A Family of Natural Language Interfaces for Databases based on ChatGPT and LangChain

Eduardo Nascimento[1], Grettel García[1], Wendy Victorio[1], Melissa Lemos[1], Yenier Izquierdo[1], Robinson Garcia[2], Luiz Leme[3] and Marco Casanova[1]

[1]*Instituto Tecgraf and Departamento de Informática, PUC-Rio, Rio de Janeiro, 22451-900, RJ, Brazil*
[2]*Petrobras, Rio de Janeiro, 20031-912, RJ, Brazil*
[3]*Instituto de Computação, UFF, Niterói, 24210-310, RJ, Brazil*

Abstract

This poster paper proposes a family of Natural Language (NL) interfaces for databases (NLIDBs) that use ChatGPT and LangChain features to compile NL sentences expressing database questions into SQL queries or to extract keywords from NL sentences, which are passed to a database keyword search tool. The use of ChatGPT reduces dealing with NL questions to a few-shot learning process for the benefit of developers interested in creating NLIDBs. The paper concludes by comparing the NLIDBs in the family.

Keywords

Natural Language Interfaces for Databases, ChatGPT, LangChain, Database Keyword Search

1. Introduction

Natural language interfaces for databases (NLIDBs) have long been investigated with relative success [1]. The recent availability of the so-called Large Language Models [2] offers a possible strategy to simplify the development of NLIDBs worth exploring. A *Large Language Model* (LLM) is a language model based on a deep neural network architecture with a very large number of parameters, trained on enormous quantities of unlabeled text, using self-supervised or semi-supervised learning. LLMs came to the foreground with the announcement of conversational interfaces, such as ChatGPT [3], that generate high-quality textual answers, among other tasks.

The primary motivation for this paper is the construction of LLM-based NLIDBs, which poses new challenges to database designers. This paper contributes to meeting such challenges by proposing strategies to build NLIDBs that pass a suitable database description to the LLM in a context prompt. In detail, the paper first introduces an NLIDB family that uses ChatGPT and LangChain features (see Section 3) to compile NL sentences expressing database questions into SQL queries or to extract keywords from NL sentences, which are passed to a database keyword search tool. Some of the NLIDBs in the family explore ChatGPT's few-shot learning to improve the accuracy of processing NL questions. Then, the paper describes early experiments to compare the NLIDBs in the family.

ER 2023 - Posters and Demos
rogerrsn@tecgraf.puc-rio.br (E. Nascimento); ggarcia@tecgraf.puc-rio.br (G. García); wendyzv@tecgraf.puc-rio.br (W. Victorio); melissa@tecgraf.puc-rio.br (M. Lemos); ytorres@tecgraf.puc-rio.br (Y. Izquierdo); lapaesleme@ic.uff.br (L. Leme); casanova@inf.puc-rio.br (M. Casanova)



CEUR Workshop Proceedings (CEUR-WS.org)

The paper is organized as follows. Section 2 summarizes related work. Section 3 covers the required ChatGPT and LangChain features. Section 4 introduces the NLIDB family. Section 5 describes experiments with the NLIDB family. Finally, Section 6 contains the conclusions.

2. Related Work

Affolters et al. [1] offers a comparative survey of 24 NLIDBs, classified as keyword-, pattern-, parsing- and grammar-based. As lessons learned, the authors indicate that, for simple questions, keyword-based systems are enough, whereas for complex questions, an NLIDB depends on manually designed rules. Diefenbach et al. [4] assumes that an NL question can be correctly interpreted considering only the semantics of the words. For example, the NL question "Give me actors born in Berlin." is reformulated as a keyword question "Berlin, actors, born in", that is, the semantics of the words "Berlin", "actors", and "born" suffice to deduce the intention of the user. The authors proceed to map the NL question into one of four templates, which limits the usefulness of the process.

As for database keyword search, QUIOW [5] is an automatic, schema-based tool that supports keyword-based query processing for relational and RDF environments. Izquierdo et al. [6] introduced an RDF keyword search tool that does not rely on an RDF schema but synthesizes SPARQL queries by exploring the similarity between instance sets observed in the RDF dataset.

Manning [2] classifies NLP approaches in roughly four eras. The last era, from 2013 to the present, extended the use of machine learning, allowing a more generalized language understanding by using self-supervised learning and deep learning-based models to represent words with dense vectors. Within this context, ChatGPT from OpenAI was released on Nov. 30th, 2022, and uses the GPT LLM family.

Differently from [1][4], the NLIDB family proposed relies on ChatGPT to process NL questions, does not depend on grammar-based rules, and is not limited to a few database query templates.

3. A Summary of the Required ChatGPT and LangChain Features

The OpenAI API provides the *Chat Completions API*[1] that takes a list of messages as input and returns a model-generated message as output. Each message comprises a *role* ("system", "user" or "assistant") and the *content*. The system message defines the assistant's behavior from specific instructions about how it should behave during the conversation. The user messages provide requests or comments for the assistant to answer. Finally, the assistant messages refer to responses from ChatGPT, which is called *assistant* in this context.

ChatGPT (and the OpenAI API) can translate NL questions into SQL queries and extract keywords from a text block[2]. To translate NL questions into SQL queries, the user can create a simple input, with practically no context passed, and rely on the knowledge of the model to perform the translation. Alternatively, the user can manually create a system message to pass more context about the data and the translation task, such as metadata describing the database tables.

[1]https://platform.openai.com/docs/guides/gpt/chat-completions-api
[2]https://platform.openai.com/examples

For example, the following system message prompts ChatGPT to generate an SQL query for the NL question "`Tell me about cities of Brazil`":

```
[{"role": "system", "content": "Given the following SQL tables, your job is to write queries given a 'users
    request. CREATE TABLE country (id int, name varchar); CREATE TABLE city (id int, name varchar);"},
{"role": "user", "content": "Tell me about cities of Brazil"}]
```

Another approach would be to use frameworks to develop LLM-powered applications, such as LangChain[3]. Chains, in LangChain, go beyond a single LLM call and involve chaining together sequences of calls (whether to an LLM or a different utility). The chain named SQLDatabaseChain inspects the schema, tables, and joins in the database, then provides context to an LLM in an automated way; the user can also manually add a prompt and choose which tables to inspect. Using such context, ChatGPT constructs an SQL query for the original NL question, which is then executed and returned by LangChain.

The following example shows how to use LangChain to generate an SQL query:

```
1. include_tables = ["airport", "borders", "city", "continent", "country"]
2. db = SQLDatabase.from_uri(uri, include_tables=include_tables)
3. db_chain = SQLDatabaseChain(llm=ChatOpenAI(temperature=0, model_name='gpt-3.5-turbo'), database=db)
4. db_chain.run("Tell me about cities of Brazil")
```

Finally, the *Chat Completions* API allows describing function calls to be used to respond when the information the LLM has is not enough or is incomplete. This feature provides a straightforward way to create an NL front-end to a database keyword search (KwS) tool.

4. The NLIDB Family

In the process of exploring ChatGPT and LangChain features to create NLIDBs, we identified two alternatives – (1) the *SQL alternative* and (2) the *KwS alternative* – which induce the proposed NLIDB family[4].

The SQL alternative translates NL questions directly into SQL queries. We explored four options to create an NLIDB in this case:
Option 1.1 – *ChatGPT-SQL*: uses only the knowledge embedded in the LLM, with no context.
Option 1.2 – *ChatGPT-SQL-Prompt*: uses a manually created system message to pass context information about the database and query examples, and runs an application on the database for validation.
Option 1.3 – *ChatGPT-SQL-LangChain*: uses LangChain with ChatGPT to generate and execute an SQL query; to avoid exceeding the ChatGPT token limit, LangChain creates chains to inspect specific tables, connects with the database, and communicates with the ChatGPT API.
Option 1.4 – *ChatGPT-SQL-LangChain-Prompt*: is the same as Option 1.3, extended with prompts.

The KwS alternative uses ChatGPT to extract keywords from an NL question and then uses a KwS tool [5][7] to find the answers. We implemented two options to create an NLIDB:
Option 2.1 – *ChatGPT-KwS*: uses the knowledge of the LLM to extract the keywords directly.
Option 2.2 – *ChatGPT-KwS-Prompt*: fine-tunes the model by passing the main terms of the domain, and examples of NL questions and their translations to keywords via prompt messages.

[3]https://python.langchain.com/docs/modules/chains/popular/sqlite
[4]The prototypes and a brief video can be found at https://github.com/dudursn/nl_interface_tools_based_chatgpt_kws

In either case, given an NL question, ChatGPT extracts the keywords and sends them to the KwS tool via the following function:

```
{
    "function_call" : {"name": "search_keywords"},
    "functions" : [{
        "name": "search_keywords",
        "description": "Send keywords to keyword search tool",
        "parameters": { "type": "object",
                        "properties": { "keywords": {"type": "string"}, },
                        "required": ["keywords"] } }]
}
```

Note that the helper always calls the function in the *function_call* parameter (which is a call to the KwS). The results of the KwS tool are passed to ChatGPT, which shows them to the user.

5. Experiments

The experiments used the Mondial database and 27 NL questions[5], which were submitted to each NLIDB of Section 4. Table 1 shows the results obtained, summarized as follows:
Option 1.1 – *ChatGPT-SQL*, which translates NL questions into SQL queries using only knowledge from the LLM, generated only incorrect SQL queries. The LLM was not schema-aware and, in some responses, did not generate an SQL query but responded with its knowledge, and in others, it asked to provide information about the database.
Option 1.2 – *ChatGPT-SQL-Prompt*, which uses a manually provided prompt with schema information, succeeded on most questions. The errors were due to mismatches between the NL question terms and table or column names.
Option 1.3 – *ChatGPT-SQL-LangChain*, which uses LangChain to provide schema information to ChatGPT, incurred more errors than Option 1.2. Some errors resulted from misplaced or non-existent column names. Also, some responses were empty due to incorrect filters.
Option 1.4 – *ChatGPT-SQL-LangChain-Prompt*, which uses a prompt with information for LangChain, showed some improvement, but repeated some of the errors of Option 1.3.
Option 2.1 – *ChatGPT-KwS*, which uses knowledge from the LLM, incurred in problems with plural names ("Country" versus "Countries") and synonyms. For some NL questions, the NLIDB returned an error the first time the question was submitted, but executed correctly if the question was re-submitted. For other NL questions, the KwS tool could not return a response for the keywords generated, and in others, ChatGPT responded using its knowledge (since GPT was trained with geographic data).
Option 2.2 – *ChatGPT-KwS-Prompt*, which uses prompts with context information and examples, was the best-performing NLIDB. ChatGPT was able to generate most of the keywords that the KwS tool expected. For example, for the NL question "*Cities with area more than 1000*", ChatGPT extracted "*City Area > 1000*", which the KwS tool could handle. However, some errors occurred because the KwS tool did not respond correctly to the keywords ChatGPT extracted.

[5]https://github.com/dudursn/nl_interface_tools_based_chatgpt_kws/blob/main/queries_results.xlsx

Table 1
Results for the NLIDB tools.

Result	ChatGPT-SQL	ChatGPT-SQL-Prompt	ChatGPT-SQL-LangChain	ChatGPT-SQL-LangChain-Prompt	ChatGPT-KwS	ChatGPT-KwS-Prompt
Correct	0	19	10	13	16	22
Failed	27	8	17	14	11	5

6. Conclusions

This paper demonstrated how ChatGPT and LangChain can help create Natural Language interfaces for databases (NLIDBs). In the SQL alternative, such tools were used to generate an SQL query from an NL question. In the KwS alternative, they were adopted to extract keywords from an NL question, which were then passed to a KwS tool, thereby extending the usefulness of the KwS tool to accept NL questions. The experiments suggest that ChatGPT-KwS-Prompt, which uses prompts with context information and examples to help extract keywords, is the best-performing NLIDB.

Acknowledgments

This work was partly funded by FAPERJ under grant E-26/200.834/2021, by CAPES under grants 88881.134081/2016-01 and 88887.694383/2022-00, and by CNPq under grant 305.587/2021-8.

References

[1] K. Affolter, K. Stockinger, A. Bernstein, A comparative survey of recent natural language interfaces for databases, The VLDB Journal 28 (2019). doi:10.1007/s00778-019-00567-8.

[2] C. D. Manning, Human Language Understanding & Reasoning, Daedalus 151 (2022) 127–138. URL: https://doi.org/10.1162/daed_a_01905, doi: 10.1162/daed_a_01905.

[3] OpenAI, Chatgpt: Optimizing language models for dialogue, https://openai.com/blog/chatgpt/, 2022.

[4] D. Diefenbach, A. Both, K. Singh, P. Maret, Towards a question answering system over the semantic web, Semant. Web 11 (2020) 421–439. doi:10.3233/SW-190343.

[5] Y. T. Izquierdo, G. M. García, E. S. Menendez, M. A. Casanova, F. Dartayre, C. H. Levy, Quiow: a keyword-based query processing tool for rdf datasets and relational databases, in: International Conference on Database and Expert Systems Applications (DEXA), Springer, 2018, pp. 259–269. doi:10.1007/978-3-319-98812-2_22.

[6] Y. T. Izquierdo, G. M. García, E. Menendez, L. A. P. Leme, A. Neves, M. Lemos, A. C. Finamore, C. Oliveira, M. A. Casanova, Keyword search over schema-less rdf datasets by sparql query compilation, Information Systems 102 (2021) 101814. doi:10.1016/j.is.2021.101814.

[7] Y. T. Izquierdo, G. M. Garcia, M. Lemos, A. Novello, B. Novelli, C. Damasceno, L. A. P. P. Leme, M. A. Casanova, A platform for keyword search and its application for covid-19 pandemic data, Journal of Information and Data Management 12 (2021). URL: https://sol.sbc.org.br/journals/index.php/jidm/article/view/1904. doi:10.5753/jidm.2021.1904.