MANGQ: Towards Natural Language Interfaces for Knowledge Graphs

Amin Harig[1], Sabine Janzen[1], Wolfgang Maaß[1,2]

{amin.harig,sabine.janzen,wolfgang.maass}@dfki.de

[1]Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany

[2]Information and Service Systems, Saarland University, Germany

Abstract

The growing number of data platforms offering large amounts of distributed, heterogeneous, but economically relevant data is more and more designed and implemented based on knowledge graphs (KGs). But despite the overall presence of KGs in data-driven systems and the growing need for intuitively accessing them, natural language querying of KGs by non-technical users is not possible. With MANGQ, we introduce a model for mapping natural language to query language expressed in GraphQL. Focusing on Question-Answering (QA) settings, we describe a question-to-query mapping approach that supports intuitive access of KGs. We present promising results from a runtime experiment with a QA system implementing the proposed approach using subsets of the CoSQL corpus and the ParaphraseBench benchmark.

Introduction

As "data are the new oil" (Economist, 2017), data platforms and marketplaces pop up offering large amounts of distributed, heterogeneous but economically relevant data (Otto and Jarke, 2019). Learning the application of semantic technologies from social networks, these data-driven systems are more and more designed and implemented on top of knowledge graphs (KGs) (Narayanasamy et al., 2022; Shinavier et al., 2019). The need for intuitively accessing these KGs by natural language interfaces (NLIs) is growing (Li and Rafiei, 2017), but up till now technical know-how and the ability to express queries in SPARQL or GraphQL are required to access the data. Related work on natural language interfaces for relational databases (NLIDBs) tackles this issue by enabling users to enter their queries in form of NL questions and mapping them into machine-readable queries (e.g., SQL) for requesting database(s). Approaches range from (1) rule-based semantic parsing of nested questions (Sheinin et al., 2018), and (2) sequence-to-sequence processing using turn-level encoding with attention (Suhr et al., 2018), to (3) syntax tree networks (Yu et al., 2018). So far, research on requesting KGs focusses on performance issues in query optimization for well-structured large KGs, e.g., (Wang et al., 2019; Zhang et al., 2021). In our research, we investigate natural language interfaces for knowledge graphs (NLIKGs). Focusing on Question-Answering (QA) settings, we consider a question-to-query-mapping approach that enables users to pose NL questions to KGs and obtain appropriate results. In this work, we propose MANGQ - a model for mapping natural language to query language expressed in GraphQL for intuitively requesting KGs. MANGQ uses the well-defined data schema in GraphQL and runs a set of schema-aware modules identifying relevant components and intentions of the user question using similarity measures. Additionally, we focus on keeping the model lightweight to ensure its performance, robustness and applicability in web-based settings, e.g., edge computing. One resulting appeal of the model is its applicability also to small or new KGs without already given big data, as proposed modules operate on the schema and do not require intensive learning. By exemplifying the model within a QA system as a NLIKG, e.g., in mobility, education and industry, we were able to evaluate the proposed approach by means of subsets of the CoSQL corpus (Yu et al., 2019) and the ParaphraseBench benchmark (Utama et al., 2018) in terms of correctness in answered user questions with promising results.

Natural Language Interfaces for Knowledge Graphs

A multitude of models for mapping NL to query language has been proposed in the past. As one main aspect of this work is the focus on portable, lightweight solutions, NLP-Reduce (Kaufmann et al., 2007) was also designed to fulfill this task on OWL without relying on expensive NLP techniques. It makes use of string similarity measures and a modular architecture that creates query component candidates and rates them by their probability of correctness. The model is therefore dependent on the user's awareness of the requested ontology's underlying structure. (Bast and Haussmann, 2015) focus on an efficient mapping (to SPARQL) and employ template matching, meaning the (recursive) insertion of identified elements into predefined query templates. Both models are robust regarding user input as they can process sentences as well as only keywords. Other examples of keyword-based querying of RDF entity graphs aim to efficiently map entered keywords to entities in the graph using an entity index of RDF data (Zhang et al., 2021). They build up an embedding space for the considered graph, in which its vertices and edges are positioned closer to each other the more they are semantically similar. This enables more efficient resolving of ambiguities in the NL question and precise identification of the relevant structures in the graph. (Utama et al., 2018) propose a deep learning-powered model for NL database requests based on neural query translation of the user questions to SQL statements. The SyntaxSQLNet model (Yu et al., 2018) is based on modules deriving specific elements of SQL statements out of user questions using an encoding of targeted data structures (table-aware column representation).

The modules generate SQL tokens (e.g., `AND`, < or a column name) that compose a syntax tree applied for recursively building up the target SQL query.

Model for Mapping Natural Language onto GraphQL

We present MANGQ, a model for mapping natural language questions to GraphQL queries consisting of three main modules following the mentioned SyntaxSQLNet (Yu et al., 2018) (cf. Figure 1): Tokenizer, Parsing Engine and Query Builder. MANGQ operates on tokens obtained from user questions by the Tokenizer and on the KG's data schema, both serving as input for the Parsing Engine. It consists of several modules charged with executing the mapping task, obtained by applying string similarity measures (i.e., Jaro-Winkler) on all pairs of input tokens and KG elements. Outputs of these modules are identified target structures of the graph, e.g., type and field names, as well as required information for GraphQL query generation, e.g., whether an aggregation shall be performed. Outputs are finally passed to the Query Builder. This component applies a multitude of pre-defined query templates, selects the most appropriate template based on the parsing engine's results and inserts corresponding elements accordingly (cf. Figure 1). This pipeline is based on low complexity procedures to ensure fast computation and high usability even in limited environments, e.g., edge computing.

For introducing the proposed approach, we will give a short example course of mapping NL onto GraphQL starting with user question and ending with GraphQL query. We apply MANGQ on a domain-specific KG extracted from the CoSQL corpus (Yu et al., 2019), consisting of pilots, aircrafts and airports related to an airline, and their relations. For the following, imagine the user poses the question: "Show me name and age of all our pilots".

Decomposition of natural language input: The Tokenizer receives the user question in form of a raw string as input (cf. Figure 1, step 1). Tokenization consists of cleaning input by removing any punctuation marks and dividing it on blank spaces. The decomposed entry represents its output as a set of single-word tokens, e.g., `[show, me, ..., pilots]` (cf. Figure 1, step 2).

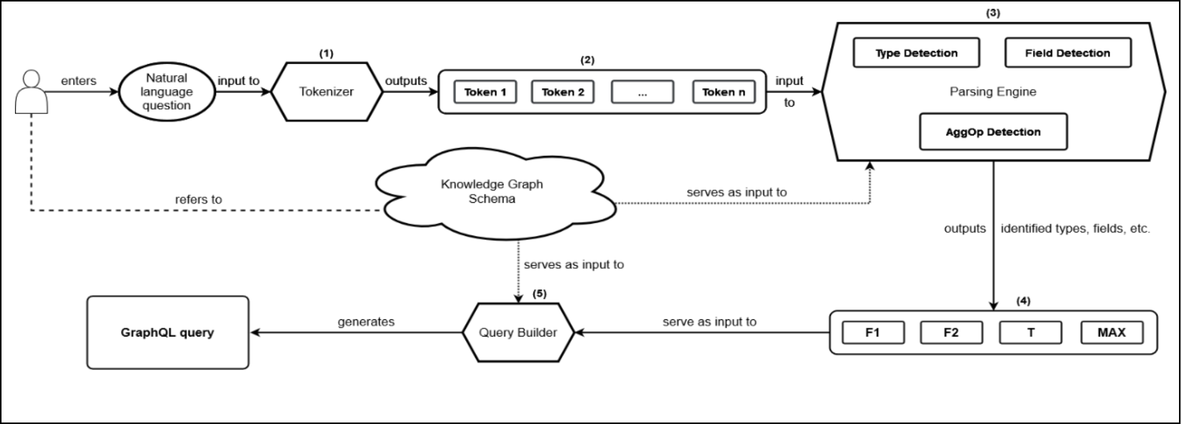

Figure 1. Model for Mapping Natural Language onto GraphQL (MANGQ)

Key component detection: The Type Detection module of the Parsing Engine processes the similarity of tokens and types specified by the schema (cf. Figure 1, step 3). The token-type-pair consisting of 'pilots' and the type `Pilot` has a Jaro-Winkler distance of 0.97, representing the highest similarity value for this token, thus leading to its selection. Other modules of the Parsing Engine are called sequentially and perform similar operations, e.g., the Field Detection module uses a similarity threshold to identify all fields mentioned in the user question. In the example, `name` and `age` are identified due to their similarity of 1.0 with tokens. Some of the other modules are the Aggregation Function Detection or the Entity Instance Detection.

Query generation: Finally, the Query Builder receives the list of identified elements (cf. Figure 1, step 5). Depending on the input and the user intention, it selects a template of the pool of

GraphQL query templates appropriate for the given user question (cf. Bast and Haussmann, 2015). The resulting GraphQL query `{Pilots {name, age}}` can then be sent to a GraphQL-powered server to obtain the data required to answer the user question.

Implementation and Evaluation

Based on the proposed model (cf. Figure 1), we implemented a text-based QA system in form of a NLIKG, e.g., in mobility, education and industry. To be able to process the obtained queries and display the results to the user, a server-client architecture has been deployed using Apollo Server for JavaScript. On client side, the interface has been implemented as a command-line based Node.js module that accepts a user question in form of plain text. Additionally, a GraphQL server was set up with the example KG(s) loaded. The resulting query is displayed to the user, sent to the server and the received JSON string is passed back to the user as an answer. A prerequisite for the QA system is the availability of data in GraphQL format including schema and resolver functions.

To evaluate our approach, we conducted two runtime studies with the implemented NLIKG prototype. Goal of these studies was to assess the performance in answered user questions. For that purpose and due to the lack of GraphQL-specific benchmarks, we selected and adapted two SQL benchmarks: the CoSQL challenge (Yu et al., 2019, reduced to a randomly selected subset due to its size) and the ParaphraseBench benchmark (Utama et al., 2018), which is composed of linguistic variations of initial, naïve user questions. The sets both target NLIDBs and consist of 212 and 399 question-query pairs covering a total range of 7 databases with a wide spread of domains, e.g., mobility, education, industry, health care. Those were remodeled as GraphQL-compatible KGs of similar structure, including the corresponding data.

Results

Model	Application Scope	Correctness
Seq2Seq (Suhr et al., 2018)	Text-to-SQL	13.9%
SyntaxSQL (Yu et al., 2018)	Text-to-SQL	14.1%
MANGQ	Text-to-GraphQL	29.0%
PICARD (Scholak et al., 2021)	Text-to-SQL	54.6%

Table 1. Correctness in comparison with models of the CoSQL challenge (Yu et al., 2019)

In summary, 212 user questions of the CoSQL corpus were selected for the experiment. 20 questions have been identified as duplicates, 28 as follow-up questions that are out of the functional scope of MANGQ and 2 questions referred to SQL-specific tasks (i.e., table operations), resulting in N_1=162 user questions considered for analysis. For evaluation on the ParaphraseBench benchmark, we exclusively considered questions lying within the scope of MANGQ. This excludes requests requiring grouping of instances as well as compound conditions over multiple fields. This led to a set of N_2=138 questions considered for analysis. The QA system was able to perfectly answer 29.01% of the CoSQL user questions. Additionally, 17.3% were answered partially, means answers included more information than requested. The model's performance in answering straightforward questions was quite good (correctness of 72.5%) whereas correctness in processing questions requiring conditional filtering of data or aggregational operations was lower (17 and 12%). On ParaphraseBench, our model was able to fully answer 55.8% of the questions. Its performance in answering the original naïve questions was very good (correctness of 89.5%), while correctness in the different paraphrased question sets exhibits a large variation from 30% (semantic paraphrases with synonyms) to 84.2% (syntactic paraphrases). While baseline models were outperformed in both benchmarks, the gap to the stronger performing models may be explained through the different depths in complexity, as MANGQ focuses on portability and does not make use of computationally intensive NLP techniques. Yet, comparison results with other models must be viewed with caution as we only evaluated on subsets of the original corpora (due to high effort of recreating the content in GraphQL and logical limitations of our model) and in a different query language. Overall, this indicates a positive evaluation of the QA system regarding its ability to map NL to GraphQL. Next steps will be to extend the model with respect to handling numerical conditions (e.g., “list all branches with a staff size below 24”), grouping by multiple variables (complex views) and handling of synonyms.

Model	Application Scope	Correctness
NaLIR (w/o feedback)	Text-to-SQL	5.5%
NSP (template only)	Text-to-SQL	10.6%
MANGQ	Text-to-GraphQL	55.8%
DBPal (Utama et al., 2018)	Text-to-SQL	75.9%

Table 2. Correctness in comparison with models of the ParaphraseBench (Utama et al., 2018)

Conclusion and Future Work

We considered natural language interfaces for knowledge graphs (NLIKGs). Despite the overall presence of knowledge graphs (KGs) in data-driven systems and the growing need for intuitively accessing these KGs, natural language querying of KGs by non-technical users is not possible. Focusing on Question-Answering (QA) settings, we introduced MANGQ, a model for mapping natural language to query language expressed in GraphQL for intuitively requesting KGs. The model was exemplified within a QA system as a NLIKG, e.g., in mobility, retailing, education, industry. We presented results from a runtime experiment with the QA system using subsets of the

CoSQL corpus (Yu et al., 2019) and the ParaphraseBench benchmark (Utama et al., 2018). Results indicate a positive evaluation of the system's performance in mapping natural language questions to GraphQL and generating appropriate answers in JSON format, while the goal of developing a portable and lightweight solution to the problem also has been achieved.

Acknowledgements

This work was partially funded by the German Federal Ministry of Economics and Climate Protection (BMWK) under the contract 01MK20015A.

References

Bast, H., & Haussmann, E. (2015). More accurate question answering on freebase. *Proc. of the 24th ACM Int. Conf. on Information and Knowledge Management*, 1431–1440.

Economist (2017). The world's most valuable resource is no longer oil, but data. The Economist.

Kaufmann, E., Bernstein, A., & Fischer, L. (2007). NLP-reduce: A naive but domain-independent natural language interface for querying ontologies. *4th European Semantic Web Conf.*, 1–2.

Li, Y., & Rafiei, D. (2017). Natural language data management and interfaces: Recent development and open challenges. *Proc. of the 2017 ACM Int. Conf. on Management of Data,* 1765–1770.

Narayanasamy, S. K., Srinivasan, K., Hu, Y.-C., Masilamani, S. K., & Huang, K.-Y. (2022). A contemporary review on utilizing semantic web technologies in healthcare, virtual communities, and ontology-based information processing systems. *Electronics, 11(3)*, 453.

Otto, B., & Jarke, M. (2019). Designing a multi-sided data platform: Findings from the International data spaces case. *Electronic Markets, 29(4)*, 561–580.

Scholak, T., Schucher, N., & Bahdanau, D. (2021). PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. *Proc. of the 2021 Conf. on Empirical Methods in Natural Language Processing*, 9895–9901.

Sheinin, V., Khorashani, E., Yeo, H., Xu, K., Vo, N. P. A., & Popescu, O. (2018). Quest: A natural language interface to relational databases. *(LREC 2018)*.

Shinavier, J., Branson, K., Zhang, W., Dastgheib, S., Gao, Y., Arsintescu, B., Özcan, F., & Meij, E. (2019). Panel: Knowledge graph industry applications. *Companion Proc. of the 2019 WWW Conf.*, 676–676.

Suhr, A., Iyer, S., & Artzi, Y. (2018). Learning to map context-dependent sentences to executable formal queries. *Proc. of the 2018 Conf. of the NAACL: Human Language Technologies, Volume 1*, 1.

Utama, P., Weir, N., Basik, F., Binnig, C., Cetintemel, U., Hättasch, B., Ilkhechi, A., Ramaswamy, S., & Usta, A. (2018). An end-to-end neural natural language interface for databases.

Yu, T., Yasunaga, M., Yang, K., Zhang, R., Wang, D., Li, Z., & Radev, D. (2018). SyntaxSQLnet: Syntax tree networks for complex and cross-domain Text-to-SQL task. *Proc. of the 2018 Conf. on Empirical Methods in Natural Language Processing*, 1653–1663.

Yu, T., Zhang, R., Er, H., Li, S., Xue, E., Pang, B., Lin, X. V., Tan, Y. C., Shi, T., Li, Z., et al. (2019). CoSQL: A conversational Text-to-SQL challenge towards cross-domain natural language interfaces to databases. *EMNLP/IJCNLP (1)*.

Zhang, H., Dong, B., Feng, B., & Wei, B. (2021). A keyword query approach based on community structure of RDF entity graph. *2021 IEEE 45th Annual Computers, Software, and Applications Conf.*, 1143–1148.