	
	Precision	Recall
CoT	0.0605(+80.6%)	0.0641(+103.4%)
Direct RAG	0.0814(+34.2%)	0.0789(+65.3%)
ReAct	0.0264(+314.0%)	0.0270(+382.9%)
Self-Ask	0.0385(+183.8%)	0.0423(+208.2%)
ICLPKG(GPT-3.5)	0.0907(+20.5%)	0.1014(+28.6%)
LPKG(Llama3)	**0.1112**	**0.1344**

Table 6: Precision and Recall result on CLQA-Wiki.

tions for fine-tuning, using planning data constructed from KG yields better performance than using planning data distilled from GPT-3.5. We believe this observation is inspiring and can be attributed to the richer reasoning types in the KG patterns, as well as the highly accurate reasoning paths in well-constructed KG.

Error Analysis To gain a deeper understanding of the model's performance, we conduct an error analysis of LPKG. Specifically, we extract 40 incorrect samples (10 per dataset) of LPKG(Llama3) and manually categorize the error cases into three types: planning error, retrieval error, and QA LLM error. As shown in Table 5, the performance of the retrieval model has the greatest impact. Among the 13 samples with planning errors, 10 of them are due to incorrect judgment of the type of questions, and 3 are due to incorrect expression of sub-questions. Future exploration directions can be based on improving the performance of the retriever model and enhancing the planning LLM's ability to identify question types.

5.3 Results on CLQA-Wiki (RQ4)

Main Results We then conduct testing based on the CLQA-Wiki benchmark. Given that answers in this benchmark may have multiple candidates, we adjust the instructions for QA LLMs to require them to output all potential answers in a specified list format. This adjustment is made to facilitate the extraction and evaluation of the responses. Since Llama3-8B-Instruct is more powerful than CodeQwen1.5-7B-Chat as shown in Table 2, we only conduct LPKG with Llama3 here. Experimental results are presented in Table 6. It can be seen that CLQA-Wiki is a very challenging dataset, but LPKG(Llama3) still outperforms the baseline methods. At the same time, compared to ICLPKG(GPT-3.5), LPKG(Llama3) has an average improvement of over 20%, highlighting the importance of using KG-sourced planning data.

In addition, we conduct more fine-grained experiments based on the type of questions, and the experimental results are shown in Figure 4. We found that LPKG(Llama3) performs more prominently on some complex questions, such as the $3p$, $2i$, and $2u$ questions, demonstrating the advantages of our framework in dealing with complex logic questions. At the same time, we also found that direct retrieval performs well on some types of questions, such as $3i$ and compare questions. This may be due to the fact that in the process of verbalizing these questions, the assembly of sub-questions into complex questions is relatively straightforward, allowing answers to each sub-questions to be obtained directly through the retrieval of complex questions or the knowledge of the LLM itself.

Case Study To more intuitively demonstrate the effectiveness of KG-source planning data, we conduct a case study on CLQA-Wiki, detailed in Figure 5 in Appendix A. When planning a $2i$ question "What sport is associated with John Madden and Ben Johnson?", GPT-3.5 generates some meaningless sub-questions and incorrectly defines the question type, which will definitely lead to incorrect answers. But LPKG(Llama3) could identify it as a $2i$ question and provide the correct sub-questions and planning steps, thereby helping to obtain the correct answer during final parsing and execution.

6 Conclusion

In this paper, we try to enhance the planning ability in retrieval-augmented LLMs using KGs. Specifically, we design a framework for Learning to Plan from KG (LPKG). The proposed LPKG framework first utilizes the rich patterns in the KGs to construct planning data, then fine-tune planning LLMs based on such data to enable them to conduct planning on downstream datasets, and ultimately get the final answer through parsing and execution. The experimental results reveal the excellent performance of the LPKG framework and also demonstrate the effectiveness of using KG-sourced data to enhance LLMs' planning ability. Finally, we construct CLQA-Wiki, providing a more challenging complex QA benchmark for the community.

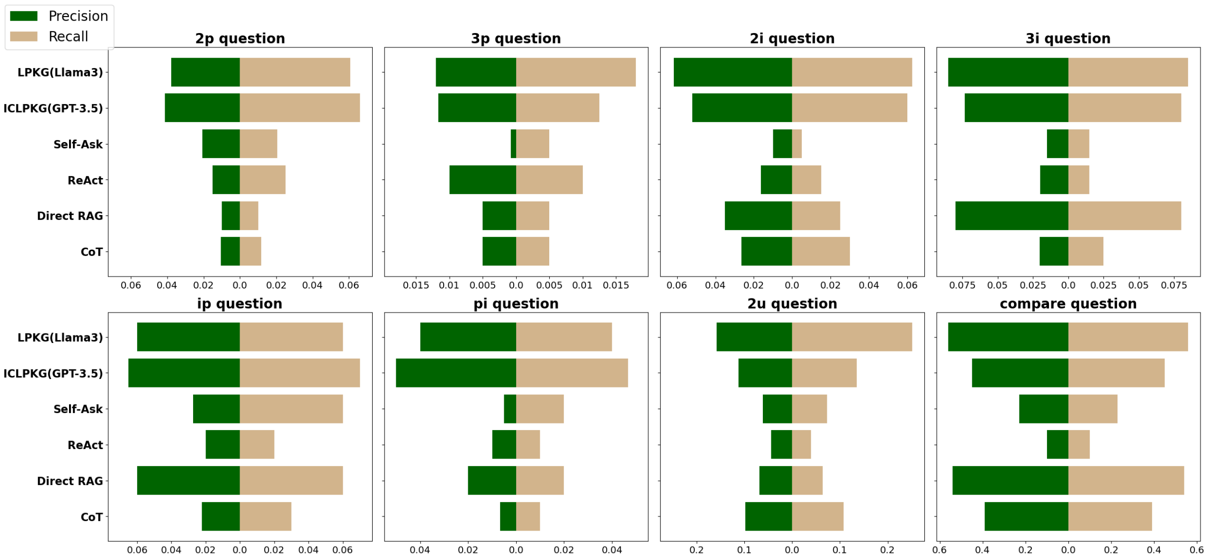


Figure 4: Fine-grained evaluation based on question types.

Limitation

In our view, the limitations of our work at the current stage mainly stem from two aspects:

(1) During the fine-tuning phase of planning LLMs, we simply mixed various types of questions together uniformly for training. We have not yet explored the impact of question type distribution on the experimental results, which could be the focus of future work.

(2) At present, the datasets we test have explicit types of questions (multi-hop, comparison, and union/intersection), but in reality, some question types may be implicit or even not be included in the types we define. The future direction of our work can be to study planning methods for these types of unclear questions.

Acknowledgement

This work is founded by National Natural Science Foundation of China (NSFC62306276/NSFCU23B2055/NSFCU19B2027), Zhejiang Provincial Natural Science Foundation of China (No. LQ23F020017), Yongjiang Talent Introduction Programme (2022A-238-G), Chang Jiang Scholars Program (J2019032), Fundamental Research Funds for the Central Universities (226-2023-00138).

References

AI@Meta. 2024. Llama 3 model card.

Renat Aksitov, Sobhan Miryoosefi, Zonglin Li, Daliang Li, Sheila Babayan, Kavya Kopparapu, Zachary Fisher, Ruiqi Guo, Sushant Prakash, Pranesh Srinivasan, et al. 2023. Rest meets react: Self-improvement for multi-step reasoning llm agent. *arXiv preprint arXiv:2312.10003*.

Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. 2021. Complex query answering with neural link predictors. In *ICLR*. OpenReview.net.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *CoRR*, abs/2310.11511.

Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. RQ-RAG: learning to refine queries for retrieval augmented generation. *CoRR*, abs/2404.00610.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023a. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.

Xiang Chen, Duanzheng Song, Honghao Gui, Chengxi Wang, Ningyu Zhang, Jiang Yong, Fei Huang, Chengfei Lv, Dan Zhang, and Huajun Chen. 2023b. Factchd: Benchmarking fact-conflicting hallucination detection. *CoRR*, abs/2310.12086.

Xuelu Chen, Ziniu Hu, and Yizhou Sun. 2022. Fuzzy logic based logical query answering on knowledge graphs. In *AAAI*, pages 3939–3948. AAAI Press.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek

Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113.

Zhangyin Feng, Xiaocheng Feng, Dezhi Zhao, Maojin Yang, and Bing Qin. 2023. Retrieval-generation synergy augmented large language models. volume abs/2310.05149.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *CoRR*, abs/2312.10997.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Retrieval augmented language model pre-training. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 3929–3938. PMLR.

William L. Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Embedding logical queries on knowledge graphs. In *NeurIPS*, pages 2030–2041.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *EMNLP*, pages 8154–8173. Association for Computational Linguistics.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

Wenyu Huang, Mirella Lapata, Pavlos Vougiouklis, Nikos Papasarantopoulos, and Jeff Z. Pan. 2023. Retrieval augmented generation with rich answer encoding. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 1012–1025.

Wenyu Huang, Guancheng Zhou, Hongru Wang, Pavlos Vougiouklis, Mirella Lapata1, and Jeff Z. Pan. 2024. Less is More: Making Smaller Language Models Competent Subgraph Retrievers for Multi-hop KGQA. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP 2024)*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *ICLR*. OpenReview.net.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*.

Xingxuan Li, Ruochen Zhao, Yew Ken Chia, Bosheng Ding, Shafiq Joty, Soujanya Poria, and Lidong Bing. 2024. Chain-of-knowledge: Grounding large language models via dynamic knowledge adapting over heterogeneous sources. In *The Twelfth International Conference on Learning Representations*.

Xi Victoria Lin, Xilun Chen, Mingda Chen, Weijia Shi, Maria Lomeli, Rich James, Pedro Rodriguez, Jacob Kahn, Gergely Szilvasy, Mike Lewis, Luke Zettlemoyer, and Scott Yih. 2023. RA-DIT: retrieval-augmented dual instruction tuning. *CoRR*, abs/2310.01352.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2023. Reasoning on graphs: Faithful and interpretable large language model reasoning. *CoRR*, abs/2310.01061.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented language models: a survey. *CoRR*, abs/2302.07842.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. 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 *NeurIPS*.

Jeff Z. Pan, Simon Razniewski, Jan-Christoph Kalo, Sneha Singhania, Jiaoyan Chen, Stefan Dietze, Hajira Jabeen, Janna Omeliyanenko, Wen Zhang, Matteo Lissandrini, ussa Biswas, Gerard de Melo, Angela Bonifati, Edlira Vakaj, Mauro Dragoni, and amien Graux. 2023. Large language models and knowledge graphs: Opportunities and challenges. *Transactions on Graph Data and Knowledge*.

J.Z. Pan, D. Calvanese, T. Eiter, I. Horrocks, M. Kifer, F. Lin, and Y. Zhao, editors. 2017a. *Reasoning Web: Logical Foundation of Knowledge Graph Construction and Querying Answering*. Springer.

J.Z. Pan, G. Vetere, J.M. Gomez-Perez, and H. Wu, editors. 2017b. *Exploiting Linked Data and Knowledge Graphs for Large Organisations*. Springer.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *EMNLP (Findings)*, pages 5687–5711. Association for Computational Linguistics.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Hongyu Ren, Weihua Hu, and Jure Leskovec. 2020. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *ICLR*. OpenReview.net.

Hongyu Ren and Jure Leskovec. 2020. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems*, 33:19716–19726.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *EMNLP (Findings)*, pages 9248–9274. Association for Computational Linguistics.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2023. REPLUG: retrieval-augmented black-box language models. *CoRR*, abs/2301.12652.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model with knowledge graph. *CoRR*, abs/2307.07697.

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, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Denny Vrandecic 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 *ICLR*. OpenReview.net.

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 *NeurIPS*.

Yike Wu, Yi Huang, Nan Hu, Yuncheng Hua, Guilin Qi, Jiaoyan Chen, and Jeff Z. Pan. 2024. CoTKR: Chain-of-Thought Enhanced Knowledge Rewriting for Complex Knowledge Graph Question Answering. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP 2024)*.

Siheng Xiong, Ali Payani, Ramana Kompella, and Faramarz Fekri. 2024a. Large language models can learn temporal reasoning. *arXiv preprint arXiv:2401.06853*.

Siheng Xiong, Yuan Yang, Ali Payani, James C Kerce, and Faramarz Fekri. 2024b. Teilp: Time prediction over knowledge graphs via logical reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16112–16119.

Zezhong Xu, Wen Zhang, Peng Ye, Hui Chen, and Huajun Chen. 2022. Neural-symbolic entangled framework for complex query answering. In *NeurIPS*.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. 2024. Corrective retrieval augmented generation. *CoRR*, abs/2401.15884.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2023. Automatic chain of thought prompting in large language models. In *ICLR*. OpenReview.net.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *ICLR*. OpenReview.net.

A Case Study

Cases are shown in Figure 5 where we mark the bad parts of the ICLPKG (GPT-3.5) planning in red, and the corresponding parts of LPKG (Llama3) in blue. In order to highlight the core difference, the "Thought" in planning is omitted in the cases.

B Prevention of Data Leakage

In the experiment, we used data from knowledge graph sources as training data, which may raise concerns about data leakage, specifically the overlap between the training data and the four multi-hop test sets(HotPotQA, 2WikiMQA, Bamboogle, MusiQue). We calculate the semantic similarity between the training and testing questions. We use the BGE-M3 model to embed the training and testing questions into high-dimensional vectors and calculate their cosine similarity. We found that the similarity between the testing and training questions does not exceed 0.8, and is generally below 0.6. This indicates that there is almost no high degree of overlap between the training data and the testing data.

In addition, when conducting experiments on our own test set CLQA-Wiki, we excluded questions similar to those in CLQA-Wiki from the training data (with similarity scores above 0.9), ensuring no high overlap between the training and testing data.

C Prompt Content

C.1 Prompt of Verbalization

Table 8,9,10,11,12,13,14,15,16 shows the different prompts of verbalizing pattern instances to their natural language questions. The specific instance that needs to be verbalized will be added to the end of the prompt.

C.2 Prompt of Planning LLM

The code-formatted input prompt for planning LLM is as follows. Due to space limitations, only some demonstrations are displayed in the prompt. In fact, we will include planning demonstrations for different types of questions in the prompt:

```
###Complete the Code Below###

from package1 import SerpAPIWrapper
from package2 import QA_LLM
search = SerpAPIWrapper()

def Search(query:str,thought:str):
    """Search relevant information about query based
         on external Search Engine.
    Attributes:
                query: The question you want to
                     search.
                thought: The reason why this query
                     is need.
    """
    if thought is not None:
        return search.run(query)
    else:
        return ("Please give your thought!")

def Get_Answer(query:str,info:str):
    """Get the answer of the query based on the
         information.
    Attributes:
    query: The question you want to search.
    info: The information relevant to the query.
    """
    ### Use the QA_LLM model to get the answer.
    return QA_LLM(query,info)

def Compare(Original_Query:str,Subquestions:list,
     Answers:list):
    """Compare the answer of the sub-questions and
         return the final answer of original query.
    Attributes:
    Original_Query: The original question.
    Subquestions: The list of sub-questions.
    Answers: The list of answers of the sub-
         questions.
    """
    query = Original_Query
    info = str()
    for i in range(len(Subquestions)):
        info += Subquestions[i] + ' : ' + Answers[i]
             + '\n'
    return QA_LLM(query,info)

def Intersection(Answer1:str,Answer2:str):
    """Find the intersection of two answer sets.
    Attributes:
    Answer1: The first answer set.
    Answer2: The second answer set.
    """
    List1 = Answer1.split(',')
    List2 = Answer2.split(',')
    return str(set(List1) & set(List2))

def Union(Answer1:str,Answer2:str):
    """Find the union of two answer sets.
    Attributes:
    Answer1: The first answer set.
    Answer2: The second answer set.
    """
    List1 = Answer1.split(',')
    List2 = Answer2.split(',')
    return str(set(List1) | set(List2))

def Finish_The_Plan(Answer:str):
    """Call this function to finish the plan and
         return the final answer.
    Attributes:
    Answer: The final answer of the original
         question.
    """
    return Answer

###################
# Example 0:
###################

Original_Question: str = "What is the ethnic group
     of Booker T. Jones?"
### Question Type: One Projection
### Decompose the original question into sub-
     questions.

Thought1: str = "An atomic question, no need to
     decompose. Search directly."
Sub_Question_1: str = "What is the ethnic group of
     Booker T. Jones?"
Info_1: str = Search(query = Sub_Question_1, thought
      = Thought1)
Ans_1: str = Get_Answer(query = Sub_Question_1, info
      = Info_1)

Final_Answer: str = Finish_The_Plan(Answer = Ans_1)
```

Question: What sport is associated with both John Madden and Ben Johnson?

ICLPKG(GPT-3.5) Planning	LPKG(Llama3) Planning
Thought1: … Sub_Question_1: str = "Who is John Madden?" Info_1: str = Search(query = Sub_Question_1, thought = Thought1) Ans_1: str = Get_Answer(query = Sub_Question_1, info = Info_1) Thought2: … Sub_Question_2: str = "Who is Ben Johnson?" Info_2: str = Search(query = Sub_Question_2, thought = Thought2) Ans_2: str = Get_Answer(query = Sub_Question_2, info = Info_2) Thought3: … Inter_Results1: str = Intersection(Answer1 = Ans_1, Answer2 = Ans_2) Thought4: … Sub_Question_3: str = f"What sport is associated with the people in {Inter_Results1}?" Info_3: str = Search(query = Sub_Question_3, thought = Thought4) Ans_3: str = Get_Answer(query = Sub_Question_3, info = Info_3) Final_Answer: str = Finish_The_Plan(Answer = Ans_3)	Thought1: … Sub_Question_1: str = "What sport is associated with John Madden?" Info_1: str = Search(query = Sub_Question_1, thought = Thought1) Ans_1: str = Get_Answer(query = Sub_Question_1, info = Info_1) Thought2: … Sub_Question_2: str = "What sport is associated with Ben Johnson?" Info_2: str = Search(query = Sub_Question_2, thought = Thought2) Ans_2: str = Get_Answer(query = Sub_Question_2, info = Info_2) Thought3: … Inter_Results1: str = Intersection(Answer1 = Ans_1, Answer2 = Ans_2) Final_Answer: str = Finish_The_Plan(Answer = Inter_Results1)

Question: What is the place of birth of Robert Cray named after?

ICLPKG(GPT-3.5) Planning	LPKG(Llama3) Planning
Thought1: … Sub_Question_1: str = "Who is Robert Cray?" Info_1: str = Search(query = Sub_Question_1, thought = Thought1) Ans_1: str = Get_Answer(query = Sub_Question_1, info = Info_1) Thought2: … Sub_Question_2: str = f"What is the place of birth of {Ans_1}?" Info_2: str = Search(query = Sub_Question_2, thought = Thought2) Ans_2: str = Get_Answer(query = Sub_Question_2, info = Info_2) Final_Answer: str = Finish_The_Plan(Answer = Ans_2)	Thought1: … Sub_Question_1: str = "Where was Robert Cray born?" Info_1: str = Search(query = Sub_Question_1, thought = Thought1) Ans_1: str = Get_Answer(query = Sub_Question_1, info = Info_1) Thought2: … Sub_Question_2: str = f"What is {Ans_1} named after?" Info_2: str = Search(query = Sub_Question_2, thought = Thought2) Ans_2: str = Get_Answer(query = Sub_Question_2, info = Info_2) Final_Answer: str = Finish_The_Plan(Answer = Ans_2)

Question: Which team's stadium has a greater maximum capacity, Zamalek SC or Selangor FA?

ICLPKG(GPT-3.5) Planning	LPKG(Llama3) Planning
Thought1: … Sub_Question_1: str = "What is the stadium of Zamalek SC?" Info_1: str = Search(query = Sub_Question_1, thought = Thought1) Ans_1: str = Get_Answer(query = Sub_Question_1, info = Info_1) Thought2: … Sub_Question_2: str = "What is the stadium of Selangor FA?" Info_2: str = Search(query = Sub_Question_2, thought = Thought2) Ans_2: str = Get_Answer(query = Sub_Question_2, info = Info_2) Thought3: … Ans_3: str = Compare(Original_Query = Original_Question, Subquestions = [Sub_Question_1,Sub_Question_2], Answers = [Ans_1,Ans_2]) Final_Answer: str = Finish_The_Plan(Answer = Ans_3)	Thought1: … Sub_Question_1: str = "What is the maximum capacity of the stadium of Selangor FA?" Info_1: str = Search(query = Sub_Question_1, thought = Thought1) Ans_1: str = Get_Answer(query = Sub_Question_1, info = Info_1) Thought2: … Sub_Question_2: str = "What is the maximum capacity of the stadium of Selangor FA?" Info_2: str = Search(query = Sub_Question_2, thought = Thought2) Ans_2: str = Get_Answer(query = Sub_Question_2, info = Info_2) Thought3: … Ans_3: str = Compare(Original_Query = Original_Question, Subquestions = [Sub_Question_1,Sub_Question_2], Answers = [Ans_1,Ans_2]) Final_Answer: str = Finish_The_Plan(Answer = Ans_3)

Figure 5: Case study on CLQA-Wiki.

```
###################
# Example 1:
###################

Original_Question: str = "Who succeeded the first
     President of Namibia?"
### Question Type: Two Projection
### Decompose the original question into sub-
     questions.

Thought1: str = "If I want to know who succeeded the
      first President of Namibia, I need to first
     know who is the first President of Namibia."
Sub_Question_1: str = "Who is the first President of
      Namibia?"
Info_1: str = Search(query = Sub_Question_1, thought
      = Thought1)
Ans_1: str = Get_Answer(query = Sub_Question_1, info
      = Info_1)

Thought2: str = "After knowing who is the first
     President of Namibia, I need to know who
     succeeded him."
Sub_Question_2: str = f"Who succeeded {Ans_1}?"
Info_2: str = Search(query = Sub_Question_2, thought
      = Thought2)
Ans_2: str = Get_Answer(query = Sub_Question_2, info
      = Info_2)

Final_Answer: str = Finish_The_Plan(Answer = Ans_2)

......(More Examples are omitted here)
###################
# Your turn! Just complete the code below and do not
      return other things.
###################

Original_Question: str =
```

C.3 Prompt of QA LLM

Table 7 shows the prompt we used for QA LLM. The "Wikipedia Docs." will be filled with retrieved Wikipedia documents based on input questions.

Instruction:
Give a question and some information that may help you answer the question. Please answer the question based on your own knowledge and the information provided.

Retrieved Information:
### Information
{Wikipeida Docs.}

Input
### Question:
{Input Question}
### Your Answer: (You only need to provide the final answer to the question. Intermediate answers are not needed. Please return your answer in the form of a list, where each element in the list is a short entity answer, such as [Apple]. When you think there are multiple answers, please divide them with a '#' symbol, such as [Apple#Banana#Origin]. If the answer is not included in the information provided, please answer based on your own knowledge. If you don't know either, please return [None].)

Table 7: Prompt for QA LLM.

Instruction:
Given a subgraph query in the knowledge graph, please transfer it into natural language. The subgraph query is expressed in the format (h,(r,)), where h and r represent the head entity and relation respectively, and the meaning of this query is to find the set of tail entities of h under the relation r. Your responsibility is to transfer it into a question in natural language form. I will give you some examples, please complete your task after reading them:

Demonstrations:
### Example 1:
Subgraph Query: (Booker T. Jones, (ethnic group,))
Natural Language Question: What is the ethnic group of Booker T. Jones?
### Example 2:
Subgraph Query: (Daniel Handler, (educated at,))
Natural Language Question: Where did Daniel Handler receive education?

### Your Turn (Just output the Natural Language Question and do not return other content):
Input:
Subgraph Query:

Table 8: Prompt of verbalization for $1p$ pattern instances.

Instruction:

Given a subgraph query in knowledge graph, please transfer it into natural language. The subgraph query is expressed in the format (h,(r1,r2,)), where h represents the head entity, and r1 and r2 represent a two-hop relation path starting from head entity h. The purpose of this query is to find the target entity associated with the head entity h under the relational path (r1, r2). Your responsibility is to first transfer it into two sub-questions and finally combine them to form a complex question. When constructing the second sub-question, you may need the answer to the first sub-question, so we will assume that the answer to the first sub-question is A1 and the answer to the second sub-question is A2, to facilitate the formulation of the sub-question. When composing the final question, please pay attention to the fluency of the language and avoid mechanically stitching sub-questions together. I will give you some examples, please complete your task after reading them:

Demonstrations:

### Example 1:
Subgraph Query:(Chongqing, (twinned administrative body, country of citizenship’))
Q1: Which city or administrative body that is twinned with Chongqing?
Q1_Answer: A1
Q2: What is the country of {A1}?
Q2_Answer: A2
Final Question: Which country has a city or administrative body that is twinned with Chongqing?
### Example 2:
Subgraph Query:(Inkheart, (cast member, educated at))
Q1: Who is the cast member of Inkheart?
Q1_Answer: A1
Q2: Where did {A1} receive education?
Q2_Answer: A2
Final Question: Where did the cast member of Inkheart receive education?

Input:

### Your Turn (Just complete your task in the above format and do not return other content):
Subgraph Query:

Table 9: Prompt of verbalization for 2p pattern instances.

Instruction:

Given a subgraph query in knowledge graph, please transfer it into natural language. The subgraph query is expressed in the format (h,(r1,r2,r3,)), where h represents the head entity, and (r1,r2,r3,) represents a three-hop relation path starting from the head entity h. The purpose of this query is to find the target entity associated with the head entity h under the relational path (r1,r2,r3,). Your responsibility is to first transfer it into three sub-questions and finally combine them to form a complex question. When constructing the second and third sub-question, you may need the answer to the previous sub-question, so we will assume that the answers to these three sub-questions are A1, A2, and A3, to facilitate the formulation of the sub-question. When composing the final question, please pay attention to the fluency of the language and avoid mechanically stitching sub-questions together. I will give you some examples, please complete your task after reading them:

Demonstrations:

### Example 1:
Subgraph Query:(Chongqing, (twinned administrative body, country of citizenship))
Subgraph Query: (Android, (developer, country, foundational text))
Q1: Who is the developer of Android?
Q1_Answer: A1
Q2: What is the country of {A1}?
Q2_Answer: A2
Q3: What is the foundational text of country {A2}?
Q3_Answer: A3
Final Question: What is the foundational text of the Android developer's country?

### Example 2:
Subgraph Query: (X-Men: The Last Stand, (cast member, place of birth, shares border with))
Q1: Who is the cast member of X-Men: The Last Stand?
Q1_Answer: A1
Q2: What is the birthplace of {A1}?
Q2_Answer: A2
Q3: Which area borders with {A2}?
Q3_Answer: A3
Final Question: Which area borders the birthplace of X-Men: The Last Stand's cast member?

Input:

### Your Turn (Just complete your task in the above format and do not return other content):
Subgraph Query:

Table 10: Prompt of verbalization for $3p$ pattern instances.

Instruction:

Given a subgraph query in knowledge graph, please transfer it into natural language. The subgraph query is expressed in the format “(h1,(r1,)) Intersection (h2,(r2,))”, where h1 and h2 represent two head entities, r1 and r2 are their corresponding relations. The purpose of this query is to find the intersection set of the tail entities of (h1,(r1,)) and (h2,(r2,)). Your responsibility is to first transfer it into two sub-questions and finally combine them to form a complex question. When composing the final question, please pay attention to the fluency of the language and avoid mechanically stitching sub-questions together. The questioning method can be adjusted appropriately, but the meaning cannot be changed. I will give you some examples, please complete your task after reading them:

Demonstrations:

### Example 1:
Subgraph Query: (Jimmy Carter, (educated at,)) Intersection (John Wells, (educated at,))
Q1: Where did Jimmy Carter receive education?
Q2: Where did John Wells receive education?
Final Question: Where did both Jimmy Carter and John Wells receive education?
### Example 2:
Subgraph Query: (Burlington County, (shares border with,)) Intersection (Trumbull County, (shares border with,))
Q1: Which areas border with Burlington County?
Q2: Which areas border with Trumbull County?
Final Question: Which areas border with Burlington County and Trumbull County at the same time?

Input:

### Your Turn (Just complete your task in the above format and do not return other content):
Subgraph Query:

Table 11: Prompt of verbalization for $2i$ pattern instances.

Instruction:

Given a subgraph query in knowledge graph, please transfer it into natural language. The subgraph query is expressed in the format "(h1,(r1,)) Intersection (h2,(r2,)) Intersection (h3,(r3,))", where h1, h2 and h3 represent three head entities, r1, r2 and r3 are their corresponding relations. The purpose of this query is to find the intersection set of the tail entities of (h1,(r1,)), (h2,(r2,)) and (h3,(r3,)). Your responsibility is to first transfer it into three sub-questions and finally combine them to form a complex question. When composing the final question, please pay attention to the fluency of the language and avoid mechanically stitching sub-questions together. The questioning method can be adjusted appropriately, but the meaning cannot be changed. I will give you some examples, please complete your task after reading them:

Demonstrations:

### Example 1:

Subgraph Query: ((Alice in Wonderland, (genre,)) Intersection (Blues Brothers 2000, (genre,)) Intersection (Pinocchio, (genre,)))

Q1: What are the genre of Alice in Wonderland?

Q2: What are the genre of Blues Brothers 2000?

Q3: What are the genre of Pinocchio?

Final Question: What are the same genre shared between Alice in Wonderland, Blues Brothers 2000 and Pinocchio?

### Example 2:

Subgraph Query:(Springfield, (capital of,)) Intersection (Ulster County, (shares border with,)) Intersection (Montgomery County, (shares border with,))

Q1: What is the capital of Springfield?

Q2: Which areas border with Ulster County?

Q3: Which areas border with Montgomery County?

Final Question: Which area is the capital of Springfield and borders with Ulster County and Montgomery County at the same time?

Input:

### Your Turn (Just complete your task in the above format and do not return other content):

Subgraph Query:

Table 12: Prompt of verbalization for $3i$ pattern instances.

Instruction:

Given a subgraph query in knowledge graph, please transfer it into natural language. The subgraph query is expressed in the format "(h1,(r1,)) Intersection (h2,(r2,)) Projection r3", where h1 and h2 represent two head entities, r1 and r2 are their corresponding relations. The purpose of this query is to fisrt get the intersection set of the tail entities of (h1,(r1,)) and (h2,(r2,)), and then find the tail entities of every entity in the previous intersection set under relation r3. Your responsibility is to first transfer it into three sub-questions and finally combine them to form a complex question. When composing the final question, please pay attention to the fluency of the language and avoid mechanically stitching sub-questions together. The questioning method can be adjusted appropriately, but the meaning cannot be changed. I will give you some examples, please complete your task after reading them:

Demonstrations:

### Example 1:

Subgraph Query: (John Williams, (educated at,)) Intersection (John Milton, (educated at,)) Projection named after

Q1: Where did John Williams receive education?

Q2: Where did John Milton receive education?

Intersection_Answer: Inter_A

Q3: The {Inter_A} was named after what?

Final Question: The place where John Williams and John Milton both received education was named after what?

### Example 2:

Subgraph Query: (The Blues Brothers, (cast member,)) Intersection (Going My Way, (cast member,)) Projection member of political party

Q1: Who are the cast members of The Blues Brothers?

Q2: Who are the cast members of Going My Way?

Intersection_Answer: Inter_A

Q3: What are the political party of {Inter_A}?

Final Question: What are the political party of people who are cast members of both The Blues Brothers and Going My Way?

Input:

### Your Turn (Just complete your task in the above format and do not return other content):

Subgraph Query:

Table 13: Prompt of verbalization for *ip* pattern instances.

Instruction:
Given a subgraph query in knowledge graph, please transfer it into natural language. The subgraph query is expressed in the format “(h1,(r1,r2,)) Intersection (h2,(r3,))”, where (h1,(r1,r2,)) represents a two-hop relational path starts from head entity h1 followed by relation r1 and r2, and (h2, (r3,)) is an one-hop relational path start from head entity h2. The purpose of this query is to find the intersection set of the tail entity of relational path (h1,(r1,r2)) and (h2,(r3,)). Your responsibility is to first transfer it into three sub-questions and finally combine them to form a complex question. When constructing second or third sub-questions, you may need the answer to the previous sub-question, so we will assume that the answer to the first sub-question is A1 and the answer to the second sub-question is A2, to facilitate the formulation of the sub-question. When composing the final question, please pay attention to the fluency of the language and avoid mechanically stitching sub-questions together. The questioning method can be adjusted appropriately, but the meaning cannot be changed. I will give you some examples, please complete your task after reading them:

Demonstrations:
### Example 1:
Subgraph Query: (Drake Bell, (place of birth, shares border with)) Intersection (Santa Ana, shares border with)
Q1: What is the birthplace of Drake Bell?
Q1_Answer: A1
Q2: Which areas border with {A1}?
Q2_Answer: A2
Q3: Which areas border with Santa Ana?
Q3_Answer: A3
Final Question: Which regions border Drake Bell’s birthplace and Santa Ana at the same time?
Final Answer: A2 Intersection A3
### Example 2:
Subgraph Query: (Fran Walsh, (spouse, sport)) Intersection (Fluminense F.C., (sport,))
Q1: Who is the spouse of Fran Walsh?
Q1_Answer: A1
Q2: What sports does {A1} play?
Q2_Answer: A2
Q3: What sports does Fluminense F.C. play?
Q3_Answer: A3
Final Question: What sports have Fluminense F.C. and Fran Walsh’s spouse played in?
Final Answer: A3 Intersection A2

Input:
### Your Turn (Just complete your task in the above format and do not return other content):
Subgraph Query:

Table 14: Prompt of verbalization for *pi* pattern instances.

Instruction:

Given a subgraph query in knowledge graph, please transfer it into natural language. The subgraph query is expressed in the format “(h1,(r1,)) Union (h2,(r2,))”, where h1 and h2 represent two head entities, r1 and r2 are their corresponding relations. The purpose of this query is to find the Union set of the tail entities of (h1,(r1,)) and (h2,(r2,)). Your responsibility is to first transfer it into two sub-questions and finally combine them to form a complex question. When composing the final question, please pay attention to the fluency of the language and avoid mechanically stitching sub-questions together. The questioning method can be adjusted appropriately, but the meaning cannot be changed. I will give you some examples, please complete your task after reading them:

Demonstrations:

### Example 1:
Subgraph Query: (Wuthering Heights, (cast member,)) Union (Traffic, (cast member,))
Q1: Who are the cast members of Wuthering Heights?
Q2: Who are the cast members of Traffic?
Final Question: Who are all the cast members from Wuthering Heights combined with the cast members from Traffic?

### Example 2:
Subgraph Query: (Eve, (director,)) Union (Cold Mountain, (cast member,))
Q1: Who is the director of Eve?
Q2: Who are the cast members of Cold Mountain?
Final Question: Please list the director of Eve as well as all the cast members from Cold Mountain.

Input:

### Your Turn (Just complete your task in the above format and do not return other content):
Subgraph Query:

Table 15: Prompt of verbalization for $2u$ pattern instances.

Instruction:

Given two triples with numerical tail entities, please create a comparison-type question based on the given triples and create the corresponding sub-questions for each triple. Finally, you should also give the answer based on the given triple. The final answer should be “Yes” or “No”. Here are some examples:

Demonstrations:

### Example 1:
Triple 1:(Vietnam male, marriageable age, 20 years old)
Triple 2:(Vietnam female, marriageable age, 18 years old)
Q1: What is the marriageable age for Vietnamese men?
Q2: What is the marriageable age for Vietnamese women
Final Question: Is the marriageable age the same for men and women in Vietnam? Answer: No

### Example 2:
Triple1:(Vietnam, population, 94660000)
Triple2:(Halifax, population, 424931)
Q1: What is the population of Vietnam?
Q2: What is the population of Halifax?
Final Question: Which company has less population, Vietnam or Halifax?
Answer: Halifax

Input:

### Your Turn (Just complete your task in the above format and do not return other content):
Subgraph Query:

Table 16: Prompt of verbalization for compare pattern instances.