A Framework for Question Answering on Knowledge Graphs Using Large Language Models

Caio Viktor S. Avila[1], Marco A. Casanova[2], and Vânia M.P.Vidal[1]

[1] Federal University of Ceará, Fortaleza, Brazil 60440-900
caioviktor@alu.ufc.br,vvidal@lia.ufc.br
[2] Department of Informatics, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil 22451-900
casanova@inf.puc-rio.br

Abstract. Currently, large language models (LLMs) are the state of the art for pre-trained language models. LLMs have been applied to many tasks, including question and answering over Knowledge Graphs (KGs) and text-to-SPARQL, that is, the translation of Natural Language (NL) questions to SPARQL queries. This paper introduces Auto-KGQA, an autonomous domain-independent framework based on LLMs for text-to-SPARQL. The framework uses as context, fragments of the KG, which the LLM uses to translate the user's NL question to a SPARQL query on the KG. Finally, it generates a natural language response for the user, based upon the result of the execution of SPARQL query over the KG.

Keywords: Question Answering · Knowledge Graph · Large Language Model.

1 Introduction

Question Answering (QA) systems are computational systems that can answer questions typically asked in Linguagem Natural (LN) [5]. Among the types of QA systems, *Knowledge Graph Question Answering* (KGQA) systems stand out for their ability to generate curated and deep responses. KGQA systems are QA systems that are based on a *Knowledge Graph* (KG)[9].

Currently, Large Language Models (LLMs) are the state of the art for pre-trained language models. LLMs achieve good results in tasks like question and answering over KGs and text-to-SPARQL[2], which opens an opportunity to develop KGQA systems based on LLMs [8]. However, for the LLM to be able to generate SPARQL queries on a specific KG it is necessary that it has knowledge about the vocabulary and structure of the KG being queried. A possible strategy for passing on such knowledge could be passing the KG via prompt, exploiting the LLM's ability to learn with the context present in the prompt (in-context learning) [3]. However, given the size limitation of the LLM prompt, passing the entire KG proves to be unfeasible in real-world scenarios. Thus, fragments must be selected, forming a sub-graph, relevant to the user's original question to be passed to the LLM, via prompt.

This paper introduces Auto-KGQA, an autonomous domain-independent framework based on LLMs for text-to-SPARQL. Given a KG K and a user's NL question Q, the framework selects fragments of the T-Box and A-Box of K, providing these as contextual input for the Large Language Model (LLM). Auto-KGQA proceeds to generate n potential SPARQL queries that interpret Q, subsequently selecting the most suitable query S based on the outcomes derived from executing these queries on K. Finally, it generates a natural language response for the user, based upon the result of the execution of S over K. The key feature of Auto-KGQA is its ability to select smaller KG fragments without prior knowledge of the expected outcome, thus reducing the number of input tokens to the LLM.

The remainder of this article is structured as follows. Section 2 covers related work. Section 3 introduces Auto-KGQA. Finally, Section 4 contains the conclusions.

2 Related Work

An approach to translate NL queries to SPARQL, called SGPT, is proposed in [7]. SGPT encodes a vector of linguistic embedding features, such as parts-of-speech (POS) and the dependency tree, in addition to the corresponding sub-graph of the question. Then, the embeddings are passed as training to a Transformers model based on GPT-2 to generate the SPARQL query.

The *Knowledge Solver* (KSL) paradigm was proposed in [4] to teach LLMs to fetch essential knowledge from external knowledge bases. In this approach, the LLM receives as input question-and-answer pairs (of the multiple choice type), where the LLM task is to learn to select the KG subgraph required to answer the original NL question.

SPARQLGEN is a one-shot approach that generates SPARQL queries by enhancing LLMs with context in a single prompt[6]. The prompt includes the question, an RDF subgraph needed for the query, and an example SPARQL query for a different question. Subgraph selection for each question is done through reverse engineering of expected correct SPARQL queries.

To the best of our knowledge, LLM-based approaches already receive a subgraph of the KG relevant to the question as input or use information from the expected response to derive it. Therefore, this current work has the advantage of being able to autonomously select the relevant sub-graph, given only the user's question.

3 Auto-KGQA

The framework[3] is composed of six components: (1) Chat Web Interface, an instant messenger web interface for sending and receiving messages to the system

[3] `https://github.com/CaioViktor/Auto-KGQA`

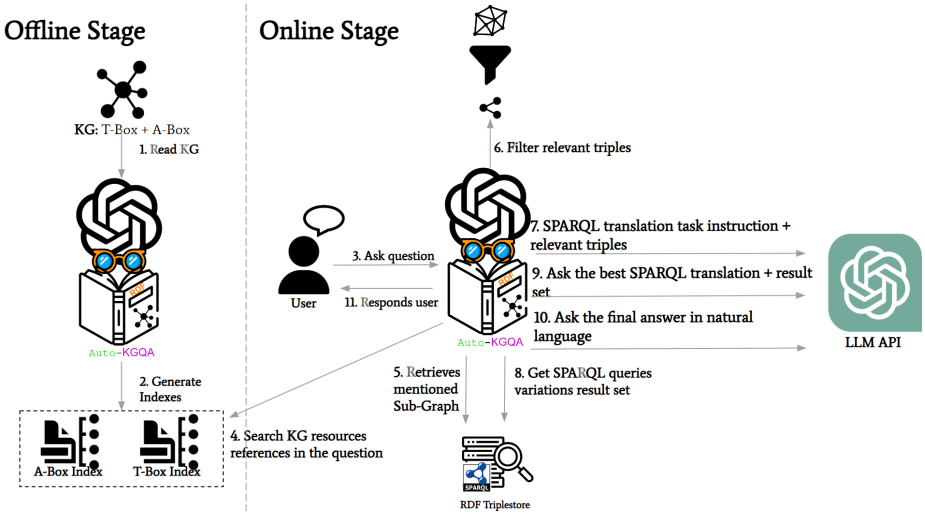

Fig. 1. Auto-KGQA framework

implemented in React; (2) Conversation API, question processing system accessible via API in Flask Python; (3) Knowledge Graph Endpoint; (4) Resources indexes, indices used to find references to KG resources in natural language questions; (5) Example dataset of queries collected during users interactions; and (6) LLM API (ChatGPT by default).

Below is a brief description of the processes carried out in the framework depicted in the Figure 1. A more detailed explanation and initial experiments of the approach, can be found in [1].

Off-line stage. The offline stage receives the KG (T-Box and A-Box) as input via a SPARQL endpoint and creates two indexes, used in the online stage to find references to KG resources in the input question. The first index generates mappings from labels of classes and properties to their URIs. The second index generates mappings from A-Box (instances) resource labels to their URIs.

The indexes generated in this step can be of three types: (1) full-text search engine, using the Whoosh; (2) Sentences vector embedding based, using FAISS library; (3) DBpedia Spotlight, using the DBpediaAPI. The first two indices can be used in any KG, both to index the T-box and the A-box. In turn, the third party can only be used when performing queries on DBpedia, indexing only its A-box.

On-line stage. Figure 1 shows the workflow followed during the online stage. At this stage, the system uses the indexes created in the offline stage to find references to KG resources present in the user's question Qi (step 3). Then, for each referenced resource, all its triples are recovered, along with those of their neighbors up to depth 2 in the KG (step 4), building a set of Ti triples. Next, the framework builds the **Query Sub-graph** (presented in more detail in the next section), Gq, a subgraph of Ti that will be passed to LLM to build the SPARQL

query . In the next step, a request is made to LLM to generate five translation variations of Qi to SPARQL over Gq (step 5). For each valid query generated by LLM, the system retrieves the result set from its execution on the KG endpoint (step 6). Next, the LLM is asked to choose the SPARQL translation that best represents the original query, also considering the previously obtained result sets (step 7). Finally, the system asks LLM to generate the final response in NL based on the SPARQL query and its result set selected in the previous step (step 8).

Query Sub-graph Construction. The selected triples, Ti, are filtered to remove triples irrelevant to the question Qi. For this, all triples of Ti are grouped by subject and property , then, for each group, a single phrase will be generated that represents a pseudo verbalization of these triples. The generated sentences follows the structure "**Subject verb complement**". The subject and verb will be formed by the labels of the resources in the original triples, while the complement will be a list of values representing the different RDF objects of each triple of the group, separated by "commas". When the RDF object is an RDF resource, it will be represented in the sentence by its label, when it is a literal, it will be used directly in the sentence. Next, the embedding vectors of all sentences are computed, which will be used to construct a langchain's faiss index, which will be used to get the sentences closest to Qi, Sc. The triples associated with Sc generate a set of triples relevant to the question, called Tr. The resources referenced in Tr are then passed as input to run a steiner-tree algorithm on Ti, to construct a connected graph Gc. Finally, the Ti triples are joined to the Gc triples, generating the Query Sub-graph, a connected graph with the most relevant triples for queries, called Gq. The Gq triples are then serialized to Turtle RDF notation so that they can be passed as a string to the LLM.

Demonstration. The Figure 2 shows a demonstration of interaction via the Auto-KGQA web interface (the image was originally one vertically). In this demo, a KG described in [1] and available in the project's github repository is used, about which questions are asked in sequence about a specific resource (someone called "John"). At the link[4] you will find a video with a demonstration in greater detail.

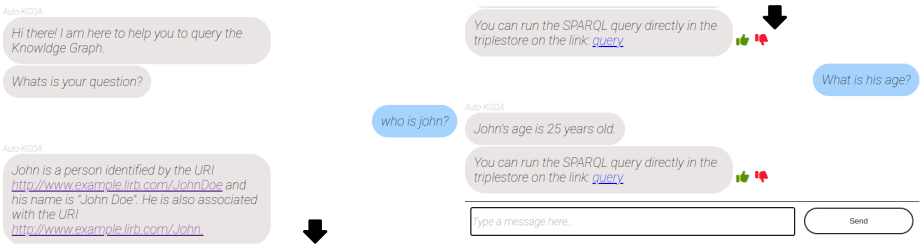

Fig. 2. Auto-KGQA interface demonstration.

[4] `https://youtu.be/GFSHySDitzU`

4 Conclusions

This paper introduced Auto-KGQA, an autonomous domain-independent framework based on LLMs for text-to-SPARQL. The main goal of Auto-KGQA is the selection of smaller KG fragments thereby reducing the number of tokens passed as input to the LLM. Experiments with Auto-KGQA suggest that the framework achieved this goal, without sacrificing performance.

Future work includes experiments using opensource LLMs in place of OpenAI's ChatGPT, as well as using a RAG engine to utilize examples acquired via user feedback to improve system performance.

Acknowledgment

This work was partly funded by FAPERJ under grants E-26/200.834/2021, by CAPES under grant 88881.134081/2016-01 and 88882.164913/2010-01, and by CNPq under grant 305.587/2021-8.

References

1. Avila, C.V.S., Vidal, V.M., Franco, W., Casanova, M.A.: Experiments with text-to-sparql based on chatgpt. In: 2024 IEEE 18th International Conference on Semantic Computing (ICSC). pp. 277–284. IEEE (2024)
2. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
3. Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., Sun, X., Xu, J., Sui, Z.: A survey for in-context learning. arXiv preprint arXiv:2301.00234 (2022)
4. Feng, C., Zhang, X., Fei, Z.: Knowledge solver: Teaching llms to search for domain knowledge from knowledge graphs. arXiv preprint arXiv:2309.03118 (2023)
5. Hirschman, L., Gaizauskas, R.: Natural language question answering: the view from here. natural language engineering **7**(4), 275–300 (2001)
6. Kovriguina, L., et al.: SPARQLGEN: one-shot prompt-based approach for SPARQL query generation. In: Proc. of the Posters and Demo Track of the 19th International Conference on Semantic Systems, Leipzing, Germany, September 20 to 22, 2023. CEUR Workshop Proceedings, vol. 3526. CEUR-WS.org (2023), `https://ceur-ws.org/Vol-3526/paper-08.pdf`
7. Rony, M.R.A.H., Kumar, U., Teucher, R., Kovriguina, L., Lehmann, J.: Sgpt: a generative approach for sparql query generation from natural language questions. IEEE Access **10**, 70712–70723 (2022)
8. Tan, Y., Min, D., Li, Y., Li, W., Hu, N., Chen, Y., Qi, G.: Can chatgpt replace traditional kbqa models? an in-depth analysis of the question answering performance of the gpt llm family. In: International Semantic Web Conference. pp. 348–367. Springer (2023)
9. Yani, M., Krisnadhi, A.A.: Challenges, techniques, and trends of simple knowledge graph question answering: a survey. Information **12**(7), 271 (2021)