SymKGQA: Few-Shot Knowledge Graph Question Answering via Symbolic Program Generation and Execution

Prerna Agarwal[*1,2], **Nishant Kumar**[1], **Srikanta Bedathur**[1]
[1]Indian Institute of Technology Delhi, India
[2]IBM Research India
{prerna.agarwal@scai, srikanta@cse}.iitd.ac.in, nishant.1906.kumar@gmail.com

Abstract

Semantic Parsing of natural language questions into their executable logical form (LF) has shown state-of-the-art (SOTA) performance for Knowledge Graph Question Answering (KGQA). However, these methods are not applicable for real-world applications, due to lack of KG-specific training data. Recent advances in the capabilities of Large Language Models (LLMs) has led towards generating low-level LFs such as SPARQL and S-Expression in a few-shot setting. Unfortunately, these methods: (1) are limited to the knowledge of underlying LLM about the LF, (2) performs inferior for the harder complex benchmarks such as KQA Pro, (3) suffers while grounding the generated LF to a specific Knowledge Graph. Recently, a new LF called KoPL (Cao et al., 2022a) has been introduced that explicitly models complex reasoning process step-by-step in a symbolic manner and has shown SOTA on KQA Pro in fully-supervised setting. Inspired by this, we propose **SymKGQA**[1] framework that generates step-by-step **Sym**bolic LF i.e., KoPL in a few-shot in-context learning setting using LLM. Our framework is not dependent on pre-trained knowledge of LLM about KoPL. We further build a Retrieval-Augmented Generation based **Qu**estion-**A**ware **C**ontextual **K**oPL (QUACK) resolver to ground the generated LF. Our experiments with different LLMs and few-shot settings demonstrate that SymKGQA outperforms all other few-shot and even many of the fully-supervised KGQA approaches.

1 Introduction

Knowledge Graph Question Answering (KGQA) has received a lot of research interest in recent years (Saxena et al., 2022; Zhang et al., 2022b; Mitra et al., 2022; Shu et al., 2022; Nie et al., 2022; Neelam et al., 2022). It aims at answering a natural language question using a structured Knowledge Graph (KG) by producing a logical form (LF) such as SPARQL, S-Expression, λ-DCS, etc. that is executed on the KG to retrieve the answer(s). When such techniques are used in applications such as intelligent assists, the annotated training data is generally not available at cold-start, thus, making it difficult to use these techniques without significant investment in building a large, generalizable (to the application) dataset to train the models on.

With recent advancements in Large Language Models (LLMs), the research is moving towards leveraging the reasoning capability of LLMs to obtain the LF for a given Natural Language Question (NLQ) in both fine-tuning (Ye et al., 2022; Shu et al., 2022; Gu et al., 2023) and few-shot settings (Gu et al., 2023; Li et al., 2023b). For few-shot setting, these works have used in-context learning (ICL) paradigm with various state-of-the-art LLMs such as GPT-3 (Brown et al., 2020) variants to generate LF. However, these methods suffer from the following limitations: (1) their LF generative capabilities are limited to the pre-trained knowledge of the underlying LLM about the grammar and semantics of the LF (SPARQL or S-expression). (2) this also results in their inferior performance for harder complex KGQA benchmarks such as KQA Pro (Cao et al., 2022a; Huang et al., 2023). These benchmarks require explicit modeling of KG complexities such as concepts, attributes, qualifiers, etc. to perform joint compositional and numerical reasoning that may not be seen by the LLM. (3) their KG grounding formulation is independent of the context of the question, which limits their performance.

Due to the recent surge in the complexity of questions that users pose to Question Answering (QA) systems, complex KGQA has particularly gained attention (Huang et al., 2023). Complex

[*]P. Agarwal is an employee at IBM Research. This work was carried out as part of PhD research at IIT Delhi. Email id: preragar@in.ibm.com

[1]Codebase and Data: https://github.com/data-iitd/symKGQA

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 10119–10140

questions require joint compositional and numerical reasoning on KG to answer them. Some of the works have demonstrated that step-by-step Chain-of-Thought (CoT) decomposition of the reasoning process is one of the key factors in enhancing the reasoning capability of LLMs for QA tasks (Wei et al., 2023; Zhou et al., 2023; Niu et al., 2023). However, a recent shift towards decomposing the problem with symbolic formalization i.e., Chain-of-Symbol (CoS) (Hu et al., 2023) has emerged as another key factor to observe further enhancements in reasoning (Zhang et al., 2022a; Wolfson et al., 2020). Additionally, some of the works (Cao et al., 2022a; Nie et al., 2022; Liu et al., 2024) have shown that using the same underlying model with LFs having high to low formalization i.e., dissimilarity to natural language has shown an inverse trend in the performance for KGQA.

Recently, KoPL (Cao et al., 2022a), a symbolic LF has been introduced that defines various functions (operations) on KG catering to various complexities. It reflects the CoS reasoning process as compared to other LFs that have to be used as a whole query (Huang et al., 2023; Liu et al., 2024). Its modular functions and their inputs make it suitable for complex compositional and numerical reasoning while being interpretable. Figure 1 shows an example of a complex question and its corresponding LFs with high (top) to low (bottom) level of formalization. As shown, KoPL is most decomposable, interpretable LF while having symbolic formalization to solve complex questions.

Another challenge that most of the existing LLM-based techniques face while generating LF for KGQA is - that LLM picks KG items (entities, relations, etc.) based on its inherent knowledge or picks an appropriate phrase from the input question. These items have to be grounded/resolved for a KG before the generated LF can be executed. To illustrate this, consider the generated KoPL in Figure 1. The functional input of Step 3 is `occupy` that does not exist in the KG and the closest relations in the KG are {`mass`, `area`, etc.}. The existing methods pick the closest relation based on the similarity between `occupy` and the candidates {`mass` (0.91[2]), `area` (0.84)}. Hence, they would pick `mass` due to its higher similarity than `area`. However, if we consider an additional context of the input question while choosing the closest relation, `area` should be chosen (desired relation). Existing methods do not take into consideration this context of the input question and hence, suffer in grounding the LF.

[2]BERT similarity score

Question:	***Which administrative territorial entity occupying over 270 square kilometers has Davenport as its capital?***
SPARQL:	SELECT DISTINCT ?e WHERE { ?e <pred:instance_of> ?c . ?c <pred:name> "administrative territorial entity" . ?e <area> ?pv . ?pv <pred:unit> "square kilometre" . ?pv <pred:value> ?v . FILTER (?v > "270"^^xsd:double) . ?e <capital> ?e_1 . ?e_1 <pred:name> "Davenport" . }
Lambda DCS:	(call SW.listValue (call SW.and (call SW.filter (call SW.getProperty (call SW.singleton administrative territorial entity) (string ! type)) (string area) (string >) (number 270 square kilometre)) (call SW.filter (call SW.getProperty (call SW.singleton administrative territorial entity) (string ! type)) (call SW.reverse (string capital)))))
Actual KoPL:	(1)FindAll() (2)FilterConcept(administrative territorial entity) (3)FilterNum(area,270 square kilometre,>) (4)FilterStr(capital, Davenport)
Generated KoPL:	(1)FindAll() (2)FilterConcept(administrative territorial entity) (3)FilterStr(occupy,270 square kilometre,>) (4)FilterStr(capital, Davenport)

Figure 1: Logical Forms with varying formalization levels for a complex question.

Addressing the above-discussed challenges, the contributions of our work are as follows:

1. We propose SymKGQA: a KGQA framework specifically designed to generate **Sym**bolic KoPL program for a given complex natural language question using LLMs in a few-shot setting.

2. We further build a Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) based **Qu**estion-**A**ware **C**ontextual **K**oPL (QUACK) resolver to ground the generated KoPL for a given KG.

3. We experiment with different LLMs on 3 datasets with varying complexity of questions and KG to demonstrate the flexibility and robustness of our approach.

We adopt the few-shot ICL paradigm with KoPL function definitions and instructions as prompt to generate CoS KoPL programs. This prompting approach makes SymKGQA independent of the pre-trained knowledge of the underlying LLM about KoPL.

Popular KGQA datasets such as MetaQA (Zhang et al., 2018), WebQSP (Yih et al., 2016), etc. has sparse coverage of complex questions. On the other hand, a recently introduced KQA Pro dataset (Cao et al., 2022a) contains much harder complex QA pairs that require compositional and numerical reasoning as compared to other datasets such as GrailQA (Gu et al., 2021; Dutt et al., 2023). Thus, this work utilizes KQA Pro as the core benchmark dataset for experiments.

Our extensive experiments demonstrate that SymKGQA outperforms all other LLM-based few-shot and even most of the SOTA fully-supervised KGQA approaches by a significant margin. *To the best of our knowledge, SymKGQA is the first few-shot KGQA framework that uses a CoS approach (using KoPLs) with RAG-based QUACK resolver.*

2 Related Work

2.1 Fully-Supervised KGQA methods

These KGQA techniques requires a large amount of annotated training data to learn the model. They are further divided into 2 sub-categories:
(1) **Semantic Parsing based techniques**: This includes methods (Shu et al., 2022; Nie et al., 2022; Neelam et al., 2022) that transform the natural language question into LF, such as SPARQL, S-expression, etc. that are directly executable on KG to retrieve the answer. KQA Pro (Cao et al., 2022a) work also provides a technique to learn questions to KoPL/SPARQL mapping using scratch training of sequence-to-sequence models such as BART. Recently, a new intermediate representation (Nie et al., 2022) i.e., GraphQ IR has been introduced that has a language-like expression to define the syntax. They demonstrate that the performance obtained with BART + GraphQ IR > BART + KoPL > BART + SPARQL. These works have shown SOTA on the KQA Pro dataset. Among these LFs, only KoPL is symbolic in nature that enables CoS.
(2) **Information Retrieval based techniques:** This includes end-to-end learning approaches that are not tied to any specific KG (Miller et al., 2016; Saxena et al., 2020; Shi et al., 2021; Zhang et al., 2022b; Mitra et al., 2022; Saxena et al., 2022). These techniques use only QA pairs with KG triples to rank candidate entities for a given question. These techniques also require a large amount of QA pairs data for learning and can't be used if the data is not available at cold-start.

Both categories of techniques have shown SOTA performance on specific datasets, however, highly sensitive to a large amount of manually annotated data for learning. Hence, none of these techniques can be adopted for practical applications where KG specific training data is not available.

2.2 Few-Shot KGQA methods

Recent advancements in the development of various pre-trained LLMs, such as T5 (Raffel et al., 2023) and Codex (Chen et al., 2021), have shown SOTA performance on various downstream tasks. Some of the LLM based frameworks have been proposed for KGQA task as well. RnG-KBQA (Ye et al., 2022), TIARA (Shu et al., 2022), KB_BINDER (Li et al., 2023a) and similar works convert natural language sequences to S-Expression with constrained decoding using LLMs. Pangu (Gu et al., 2023) capitalizes on the discriminative ability of LLMs rather than generation. BYOKG (Bring Your Own KG) (Agarwal et al., 2023) leverages an LLM-based symbolic agent to comprehend KG information through exploration. It starts at random nodes, inspecting the labels of adjacent nodes and edges, and combining them with their prior world knowledge. FlexKBQA (Li et al., 2023b) introduces an execution-guided self-training method and surpasses all other few-shot methods. These techniques generate low-level LFs such as SPARQL, and S-expressions but suffers from various limitations (discussed in Section 1).

3 Proposed Framework: SymKGQA

We first briefly define the problem statement and then introduce the details of SymKGQA framework shown in Figure 2, which consists of two stages: (1) KoPL Generation: using Few-Shot ICL; and (2) RAG based QUACK Resolver.

3.1 Problem Statement

The goal of the SymKGQA framework is:

1. Generate the CoS KoPL program steps (P) for a given natural language question (Q) using a pre-trained LLM (M) by providing a prompt (Y).

2. Execute P, on KG K, with the help of QUACK resolver, to obtain the final answer(s). Here, $K \subset E \times R \times (E \cup L \cup C)$, where C is the set of concepts[3], E is the set of entities, L is the set of attributes/literals and R a set of binary relations.

3.2 Generation: Few-Shot ICL Prompting

As mentioned earlier, we adopt the ICL paradigm to generate the CoS KoPL steps for a given complex natural language question using a pre-trained LLM. We first create a prompt $Y = (I, F, D)$ that consists of a set of Instructions I, KoPL Function Information F, and Few-Shot Demonstrations D (also shown in Figure 2).
Instructions I: It describes the task and outlines generation policies for program steps P. Few instructions are shown below. Refer to Appendix A.1 for a full set of instructions.

Instructions

- The task is to come up with the steps of functions for the given test question using the list of functions provided.
- To generate these steps you should match the output of the current step function with functional inputs of the next step function. ....

[3] A concept is an abstraction of a set of entities

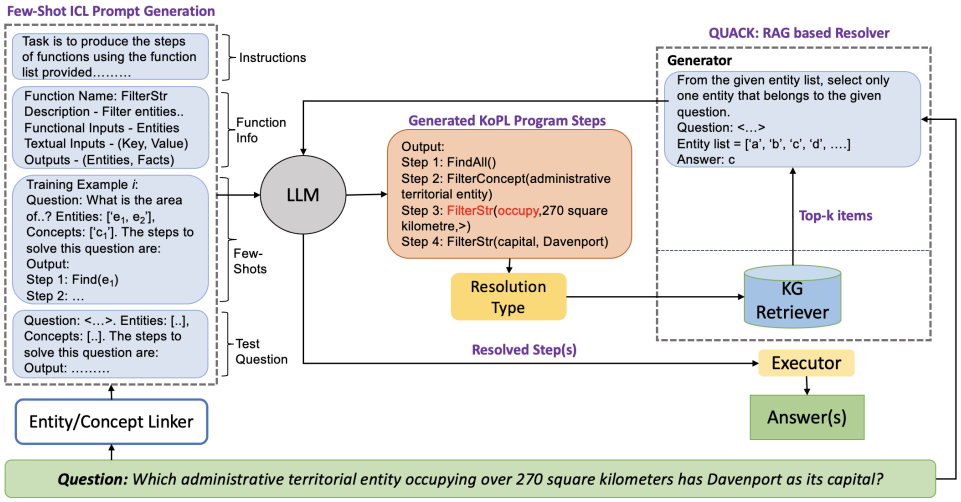

Figure 2: SymKGQA: A Few-Shot KGQA Framework based on ***Sym***bolic Program Generation and Execution

Function Information F: We provide the information of all 27 KoPL functions for the LLM to understand and use them in the generation process (shown below). Refer to Appendix A.1 to find the details of all 27 KoPL functions. We specifically provide the function name, description, inputs (functional and textual), and outputs. The functional inputs come from the previous program steps whereas the textual inputs come from the question.

Function Information

Function Name: Relate
Description - Find entities that have a specific relation with the given entity in the knowledge graph
Functional Inputs - Entity
Textual Inputs - (Predicate, Direction)
Outputs - (Entities, Facts)

Demonstration Selection D: The demonstration selection is a crucial part of generation. The demonstration questions should be selected such that their KoPL program steps cover all functions. Here, we don't require annotated KoPLs for the full training dataset, we just need them to create a pool of few-shots (approx. $10 - 100$ questions), as described below. We experiment with the below methods to select $n = 10$ demonstration questions for ICL:

- **Manual selection**: Here, we manually select $n = 10$ natural language questions from the train set keeping in mind that their KoPL program steps, when annotated, would cover a maximum number of functions with minimal overlap to obtain diverse yet complete demonstration set D. For all the test questions of a particular dataset, D will remain fixed. Refer to Appendix A.1 for the n manually chosen set D.

- **Dynamic selection**: We perform function coverage-based pool selection of N (100) natural language questions from the train set inspired from "bottom-up matching" in (Drozdov et al., 2022).

1. *Pool Creation (N)*: Since, `Find` and `FindAll` functions appear with almost all other remaining 25 functions, we randomly select $k = N/25$ questions that specifically demonstrate each of the 25 functions (except `Find` and `FindAll`). This ensures diversity and coverage of all functions.
2. *Obtaining D (n)*: We build a retriever with the pool to retrieve top $n = 10$ nearest neighbor questions to the test question as D. After experimentation, we chose ChromaDB[4], a vector-based retriever, due to its superior performance $(5 - 7\%)$ as compared to popular BM25 retriever (Robertson and Zaragoza, 2009). We mask the entities and concepts in the question to exclude their interference in determination of semantic similarity. Cosine similarity is used as the distance metric.

The value of n is also dependent on the prompt length supported by an LLM. The demonstration set D of size $n = 10$ thus obtained is added to the prompt using the template shown.

The template, however, requires the set of entities and concepts (if present) to be annotated for each demonstration $d \in D$ in the prompt using the KG. The same template is used to add the test question also in the prompt.

Entity and Concept Identification: For a given complex natural language question Q during testing, the set of concepts $C' \subset C$ and the set of entities $E' \subset E$ present in Q can be annotated using any off-the-shelf entity linker such as (Gu et al.,

[4] `https://docs.trychroma.com/`

2023; Mohammed et al., 2018) popularly used in KGQA, Named Entity Recognition (NER) methods, POS tags, or can be explicitly provided. We use NER as the entity linker for SymKGQA.

Training Example i

Question: Does Pierce County that is located in Washington or Grays Harbor County have less area?
Entities: ['Washington', 'Pierce County', 'Grays Harbor County'], Concepts: None. The steps to solve this question are:
Output:
Step 1: Find(Washington)
Step 2: Relate(located in the administrative territorial entity, backward)
Step 3: Find(Pierce County)
Step 4: And()
Step 5: Find(Grays Harbor County)
Step 6: SelectBetween(area, less)

After assembling the prompt Y using I, F and D, the LLM is now supposed to generate the correct CoS KoPL program steps for each test question conditioned on prompt Y and Q.

Adding I and F makes Y independent of the pretrained knowledge of the underlying LLM about the LF (i.e., KoPL) while handling complex questions. This prompting technique is applicable to any LF supporting CoS-based reasoning. Appendix A.1 contains the complete prompt for a sample test question. We compare prompt templates used by existing approaches and SymKGQA (Appendix A.2). Hyperparameter details for generation are provided in Section 4.4.

3.3 RAG based QUACK (*Qu*estion-*A*ware *C*ontextual *K*oPL) Resolver

The KoPL steps generated by LLM can have discrepancies in terms of: (a) entity and concept names; (b) relations and attributes; (c) numerical operators; (d) numerical quantity and their units; and (e) symbolic functions. This would hinder the direct execution of the generated steps on the KG. Hence, we resolve each program step with the help of RAG before executing it.

Resolution: We resolve each of the above defined discrepancy, as described below:

1. *Entity and Concept Names:* The entity and concept names present in the generated program can be semantically similar but not grounded to a KG due to: (a) imperfect identification of E' and C' by entity linker during the prompt creation. (b) LLM discards the identified E' and C' and hallucinates based on its inherent knowledge. Hence, to resolve such cases, we use the following approach: (a) **Retrieval**: Top 10 similar entities/concepts from the KG for each entity/concept name present in the generated step are retrieved. BERT based semantic similarity retriever is used to achieve this. (b) **Generation**: LLM is prompted with the retrieved list along with the question to select the most probable entity/concept from the given list. to prevent hallucinations. The prompt template used at this step is shown in Figure 2.
More details on the RAG approach are provided in Appendix A.3.

2. *Relation and Attribute Names:* LLM chooses relations/attributes present in Q itself while generating program steps due to the absence of relation/attribute linkers. Hence, it needs to be grounded for the KG explicitly. We use the same RAG approach as (1). At the retrieval step, the top 10 similar relations/attributes from the KG for each generated relation/attribute name are retrieved. At the Generation step, LLM selects the most probable relation/attribute in the context of Q. A sample prompt is shown below:

Resolution Prompt

From below relation list, select only top relation that is most similar to the relation extracted from question.

Question: What is the higher education institution is headquartered in the city whose postal code is 20157?

Relation list = ['headquarters location', 'work location', 'located in the administrative territorial entity', 'capital of', 'located in time zone', 'residence', 'capital', 'country', 'filming location', 'place of birth']
Relation extracted from the Question: ['headquartered in']
Answer:

3. *Numerical Operators:* Sometimes, the numerical operators such as $\{<, >, =\}$, etc. are either missing from the "textual input" of the KoPL function or represented in textual form (such as "less than", "greater than", etc). However, the KoPL executor requires the numerical operator in the form of symbol. Hence, to correctly identify the numerical operator symbol from Q, we provide the list of pre-defined numerical operators[5] for the LLM to pick the right one in the context of Q using the same RAG approach as (1).

4. *Numerical Quantity and Units:* The numerical quantity unit present in Q and the unit present in the KG for that attribute can be different. Hence, it is important to normalize the numerical quantity

[5]Refer to Appendix A.4 for the list of numerical operators

present in Q into the unit present in the KG. We use *sympy*[6] python library to achieve this.

5. *Symbolic Function Names:* As illustrated in Figure 1, sometimes LLM picks wrong symbolic function names but generates logically correct steps. In Figure 1, the model generated `FilterStr` at Step 3 but the functional input is a number with an operator. Considering the context of the question, the correct function that should be executed is `FilterNum`. To rectify such functional errors, we use the same RAG approach as (1). At retrieval step, the set of functions with the same functional inputs (from the previous step) and textual inputs (corrected using the above-described resolutions) that are possible at the current step are retrieved. Then, the most probable function to be used in the context of Q is selected at Generation step.

Most of the existing methods such as (Li et al., 2023a) use only lexicon-based similarity to select top-k closest entities and relations (without considering Q as context) to resolve the LF. On the other hand, *our approach utilize Q as the context so that the interconnected constraints among entities, concepts, relations, etc. are considered in a unified way while choosing the most relevant KG item.*

Execution: The resolved KoPL program steps are then executed on the KG as per the default KoPL executor provided (Cao et al., 2022a).

4 Experiments

Our experiments are outlined to answer the following research questions: (1) How does SymKGQA compare against existing KGQA frameworks in few-shot setting? (2) How does the performance of SymKGQA vary with different pre-trained LLMs and demonstration selection methods? (3) What are the contributions of each stage of SymKGQA?

4.1 Datasets

We experiment with 3 datasets with different: (a) question complexities; (b) KG domain; and (c) number of inference hops required. The statistics of these datasets are shown in Table 1.

1. **KQA Pro** (Cao et al., 2022a): This is a very recent dataset based on Wikidata KG and contains complex QA pairs with numerical quantities, concepts and entities for multi-hop compositional and numerical reasoning. It contains questions that require up to 10 hops to obtain the answer(s). The test set is closed, hence, we use the validation set as the test set for our experiments.

2. **MetaQA** (Zhang et al., 2018): It is a large-scale multi-hop dataset widely used for domain-specific KGQA (movie domain). It provides 1-hop, 2-hop, and 3-hop QA pairs.

3. **WebQSP** (Yih et al., 2016): This dataset contains natural language questions up to 2 hops from Google query logs answerable through Freebase KG. We sample $N = 100$ questions (explained in Section 3.2) to create the pool for dynamic demonstration selection and annotate their KoPLs.

Dataset	KoPL	Train	Val	Test
KQA Pro	✓	94,376	11,797	11,797
MetaQA 1-hop	✓	96,106	9,992	9,947
MetaQA 2-hop	✓	118,948	14,872	14,872
MetaQA 3-hop	✓	114,196	14,274	14,274
WebQSP	×	2,998	100	1,639

Table 1: Statistics of the Datasets used

4.2 Models

We experiment with below SOTA LLMs with varying number of parameters:

1. **CodeLlama Instruct (34B)** (Rozière et al., 2023): A code-focused LLM, built upon Llama 2. It can follow programming instructions without prior training for various programming tasks. The prompt/context limit of this model is $4K$ tokens.

2. **Llama-2 (70B)** (Touvron et al., 2023): It is an auto-regressive model that uses an optimized transformer architecture. The prompt/context limit of this model is $4K$ tokens.

3. **PaLM 2 (540B)** (Chowdhery et al., 2022): One of the SOTA LLM with improved multilingual, reasoning, and coding capabilities. We use `text-bison-001` version of this model. The prompt/context limit of this model is $8K$ tokens.

4. **GPT3.5-turbo**: Due to the costs associated with the API, we randomly sample 100 questions from the test set of each of the three datasets and compare the performance with other models.

4.3 Baselines

We select the following 3 categories of baselines:

• **Fully Supervised**: We compare with SOTA fully supervised techniques for each dataset. It includes KVMemNet (Miller et al., 2016), EmbedKGQA (Saxena et al., 2020) for all datasets and:

1. MetaQA: SRN (Qiu et al., 2020), PullNet (Sun et al., 2019), NSM (He et al., 2021) and TransferNet (Shi et al., 2021).

[6] https://pypi.org/project/sympy/

Method	Models	F1
Fully Supervised	Program Transfer	74.6
	EmbedKGQA	66.6
	KVMemNet	46.7
	DecAF	78.8
	Subgraph Retrieval	66.7
Few-Shot (100 Shots)	FlexKBQA (S-Expression + GPT3.5)	60.6
	KB_BINDER (1) (S-Expression + Codex)	52.5
	Pangu (S-Expression + Codex)	54.5
Direct QA	CodeLlama Ins.	39.6
	Llama-2	46.7
	PaLM 2	33.5
SymKGQA Manual Few-Shot (10 Shots)	CodeLlama Ins.	61.9
	Llama-2	65.2
	PaLM 2	70.6
SymKGQA Dynamic Few-Shot (100 Shots)	CodeLlama Ins.	71.2
	Llama-2	<u>71.9</u>
	PaLM 2	**75.4**

(a) WebQSP

Method	Models	Hits@1		
		1-hop	2-hop	3-hop
Fully Supervised	KVMemNet	96.2	82.7	48.9
	EmbedKGQA	97.5	98.8	94.8
	SRN	97.0	95.1	75.2
	PullNet	97.0	99.9	91.4
	NSM	97.2	99.9	98.9
	TransferNet	97.5	100	100
Direct QA	CodeLlama Ins.	27.9	28.8	46.7
	Llama-2	58.0	37.6	55.8
	PaLM 2	27.6	9.1	21.5
SymKGQA Manual Few-Shot (10 Shots)	CodeLlama Ins.	98.1	**99.9**	<u>99.8</u>
	Llama-2	99.1	99.7	**99.9**
	PaLM 2	99.7	99.7	**99.9**
SymKGQA Dynamic Few-Shot (100 Shots)	CodeLlama Ins.	<u>99.8</u>	**99.9**	<u>99.8</u>
	Llama-2	**99.9**	**99.9**	**99.9**
	PaLM 2	99.7	<u>99.8</u>	**99.9**

(b) MetaQA

Table 2: Results on WebQSP and MetaQA (*Bold and underline fonts denote the best and the second best performance **only among few-shot setting**. Rows in grey color denote fully-supervised methods. There are no few-shot baselines for MetaQA[10].)

Method	Models	Hits@1
Fully Supervised	KVMemNet	6.9
	EmbedKGQA	20.27
	SRN	11.84
	RGCN	29.12
	Subgraph Retrieval	22.82
	BART + KoPL	83.28
	GraphQ IR	79.13
Few-Shot (100 Shots)	FlexKBQA (SPARQL + GPT3.5)	42.68
	LLM-ICL (SPARQL + Codex)	27.75
Direct QA	CodeLlama Ins.	28.7
	Llama-2	31.2
	PaLM 2	21.5
SymKGQA Manual Few-Shot (10 Shots)	CodeLlama Ins.	<u>50.9</u>
	Llama-2	38.9
	PaLM 2	39.0
SymKGQA Dynamic Few-Shot (100 Shots)	CodeLlama Ins.	**51.1**
	Llama-2	45.6
	PaLM 2	42.7

Table 3: Results on KQA Pro*

2. KQA Pro: SRN, RGCN (Schlichtkrull et al., 2018), Subgraph Retrieval (Zhang et al., 2022b), BART + KoPL (Cao et al., 2022a), GraphQ IR (Nie et al., 2022).
3. WebQSP: DecAF (Yu et al., 2023), Subgraph Retrieval (Zhang et al., 2022b) and Program Transfer (Cao et al., 2022b).

• **Few-Shot**: We compare with few-shot techniques that have shown SOTA for each dataset:

1. KQA Pro: FlexKBQA (Li et al., 2023b) and LLM-ICL[7] both generating SPARQL.
2. WebQSP: FlexKBQA, KB_BINDER (Li et al., 2023a) and Pangu (Gu et al., 2023) all generating S-Expression. We use KB_BINDER (1) setting for fair comparison.

There are no few-shot baselines that has benchmarked MetaQA[8].

• **Direct QA**: LLMs are pre-trained on a large body of knowledge. Hence, we ask the question directly to the LLM to obtain the answer(s).

Refer to Appendix A.6 for more details on the techniques adopted by each of the baseline.

4.4 Implementation and Hyper-parameters

PyTorch[9] Python framework is used to implement SymKGQA. Greedy decoding is used to obtain the KoPL program steps from the LLMs (for reproducibility). Additionally, the program steps are obtained using 2 iterations of sampling decoding (for exploration) with temperature=0.3 and top_k=30 to handle the cases where the program steps generated from greedy decoding (after applying QUACK resolution) results in error. We use $n = 10$ and $N = 100$ shots for manual and dynamic demonstration selection. `all-distilroberta-v1`[10] BERT model is used at the retrieval step in QUACK. Hits@1 for KQA Pro and MetaQA, F1 score for WebQSP are the evaluation metrics used to evaluate and compare the performance of SymKGQA with baselines.

5 Results and Analysis

We discuss the results of SymKGQA on each dataset in detail. Qualitative analysis examples are provided in Appendix A.5.

[7]Alternate to Pangu for evaluation (as used in FlexKBQA)

[8]Benchmarking FlexKBQA, KB_BINDER and Pangu was not feasible due to closed access of GPT3.5 models.

[9]https://pypi.org/project/torch/

[10]https://huggingface.co/sentence-transformers/all-distilroberta-v1

Model	**Overall**	Multihop	Qualifier	Comparison	Logical	Count	Verify	Zero-Shot
CodeLlama Ins.	51.11	44.29	32.50	49.02	37.28	36.95	54.32	56.99
Llama-2	45.62	38.28	29.90	40.80	28.84	34.90	44.57	54.06
Palm-2	42.75	33.61	17.57	40.89	28.03	31.34	50.52	45.07

Table 4: Category-wise accuracy of different models on KQA Pro dataset.

5.1 Results on KQA Pro

The results of SymKGQA compared with different baselines are shown in Table 3. The detailed category-wise performance is provided in Table 4. As shown, SymKGQA with CodeLlama Instruct model outperforms all few-shot baselines and *achieves a new SOTA in the few-shot setting for KQA Pro*. Specifically, SymKGQA beats FlexKBQA by 8% and LLM-ICL by 24% (both generates SPARQL using GPT-3 variants).

The performance with PaLM 2 and Llama-2 LLMs are comparable but inferior to what we observe with CodeLlama Instruct. The performance with dynamic demonstration selection is higher than the manual selection method. SymKGQA even outperforms all fully-supervised baselines except BART + KoPL and GraphQ IR. For the sampled test set with 100 questions (shown in Table 6), the performance of SymKGQA with GPT3.5-turbo beats all other LLMs by a huge margin. Hence, this encourages us to believe that SymKGQA when used with GPT3.5-turbo has the potential to surpass BART + KoPL and GraphQ IR baselines when evaluated on a full test set.

5.2 Results on WebQSP

The results of SymKGQA for the WebQSP dataset are shown in Table 2a. As shown, *SymKGQA achieves a new SOTA in the few-shot setting for WebQSP* outperforming all few-shot baselines. All few-shot baselines generates S-Expression. SymKGQA even beats all fully supervised baselines except DecAF where the performance of SymKGQA is slightly less. The performance with dynamic demonstration selection is higher than the manual selection method. For the sampled test set with 100 questions (see Table 6), the performance of SymKGQA with GPT3.5-turbo is a bit lower than other LLMs except for CodeLlama Instruct.

5.3 Results on MetaQA

The results of SymKGQA for the MetaQA dataset are shown in Table 2b. As shown, *SymKGQA achieves a new SOTA for 1-hop* with all 3 LLMs. For 2-hop and 3-hop, SymKGQA even beats all fully supervised baselines except TransferNet where the performance is nearly the same. The performance with dynamic demonstration selection is slightly higher than the manual selection method. For the sampled test set (shown in Table 6), the performance of SymKGQA with GPT3.5-turbo is nearly the same as all other LLMs.

5.4 Observations

Overall, we observed the following:

1. The low performance of the Direct QA baselines for all datasets indicates that LLMs inherently don't have sufficient knowledge to answer the questions and hence, are not biased by their inherent knowledge during KoPL generation.
2. The difference in the performance with fully-supervised baselines is attributed to the use of a large amount of annotated LF data as strong supervision signals during learning, whereas, our framework addresses a more challenging few-shot setting, assuming only few-shot availability.
3. The performance of SymKGQA with dynamic selection is expected to increase further as N increases.
4. GPT3.5-turbo has the better ability to provide reasoning over complex questions (KQA Pro) as compared to other models, whereas, all the models possess similar ability to handle simpler questions (MetaQA and WebSQP).

Note: *Few-shot and fully supervised approaches can not be compared directly. However, we make this comparison only to show that SymKGQA is able to perform better than fully supervised methods in many scenarios. There is still some performance gap with fully supervised approaches that suggests potential avenues for future research.*

5.5 Ablation Study

Table 5 shows the SymKGQA performance when:

1. Programs generated using sampling decoding (for exploration) are eliminated while obtaining the answers. On this elimination, the performance dropped by 6-14% for KQA Pro dataset. This high-

Dataset	Config	CodeLlama Ins.	Llama-2	PaLM 2
KQA Pro	- Sampling	44.4 (↓ 6.5%)	38.0 (↓ 13%)	37.3 (↓ 14%)
	- QUACK	26.0 (↓ 25%)	23.1 (↓ 28%)	23.3 (↓ 27.5%)
MetaQA (1-hop)	- Sampling	99.7 (↓ 0.1%)	99.9 (↓ 0%)	99.1 (↓ 0.6%)
	- QUACK	14.2 (↓ 85.6%)	14.1 (↓ 85.7%)	14.1 (↓ 85.4%)
MetaQA (2-hop)	- Sampling	99.8 (↓ 0.1%)	99.9 (↓ 0%)	99.7 (↓ 0.1%)
	- QUACK	0.0 (↓ 99.9%)	0.0 (↓ 99.9%)	0.0 (↓ 99.8%)
MetaQA (3-hop)	- Sampling	99.5 (↓ 0.3%)	99.8 (↓ 0.1%)	99.6 (↓ 0.1%)
	- QUACK	0.0 (↓ 99.8%)	0.0 (↓ 99.9%)	0.0 (↓ 99.9%)
WebQSP	- Sampling	70.9 (↓ 0.3%)	71.7 (↓ 0.2%)	75.1 (↓ 0.3%)
	- QUACK	13.7 (↓ 57.5%)	13.5 (↓ 58.4%)	20.0 (↓ 55.4%)

Table 5: Ablation Study (on Dynamic Few-Shot Setting)

Models	KQA Pro	MetaQA	WebQSP
CodeLlama Ins.	46.0	98.0	62.0
Llama-2	40.0	**100.0**	71.0
PaLM 2	28.0	99.0	**72.0**
GPT3.5-turbo	**88.0**	99.0	68.0

Table 6: Comparison with GPT3.5 (Sampled Test Set)

lights the importance of the sampling decoding in the framework. However, not much variation is observed in the generation and hence, in performance for MetaQA and WebQSP due to the simplicity of the programs for this dataset.

2. QUACK resolver is eliminated while obtaining the answers. For MetaQA 2 and 3-hop, none of the generated KoPLs were directly executable and required one or more resolutions. Hence, removal of the QUACK resolver shows a huge drop in the performance of the model. However, the proposed QUACK resolver is able to resolve most of the generated KoPL program steps and interpret them semantically and not just syntactically.

The quantification of decrease in the performance demonstrates the importance and contribution of each stage.

5.6 Error Analysis

We analyze the following sources of errors in SymKGQA (on an average across different LLMs):

• *Entity Linking*: Incorrect entity and concept linking after the QUACK resolver is observed only in KQA Pro, for 1.5% of questions. No such issues are observed in MetaQA and WebQSP.

• *Mismatch in Input-Output types of the generated steps*: Errors in program execution due to mismatch in input-output types of the steps generated are seen only for KQA Pro, for 29% of questions. No such errors are seen for MetaQA and WebQSP.

• *Incorrect KG Grounding by QUACK Resolution*: The wrong answers obtained due to incorrect grounding of KG components (other than entity and concepts) by the QUACK resolver is observed in KQA Pro, for 14% and WebQSP, for 20% of questions. No such issue is observed for MetaQA. A detailed analysis is provided in Appendix A.3.

5.7 Inference Latency

We make API calls LLMs for the generation of KoPL programs with average latency of 5s/call that runs on CPU. We use NVIDIA V100 GPU with 32 GB RAM for QUACK resolution and execution. On an average it consumes following GPU hours:

• 3 Hrs: Execution and Resolution for 11,797 test questions of KQA Pro

• 2 Hrs: Execution and Resolution for 1,639 test questions of WebQSP

• 3 Hrs: Execution and Resolution for 13,031 test questions of MetaQA (average of all hops).

6 Conclusion

We propose SymKGQA, that generates and executes symbolic programs (KoPL) in a few-shot setting. To the best of our knowledge, it is the first one to generate KoPL LF using LLMs for complex KGQA. We provide a method for LLM prompt generation with manual and dynamic demonstration selection for few-shot ICL. We further propose a RAG based *Qu*estion-*A*ware *C*ontextual *K*oPL (QUACK) Resolver that grounds the generated KoPL for a KG. Our extensive experiments shows that SymKGQA surpass all few-shot and most of the fully supervised SOTA baselines, setting a new benchmark for various complex KGQA datasets. Our approach being independent of underlying LLM knowledge, makes it applicable for any other symbolic LF while being interpretable.

7 Acknowledgements

This work was supported by an IBM AI Horizons Network (AIHN) grant. Srikanta Bedathur was partially supported by a DS Chair of AI fellowship.

8 Limitations

Despite the strong reasoning capabilities of SOTA LLMs, obtaining step-by-step CoS program may be prone to errors, which can impact the performance of SymKGQA, as discussed in Section 5.6. Most of the errors stem from a mismatch between the input-output type of the subsequent generated steps. Fortunately, these errors are generally interpretable and can be rectified with human intervention.

9 Risks

Our work does not have any obvious risks that we are aware of.

References

Dhruv Agarwal, Rajarshi Das, Sopan Khosla, and Rashmi Gangadharaiah. 2023. Bring your own kg: Self-supervised program synthesis for zero-shot kgqa.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

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

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

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. 2022. Palm: Scaling language modeling with pathways.

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. 2022. Compositional semantic parsing with large language models.

Ritam Dutt, Sopan Khosla, Vinayshekhar Bannihatti Kumar, and Rashmi Gangadharaiah. 2023. GrailQA++: A challenging zero-shot benchmark for knowledge base question answering. 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 (Volume 1: Long Papers)*, pages 897–909, Nusa Dua, Bali. Association for Computational Linguistics.

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 iid: three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, pages 3477–3488. ACM.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. WSDM '21, page 553–561, New York, NY, USA. Association for Computing Machinery.

Hanxu Hu, Hongyuan Lu, Huajian Zhang, Wai Lam, and Yue Zhang. 2023. Chain-of-symbol prompting elicits planning in large langauge models.

Xiang Huang, Sitao Cheng, Yuheng Bao, Shanshan Huang, and Yuzhong Qu. 2023. MarkQA: A large scale KBQA dataset with numerical reasoning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10241–10259, Singapore. Association for Computational Linguistics.

Patrick 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. NIPS'20, Red Hook, NY, USA. Curran Associates Inc.

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.

Jinxin Liu, Shulin Cao, Jiaxin Shi, Tingjian Zhang, Lei Hou, and Juanzi Li. 2024. Probing structured semantics understanding and generation of language models via question answering.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1400–1409, Austin, Texas. Association for Computational Linguistics.

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

Salman Mohammed, Peng Shi, and Jimmy Lin. 2018. Strong baselines for simple question answering over knowledge graphs with and without neural networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 291–296, New Orleans, Louisiana. Association for Computational Linguistics.

Sumit Neelam, Udit Sharma, Hima Karanam, Shajith Ikbal, Pavan Kapanipathi, Ibrahim Abdelaziz, Nandana Mihindukulasooriya, Young-Suk Lee, Santosh Srivastava, Cezar Pendus, Saswati Dana, Dinesh Garg, Achille Fokoue, G P Shrivatsa Bhargav, Dinesh Khandelwal, Srinivas Ravishankar, Sairam Gurajada, Maria Chang, Rosario Uceda-Sosa, Salim Roukos, Alexander Gray, Guilherme Lima, Ryan Riegel, Francois Luus, and L V Subramaniam. 2022. SYGMA: A system for generalizable and modular question answering over knowledge bases. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3866–3879, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Lunyiu Nie, Shulin Cao, Jiaxin Shi, Jiuding Sun, Qi Tian, Lei Hou, Juanzi Li, and Jidong Zhai. 2022. GraphQ IR: Unifying the semantic parsing of graph query languages with one intermediate representation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5848–5865, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yilin Niu, Fei Huang, Wei Liu, Jianwei Cui, Bin Wang, and Minlie Huang. 2023. Bridging the gap between synthetic and natural questions via sentence decomposition for semantic parsing. *Transactions of the Association for Computational Linguistics*, 11:367–383.

Yunqi Qiu, Yuanzhuo Wang, Xiaolong Jin, and Kun Zhang. 2020. Stepwise reasoning for multi-relation question answering over knowledge graph with weak supervision. WSDM '20, page 474–482, New York, NY, USA. Association for Computing Machinery.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code.

Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. 2022. Sequence-to-sequence knowledge graph completion and question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4498–4507, Online. Association for Computational Linguistics.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *The Semantic Web*, pages 593–607, Cham. Springer International Publishing.

Jiaxin Shi, Shulin Cao, Lei Hou, Juanzi Li, and Hanwang Zhang. 2021. TransferNet: An effective and transparent framework for multi-hop question answering over relation graph. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4149–4158, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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.

Haitian Sun, Tania Bedrax-Weiss, and William Cohen. 2019. PullNet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2380–2390, Hong Kong, China. 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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break it down: A question understanding benchmark. *Transactions of the 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, 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 Wang, Zhiguo Wang, and Bing Xiang. 2023. Decaf: Joint decoding of answers and logical forms for question answering over knowledge bases.

Hanlin Zhang, YiFan Zhang, Li Erran Li, and Eric Xing. 2022a. The impact of symbolic representations on in-context learning for few-shot reasoning. In *NeurIPS 2022 Workshop on Neuro Causal and Symbolic AI (nCSI)*.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. 2022b. 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.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J. Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. AAAI'18/IAAI'18/EAAI'18. AAAI Press.

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

A Appendix

A.1 Few-Shot ICL Prompt

Full prompt for a sample test question of KQA Pro is shown in Figure 3.

A.2 Few-Shot ICL Prompt Templates of Existing Methods

Full prompt for a sample test question of KQA Pro is shown in Table 10.

A.3 RAG based QUACK Resolver

Retriever: The full set of each KG item i.e., entities, concepts, relations, attributes, qualifiers, etc. are indexed separately in ChromaDB. The KG item generated at each step is used as the input query to retrieve the top-10 closest corresponding KG item. The retrieved list is then passed to the LLM using the prompt shown in Figure 2 (Generation). This approach of encoding each KG item set separately provides the following advantages over indexing triples for retrieval:

For the grounding task, the grounding of each generated KG item has to be done w.r.t that item only. For example: if we consider the grounding of the generated relations, it has to be picked from the set of relations in the KG only. It will never be picked up from the set of other KG items i.e., entities, concepts, etc. Hence, indexing each KG item set separately provides constraint retrieval.

Given that the input query will be the generated relation, on the other hand, if the triples of the KG are indexed, then the retrieved triples will not be constrained to the relations. It will contain the corresponding entities as well that could be irrelevant for a given question and hence introduce some noise in the retriever output.

We experiment with both these approaches and found out that indexing KG items separately provides provides superior performance ($8 - 10\%$) than indexing KG triples.

Generation: Here, we adapt the generation step in RAG by using the discriminative ability of the LLM instead of generation to select the most probable entity/concept from the given list to prevent hallucinations.

We further perform in-depth analysis of effectiveness of our proposed QUACK resolver on KQA Pro dataset. Table 7 shows the QUACK performance (Accuracy) mainly for entities/concepts and relations/attributes.

Model	Entity/Concept	Relation/Attribute
CodeLlama Ins.	87.0	60.4
Llama 2	74.9	55.0
PaLM 2	70.7	52.0

Table 7: QUACK Resolution Performance on KQA Pro

A.4 List of Numerical Operators

The list of the numerical operators supported are as follows: `[`$<, \leq, >, \geq, =, \neq$`, argmin, argmax]`

A.5 Qualitative Analysis of the Generated KoPLs

The examples of the generated KoPL steps that are not directly executable before QUACK resolver and executes correctly after QUACK resolution for each dataset are shown in Table 8. The examples of the generated KoPL steps that resulted in error (failure cases) during execution even after QUACK resolver are shown in Table 9.

A.6 Baselines

- **Pangu** (Gu et al., 2023) is a recent SOTA KGQA model. It uses LLMs for discrimination rather than generation for grounding the generated draft. It incrementally constructs plans in a step-wise fashion to handle large search spaces.

- **KB_BINDER** (Li et al., 2023a) enables few-shot learning for KBQA using LLMs through two key stages: Draft Generation, where given a question, an LLM generates a preliminary "draft" logical form using few-shot examples; and Knowledge Base Binding, where entities and relations in the draft are grounded to the target KG using string matching and similarity search.

- **LLM-ICL** is an in-context learning-based baseline we implement for KQA Pro. As there are no experimental results of Pangu and KB_BINDER on KQA Pro, we use LLM-ICL as an alternative. Since KQA Pro models do not include an entity linking stage, LLM-ICL directly generates SPARQL queries without further grounding stage, ensuring a fair comparison.

- **KVMemNet** (Miller et al., 2016) performs QA by first storing facts in a key-value structured memory before reasoning on them in order to predict an answer. At each reasoning step, the collected information from the

memory is cumulatively added to the original query to build context for the next reasoning iteration.

- **SRN** (Qiu et al., 2020) model starts from the question entity and uses a path search technique to predict the relation path sequence to reach the target entity.
- **RGCN** (Schlichtkrull et al., 2018) uses a graph convolution network-based technique to encode the KG into graph form and perform QA.
- **EmbedKGQA** (Saxena et al., 2020) uses KG embeddings to perform multi-hop reasoning using a RoBERTa-based question encoder.
- **Subgraph Retrieval** (Zhang et al., 2022b) use a dual-encoder that provides better retrieval as compared to the existing retrieval methods.
- **PullNet** (Sun et al., 2019) extracts a question specific subgraph from the entire relation graph using a graph CNN instead of heuristics and then retrieves the answer.
- **NSM** (He et al., 2021) propose a teacher-student approach. The student network aims to find the correct answer to the query, while the teacher network tries to learn intermediate supervision signals for improving the reasoning capacity of the student network. They utilize both forward and backward reasoning to enhance the learning of intermediate entity distributions.
- **BART + KoPL** (Cao et al., 2022a) is an end-to-end generation model that directly produces the corresponding KoPL program steps given a question. It is worth noting that the pre-trained BART model is forced to have the capability to memorize the relations and entities present in the KG.
- **GraphQ IR** (Nie et al., 2022) proposes a unified intermediate representation for graph query languages, named GraphQ IR. It has a natural-language-like expression that bridges the semantic gap and formally defined syntax that maintains the graph structure. A neural semantic parser is used to convert user queries into GraphQ IR, which can be later losslessly compiled into various downstream graph query languages such as SPARQL, Lambda DCS, etc.
- **TransferNet** (Shi et al., 2021) answers multi-hop questions by attending to different parts of the question at each step. It then computes activated scores for relations, and then transfers the previous entity scores along activated relations in a differentiable way.
- **FlexKBQA** (Li et al., 2023b) utilizing Large Language Models (LLMs) as program translators. It leverages automated algorithms to sample diverse programs, such as SPARQL queries, from the knowledge base, which are subsequently converted into natural language questions via LLMs. They use this synthetic dataset to facilitate training of a specialized lightweight model for a KG.
- **DecAF** (Yu et al., 2023) jointly generates both logical forms and direct answers and then combines the merits of them to get the final answers. They treat logical forms as regular text strings just like answers during generation, reducing efforts of hand-crafted engineering. DecAF linearizes KG into text documents and leverages free-text retrieval methods to locate relevant sub-graphs.

Figure 3: Few-Shot ICL Prompt for a Sample Test Question in KQA Pro

You have to follow the below instructions to generate the logical form called as Knowledge Oriented Programming Language (KoPL).

Always validate the output using below instructions, and don't try to generate anything which you think is wrong.
Instructions -
- The task is to come up with the steps of functions for the test question using the list of functions provided below.
- Each function can take "functional inputs" which is the "output" from the previous step, and "textual inputs" which you have to generate from a given question. If textual input is None for any function then you don't have to generate any textual input for that function.
- To generate these steps you should match "output" of the current step function with "functional inputs" of the next step function.
- Use the training examples to understand the step generation process and stick only to the output format provided in the training examples. Do not generate any explanation text.
- Do not use entities and concepts outside of the list provided in each test question. If "None" is mentioned in concept in question then it means that their is no concept present in the test question and you can't generate any concept related function.
- "Functional input" of current step can be subset of previous step "output", but can't be superset, for example if function input of current step is (entity, entity) then previous step output can't be entity only, it should be at least (entity, entity).
- Or function is always come by at least after two Find or FindAll functions.
- And function is always come by at least after two Find or FindAll functions.
- If Concept is None in question, then you is not allowed to generate FilterConcept function.

Each **Function Information** is provided using the below template:
- FunctionName:
Description - <Description of function>
Functional Inputs - <Inputs to the current step function from previous step function output >
Textual Inputs - <Textual inputs to the current step function>
Outputs - <Outputs of the current step function>

List of Functions:
- FindAll
Description - Return all entities in the knowledge graph
Functional Inputs - None
Textual Inputs - None
Outputs - Entities

- Find
Description - Return all entities with the given name in the knowledge graph
Functional Inputs - None
Textual Inputs - name
Outputs - Entities

- FilterConcept
Description - Find those belonging to the given concept in the knowledge graph
Functional Inputs - Entities
Textual Inputs - concept name
Outputs - Entities

- FilterStr
Description - Filter entities with an attribute condition of string type in the knowledge graph
Functional Inputs - Entities
Textual Inputs - (Key, Value)
Outputs - (Entities, Facts)

- FilterNum
Description - Similar to FilterStr, except that the attribute type is number in the knowledge graph
Functional Inputs - Entities
Textual Inputs - (Key, Value, Operation)
Outputs - (Entities, Facts)

- FilterYear
Description - Similar to FilterStr, except that the attribute type is year in the knowledge graph
Functional Inputs - Entities
Textual Inputs - (Key, Value, Operation)
Outputs - (Entities, Facts)

- FilterDate
Description - Similar to FilterStr, except that the attribute type is date in the knowledge graph
Functional Inputs - Entities
Textual Inputs - (Key, Value, Operation)
Outputs - (Entities, Facts)

- QFilterStr
Description - Filter entities and corresponding facts with a qualifier condition of string type in the knowledge graph
Functional Inputs - Entities
Textual Inputs - (Qualifier Key, Qualifier Value)
Outputs - (Entities, Facts)

- QFilterNum
Description - Similar to QFilterStr, except that the qualifier type is number in the knowledge graph
Functional Inputs - Entities
Textual Inputs - (Qualifier Key, Qualifier Value, Operation)
Outputs - (Entities, Facts)

- QFilterYear
Description - Similar to QFilterStr, except that the qualifier type is year in the knowledge graph
Functional Inputs - Entities
Textual Inputs - (Qualifier Key, Qualifier Value, Operation)
Outputs - (Entities, Facts)

- QFilterDate
Description - Similar to QFilterStr, except that the qualifier type is date in the knowledge graph
Functional Inputs - Entities
Textual Inputs - (Qualifier Key, Qualifier Value, Operation)
Outputs - (Entities, Facts)

- Relate
Description - Find entities that have a specific relation with the given entity in the knowledge graph
Functional Inputs - Entity
Textual Inputs - (Predicate, Direction)
Outputs - (Entities, Facts)

- And
Description - Return the intersection of two entity sets in the knowledge graph
Functional Inputs - (Entities, Entities)
Textual Inputs - None
Outputs - Entities

- Or
Description - Return the union of two entity sets in the knowledge graph
Functional Inputs - (Entities, Entities)
Textual Inputs - None
Outputs - Entities

- QueryName
Description - Return the entity name in the knowledge graph
Functional Inputs - Entity
Textual Inputs - None
Outputs - string

- Count
Description - Return the number of entities in the knowledge graph
Functional Inputs - Entity
Textual Inputs - None
Outputs - number

- QueryAttr
Description - Return the attribute value of the entity in the knowledge graph
Functional Inputs - Entity
Textual Inputs - Key
Outputs - Value

- QueryAttrUnderCondition
Description - Return the attribute value whose corresponding fact should satisfy the qualifier condition in the knowledge graph
Functional Inputs - Entity
Textual Inputs - (Key, Qualifier Key, Qualifier Value)
Outputs - Value

- QueryRelation
Description - Return the relation between two entities in the knowledge graph
Functional Inputs - (Entity, Entity)
Textual Inputs - None
Outputs - Predicate

- SelectBetween
Description - From the two entities, find the one whose attribute value is greater or less and return its name in the knowledge graph
Functional Inputs - (Entity, Entity)
Textual Inputs - (Key, Operation)
Outputs - string

- SelectAmong
Description - From the entity set, find the one whose attribute value is the largest or smallest in the knowledge graph
Functional Inputs - Entities
Textual Inputs - (Key, Operation)
Outputs - string

- VerifyStr
Description - Return whether the output of QueryAttr or QueryAttrUnderCondition and the given value are equal as string in the knowledge graph
Functional Inputs - Value
Textual Inputs - Value
Outputs - boolean

- VerifyNum
Description - Return whether the two numbers satisfy the condition in the knowledge graph
Functional Inputs - Value
Textual Inputs - (Value, Operation)
Outputs - boolean

- VerifyYear
Description - Return whether the two years satisfy the condition in the knowledge graph
Functional Inputs - Value
Textual Inputs - (Value, Operation)
Outputs - boolean

- VerifyDate
Description - Return whether the two dates satisfy the condition in the knowledge graph
Functional Inputs - Value
Textual Inputs - (Value, Operation)
Outputs - boolean

- QueryAttrQualifier
Description - Return the qualifier value of the fact (Entity, Key, Value) in the knowledge graph
Functional Inputs - Entity
Textual Inputs - (Key, Value, Qualifier Key)
Outputs - Qualifier Value

- QueryRelationQualifier
Description - Return the qualifier value of the fact (Entity, Pred, Entity) in the knowledge graph
Functional Inputs - (Entity, Entity)
Textual Inputs - (Predicate, Qualifier Key)
Outputs - Qualifier Value

Training Examples:

Training Example 1:
Question: What is the connection between A Serious Man to Ireland (the one whose nominal GDP is 239389340720.488 United States dollar)? Entities: ['A Serious Man', 'Ireland'], Concepts: None. The steps to solve this question are:
Output:
Step 1: Find(A Serious Man)
Step 2: Find(Ireland)
Step 3: FilterNum(nominal GDP, 239389340720.488 United States dollar, =)
Step 4: QueryRelation()
Done

Training Example 2:
Question: Which first-level administrative country subdivision established post-1829 covers the biggest area? Entities: None, Concepts: ['first-level administrative country subdivision']. The steps to solve this question are:
Output:
Step 1: FindAll()
Step 2: FilterYear(inception, 1829, >)
Step 3: FilterConcept(first-level administrative country subdivision)
Step 4: SelectAmong(area, largest)
Done

Training Example 3:
Question: What is the ISNI of John Broome (the one born in 1738-01-01)? Entities: ['John Broome'], Concepts: None. The steps to solve this question are:
Output:
Step 1: Find(John Broome)
Step 2: FilterDate(date of birth, 1738-01-01, =)
Step 3: QueryAttr(ISNI)
Done

Training Example 4:
Question: Does the sovereign state that has a diplomatic relation with Malaysia (the subject of this statement is East Timor–Malaysia relations), have the CIVICUS Monitor country entry of saint-lucia? Entities: ['Malaysia'], Concepts: ['sovereign state']. The steps to solve this question are:
Output:
Step 1: Find(Malaysia)
Step 2: Relate(diplomatic relation, forward)
Step 3: QFilterStr(statement is subject of, East Timor–Malaysia relations)
Step 4: FilterConcept(sovereign state)
Step 5: QueryAttr(CIVICUS Monitor country entry)
Step 6: VerifyStr(saint-lucia)
Done

Training Example 5:
Question: What is the umber of episodes in TV series with Twitter username ThomasFriends (the subscription number of this statement is 15947)? Entities: None, Concepts: ['television series']. The steps to solve this question are:
Output:
Step 1: FindAll()
Step 2: FilterStr(Twitter username, ThomasFriends)
Step 3: QFilterNum(number of subscribers, 15947, =)
Step 4: FilterConcept(television series)
Step 5: QueryAttr(number of episodes)
Done

Training Example 6:
Question: When was born the person that was nominated for Tony Award for Best Actor in a Musical in 1967? Entities: ['Tony Award for Best Actor in a Musical'], Concepts: ['human']. The steps to solve this question are:
Output:
Step 1: Find(Tony Award for Best Actor in a Musical)
Step 2: Relate(nominated for, backward)
Step 3: QFilterYear(point in time, 1967, =)
Step 4: FilterConcept(human)
Step 5: QueryAttr(date of birth)
Done

Training Example 7:
Question: Does Pierce County that is located in Washington or Grays Harbor County have less area? Entities: ['Washington', 'Pierce County', 'Grays Harbor County'], Concepts: None. The steps to solve this question are:
Output:
Step 1: Find(Washington)
Step 2: Relate(located in the administrative territorial entity, backward)
Step 3: Find(Pierce County)
Step 4: And()
Step 5: Find(Grays Harbor County)
Step 6: SelectBetween(area, less)
Done

Training Example 8:
Question: How many researchers are the occupation of Aristotle or practice motivational speaking? Entities: ['Aristotle', 'motivational speaking'], Concepts: ['researcher']. The steps to solve this question are:
Output:
Step 1: Find(Aristotle)
Step 2: Relate(occupation, forward)
Step 3: FilterConcept(researcher)
Step 4: Find(motivational speaking)
Step 5: Relate(practiced by, forward)
Step 6: FilterConcept(researcher)
Step 7: Or()
Step 8: Count()
Done

Training Example 9:
Question: Is the nominal GDP of Guinea-Bissau over 69000000 United States dollars on the date 1996-01-01? Entities: ['Guinea-Bissau'], Concepts: None. The steps to solve this question are:
Output:
Step 1: Find(Guinea-Bissau)
Step 2: QueryAttrUnderCondition(nominal GDP, point in time, 1996-01-01)
Step 3: VerifyNum(69000000 United States dollar, >)
Done

Training Example 10:
Question: Which university has fewer students, George Washington University or University of Hamburg? Entities: ['George Washington University', 'University of Hamburg'], Concepts: None. The steps to solve this question are:
Output:
Step 1: Find(George Washington University)
Step 2: Find(University of Hamburg)
Step 3: SelectBetween(students count, less)
Done

Test Question:
Question: Who was the prize winner when Mrs. Miniver got the Academy Award for Best Writing, Adapted Screenplay? Entities: ['Mrs. Miniver', 'Academy Award for Best Writing, Adapted Screenplay'], Concepts: None. The steps to solve this question are:
Output:

KQA Pro	Question: ***What is the higher education institution is headquartered in the city whose postal code is 20157?***	
Gold KoPL	Step 1: FindAll(), Step 2: FilterStr(postal code, 20157), Step 3: FilterConcept(city), Step 4: Relate(headquarters location, backward), Step 5: FilterConcept(higher education institution), Step 6: What()	
Gold Answer	Politecnico di Milano	
Generated KoPL	Step 1: FindAll(), Step 2: FilterStr(postal code, 20157), Step 3: FilterConcept(city), Step 4: Relate(headquartered in, backward), Step 5: FilterConcept(higher education institution), Step 6: What()	×
After QUACK Resolver	Step 1: FindAll(), Step 2: FilterStr(postal code, 20157), Step 3: FilterConcept(city), Step 4: Relate(headquarters location, backward), Step 5: FilterConcept(higher education institution), Step 6: What()	✓
MetaQA (1-hop)	Question: ***What is the language spoken in the movie Priceless?***	
Gold KoPL	Step 1: Find(Priceless), Step 2: Relate(in_language)	
Answer	French	
Generated KoPL	Step 1: Find(Priceless), Step 2: Relate(language)	×
After QUACK Resolver	Step 1: Find(Priceless), Step 2: Relate(in_language)	✓
MetaQA (2-hop)	Question: ***What are the genres of the films directed by Mike Nichols?***	
Gold KoPL	Step 1: Find(Mike Nichols), Step 2: Relate(directed_by,backward), Step 3: Relate(has_tags,forward), Step 4: What()	
Answer	Drama, Horror, Comedy, War, Thriller	
Generated KoPL	Step 1: Find(Mike Nichols), Step 2: Relate(directed by,backward), Step 3: Relate(has genre,forward), Step 4: What()	×
After QUACK Resolver	Step 1: Find(Mike Nichols), Step 2: Relate(directed_by,backward), Step 3: Relate(has_tags,forward), Step 4: What()	✓
MetaQA (3-hop)	Question: ***who are the screenwriters that the actors in their movies also appear in the movie Medium Cool?***	
Gold KoPL	Step 1: Find(Medium Cool), Step 2: Relate(starred_actors,forward), Step 3: Relate(starred_actors,backward), Step 4: Relate(written_by,forward), Step 5: What()	
Answer	Dan Cohen, David Mamet, Daniel Chuba, John Sayles, William Malone, Abe Polsky, Corey Allen, Quentin Tarantino, Alan Sharp, David C. Wilson, Elmore Leonard	
Generated KoPL	Step 1: Find(Medium Cool), Step 2: Relate(starred actors,forward), Step 3: Relate(starred actors,backward), Step 4: Relate(written by,forward), Step 5: What()	×
After QUACK Resolver	Step 1: Find(Medium Cool), Step 2: Relate(starred_actors,forward), Step 3: Relate(starred_actors,backward), Step 4: Relate(written_by,forward), Step 5: What()	✓
WebQSP	Question: ***Who is Michael Buble?***	
Gold KoPL	Step 1: Find(Michael Buble), Step 2: Relate(profession, forward)	
Answer	Actor, Singer, Songwriter	
Generated KoPL	Step 1: Find(Michael Buble), Step 2: Relate(occupation, forward)	×
After QUACK Resolver	Step 1: Find(Michael Buble), Step 2: Relate(profession, forward)	✓

Table 8: KoPL Program Steps before and after QUACK resolution

KQA Pro	Question: ***Which has less mass, Chuck Connors who is in the United States Army or Chris Carter of the Boston Red Sox?***	
Gold KoPL	Step 1: Find(United States Army), Step 2: Relate(military branch, backward), Step 3: Find(Chuck Connors), Step 4: And(), Step 5: Find(Boston Red Sox), Step 6: Relate(member of sports team, backward), Step 7: Find(Chris Carter), Step 8: And(), Step 9: SelectBetween(mass, less)	
Gold Answer	Chuck Connors	
Generated KoPL	Step 1: Find(United States Army), Step 2: Relate(member, backward), Step 3: Find(Chuck Connors), Step 4: And(), Step 5: Find(Boston Red Sox), Step 6: Relate(member, backward), Step 7: Find(Chris Carter), Step 8: And(), Step 9: SelectBetween(mass, less)	×
After QUACK Resolver	Step 1: Find(United States Army), Step 2: Relate(part of, backward), Step 3: Find(Chuck Connors), Step 4: And(), Step 5: Find(Boston Red Sox), Step 6: Relate(member of sports team, backward), Step 7: Find(Chris Carter), Step 8: And(), Step 9: SelectBetween(mass, less)	×
Predicted Answer	Chris Carter	
MetaQA (1-hop)	Question: ***Who stars in Psycho II?***	
Gold KoPL	Step 1: Find(Psycho II), Step 2: Relate(starred_actors,forward), Step 3: What()	
Gold Answer	Vera Miles, Anthony Perkins, Robert Loggia, Meg Tilly	
Generated KoPL	Step 1: Find(Psycho I), Step 2: Relate(starred actors,forward), Step 3: Find(starred actors.0), Step 4: Relate(starred in,forward), Step 5: What()	×
After QUACK Resolver	Find(Psycho), Step 2: Relate(starred_actors,forward), Step 3: Find(actors), Step 4: Relate(has_tags), Step5: What()	×
Predicted Answer	Treasure Island	
MetaQA (2-hop)	Question: ***Who are movie co-writers of Bob Badway?***	
Gold KoPL	Step 1: Find(Bob Badway), Step 2: Relate(written_by,backward), Step 3: Relate(written_by,forward), Step 4: What()	
Gold Answer	Michael Emanuel	
Generated KoPL	Step 1: Find(Bob Badway), Step 2: Relate(co-writer,backward), Step 3: What()	×
After QUACK Resolver	Step 1: Find(Bob Badway), Step 2: Relate(written_by,backward), Step 3: What()	×
Predicted Answer	Scary or Die	
MetaQA (3-hop)	Question: ***Which person wrote the films directed by the director of The Leading Man?***	
Gold KoPL	Step 1: Find(The Leading Man), Step 2: Relate(directed_by,forward), Step 3: Relate(directed_by,backward), Step 4: Relate(written_by,forward), Step 5: What()	
Gold Answer	Naomi Wallace, Dick Christie, John Duigan	
Generated KoPL	Step 1: Find(The Leading Man), Step 2: Relate(directed by,forward), Step 3: Relate(written by,forward), Step 4: What()	×
After QUACK Resolver	Step 1: Find(The Leading Man), Step 2: Relate(directed_by,forward), Step 3: Relate(written_by,forward), Step 4: What()	×
Predicted Answer	The Year My Voice Broke, Flirting, Careless Love, Sirens, Head in the Clouds	
WebQSP	Question: ***What countries are part of the UK?***	
Gold KoPL	Step 1: Find(United Kingdom), Step 2: Relate(administrative_children), Step 3: FilterStr(administrative_area_type, UK constituent country)	
Gold Answer	England, Northern Ireland, Scotland, Wales	
Generated KoPL	Step 1: Find(United Kingdom), Step 2: Relate(countries, forward)	×
After QUACK Resolver	Step 1: Find(United Kingdom), Step 2: Relate(continent, forward)	×
Predicted Answer	Europe	

Table 9: Case-based error analysis of KoPL Program Steps before and after QUACK resolution

Method	Prompt Templates	Logical Form
Pangu	Please translate the following questions to Lisp-like programs. question: <> program: <>; question: <> program: <>;.....; question: <> program:	S-Expression
KB_BINDER	Question: <> Logical Form: <>; Question: <> Logical Form: <>;.....; Question: <> Logical Form:	S-Expression
FlexKBQA	Convert the s-expressions to natural language questions. question: <> s-expression: <>; question: <> s-expression: <>;.....; question: <> s-expression:	SPARQL and S-Expression
SymKGQA	You have to follow the below instructions to generate the logical form called as Knowledge Oriented Programming Language (KoPL). Instructions Function Definitions ICL Examples	KoPL

Table 10: Prompt Templates used by Existing Methods