Toward Exploring Knowledge Graphs with LLMs

Guangyuan Piao[1], Mike Mountantonakis[2], Panagiotis Papadakos[2], Pournima Sonawane[1] and Aidan OMahony[1]

[1]*Dell Technologies, Ireland*
[2]*W3C, ERCIM, France*

Abstract

Interacting with knowledge graphs (KGs) is challenging for non-technical users with information needs who are unfamiliar with KG-specific query languages such as SPARQL and the underlying KG schema. Previous KG question answering systems require ground-truth pairs of questions and queries or fine tuning (Large) Language Models (LLMs) for a specific KG, which is time-consuming and demands deep expertise. In this poster, we present a framework for exploring KGs for question answering using LLMs in a zero-shot setting for non-technical end users, without the need for ground-truth pairs of questions and queries or fine-tuning LLMs. Additionally, we evaluate an example implementation in a simple yet challenging setting using LLMs *exclusively* based on the framework, without the extra effort of maintaining the embeddings or indexes of entities from KG for retrieving relevant ones to a given question. We share preliminary experimental results indicating that exploring a KG using LLM-generated SPARQL queries with reasonable complexity is possible in such a challenging setting.

1. Introduction

Exploring knowledge graphs (KGs) for question answering poses challenges, particularly for non-technical users lacking sufficient knowledge of KG-specific query languages such as SPARQL and Cypher. In previous studies of KG question answering systems [1, 2], they either require ground-truth pairs of questions and queries or fine-tuning of (Large) Language Models (LLMs) for any KG of interest. However, creating such ground-truth data is time-consuming and demands significant effort and expertise. The requirement for fine-tuning necessitates machine learning expertise and restricts the use of many proprietary LLMs, such as ChatGPT, which is only accessible through APIs but exhibits outstanding performance. The *goal* of the poster is to present a general framework for exploring KGs with LLMs for question answering, without these requirements (Section 2). In addition, we present an example implementation with all components defined in the framework, along with preliminary experimental results (Section 3). In contrast to building and maintaining indexes or embeddings of entities for retrieving relevant ones from a KG, we focus on a simpler yet more challenging setting: using LLMs exclusively by prompting the chosen LLM to automatically infer entity IRIs (Internationalized Resource Identifiers). Finally, we discuss some challenges and future work in Section 4.

SEMANTiCS'24: International Conference on Semantic Systems, September 17–19, 2024, Amsterdam, Netherlands
guangyuan.piao@dell.com (G. Piao); mike.mountantonakis@ercim.eu (M. Mountantonakis); panagiotis.papadakos@ercim.eu (P. Papadakos); pournima.sonawane@dell.com (P. Sonawane); aidan.omahony@dell.com (A. OMahony)

2. Framework of Question Answering with a KG using LLMs

Figure 1 illustrates our framework for question answering with a knowledge graph using LLMs. The top three components depict a straightforward pipeline. Specifically, the top-left component indicates the **input** to an LLM. In addition to a question, a user can provide any *user-input context*, such as a description about the KG. The *extracted context* refers to any context automatically retrieved from the system. This can include, but is not limited to, the KG schema or a subset of triples in the KG that might be relevant to the question. The **LLM** component refers to any LLMs, such as Code Llama [3], used for KG-specific query generation for answering the question. The **output** component executes the provided query to obtain the answer. As a special case, the LLM can also generate the output or asnwer directly based on the given question and context derived from the KG without generating any queries as in KG-RAG [4].

In addition to these three basic components, the framework includes a set of optional components indicated by dotted boxes. The **context extractor** aims to automatically extract any useful context for answering the question. For example, it can extract a set of *predicates* or *class types* that are relevant to the question. The **context parser** and **enhancer** process the output of the extractor and enhance it by validating, updating or pruning as necessary. For example, they can check whether the extracted predicates actually exist in the KG schema or revisit the context extractor if necessary. KG-RAG [4] implements a context extractor based on embedding similarities between the question sentence or a set of extracted entities and precomputed entity embeddings with a small language model to retrieve relevant entities from the KG. A set of triples associated with each entity is retrieved, and then parsed and pruned based on their relevance to the given question. Auto-KGQA [5] implements the retrieval of relevant entities by building and maintaining KG resource indexes for text- or embedding-based approaches. For each entity, all its triples are retrieved along with their neighbors up to a predefined depth. These triples are then parsed and pruned to construct a sub-graph containing the most relevant triples to the question. The **query parser** and **enhancer** parse the LLM output, extract the query, and refine the query if necessary. For instance, they can regenerate the query in cases where it is not executable or returns no results. Auto-KGQA [5] prompts a LLM to generate several SPARQL queries and parse the results, and then let the LLM choose the best one.

In the framework, edges highlighted in blue and orange indicate repeatable loops. For example, the loop of ❺❻❼❽ can be repeated multiple times, with each iteration providing a query as extracted context. The query extracted from the previous loop can be used in the next query generation process to enhance it. The framework also allows for the extraction of different types of context. That is, one can have several sets of context extractors, parsers, and enhancers

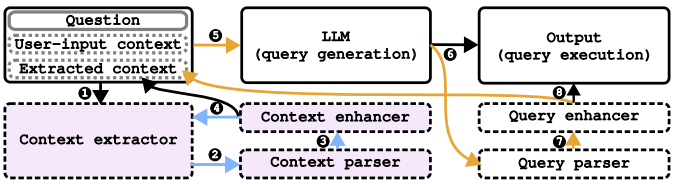

Figure 1: The framework of question answering with a KG using LLMs. It can include several sets of shaded components to extract different contexts. The colored loops can be repeated multiple times.

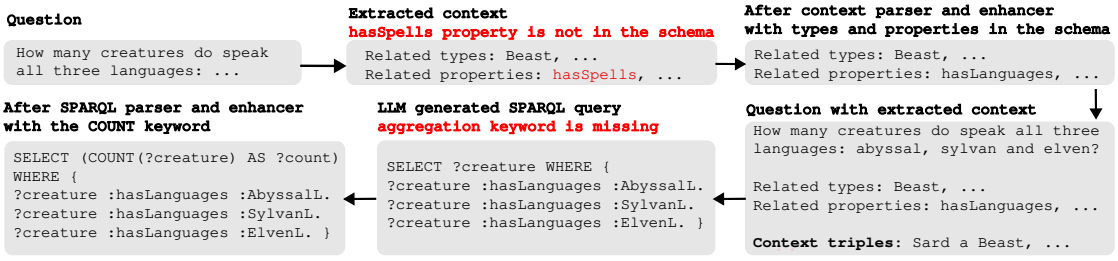

Figure 2: Example workflow for generating the SPARQL query of a question with the framework.

(shaded boxes in Figure 1). For instance, one set can be used for extracting the KG schema, while another can be used for extracting a set of relevant triples to the question.

3. Example Implementation and Experimental Results

Here, we present an example implementation[1] built upon the proposed framework with all components, using Code Llama Instruct 7B [3] as our LLM. Specifically, we are interested in exploring an implementation that relies solely on LLMs, i.e., without maintaining indexes or embeddings of entities in the KG for retrieving relevant entities, which would otherwise require extra effort and expertise [4, 5]. To this end, we consider questions with extracted context as input to the LLM. We use two types of context extractors. The first extractor extracts class types and predicates from the KG schema as context. The second one prompts the same LLM to infer the top-k class types and predicates relevant to the question. Next, the context parser and enhancer check the extracted class types and predicates, and rerun the context extractor if non-existent class types or predicates are detected. Based on these extracted class types and predicates, we can retrieve p triple(s) for each class type and predicate, which are provided as context triples (k=5 and p=1 in our experiments). Afterwards, the LLM generates the output for the given question and extracted context. Subsequently, the query parser and enhancer parse the output to obtain the query and enhance it if necessary. Again, we prompt the same LLM to check the generated query and add, remove, or modify it if necessary. Steps ❺❻❼❽ can be repeated multiple times before finalizing the SPARQL query as our output. Figure 2 illustrates an example workflow of generating the final SPARQL query for Q14 in Listing 1.

Experimental settings. We use a custom Bestiary KG [2] which contains diverse information about over 4,000 creatures from a fantasy role-playing game, comprising 98,070 triples. However, upon careful investigation of each question in the dataset, we noticed that the majority of 100 questions from [2] require high-complexity SPARQL queries. In our challenging setting, using those questions for evaluation may not be feasible (as the majority cannot be answered), and could potentially hinder the exploration of new directions by beginning with those queries. After careful investigation of those questions, we empirically chose eight questions that are possible to answer in our setting but also have varying complexities. Listing 1 shows the set of questions and the corresponding generated SPARQL queries for our discussion. It shows

[1]The source code, prompt templates, and examples are available at https://github.com/parklize/LLM4SPARQL.

Listing 1: Eight example questions and their corresponding generated SPARQL queries.

```
Q0: which creatures not speaking draconic language do have chaotic good alignment?
SELECT ?creature WHERE {?creature a :Beast. ?creature :hasAlignment :chaoticGood.
FILTER NOT EXISTS {?creature :hasLanguages :DraconicL.}}
```

```
Q9: what creatures do have cold resist?
SELECT ?creature WHERE {?creature :hasResists :cold.}
```

```
Q13: what creatures do speak both common and undercommon languages?
SELECT ?creature WHERE {?creature :hasLanguages :CommonL. ?creature :hasLanguages :UndercommonL.}
```

```
Q14: how many creatures do speak all three languages: abyssal, sylvan and elven?
SELECT (COUNT(?creature) AS ?count) WHERE {?creature :hasLanguages :AbyssalL.
?creature :hasLanguages :SylvanL. ?creature :hasLanguages :ElvenL.}
```

```
Q58: what creatures speaking dwarven language do have armor class greater than 12?
SELECT ?creature WHERE {?creature a :Beast. ?creature :hasLanguages :DwarvenL. ?creature :hasACValue ?ac. FILTER( ?ac > 12)}
```

```
Q64: what is the average number of health points for creatures speaking gnome language?
SELECT (AVG(?hp) AS ?hp_AVG) WHERE {?creature rdf:type :Beast. ?creature :hasLanguages :GnomeL. ?creature :hasHPvalue ?hp.}
```

```
Q83: which creatures speaking necril and abyssal languages do have wisdom attribute more than 4?
SELECT ?creature WHERE {?creature rdf:type :Beast. ?creature :hasLanguages :NecrilL. ?creature :wis ?wis. FILTER(?wis > 4)}
```

```
Q94: what is the average dexterity attribute for Phoenix and Sleipnir?
SELECT (AVG(?dex) AS ?dex_AVG) WHERE {?beast rdf:type :Beast. ?beast :dex ?dex. FILTER(?beast = :Phoenix || ?beast = :Sleipnir)}
```

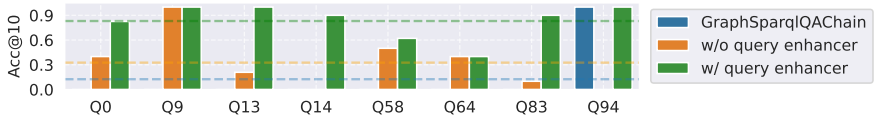

Figure 3: Acc@10 for each of the eight questions, with the average indicated by a dashed line.

that even exclusively using LLMs, we can still answer questions of reasonable complexity in a zero-shot setting. This includes questions with *negation* (Q0), *aggregation* (Q64), or even those with multiple $?s\ p\ o$ patterns with *filtering* (Q83), where $?s$ indicates a variable.

As LLMs can produce different answers each time, we use $Acc@10$, which measures the percentage of correct answers obtained out of 10 runs for a given question, to evaluate the performance. The most relevant work to ours is GraphSparqlQAChain[2], which uses only the KG schema as extracted context in the prompt to generate the query for a given question. We use this as our baseline. In addition, we include two variants of our implementation – one *without* the query enhancer and the other *with* the enhancer component in our framework.

Results. Figure 3 shows the results for the eight questions. As illustrated by dashed lines, the average $Acc@10$ scores over these questions using GraphSparqlQAChain and the two variants – one without and one with the enhancer – are 0.125, 0.326, and 0.831, respectively. As we can see from the figure, GraphSparqlQAChain, which uses only the KG schema as the context, could not generate the majority of queries because it is not aware of the IRI patterns of entities in the KG. The results with and without the query enhancer component clearly indicate that the enhancer consistently improves the quality of generated queries. For example, the $Acc@10$ is zero for both Q14 and Q94 without the query enhancer, while with the enhancer, it increases to 0.9 and 1.0, respectively. Q14 in Listing 1 shows an example where the initial query (without blue part)

[2]https://python.langchain.com/docs/use_cases/graph/graph_sparql_qa

has been enhanced (with blue part). For quantitative evaluation, we manually extended the initial eight questions by adding 22 similar ones, resulting in a total of 30 questions. The average $Acc@10$ scores using GraphSparqlQAChain, without query enhancer, and with query enhancer are 0.14, 0.22, and 0.57 respectively ($\alpha < .05$). Although it is clear that the performance improves with the query enhancer, it is worth noting that this improvement comes with the extra cost of prompting LLMs *n* more times, where *n* is a predefined parameter indicating how many times we want to repeat the enhancing process.

4. Discussion and Future Work

In this work, we presented a general framework for exploring KGs with LLMs. In addition, we investigated an example implementation using all components defined in the framework in a challenging setting, which exclusively uses LLMs in a zero-shot setting without ground-truth data and fine-tuning. While the example implementation eliminates the need for maintaining entity embeddings for embedding-based entity retrieval, it may result in *hallucinations*, where non-existent entities are used as subjects or objects in the generated queries. In addition, answering questions that require complex SPARQL queries is challenging due to the current limitations of LLM in generating such queries. Further investigation with other LLMs, including specialized open-source LLMs trained on open question-query datasets, is required. Additionally, using all 100 questions from [2] for evaluation in our setting – without ground-truth and without fine-tuning – is challenging. Those questions contain many complex queries, such as those requiring regex patterns, and might exclude the interesting possibility of exploring this research direction. Hence, a benchmark dataset with varying query complexities would be beneficial.

Acknowledgments

This work was funded by the GLACIATION Horizon Europe project (No. 101070141).

References

[1] S. Yang, M. Teng, X. Dong, F. Bo, Llm-based sparql generation with selected schema from large scale knowledge base, in: CCKS, Springer, 2023, pp. 304–316.
[2] L. Kovriguina, R. Teucher, D. Radyush, D. Mouromtsev, Sparqlgen: One-shot prompt-based approach for sparql query generation, in: SEMANTiCS, 2023.
[3] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, et al., Code llama: Open foundation models for code, arXiv:2308.12950 (2023).
[4] K. Soman, P. W. Rose, J. H. Morris, R. E. Akbas, B. Smith, B. Peetoom, C. Villouta-Reyes, G. Cerono, Y. Shi, A. Rizk-Jackson, S. Israni, C. A. Nelson, S. Huang, S. E. Baranzini, Biomedical knowledge graph-optimized prompt generation for large language models, arXiv:2311.17330 (2024).
[5] C. V. S. Avila, M. A. Casanova, V. M. Vidal, A framework for question answering on knowledge graphs using large language models, in: ESWC, 2024.