Integrating Knowledge Graphs and Large Language Models for Querying in an Industrial Environment

Domen Hočevar
domenhocevar1@gmail.com
Jožef Stefan Institute
Ljubljana, Slovenia

Klemen Kenda
klemen.kenda@ijs.si
Jožef Stefan Institute
Ljubljana, Slovenia

Abstract

Knowledge graphs have traditionally required the use of specific query languages, such as SPARQL, to retrieve relevant data. In this paper, we present a system capable of performing natural language queries on knowledge graphs by leveraging retrieval-augmented generation (RAG) and large language models (LLMs). Our system can ingest large knowledge graphs and answer queries using two approaches: first, by utilizing LLMs to extract information directly from subgraphs; and second, by generating SPARQL queries with LLMs and using the results to inform further inference, such as counting the number of items.

Keywords

knowledge graph, semantic inference, Industry 4.0, LLM, RAG

1 Introduction

In the context of Industry 4.0, knowledge graphs play a crucial role in mapping and describing the entire production vertical, from supply and demand dynamics to intricate details within the production process. This includes the configuration of shop floors, production lines, machines, and data setups, extending even to specific datasets generated during operations. Knowledge graphs can also include relevant information about the tools required for particular processes, as well as details about personnel, including their skills and roles.

A key standard for representing such data within the Industry 4.0 initiative is the Asset Administration Shell (AAS) [3], which provides a logical representation for a factory asset (can also be a piece of software, etc.). By adopting AAS, industries can ensure interoperability and standardization, enabling more efficient data exchange and integration across various systems, ultimately enhancing the agility and responsiveness of manufacturing processes.

Querying knowledge graphs can be a challenging task for end users, as it often requires expertise in specialized query languages such as SPARQL [8] — a skill that is not widely known among non-experts. Working with SPARQL SELECT queries remains a challenge also for LLMs, with performance varying significantly depending on the specific model and task complexity. While the leading LLMs can reliably address basic syntax errors, generating semantically accurate SPARQL SELECT queries remains difficult in many cases [10]. Similar work has been done on interaction with databases, however even with SQL query generation the results of GPT-4 are still far behind human ability (approx. 55% execution accuracy) [9].

Information Society 2024, 7–11 October 2024, Ljubljana, Slovenia

https://doi.org/10.70314/is.2024.sikdd.5

To overcome these challenges, we propose a system that enables users to interact with knowledge graphs through natural language queries. The system leverages LLMs' capabilities to interpret knowledge graphs while compensating for their limited ability to generate fully syntactically and semantically correct SPARQL queries. Proposed system, depicted in Figure 1, leverages large language models (LLMs) [11] to process natural language inputs and provide responses in natural language. Our approach integrates retrieval-augmented generation (RAG) techniques alongside the automatic generation of SPARQL queries based on natural language input [2].

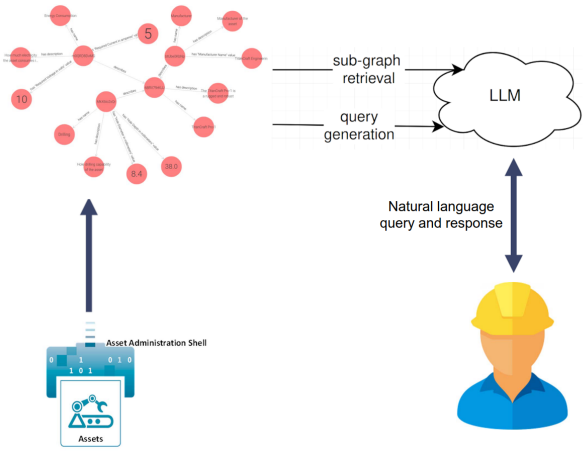

Figure 1: Intended usage of the system: AAS instances are converted into a knowledge graph, enabling natural language queries by the user.

By doing so, our system not only simplifies the querying process but also ensures that the responses are accurate and contextually relevant, making knowledge graphs more accessible and usable for a broader range of users. Additionally, the use of LLMs in combination with SPARQL querying enables the system to handle complex tasks, including those that require logical reasoning, aggregation, or interpretation of data, thus enhancing its utility in real-world applications. For example, our system is able to answer queries such as: "Give me all machines that are capable of drilling a hole with 2cm perimeter".

Finally, question answering with the help of knowledge graphs and language models has been tackled before [16], however, the development of retrieval-augmented generation (RAG) systems has seen significant growth recently. In 2024, several preprints have emerged showcasing the application of the RAG approach to knowledge graphs [12, 13, 14]. This paper contributes to this rapidly evolving field by presenting our own advancements and findings.

2 Data

This study uses a generated dataset representing a hypothetical factory with various machine models, designed to test the capabilities of the developed application. The work is part of the Smart Manufacturing pilot in the EU-funded HumAIne project [7], with the aim of eventually using real-world data from participating factories.

The mock factory includes models of "DRILLERS", "CIRCLE CUTTERS", and "CIRCULAR SAWS", each with unique names, manufacturers, and descriptions. These models are represented using AASs with relevant submodels for energy consumption, manufacturer details, and operation-specific parameters like hole diameter or depth of cut.

We created AASs for 7 drilling machine models, 7 circle cutter models, and 10 circular saw models, along with 1,000 machine instances randomly assigned to these models. Numerical values and availability were populated randomly for testing, reflecting potential real-world variations.

The initial step after acquiring AAS data is to convert it into a knowledge graph. This process involves transforming JSON-serialized AASs into RDF triples, which represent the semantic information of the data. Once the RDF triples are generated, they are stored in a GraphDB[1] repository. To enable semantic data retrieval, we employ a connector that interfaces with the ChatGPT Retrieval Plugin[2], which operates alongside the server application. When new triples are added to the GraphDB repository, the connector triggers the plugin to generate vector embeddings of the text representations of the new nodes. These embeddings are created using a language model and are stored in a separate vector database. The ChatGPT Retrieval Plugin enables interacte to a selection of different vector databases, in our case we employed the Milvus vector database. The system is also designed to maintain consistency; if any triples are removed from the GraphDB repository, the corresponding vector embeddings are automatically deleted from the vector database.

3 Methodology

The system architecture is illustrated in Figure 2. The user interacts with the system through a client application, developed using ReactJS, which serves as the graphical user interface (GUI). This client application communicates with the system's middleware, which is built on the Flask framework. Users have the capability to upload AAS data to construct and enhance the knowledge graph, as well as to issue natural language queries.

The middleware acts as the core of the system, facilitating communication between the client application, the knowledge graph stored in a GraphDB database, and OpenAI's GPT models. The AAS data uploaded by the user is first converted into RDF triples and then stored in the GraphDB repository. The Flask-based middleware also integrates with the ChatGPT Retrieval Plugin, which is responsible for generating vector embeddings of the knowledge graph nodes using OpenAI's `text-embedding-ada-002` model.

These vector embeddings are stored in the Milvus vector database [15]. The ChatGPT Retrieval Plugin allows the system to efficiently retrieve the most relevant embeddings in response to user queries, ensuring that the system can provide accurate and contextually appropriate answers. Additionally, the middleware leverages LlamaIndex[3] to manage sub-graph retrieval and query generation, which are essential for responding to complex queries by the user.

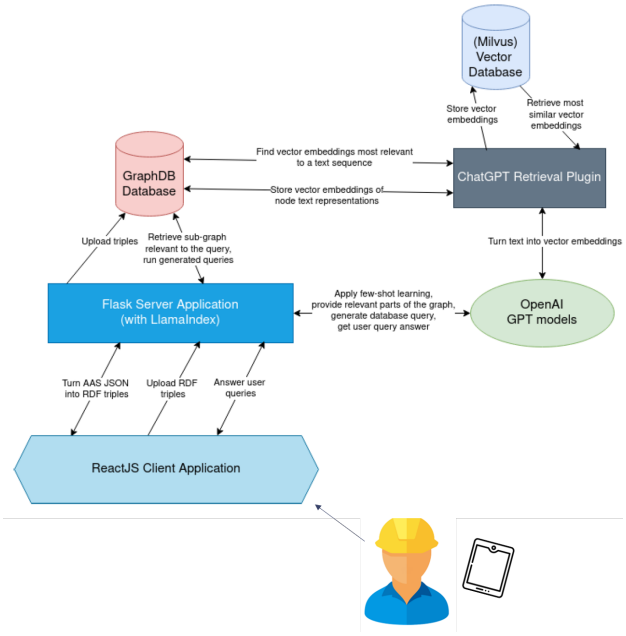

Figure 2: System architecture for retrieval augmented generation with knowledge graphs in Industry 4.0.

In summary, the architecture is designed to streamline the process of building a knowledge graph from AAS data and enables users to query this graph with retrieval-augmented generation (RAG) using natural language, with the system handling the complexities of data storage, retrieval, and natural language processing in the background.

The sequence diagram in Figure 3 illustrates the interaction between system components during query processing. Our system enables two distinct approaches to handle natural language queries, often combining both to generate a comprehensive answer for the user.

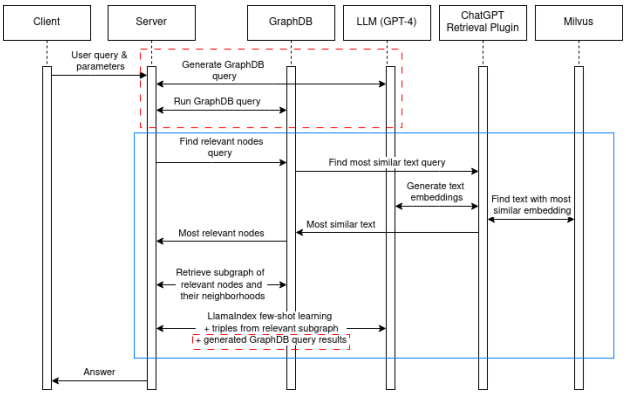

Figure 3: Sequence diagram of different approaches for data extraction. The blue box represents the RAG approach and the red box represents the SPARQL query generation approach. Note that RAG approach utilizes results from SPARQL queries on the knowledge graph.

[1] https://graphdb.ontotext.com/
[2] https://github.com/openai/chatgpt-retrieval-plugin
[3] https://www.llamaindex.ai/

The first approach utilizes a Retrieval-Augmented Generation (RAG) method. Upon receiving a query, the system analyzes the query to identify relevant concepts and generates vector embeddings for these concepts [5]. These embeddings are then matched against the knowledge graph stored in GraphDB to find the most relevant nodes. Once the relevant nodes are identified, a naive neighborhood expansion is performed, capturing additional related nodes to ensure a more complete context. The search is parameterized using parameters: scope, how many nodes from the graph to retrieve; breadth, from how many relevant nodes to start the neighborhood expansion; score weight, how many more nodes are visited from the identified relevant nodes that are deemed more relevant using embedding similarity. This subgraph, along with a few examples for context, is then fed into the Large Language Model (LLM) using a few-shot [1] learning technique to generate a response [4]. The LlamaIndex framework provides a general context query for turning triples into natural language. This method is particularly effective for queries requiring contextual understanding and extraction of complex information from the knowledge graph.

The second approach involves generating a SPARQL query based on the natural language query and the ontology used within the knowledge graph. The system attempts to execute this SPARQL query in the GraphDB database. If the query runs successfully, the resulting data is passed to the LLM to formulate the final answer. This approach is especially beneficial for tasks that involve counting instances or performing specific data aggregation operations, where LLMs alone might struggle. This approach benefits from the first approach as it can use it as backup or to enrich the SPARQL query results with additional context.

4 Results

To thoroughly evaluate the system, we employed three different evaluations: (a) assessing the accuracy of data retrieval based on query parameters (not using query generation), (b) evaluating the system's ability to correctly fetch the number of instances (testing query generation), and (c) conducting a manual assessment of most relevant user queries.

4.1 Accuracy of Data Retrieval

The first approach involved testing the system's ability to accurately retrieve data that met specific query conditions without employing SPARQL query generation. We focused on queries where the user requested a list of machines of a particular type with a voltage requirement less than or equal to a specified value. An example query would be: "Return all drilling machines that consume at most 4 volts and specify their consumption."

We conducted these tests on three types of machines: "drilling machines", "circle cutters", and "circular saws". The voltage values specified in the queries ranged from 0 to 10 volts, inclusive. The evaluation was designed to measure how accurately the system could identify and return the correct set of machines based on these voltage constraints.

For these tests, the following parameters were used (scope: 100, breadth: 1000, score weight: 100, model: `gpt-4-1106-preview`, query generation strategy: **disabled**).

The system's performance was assessed by comparing the retrieved data against the expected results, specifically checking the number of machines that met the voltage criteria and identifying any errors, such as incorrect voltage values or unnecessary machine retrievals. Results are depicted in Figures 4 and 5.

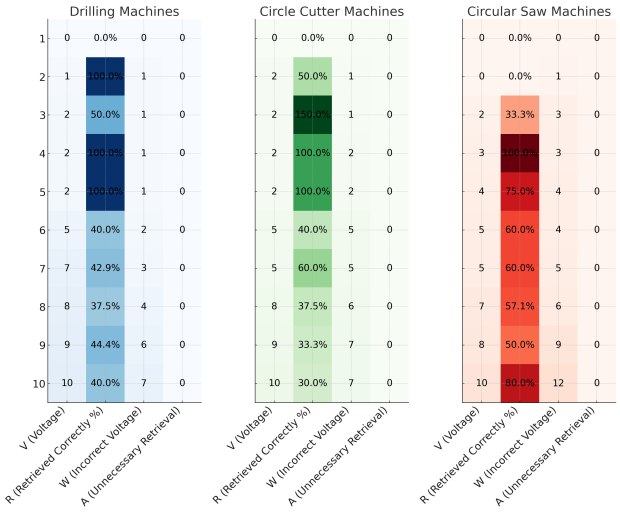

Figure 4: Performance of the system by the type of the machine and query.

In Figure 4, each table contains four columns: "V" (voltage specified in the query), "R" (percentage of correctly retrieved machines), "W" (number of machines with incorrect voltage), and "A" (number of unnecessary machine retrievals). Figure 5 summarizes the results: "Fully Correct Answers" shows the percentage of queries that returned all requested information without errors; "Share of Expected Information Found" indicates the proportion of requested information retrieved; and "Share of Incorrectly Displayed Voltages" represents the percentage of retrieved voltages that were incorrect.

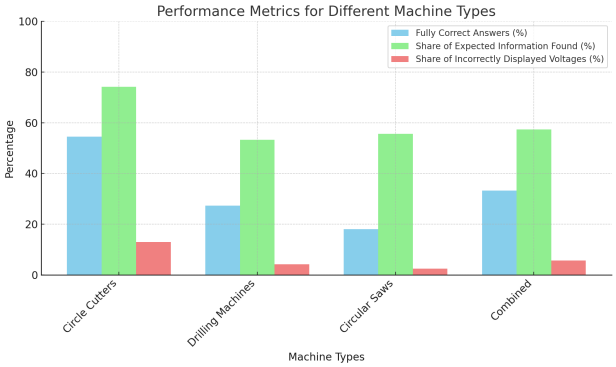

Figure 5: Combined performance.

The results show that sometimes the LLM would incorrectly generate a different voltage requirement for a machine, making it appear to satisfy the query conditions. However, the retrieved machines were always of the correct type. For example, a query like "Name all drilling machines and specify their voltage requirements" correctly retrieves all machines with the right specifications, suggesting the issue may lie with the LLM rather than the knowledge retrieval process.

To address this, users can try adjusting query parameters or rewording the query to verify the information's accuracy. If this

type of query is crucial, incorporating voltage-specific queries into the query generation strategy could improve reliability, although the LLM may struggle with large lists due to its context window limitations. As shown in Figure 5, these types of queries often do not reliably provide all requested information in one answer, so users should run multiple queries to increase the likelihood of retrieving all necessary data.

4.2 Instance Fetching Accuracy

In these tests, we tested query generation strategy. The following parameters were used (scope: 100, breadth: 1000, score weight: 100, model: `gpt-4-1106-preview`, query generation strategy: **enabled**).

The queries asked for the number of available instances for selected machine models, such as "Get the number of available [name of the machine 1], [name of the machine 2] machine instances. Specify the number for each machine type separately.". The query format was picked such that the LLM will benefit from query generation (availability property is specified in the schema supplied for query generation).

A total of 100 queries were run, with 10 queries for each number of specified machine models (ranging from 1 to 10 models). The share of fully correct answers for each query type was between 80 and 100%. The overall accuracy was 96%. This supports our hypothesis that the query generation strategy provides more accurate answers for slightly more complex queries.

4.3 Manual Evaluation of Example Queries

This evaluation was initially performed to identify several shortcomings in our methodologies as mentioned in the previous subsections. By manually evaluating specific queries relevant to end users, we were able to partially address these issues and fine-tune parameters to achieve more accurate results. For instance, while the system's initial results were often incomplete (e. g., query did not return all the machines satisfying certain criteria), increasing the breadth parameter to include a larger subgraph and allowing LLMs to traverse a broader neighborhood improved the results. Additionally, we demonstrated that subgraph retrieval and query generation can complement each other, further enhancing overall performance. All the results are commented in detail in [6].

5 Conclusions

In this paper, we presented a system that bridges the gap between natural language processing and querying knowledge graphs, specifically within the context of Industry 4.0. By leveraging large language models (LLMs) and retrieval-augmented generation (RAG), our system allows users to interact with complex knowledge graphs using natural language queries, thereby simplifying access to detailed manufacturing data.

Our evaluation demonstrated the usability of our system, however with the integration of LLMs for natural language understanding, some challenges remain. These include occasional inaccuracies in data retrieval and the LLM's limited ability to handle large datasets or specific queries. By adjusting subgraph retrieval parameters such as breadth and scope, and by combining it with SPARQL query generation, we were able to significantly enhance the system's accuracy and reliability.

This work highlights the potential of combining knowledge graphs with LLMs to create more intuitive and effective query systems in industrial environments. Future improvements could focus on refining query strategies and further optimizing the balance between subgraph retrieval and SPARQL generation to ensure even more robust and comprehensive query handling.

Acknowledgements

This work was supported by the European Commission under the Horizon Europe project HumAIne, Grant Agreement No. 101120218. We would like to express our gratitude to all project partners for their contributions and collaboration.

References

[1] Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

[2] Diego Bustamante and Hideaki Takeda. 2024. Sparql generation with entity pre-trained gpt for kg question answering. *arXiv preprint arXiv:2402.00969*.

[3] 2022. Details of the asset administration shell. https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.pdf?__blob=publicationFile&v=1 (visited on 02/22/2024). (2022).

[4] Chao Feng, Xinyu Zhang, and Zichu Fei. 2023. Knowledge solver: teaching llms to search for domain knowledge from knowledge graphs. *ArXiv*, abs/2309.03118. https://api.semanticscholar.org/CorpusID:261557137.

[5] Luis Gutiérrez and Brian Keith. 2019. A systematic literature review on word embeddings. In *Trends and Applications in Software Engineering: Proceedings of the 7th International Conference on Software Process Improvement (CIMPS 2018) 7*. Springer, 132–141.

[6] Domen Hočevar. 2024. *Integrating Knowledge Graphs and Large Language Models for Querying in an Industrial Environment*. Bachelor's Thesis. University of Ljubljana, Faculty of Computer, Information Science, Faculty of Mathematics, and Physics, Ljubljana, Slovenia, (Aug. 2024). Interdisciplinary University Study Program, First Cycle, Computer Science and Mathematics.

[7] Humaine Horizon. 2024. Humaine horizon. https://humaine-horizon.eu/. Accessed: 2024-08-26. (2024).

[8] Pérez Jorge. 2006. Semantics and complexity of sparql. In *Proc. 5th Int. Semantic Web Conference (ISWC2006)*.

[9] Jinyang Li et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

[10] Lars-Peter Meyer, Johannes Frey, Felix Brei, and Natanael Arndt. 2024. Assessing sparql capabilities of large language models. (2024). https://arxiv.org/abs/2409.05925 arXiv: 2409.05925 [cs.DB].

[11] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2023. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*.

[12] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: a roadmap. *IEEE Transactions on Knowledge and Data Engineering*.

[13] Diego Sanmartin. 2024. Kg-rag: bridging the gap between knowledge and creativity. *arXiv preprint arXiv:2405.12035*.

[14] Bhaskarjit Sarmah, Benika Hall, Rohan Rao, Sunil Patel, Stefano Pasquali, and Dhagash Mehta. 2024. Hybridrag: integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction. *arXiv preprint arXiv:2408.04948*.

[15] Jianguo Wang et al. 2021. Milvus: a purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, 2614–2627.

[16] Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. Qa-gnn: reasoning with language models and knowledge graphs for question answering. *arXiv preprint arXiv:2104.06378*.