# Few-shot learning or RAG in LLM-based text-to-SPARQL? Why not both?

Caio Viktor S. Avila*, Vânia M.P.Vidal*, Wellington Franco* and Marco A. Casanova†
*Departament of Computing, Federal University of Ceará
Fortaleza, Brazil 60440-900
Email: caioviktor@alu.ufc.br, vvidal@lia.ufc.br, wellington@crateus.ufc.br
†Department of Informatics, Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Brazil 22451-900
Email: casanova@inf.puc-rio.br

2025 19th International Conference on Semantic Computing (ICSC) | 979-8-3315-2426-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICSC64641.2025.00016

***Abstract*—This paper introduces a hybrid approach for the text-to-SPARQL task involving Large Language Models (LLMs) by integrating Retrieval-Augmented Generation (RAG) with few-shot learning techniques. The approach enhances the Auto-KGQA framework by incorporating query examples into the LLM while selecting minimal knowledge graph (KG) subgraphs as context, thereby improving its ability to interpret and answer natural language queries. Experimental results on the SciQA benchmark indicate that this integration increased the F1-score by 0.16. Notably, the framework excelled in the zero-shot setting with an F1-score of 0.73, significantly outperforming the prior score of 0.26. Additionally, the paper introduces an automated procedure to extract a minimal T-Box from KGs lacking an explicit schema, optimizing query processing by limiting deep neighborhood exploration. These findings suggest that combining KG context with query examples is an effective strategy, particularly for developing generalizable systems.**

## I. Introduction

Question Answering (QA) systems are designed to process questions posed in Natural Language (NL) and provide responses, with Knowledge Graph Question Answering (KGQA) systems distinguished by their ability to generate precise and comprehensive answers through integration with Knowledge Graphs (KGs). These systems translate NL queries into formal queries, such as SPARQL, enabling more structured and accurate information retrieval [1]. Recent advancements in KGQA architectures have embraced Deep Learning techniques, with Large Language Models (LLMs) emerging as state-of-the-art tools for tasks like question answering over KGs and NL-to-SPARQL translation, demonstrating exceptional performance in these applications.

Despite their potential, LLMs require awareness of KG schemas to generate accurate SPARQL queries. In cases where models lack prior exposure to specific KGs, techniques like Retrieval-Augmented Generation (RAG) and few-shot learning offer effective alternatives without necessitating model retraining. RAG enhances response accuracy by retrieving relevant information from databases to inform text generation, while few-shot learning enables LLMs to perform new tasks with minimal examples, demonstrating adaptability in contexts with limited data or requiring rapid customization. Frameworks such as Auto-KGQA leverage RAG to autonomously identify relevant KG subgraphs, whereas approaches employing few-shot learning excel in handling complex queries that resemble known examples [2].

This paper introduces a hybrid approach that integrates RAG and few-shot learning for the NL-to-SPARQL task, enhancing the Auto-KGQA framework to incorporate the few-shot learning paradigm. By evaluating this methodology using the SciQA benchmark [3], the study assesses its efficacy and demonstrates improvements in processing domain-specific queries. Additionally, the paper proposes an automated mechanism to extract a minimal T-Box from KGs without explicitly defined schemas, optimizing query execution by limiting the exploration of deep neighborhoods, thus advancing the efficiency and scope of the Auto-KGQA framework.

The remainder of this paper is structured as follows. Section II covers related work. Section III introduces Auto-KGQA. Section IV presents the results and discussions about the experiments with the new Auto-KGQA framework. Finally, Section V contains the conclusions.

## II. Related Work

The study in [4] introduces Neural SPARQL Machines, a method for translating natural language questions into SPARQL queries using traditional machine translation and deep neural networks. While effective in addressing issues like vocabulary mismatches, this approach was limited to the DBpedia classes used during training and required large datasets to adapt to new knowledge graphs.

In contrast, Auto-KGQA [2], [5] offers a domain-independent framework that utilizes large language models (LLMs) for text-to-SPARQL translation. It operates by selecting relevant fragments of a knowledge graph's T-Box and A-Box, enabling an LLM to generate multiple SPARQL queries from a user's natural language question and select the best-performing query. The framework aims to minimize the number of tokens inputted to the LLM, thus reducing irrelevant information and improving support for large knowledge graphs. The paper notes that Auto-KGQA successfully decreases token usage without compromising performance.

The study in [3] evaluates various LLMs on the SciQA benchmark, focusing on the effects of fine-tuning and few-

2472-9671/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSC64641.2025.00016

shot learning. The best performance, an F1-score of 0.97, was achieved by T-5 with fine-tuning. GPT-3.5 Turbo excelled in in-context learning, reaching F1-scores of 0.26 (0-shot) and 0.96 (3-shot) respectively. The experiments underscore the potential of LLMs to perform well with available examples, but also their limitations without them.

## III. Auto-KGQA: An LLM-Based Framework for text-to-SPARQL

This section presents the hybrid approach followed by the new Auto-KGQA framework[1]. The framework workflow is divided into two stages. At the link[2] you will find a video with a demonstration in greater detail.

### A. Offline Stage

In the offline stage, the framework receives the KG as input via a SPARQL endpoint. Then, it extracts a minimum T-Box that describes the KG classes and properties. Next, it creates two indexes that are used in the online stage. Optionally, the offline stage may construct an index to assist in selecting examples of similar queries that are already known.

**Minimum T-Box Extraction**

Given the *schemaless* nature of RDF KGs, where the definition of existing classes and properties is not mandatory, the *Minimum T-Box Extraction* process extracts a minimum T-Box that describes the KG classes and properties.

Let $TBox_{min}$ be the RDF KG containing the minimum T-Box of KG. $TBox_{min}$ contains the set of triples defining the classes and properties in KG. Furthermore, each class and property is accompanied by its *rdfs:label*, *rdfs:comment*, *rdfs:subClassOf* (in the case of classes), *rdfs:subPropertyOf*, *rdfs:domain* and *rdfs:range* (in the case of properties).

The Minimum T-Box Extraction process goes as follows: (1) Retrieve the list of explicit classes (*labels*, comments, superclasses); (2) Retrieve the list of implicit classes (*labels*, comments, superclasses); (3) Retrieve the number of instances of each class (relevance); (4) Retrieve the list of explicit properties (*labels*, comments, superproperties, *domain*, *range*); (5) Retrieve the list of implicit properties (*labels*, comments, superproperties, *domain*, *range*); (6) Retrieve the number of triples in which each property appears (relevance).

**KG indexes construction**

The KG indexes construction process takes as input the endpoint where the KG is hosted and the desired types of indices specified through the parameters t_box_index_type and a_box_index_type. Three types of indices can be generated: (1) Full-text search index, which supports the search of KG elements by labels partially matching an input string, offering low computational cost; (2) Sentence vector embeddings index, which identifies KG elements with labels semantically close to an input string, enabling the retrieval of synonyms and related terms; and (3) DBpedia Spotlight, which is limited to DBpedia queries and annotates references to DBpedia resources using its Spotlight API.

The T-Box index maps class and property labels to their Uniform Resource Identifiers (URIs), facilitating the connection between Natural Language (NL) terms and T-Box resources. By default, it employs the sentence vector embeddings index type, enabling the recognition of synonyms and related terms in NL queries. Meanwhile, the A-Box index maps instance labels to their URIs, linking NL terms to A-Box resources. This index defaults to the full-text search index type, ensuring efficient handling of large-scale KG entity data.

This indexing process enhances the interpretability and accessibility of KGs, with each index type tailored to specific use cases, such as syntactic matches, semantic similarity, or domain-specific annotation. These features collectively optimize the utility of KGs in diverse query scenarios.

**Example Index Generation**

To use the few-shot learning technique, an index is constructed to assist in the task of finding examples of known questions that are most similar to the question being asked. This index is characterized as a distance index between embedding vectors. Its creation process is similar to that previously described, except that this index has as keys the questions in natural language and has as its body a fragment of *prompt* containing the question and its respective SPARQL query.

Thus, each example of a query (SPARQL question) gives rise to a document in the index, considering only examples evaluated as correct.

### B. Online Stage

The online stage of the methodology is designed to answer user questions effectively. It begins by normalizing the user input to ensure uniformity. An N-Grams analysis is performed on the T-Box to identify candidate resources, with corresponding triples retrieved for each identified n-gram. This process is similarly applied to the A-Box, selecting relevant resources and gathering triples.

Next, verbalizations of these retrieved triples are generated and evaluated for their relevance to the user's question. The most pertinent verbalizations are chosen, and a Steiner-tree algorithm is employed to form a connected graph of these triples. This graph is further enriched with metadata, compiled into RDF Turtle format for subsequent use.

The approach further identifies query examples from an indexed dataset that resemble the current question. These examples help create a prompt for ChatGPT, which generates five SPARQL query variations executed on a triplestore. Based on the results, ChatGPT selects the optimal SPARQL query, which is then used to formulate a natural language response to the user query. The selected query example is documented for future reference.

All prompts used with the LLM during this stage mirror those from previous work [5], with the addition of few-shot examples integrated into the SPARQL query generation prompt.

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

[2] https://youtu.be/GFSHySDitzU

### Identifying references to KG resources in the question

The process of identifying matches between a natural language question and system indices is structured into sequential steps. Initially, the input question is normalized to remove noise and standardize the text, ensuring effective search operations. A sliding window algorithm with decreasing size then scans the input, extracting word sequences to locate matches in the available indices. Each sequence is checked against the indices, and, when multiple candidates are identified, their relevance is calculated using the formula:

*Relevance = (2 * score_index + qtd_triples_resource) / 3*, where *score_index* reflects the match's quality, and *qtd_triples_resource* denotes the number of triples associated with the resource. Matches are accepted only if their relevance exceeds a threshold of 0.85, prioritizing highly relevant results.

For each identified resource, the system retrieves all associated triples and recursively collects triples from neighboring resources, up to a configurable number of iterations (default is 2). If the resource is a class, the process extends to the class member with the highest number of triples. At the conclusion of this stage, all triples related to the resources referenced in the question are gathered. A filtering step then isolates the most relevant triples to align with the question's context, enabling precise query resolution.

### Filtering triples relevant to the question

The filtering process refines the triples retrieved during query processing by eliminating those derived from semantically irrelevant inferences. The remaining triples are grouped by subject and predicate to improve data organization and ensure variety. For each group, a pseudo-verbalization in Natural Language is generated with the structure ”Subject Verb Complement,” where the subject and verb are derived from the labels of the triples' subject and property, and the complement lists the values of the triples' objects. RDF resources are represented by their labels, and literals use their direct values. This step avoids redundant data and enhances the representation of meaningful information.

Verbalizations are transformed into vector embeddings and compared with the query's embedding vector to identify the most semantically relevant triples. The closest verbalizations are selected, and the corresponding triples are retrieved. A Steiner-tree algorithm is then applied to the selected triples, generating a concise graph that captures key relationships while ensuring connectivity. If no direct paths exist between query-referenced resources in the A-Box, the algorithm leverages the T-Box to incorporate class and property connections. Metadata associated with the triples is also retrieved to enrich the results' interpretability.

The final step converts the selected subgraph into a turtle RDF serialization to be included in the prompt for the LLM. This subgraph provides contextual information about the KG's structure, serving as the output of the RAG mechanism. This ensures that the LLM can effectively utilize the KG's structure in processing the input query.

### Retrieving sample questions

This is an optional step that uses the examples queries index to retrieve examples of questions closest to the input question, based on the distance between the embedding vectors of the input question and the known examples. The step considers the $N$ best examples, where $N$ is a configurable value (the experiments reported in Section IV use $N = 3$). The examples returned consist of the natural language question and their SPARQL translation pairs.

The dataset of examples used in this step is not static, and is not limited to only examples given beforehand, but instead, it is incremented during the execution of the framework. When answering questions, the framework saves all the questions asked, the selected subgraph, the generated SPARQL question, and the user's feedback on its correctness in the *Queries Dataset*. Thus, translations evaluated as correctly generated will also be used in the following rounds as examples of queries for the few-shot learning approach.

### Text-to-SPARQL Translation and NL Answer Generation

This step employs three context vectors for distinct purposes: SPARQL query translation, selecting the best SPARQL query, and generating the final natural language (NL) response. Each context vector consists of two components: an Instruction Vector, which defines the specific task for the LLM, and a Message History Vector, which stores a queue of recent interactions (default size: 7) to preserve conversational context while avoiding outdated information. This design ensures the LLM has relevant context for accurate translations, query selection, and NL response generation.

The text-to-SPARQL translation process begins by instructing the LLM to generate multiple SPARQL queries based on the user's question. The LLM is provided with key contextual elements, such as the task definition, selected context subgraph, constraints to ensure query correctness, optional examples of similar queries, and the original user input. Once queries are generated, they are executed on the KG endpoint, and malformed queries are excluded. Valid queries return up to 10 rows of results, which are then evaluated. The LLM selects the query that best represents the user's intent by analyzing the query-result pairs. If the selection process fails, the system defaults to the first query.

The framework optionally generates an NL response summarizing the selected query's results and provides links to an RDF graph navigation tool.

### Execution storage

Finally, the framework stores all the information used during its execution process, such as input question, selected subgraph, generated SPARQL queries, selected SPARQL query, examples used in the few-shot examples approach, and user feedback. These data are used as a source of knowledge, especially as examples in the few-shot examples approach, for future questions.

## IV. Experiments and Discussions

This section evaluates the impact of the few-shot learning technique on the performance of the Auto-KGQA framework, and compares Auto-KGQA with the work reported in [3]. Auto-KGQA was instantiated with the SciQA dataset [6] as the KG and GPT-3.5 Turbo as the LLM, since this is the only model supported by both Auto-KGQA and [3].

Due to resource limitations, these experiments did not use all 513 questions from the test dataset. Instead, 100 questions were randomly selected, following the same proportion of automatically generated (96%) and manually generated (4%) questions from the full test dataset. Furthermore, these experiments were not exhaustive, focusing instead on the performance of the framework under two configurations: 0-shot learning and 3-shot learning.

Table I presents the results obtained with Auto-KGQA, compared with the results reported in [3]. The F1-score was the metric used for performance evaluation. The last column of the table measures the coverage rate of the KG resources needed to answer the question passed to the LLM.

TABLE I: Experiment results on the SciQA dataset

| Approach | F1 (0-shot learning) | F1 (3-shot learning) | KG coverage rate |
|---|---|---|---|
| [3] (GPT 3.5 Turbo) | 0.26 | 0.96 | 0.0% |
| Auto-KGQA (GPT 3.5 Turbo) | 0.73 | 0.90 | 100% |

The results demonstrate that Auto-KGQA achieved an F1-score of 0.90 in the 3-shot learning scenario, slightly below the 0.96 reported in [3]. However, in the 0-shot learning scenario, Auto-KGQA outperformed significantly, achieving an F1-score of 0.73 compared to 0.26 in [3], which relies solely on training set examples to convey KG knowledge to the LLM. Auto-KGQA autonomously selected and passed relevant KG subgraphs to the LLM, ensuring 100% term coverage for the SPARQL query and providing context even in 0-shot scenarios. While examples improved the F1-score by 0.16 in Auto-KGQA, their impact was less critical than in [3], where they accounted for a 0.70 improvement, highlighting Auto-KGQA's reduced dependence on examples.

The experiments underscore the benefits of using query examples, enhancing KGQA task performance even when KG fragments are relayed to the LLM via RAG, as with Auto-KGQA. For the SciQA dataset, where training and test set queries are variations, the LLM excels with only these examples, per results in [3]. However, few-shot learning alone may underperform in contexts lacking such examples. Thus, for more generalizable systems, using RAG mechanisms alongside query examples is a promising strategy.

## V. Conclusions

This paper proposed a hybrid approach for the text-to-SPARQL task using LLMs by integrating RAG with the few-shot learning technique. The Auto-KGQA framework was extended to include examples alongside the original RAG strategy, which selects a minimal KG subgraph as context for answering questions. Experiments conducted with the SciQA benchmark demonstrated that passing examples to the LLM improved performance, with the framework achieving an F1-score of 0.90 in the 3-shot configuration compared to 0.73 in the 0-shot setting. This highlights the potential of combining contextual subgraphs with few-shot learning for enhanced query accuracy.

Despite these improvements, the 3-shot learning configuration of Auto-KGQA did not surpass the F1-score of 0.96 reported by other approaches, likely due to the context subgraph introducing irrelevant information that detracted from the examples' impact. However, in the 0-shot setting, Auto-KGQA significantly outperformed previous results, achieving an F1-score of 0.73 compared to 0.26 reported in the literature. These findings underscore the framework's potential for generalizability, particularly when combining KG context with query examples, which proved effective in improving benchmark results even in challenging scenarios.

Additionally, the paper introduced an automated procedure for extracting a minimal T-Box from KGs without explicitly defined schemas, optimizing query processing by constructing minimal spanning trees that preserve the connections between KG nodes. This innovation reduces the need for deep neighborhood exploration, enhancing framework efficiency. The study concludes that LLMs hold considerable promise for KGQA tasks, delivering robust performance without requiring extensive training data. Future directions include deploying the framework in real-world applications and evaluating the use of open-source LLMs as alternatives to proprietary models like OpenAI's ChatGPT.

## 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] M. Yani and A. A. Krisnadhi, "Challenges, techniques, and trends of simple knowledge graph question answering: a survey," *Information*, vol. 12, no. 7, p. 271, 2021.

[2] C. V. S. Avila, V. M. Vidal, W. Franco, and M. A. Casanova, "Experiments with text-to-sparql based on chatgpt," in *2024 IEEE 18th International Conference on Semantic Computing (ICSC)*. IEEE, 2024, pp. 277–284.

[3] J. Lehmann, A. Meloni, E. Motta, F. Osborne, D. R. Recupero, A. A. Salatino, and S. Vahdati, "Large language models for scientific question answering: An extensive analysis of the sciqa benchmark," in *European Semantic Web Conference*. Springer, 2024, pp. 199–217.

[4] T. Soru, E. Marx, D. Moussallem, G. Publio, A. Valdestilhas, D. Esteves, and C. B. Neto, "Sparql as a foreign language." in *SEMANTiCS (Posters & Demos)*, 2017.

[5] C. V. S. Avila, M. A. Casanova, and V. M. Vidal, "A framework for question answering on knowledge graphs using large language models," *Extended Semantic Web Conference (ESWC)*, 2024.

[6] S. Auer, D. A. Barone, C. Bartz, E. G. Cortes, M. Y. Jaradeh, O. Karras, M. Koubarakis, D. Mouromtsev, D. Pliukhin, D. Radyush *et al.*, "The sciqa scientific question answering benchmark for scholarly knowledge," *Scientific Reports*, vol. 13, no. 1, p. 7240, 2023.