A DB-First Approach to Query Factual Information in LLMs

Mohammed Saeed
Sophia Antipolis, France
Turin, Italy
Paolo Papotti
EURECOM
Sophia Antipolis, France
Nicola De Cao
Google AI
London, UK

1 Introduction

Declarative querying is one of the main features behind the popularity of database systems. However, SQL can be executed only on structured datasets, leaving out of immediate reach information expressed as unstructured text. Several technologies have been deployed to extract structured data from unstructured text and to model such data in relations or triples \cite{Zhang2017, Chiticariu2018}. While these methods have been studied for more than 20 years, creating well-formed structured data from text is still time consuming and error prone.

While declarative querying of text is a big challenge, there has recently been progress in question answering (QA) over text \cite{Rogers2023}. In this setting, a question in natural language (NL) is answered by gathering information from a corpus of text documents. Large Language Models (LLMs) \cite{Radford2018, Brown2020} can answer complex questions in a closed-book fashion \cite{Roberts2020} (example (2) in Figure 1). While it has been shown that such models store high quality factual information \cite{Petroni2019, Lewis2020}, they are not trained to answer complex SQL queries and may fail short with such input.

We argue that query pre-trained LLMs with SQL scripts is beneficial to overcome the problem of information extraction from text and enables data applications on top of LLMs. As depicted in example (1) in Figure 1, a pre-trained LLM can act as the data storage containing the information to answer a declarative query. Our solution preserves the characteristics of SQL when executed over this new source of data: (i) queries are written in arbitrary SQL over a user defined relational schema, enabling data application 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 still make factual mistakes, this work shows that it is already possible to collect tuples from them with promising results. With the ongoing efforts in LLMs, there is evidence that their factuality and coverage is improving \cite{Elazar2021, Tam2022}.

Being able to SQL query LLMs is appealing, but it is not clear on which architecture a solution should pivot on. Looking at the architectures for LLMs and DBMSs, there are different paths to
explore. One is LLM-first, where external information (including structured data) is accessed by the LLM (Borgeaud et al., 2022; Peng et al., 2023). While this approach is gaining visibility with Retrieval Augmented Generation, the limited context size in LLMs does not allow yet to execute queries that require a large number of tuples as input, such as aggregate queries over thousands of rows. The alternative path is DB-first, where LLMs are used as a component in a traditional DB query processing architecture, as we do in this work. 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 (Wei et al., 2022; Khattab et al., 2022). Our contributions are summarized in the following points:

1. We introduce the problem of querying with SQL existing pre-trained LLMs. We introduce GALOIS, a DB-first prototype that executes SPJA queries under assumptions that enable a large class of data applications (code available at https://gitlab.eurecom.fr/saeedm1/galois).

2. The logical query plan breaks down the complex task into simpler steps that can be handled effectively by the LLM. Physical operators in the query plan are implemented as textual prompts for LLMs. Such prompts are generated from the input schema and the logical operators.

3. We show that GALOIS’s results for 46 queries on top of popular LLMs are (i) comparable to those obtained by executing the same queries on DBMS and (ii) better than those obtained by manually rewriting the queries (and parsing the results) in NL for QA over the same LLM.

**Background.** Our effort is different from the problem of semantic parsing, i.e., the task of translating NL questions into SQL (Yu et al., 2018; Katsogiannis-Meimarakis and Koutrika, 2021). Our goal is also different from querying an existing relational database to answer a NL question (Herzig et al., 2020; Papicchio et al., 2023). 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. Indeed, QA systems are optimized for answering questions with a text, while SQL queries return results in the form of tuples, possibly with complex operations to combine intermediate values, such as aggregates, where LLMs fail short (Ribeiro et al., 2020). To overcome some of these limits, it has recently been shown that a series of intermediate reasoning steps (“chain of thought” and question decomposition (Wolfson et al., 2020)) improve LLMs’ ability in complex tasks (Wei et al., 2022).

## 2 Design Considerations

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”). However, a query asking for city names may assume that a name identifies a city, which is not the case in reality, e.g., a Rome city in Georgia, USA. This problem is solved with keys defined with multiple attributes; a composite key defined over (name, state, country) distinguishes the Rome in Italy from the one in Georgia. In GALOIS, 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 to the issue with entities, words, including attribute labels, can have multiple meanings. These alternatives are represented differently in the LLMs. In our setting, a given 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 (Veltri et al., 2022). In this work, we assume that meaningful labels for attributes and relations are used in the queries.

3. **Factual Knowledge in LLMs.** LLMs do not know what they know. The model returns the next token in a stream. Such token may be based on either reliable acquired knowledge, or it may be a guess. For this reason, a query result obtained LLMs is not 100% reliable. However, with
GALOIS, we experimentally demonstrate that it is possible to extract factual information from LLMs to answers SQL queries. Moreover, new models keep increasing the factuality of their answers.

3 Overview

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 a set of 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 the other attributes as we go across the plan. For example, if the selection operator is defined on an attribute different from the key, the corresponding implementation is a prompt that filters every key attribute based on the selection condition, e.g., "Has city \textit{c.name} more than 1M population?", where \textit{c.name} iterates over the set of key values. A prompt is obtained for each operator by combining a set of operator-specific prompt templates with the labels/selection conditions in the given SQL query. If a join or a projection involve an attribute that has not been collected for the tuple, this is retrieved with a special node injected right before the operation. For example, if a join involves an attribute "currentMayor", the corresponding attribute values are retrieved with a prompt that collects it for every key. Once the tuples are completed, regular operators, implemented in Python in our prototype, are executed on those, e.g., joins and aggregates.

On one hand, the query plan acts as a chain of thought decomposition of the original task, i.e., the plan spells outs intermediate steps. On the other hand, the operators that manipulate data fill up the limitations of LLMs, e.g., in computing average values or comparing quantities (Lewkowycz et al., 2022). Together, these two features make the LLM able to execute complex queries.

Two critical steps enable the practical use of GALOIS. First, as relations can be large, we iterate with the a prompt until we stop getting new results. A second issue is the cleaning of the data from the LLM. For example, numerical data can be retrieved in different formats. We normalize every string expressing a numerical value (say, 1k) into a number (1000). The enforcing of type and domain constraints is a simple but crucial step to limit the incorrect output.

4 Experiments

GALOIS is written in Python and all LLMs have been executed locally with the exception of ChatGPT, for which we used the API. Query plans are obtained from DuckDB.

Dataset. Spider is a Text2SQL dataset with 200 databases, each with a set of SQL queries (Yu et al., 2018). For each query, it provides its paraphrase as a NL question. We focus on a subset of 46 queries for which we expect to obtain answers from an existing LLM. More precisely, we leave out queries that are specific to the relational dataset provided by Spider and use in our evaluation only queries about generic topics, such as world geography and airports.

Setup. We test four LLMs. \textit{Flan-T5-large} (\textit{Flan}): T5 fine-tuned on datasets described via instructions (783M parameters). \textit{TK-instruct-large} (\textit{TK}): T5 with instructions and few-shot with positive
### Table 1: Average difference in the cardinality of GALOIS’s output relations ($R_M$) w.r.t. the ground truth results ($R_D$) for the 46 Spider queries. Closer to 0 is better.

<table>
<thead>
<tr>
<th></th>
<th>Flan</th>
<th>TK</th>
<th>GPT-3</th>
<th>ChatGPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difference as % of $R_D$ size</td>
<td>-47.4</td>
<td>-43.7</td>
<td>+1.0</td>
<td>-19.5</td>
</tr>
</tbody>
</table>

Table 2: Cell value matches (%) between the result returned by a method and the same query executed on the ground truth data ($R_D$) for the 46 Spider queries. Averaged results for ChatGPT.

<table>
<thead>
<tr>
<th></th>
<th>All Selections</th>
<th>Aggregates</th>
<th>Joins</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_M$ (SQL Queries)</td>
<td>50</td>
<td>80</td>
<td>29</td>
</tr>
<tr>
<td>$T_M$ (NL Questions)</td>
<td>44</td>
<td>71</td>
<td>20</td>
</tr>
</tbody>
</table>

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. As NL questions always return text paragraphs, we cannot include their results in this analysis. For GALOIS, 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. We compute the ratio of the sizes as $f = \frac{|2 \times R_D|}{|R_D + R_M|}$, where the interval for $f$ is $[0,2]$ and best result occurs when $R_D = R_M$ ($f=1$). Table 1 reports the difference as percentage (averaged over all queries with non-empty results) with the formula $1-f$. Results show that smaller models do worse and 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, as we discuss next.

(2) **Content.** Second, we measure the quality of the results by comparing the content of each cell value 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 values as records. We consider a numerical value in ($R_M$, $T_M$) as correct if the relative error w.r.t. $R_D$ is less than 5%. As this analysis requires to manually verify every result, we conduct it only for one LLM. Results in Table 2 show that GALOIS executes the queries on ChatGPT 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.

### 5 Conclusion

This work shows how GALOIS integrates a LLM into a DBMS through SQL to create a novel query execution environment. This approach could lead to new applications, by unlocking information from unstructured text data, and foster collaboration between DB and NLP communities. The research leaves open many questions on logical and physical query optimization. For logical planning, 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 [Urban et al., 2023] or pre-defined embeddings for attribute types [Saeed and Papotti, 2022].

---

1 In the full paper [Saeed et al., 2024], we report also a baseline that uses chain-of-thought reasoning; its results are better than simple prompting but lower than those from GALOIS.
References


Mohammed Saeed, Nicola De Cao, and Paolo Papotti. 2024. Querying Large Language Models with SQL. In EDBT.


