Large Language Models as Storage for SQL Querying

Paolo Papotti
EURECOM France
paolo.papotti@eurecom.fr

I. Introduction

Declarative querying is one of the main features behind the popularity of databases. However, SQL can be executed only on structured datasets, leaving out of immediate reach information in unstructured text. Technologies have been deployed to extract structured data from unstructured text and to model such data in relations or triples, but creating well-formed data from text is still time consuming and error prone.

While declarative querying of text is a challenge, there has recently been progress in *question answering* (QA) over text. In this setting, Large Language Models (LLMs) can answer complex questions expressed in natural language (NL) (example (2) in Figure 1). Such models store high quality factual information, but they are not trained to answer complex SQL queries and may fail short with such input.

We argue that if pre-trained LLMs could be queried with SQL scripts, it would overcome the problem of information extraction from text, thus enabling data applications on top of LLMs. As depicted in example (1) in Figure 1, we aim at using a pre-trained LLM as the data storage containing the information to answer a SQL query. Our solution preserves the characteristics of SQL when executed over this new data source: (i) queries are written in arbitrary SQL over a user defined relational schema, enabling data applications to be executed on LLM's factual data; (ii) answers are *correct* and *complete* w.r.t. the information stored in the LLM. This last point requires the correct execution of the queries and does not assume that LLMs always return perfect information. While LLMs can make factual mistakes, this work shows that it is already possible to collect valid tuples from them.

In this work, LLMs are used as a component in a traditional DB query processing architecture. Our core idea is that the query plan is a natural decomposition of the (possibly complex) process to obtain the result, in analogy with the approaches in NLP showing that breaking a complex task in a chain of thoughts is key to get the best results. We therefore introduce the problem of querying with SQL scripts an existing pre-trained LLM. These ideas are implemented in GALOIS, a prototype that executes SPJA queries over LLMs under assumptions that enable a large class of data applications (code available at https://gitlab.eurecom.fr/saeedm1/galois).

Background. Our effort is different from the problem of semantic parsing, i.e., the task of translating NL questions into SQL [1]. Our goal is also different from querying an existing relational database to answer a NL question [2]. We retrieve data from the LLM with SQL queries, with the traditional semantics and with the output expressed in the relational model, as if the query were executed on a DBMS. While some of these facts can be retrieved with QA, (i) the SQL query must be rewritten as an equivalent question in NL, which is not practical for complex scripts, (ii) the textual result must be parsed into a relation, (iii) current LLMs in some cases fail in answering complex queries expressed as NL.

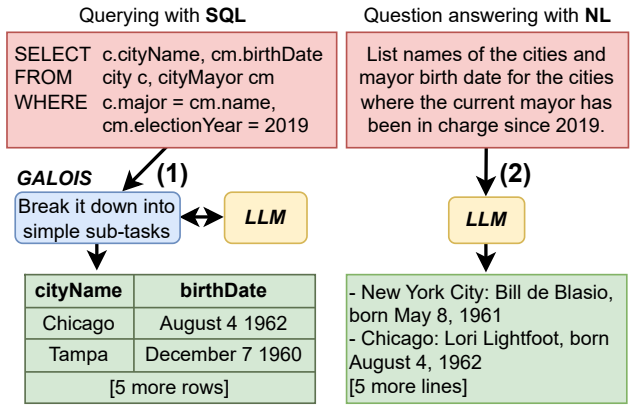

Fig. 1. Querying a LLM with SQL: GALOIS executes the query over the information stored in a LLM and outputs a relation (1). The corresponding QA task consumes and produces NL text (2).

II. Overview

Our goal is to execute SQL query over the data stored into LLMs. When we look at these models from a DB perspective, they have extensive coverage of facts from textual sources. However, LLMs have their shortcomings. We delve into three issues that have impacted the design of GALOIS.

1. Tuples and Keys. LLMs do not have a concept of schema or tuple, but they model existing relationships between entities ("Rome is located in Italy") or between entities and their properties ("Rome has 3M residents"). In the current prototype, we assume that every relation in the query has a key and that the key is expressed with one attribute, e.g., its name.

2. Schema Ambiguity. Similarly, attribute labels can have multiple meanings. These alternatives are represented differently in the LLMs. An attribute label in the query can be mapped to multiple real world attributes in the LLM, e.g., *size* for a city can refer to population or urban area [3]. We assume that meaningful labels are used in the queries.

3. Factual Knowledge in LLMs. LLMs return the next token in a stream. Such token may be based on either reliable acquired knowledge, or it may be a guess. However, we

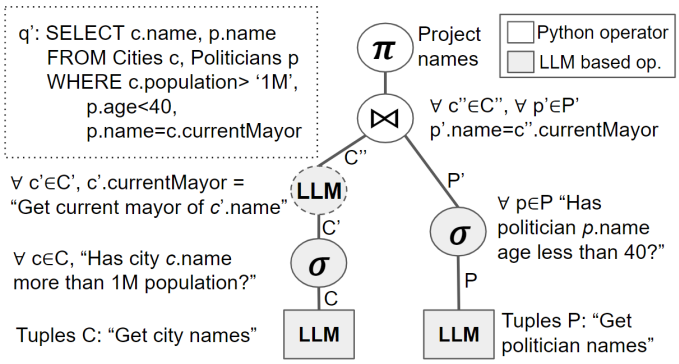

Fig. 2. Plan for query q'. Base relations are retrieves as tuple sets (*C*, *P*) with one key attribute (*name*) from the LLM.

experimentally demonstrate that it is possible to extract factual information from LLMs to answers SQL queries.

The high-level architecture of GALOIS is presented in Figure 1. We assume that the schema (but no instances) is provided together with the query. The system processes SQL queries over data stored in a pre-trained LLM. This design enables developers to implement their data applications in a conventional manner, as the complexities of using a LLM are encapsulated within GALOIS.

We use LLMs to implement specialized physical operators in a traditional query plan, as in Figure 2. As tuples are not directly available, we implement the access to the base relations (leaf nodes) with the retrieval of the key attribute values. We then retrieve other attributes as we go across the plan. A prompt is obtained for each operator by combining a set of operator-specific prompt templates with the labels/selection conditions in the given query. If a join or a projection involve an attribute that has not been collected for the tuple, this is retrieved with a node injected right before the operation. Once the tuples are completed, operators implemented in Python in our prototype are executed on those, e.g., joins and aggregates.

III. EXPERIMENTS

GALOIS is written in Python and all LLMs have been executed locally with the exception of ChatGPT.

Dataset. Spider is a Text2SQL dataset with 200 databases, each with a set of SQL queries [1]. For each query, it provides its paraphrase as a NL question. We focus on a subset of 46 queries for which we can obtain answers from an LLM.

Setup. We test four LLMs. **Flan**: T5 fine-tuned on datasets described via instructions (783M parameters). **TK**: T5 with instructions and few-shot with positive and negative examples (783M parameters). **GPT-3**: fine-tuned GPT-3 using instructions from humans (175B parameters). **ChatGPT**: chat model in the OpenAI API (175B parameters). We construct prompts for each model. For a given LLM M and a SQL query q with its Spider relation D and the corresponding NL question t, we collect three results: (a) relation R_M from GALOIS executing q over M, (b) relation R_D by executing q over D, (c) text T_M by asking t over M[1]. Only (b) uses the relations from Spider, (a) and (c) get the data from the LLM.

[1]In the full paper [4], we report also a baseline that uses chain-of-thought reasoning; its results are lower than those from GALOIS.

TABLE I. CELL VALUE MATCHES (%) BETWEEN THE METHOD'S RESULT AND THE SAME QUERY RAN ON THE GROUND TRUTH DATA (R_D) FOR THE 46 SPIDER QUERIES. AVERAGED RESULTS FOR CHATGPT.

	All	Selections only	Aggregates	Joins
R_M (SQL Queries)	50	80	29	0
T_M (NL Questions)	44	71	20	8

Evaluation. We analyze the results across two dimensions. *(1) Cardinality.* We measure to which extent GALOIS returns correct results in terms of number of tuples. All output relations have the expected schema, i.e., every R_M has the same schema as every R_D. However, in terms of number of tuples there are differences. Smaller miss lots of result rows, up to 47.4% w.r.t. the size of results from the SQL execution R_D. For GPT models, almost all queries return a number of tuples close to R_D. Most differences are explainable with errors in the results of the prompts across the query plan execution .

(2) Content. Second, we measure the quality of the results comparing cell values after manually mapping tuples between R_D on one side (ground truth) and (R_M, T_M) on the other. As T_M contains NL text, we manually postprocess them to extract the records. We consider a numerical value in (R_M, T_M) as correct if the error w.r.t. R_D is less than 5%. As this analysis requires to manually verify every result, we conduct it only for ChatGPT. Results in Table I show that GALOIS executes the queries with a better average accuracy in the results compared to the same queries expressed as questions in NL. We believe this is a promising result, as one can think that the results coming from the NL QA task are the upper bound for what the LLM knows. For the easiest subclass of queries, selection-only, the query approach returns correct values in 80% of the cases. Joins are the most problematic, as the equality test fails due to different formats of the same text, e.g., an attempt to join the country code "IT" with "ITA" for entity Italy.

IV. CONCLUSION

This work shows how a LLM can be used as a storage layer in a DBMS. The research leaves open many questions on logical and physical query optimization. Future work can explore combining operators over the LLM to reduce the number of calls to the model. In the physical optimization, it would be valuable to explore the automatic generation of more precise textual prompts, for example using data samples [5] or pre-defined embeddings for attribute types [6].

REFERENCES

[1] T. Yu and et al, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *EMNLP*, 2018, pp. 3911–3921.

[2] S. Papicchio, P. Papotti, and L. Cagliero, "Qatch: Benchmarking sql-centric tasks with table representation learning models on your data," in *NeurIPS*, 2023.

[3] E. Veltri, G. Badaro, M. Saeed, and P. Papotti, "Data ambiguity profiling for the generation of training examples," in *ICDE*. IEEE, 2023.

[4] M. Saeed, N. D. Cao, and P. Papotti, "Querying large language models with SQL," in *EDBT*, 2024.

[5] M. Urban, D. D. Nguyen, and C. Binnig, "Omniscientdb: A large language model-augmented DBMS that knows what other dbmss do not know," in *aiDM*. ACM, 2023, pp. 4:1–4:7.

[6] M. Saeed and P. Papotti, "You are my type! type embeddings for pre-trained language models," in *EMNLP*, ACL, Ed., 2022.