SQL Meets LLMs: Query Optimization for Improved Quality and Efficiency

Dario Satriani, Enzo Veltri, Donatello Santoro
{dario.satriani|enzo.veltri|donatello.santoro}@unibas.it
Universitá della Basilicata
Italy

Sara Rosato, Simone Varriale, Paolo Papotti
{rosato|varriale|papotti}@eurecom.fr
EURECOM
France

ABSTRACT

Interacting with Large Language Models (LLMs) via declarative queries is increasingly popular for tasks like question answering and data extraction, thanks to their ability to process vast unstructured data. However, LLMs often struggle with answering complex factual questions, exhibiting low precision and recall in the returned data.

This challenge highlights that executing queries on LLMs remains a largely unexplored domain, where traditional data processing assumptions often fall short. Conventional query optimization, typically cost-driven, overlooks LLM-specific quality challenges such as contextual understanding. Just as new physical operators are designed to address the unique characteristics of LLMs, optimization must consider these quality challenges. Our results highlight that adhering strictly to conventional query optimization principles fails to generate the best plans in terms of result quality.

To tackle this challenge, we present a novel approach to enhance SQL results by applying query optimization techniques specifically adapted for LLMs. We introduce a database system, Galois, that sits between the query and the LLM, effectively using the latter as a storage layer. We design alternative physical operators tailored for LLM-based query execution and adapt traditional optimization strategies to this novel context. For example, while pushing down operators in the query plan reduces execution cost (fewer calls to the model), it might complicate the call to the LLM and deteriorate result quality. Additionally, these models lack a traditional catalog for optimization, leading us to develop methods to dynamically gather such metadata during query execution.

Our plug-and-play solution is compatible with any LLM and balances the trade-off between query result quality and execution cost. Experiments show up to 144% quality improvement over questions in Natural Language and 29% over direct SQL execution, highlighting the advantages of integrating database solutions with LLMs.

KEYWORDS

Large language models, SQL, query optimization

Conference'17, July 2017, Washington, DC, USA
© 2024 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Motivation. In the rapidly evolving landscape of data management, the interaction with Large Language Models (LLMs) through declarative queries has marked a significant stride forward [1, 23, 24, 34, 39]. Leveraging LLMs, like GPT-4 and LLaMA 3, has become increasingly popular for applications in direct question answering and data extraction tasks, due to their remarkable ability to process and interpret vast amounts of unstructured text [4]. These models offer a sophisticated alternative to traditional extraction methods, which require labor-intensive scripting or data annotation efforts to harness information from unstructured sources [9, 44]. By directly querying the internal representations within these models, users can extract meaningful data without the need for complex, manual parsing processes [1, 23]. This has opened avenues where developers can issue natural language (NL) or SQL-like queries, thus integrating LLMs into mainstream data retrieval processes [18, 37]. As a result, LLMs are becoming indispensable tools in scenarios where interpreting and extracting data from extensive text corpora are essential, streamlining access to information captured within the neural fabric of these models [24, 34, 39, 45].

The Problem with Data Outputs. NL question answering involves querying the knowledge embedded within pre-trained language models [32] or documents provided at runtime through techniques such as RAG [20]. However, complex NL questions that aim at obtaining data outputs often challenge these models, leading to inaccuracies and missing data in the response.

For instance, consider a scenario with a question for which we do not have a database to answer it. We can then ask the NL question to a LLM: "What are size and population of European cities with more than 1M people and more than fifteen private hospitals?". The model gives an answer, but with errors and missing data, as depicted in the top part of Figure 1. Recent advancements allow LLMs to process SQL queries directly from prompts, obtaining more precise answers compared to the corresponding questions in NL. We therefore translate the question into SQL:

```
Q1: SELECT name, size, population
    FROM EU_Cities
    WHERE population>1M AND num_private_hospitals>15
```

The query output from the LLM is improved w.r.t. the NL equivalent question, but it still reports incorrect data, as shown in Figure 1. The qualitative results for $Q1$ reflects our experiments on 92 queries over several topics, both over the LLM internal knowledge and over documents passed in the prompt (as in RAG setting). Direct NL prompting yields the least accurate results. Since NL questions can always be translated into SQL queries, either manually or with text-to-SQL techniques [18], we focus on SQL queries as input in

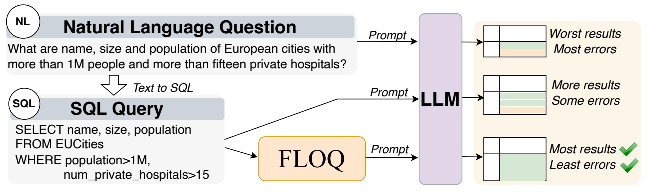


Figure 1: Comparison of query methods for obtaining data on European cites with specific criteria from the information embedded in the LLM. Galois executes SQL queries over LLMs with logical and physical optimizations, achieving superior accuracy and completeness over direct natural language and SQL prompts. The approach is effective also when executing queries over documents included in the LLM's context.

the following. However, direct SQL queries prompting improve the quality, but still fall short of the desired precision and recall.

These limitations underscore the inherent design constraints of LLMs. When queries require intricate reasoning, such as with selection conditions and aggregates, they push beyond the natural reasoning limits of these models. At the same time, albeit it is clear that LLMs indeed house, or can extract, vast amounts of data, accessing it effectively solely through NL prompts or SQL scripts often proves suboptimal. Therefore, we argue that for complex queries we should use a database management system to handle query execution, while using the LLM as a storage layer, to get better results, as in the bottom part of Figure 1. This integration leverages established database technologies but also necessitates adapting them to accommodate the distinct characteristics of LLMs, differentiating them from traditional data storage.

Challenges. A significant challenge in querying LLMs with SQL lies in balancing the trade-off between query accuracy and execution cost. Direct execution of SQL queries within LLMs comprises the most cost-effective approach. However, decomposing these queries into a sequence of operator executions, akin to the plans in a DBMS, yields superior result quality. While pushing down selections can reduce the number of interactions with the LLM, crafting simpler requests within the LLM prompt enhances the response quality. The traditional optimization principles, primarily focused on cost, are only partially applicable here. Once a logical plan is crafted from the SQL script, optimizing it to achieve both cost efficiency and result quality remains a complex task.

Additionally, unlike conventional DBMS options, LLMs do not provide direct access to crucial metadata such as schema details, column statistics, or histograms, significantly complicating query optimization efforts. The absence of this metadata requires developing novel methods to dynamically collect such information during runtime. This gap calls for solutions to ensure effective and efficient query execution over LLMs, pushing the boundaries of conventional database techniques to accommodate the unique nature of these advanced, yet inherently opaque, model architectures.

Contributions. In response to the challenges of optimizing SQL queries for LLMs, we present Galois[1], a framework designed to enhance query execution processes. First, we propose a novel physical "Scan" operator crafted specifically for interfacing with LLMs, which adeptly balances the efficiency and accuracy of data retrieval. Second, we introduce dynamic methods for acquiring essential metadata during query execution, focusing on evaluating the confidence of LLM output to bridge the gap typically filled by catalog information. Third, we develop a cost/quality model and accompanying optimization techniques that balance execution cost with query accuracy, thus enhancing the overall retrieval quality. Lastly, our experimental evaluations show the effectiveness of our approach, reporting improvements in result quality —such as enhanced accuracy and completeness— while maintaining a competitive edge in terms of resource efficiency and execution speed. These findings underscore the significant potential of integrating database management solutions with LLM technology, paving the way for sophisticated and efficient data querying practices.

[1]Framework for LLM-Optimized Querying - Available at https://anonymous.4open.science/r/floq-66B2

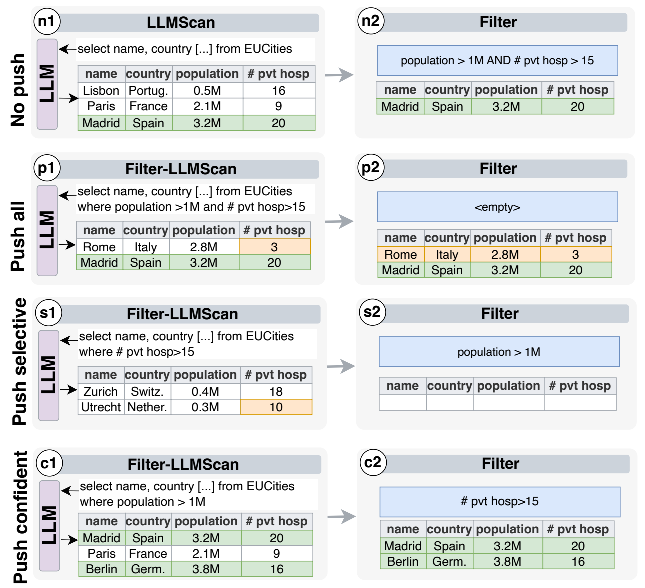
2 PROBLEM FORMULATION AND CHALLENGES

The core problem involves executing SQL queries that cannot be answered on the databases at hand. The goal is to feed the queries to an LLM, thus obtaining structured data as output from its internal knowledge, obtained from the pre-training process over a large corpus, or from the documents passed as input in the prompt, as in a RAG setting. This approach raises novel challenges due to the LLMs' fundamentally different nature from traditional databases. Unlike traditional databases, which primarily use relation-based storage and deterministic query execution plans, LLMs require to consider factors such as the generation quality of responses.

In LLM query processing, traditional optimizations aimed at reducing computational costs and execution time fall short of ensuring high-quality results. Moreover, a critical limitation is the absence of a catalog. Both traditional metadata and new, LLM-specific metadata are not readily available in a language model, but information such as column statistics is mandatory to enable query optimization in a DBMS. Finally, the solution must enable both querying the information encoded within the LLM during pre-training and the dynamic extraction of data from documents fed to the model's context at runtime, as in RAG settings.

Logical Level. At the logical level, traditional DBMSs focus on cost-effectiveness by minimizing resources such as CPU time and memory usage. However, when dealing with LLMs, quality also becomes a key metric. Consider the running example for query *Q1* in the top part of Figure 2. In the simplest logical plan, the data is gathered from the LLM with a single prompt that collects all the tuples (operator **n1**), followed by a filtering step that does not involve the LLM (operator **n2**). Notice that in the example the precision is high, but the recall is low as only one tuple is returned.

A crucial logical optimization operation is the *condition push-down*. While pushing down conditions reduces the number of interactions with the LLM, leading to lower execution costs, it often results in complex queries that are difficult for LLMs to process, thus degrading the quality of the outcomes. For example, in Figure 2, pushing both the *population* and *hospital* conditions (operator **p1**) simplifies execution from a cost perspective with fewer tokens, but fails to consider the LLM's ability to handle multi-faceted queries.



n1: select name, country [...] from EUCities

name	country	population	# pvt hosp
Lisbon	Portug.	0.5M	16
Paris	France	2.1M	9
Madrid	Spain	3.2M	20

n2: population > 1M AND # pvt hosp > 15

name	country	population	# pvt hosp
Madrid	Spain	3.2M	20

p1: select name, country [...] from EUCities where population >1M and # pvt hosp>15

name	country	population	# pvt hosp
Rome	Italy	2.8M	3
Madrid	Spain	3.2M	20

p2: <empty>

name	country	population	# pvt hosp
Rome	Italy	2.8M	3
Madrid	Spain	3.2M	20

s1: select name, country [...] from EUCities where # pvt hosp>15

name	country	population	# pvt hosp
Zurich	Switz.	0.4M	18
Utrecht	Nether.	0.3M	10

s2: population > 1M

name	country	population	# pvt hosp

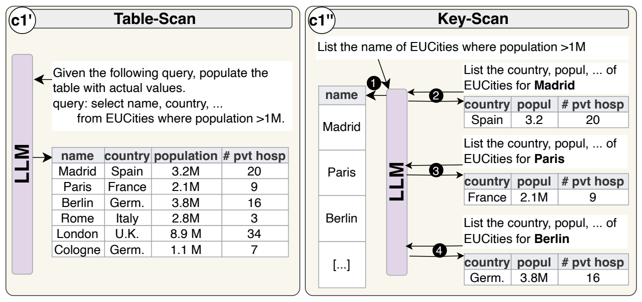
c1: select name, country [...] from EUCities where population > 1M

name	country	population	# pvt hosp
Madrid	Spain	3.2M	20
Paris	France	2.1M	9
Berlin	Germ.	3.8M	16

c2: # pvt hosp>15

name	country	population	# pvt hosp
Madrid	Spain	3.2M	20
Berlin	Germ.	3.8M	16

Figure 2: Different logical plans for querying LLMs. Varying the conditions pushed down in the scan operator impacts the precision and recall of the results.



Given the following query, populate the table with actual values.
query: select name, country, ...
from EUCities where population >1M.

name	country	population	# pvt hosp
Madrid	Spain	3.2M	20
Paris	France	2.1M	9
Berlin	Germ.	3.8M	16
Rome	Italy	2.8M	3
London	U.K.	8.9 M	34
Cologne	Germ.	1.1 M	7

List the name of EUCities where population >1M

name
Madrid
Paris
Berlin
[...]

List the country, popul, ... of EUCities for **Madrid**

country	popul	# pvt hosp
Spain	3.2	20

List the country, popul, ... of EUCities for **Paris**

country	popul	# pvt hosp
France	2.1M	9

List the country, popul, ... of EUCities for **Berlin**

country	popul	# pvt hosp
Germ.	3.8M	16

Figure 3: Two alternative physical operators for implementing logical operator c1 in Figure 2. Table-scan (c1') gets entire tuples, while Key-Scan (c1") gets key values first.

Other pushing decisions also affect the results. Pushing the most selective condition (operator **s1**) might lead to errors, preventing the identification of relevant data and ultimately producing an empty result. Balancing cost and quality necessitates evaluating each query's elements and their influence on LLM performance, rather than just attempting to reduce computational cost. In the last example in Figure 2, pushing the condition for which the model is most confident leads to the best results (operator **c1**).

Physical Level. The next challenge is to adapt query execution to treat LLMs as the data storage layer. In the physical plan, new operators are required, as data access is conducted through NL prompts to the LLM. A critical aspect is the generation of such prompts, which must be optimized to manage the variability of data retrieval quality and cost. Unlike traditional database systems, where a single command retrieves a dataset from a flat file or relational storage, LLMs require a more flexible approach.

For instance, a straightforward scanning technique retrieves the entire tuple set directly with a prompt, as in the *Table-Scan* physical operator **c1'** in Figure 3. Table-Scan minimizes LLM interactions, but can produce low-quality results for some queries. An alternative physical scan operator first identifies values for key attributes and then iteratively requests additional data, as with *Key-Scan* for operator **c1"** in Figure 3. This operator uses focused prompts that lead to better result quality. However, this comes with the cost of executing more calls to the LLM, as collecting the tuples in **c1"** requires a prompt execution for each key value. This example underscores the complexity of designing prompt-based operators striking an optimal balance between cost efficiency and output quality.

Moreover, LLMs have two limits that make difficult the extraction of their knowledge. First, they are trained to suggest the most likely next word based on the previously generated ones, thus their responses contain the most frequent values in the training corpus, while getting uncommon data in the training set may require to keep interacting with the LLM. Second, LLMs are limited in the generation of the size output response, thus we cannot expect that a single interaction produces all the correct data in general.

These challenges highlight the need for new operators and optimization techniques tailored for LLM environments. While traditional solutions offer a foundational understanding, they require augmentation to accommodate LLM-based data processing. Dynamically generating metadata is also a critical task, considering how it affects both logical and physical query plans.

3 METHODOLOGY

In a classical DBMS, executing a query involves two steps. First, the DBMS generates the logical plan, which specifies the steps needed to produce the results. Then, the DBMS derives the physical plan, which defines how the query will be executed. When it comes to querying LLMs, a distinct set of strategies is required compared to traditional DBMS to effectively manage logical and physical optimization. This section describes these strategies.

Table 1: Logical Operators Supported by GALOIS

Operator	Symbol	Description
LLMScan	$\mathcal{S}$(LLM)	Fetch data from LLM
Filter-LLMScan	$\mathcal{S}_{cond}$(LLM)	Fetch data from LLM w.r.t. cond
Selection	σ_{cond}	Select tuples w.r.t. cond
Projection	π_{attrs}	Extract attrs from tuples
Join	$\bowtie_{cond}$	Join two table given cond
Distinct	δ	Removes duplicate tuples
Grouping	γ_f	Groups tuples on common values and compute f over groups

3.1 Logical Plan

Starting from user-provided query q and schema s, we decompose q into the corresponding logical plan without any optimization. GALOIS supports the relational algebra operators in Table 1. All operators are executed in memory. The only operators interacting

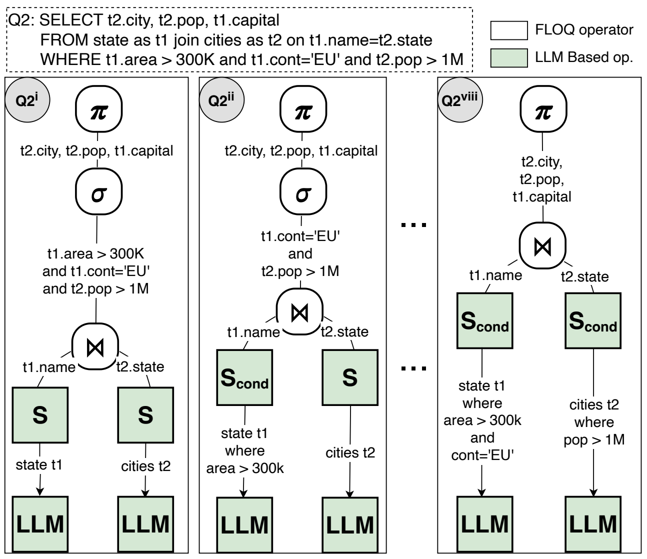

Q2: SELECT t2.city, t2.pop, t1.capital
FROM state as t1 join cities as t2 on t1.name=t2.state
WHERE t1.area > 300K and t1.cont='EU' and t2.pop > 1M
FLOQ operator
LLM Based op.
Q2i
t2.city, t2.pop, t1.capital
t1.area > 300K and t1.cont='EU' and t2.pop > 1M
t1.name
t2.state
S
state t1
cities t2
LLM
Q2ii
t1.cont='EU' and t2.pop > 1M
Scond
state t1 where area > 300k
Q2viii
state t1 where area > 300k and cont='EU'
cities t2 where pop > 1M

Figure 4: Different Logical Plans for query $Q2$. The first plan does not use any pushdown of the conditions in $\mathcal{S}$(LLM). The second and the third plans uses $\mathcal{S}_{cond}$(LLM) with a single condition, while the last uses $\mathcal{S}_{cond}$(LLM) with both conditions.

with the LLM are the *LLMScan* operator, and its variant *Filter-LLMScan*.

Traditional database systems rely on the Scan operator to transform stored tables, stored as files and pages, into a sequential flow of rows. In contrast, our proposed LLMScan operator fetches data on the fly by prompting an LLM. This dynamic data retrieval implies that LLMScan does not have complete data readily available, precluding the direct application of many traditional database optimizations.

Based on the previous observations, the initially generated logical plan could not be suitable for execution over an LLM. To address this issue, Galois generates multiple query plans using the pushdown of the selection conditions directly into a different scan operator, i.e., the Filter-LLMScan operator. Galois supports three variants of the pushdown: 1) *no-pushdown*, where the LLMScan retrieves all the tuples as in the original query plan; 2) *all conditions pushdown*, where the Filter-LLMScan retrieves the tuples that meet all the given conditions; 3) *single pushdown* of a condition, where the Filter-LLMScan retrieves the tuples that match a single condition.

Consider for example query $Q2$:

```
Q2: SELECT t2.city, t2.pop, t1.capital
    FROM state as t1 join cities as t2 on t1.name=t2.state
    WHERE t1.area > 300k and t1.cont='EU' and t2.pop > 1M
```

$Q2$ produces 8 different logical plans. Figure 4 presents three of the possible different logical plans. For example, the logical plan $Q2^{i}$ represents the no-pushdown strategy for both the tables involved, while the logical plan $Q2^{ii}$ represents a mix of pushdowns strategies for the different tables. For table *states* it consists of a single pushdown of the condition *area* > 300k, while for *cities* it uses the no-pushdown strategy. Finally, plan $Q2^{viii}$ represents the all conditions pushdown for both tables.

Algorithm 1: Table-Scan

```
Input: SQL query $q$, table $t_{name}$ , db schema $s$, max iter. $maxIter$, language model $LLM$
Output: tuple set $t$
1  $i = 0$; $prompt$ = ""; $context$=[]; $t = \{\}$;
2  while $i < maxITer$ do
3      if $i == 0$ then
4          $prompt = genFirstPrompt(t_{name}, s, q)$;
5      else
6          $prompt = genIterativePrompt()$;
7      $jsonResponse = LLM.request(prompt, context)$;
8      $parsedTuples = parse(jsonResponse, t_{name}, s)$;
9      if $noNewTuples(parsedTuples, t)$ then
10         break;
11     else
12         $context.add(prompt)$;
13         $context.add(jsonResponse)$;
14         $t.addAll(parsedTuples)$;
15     $i$++;
16 return $t$;
```

Despite the inherent limitations in the considered pushdown strategies, the number of possible logical plans for a given query can still be substantial. Indeed given a query q the number of possible logical plans can be calculated as follows:

$$push(t) = \begin{cases} 1 & if\, cond(t)=0 \\ 2 & if\, cond(t)=1 \\ cond(t)+2 & otherwise \end{cases} \tag{1}$$

$$plans(q) = \prod_{t}^{t\in q} push(t) \tag{2}$$

Where $cond(t)$ represents the number of conditions in q over the table t.

Therefore, the number of possible plans, denoted as $plans(q)$, for a given query q is calculated as the product of the possible pushdown combinations across all tables $t \in q$.

Before delving into the optimization strategies employed to select the most efficient query plan, we first elaborate on the implementation of the scan operators, which form the foundation of our execution framework.

3.2 Physical Plan

The goal of LLMScan and Filter-LLMScan is to gather structured data from the LLM parameters, such data is then further processed in the generated logical plan. Galois generates prompts in natural language to obtain the structured data from the LLM.

The most natural way to implement the scan is to prompt the LLM model to extract the tuples involved in the given query q; we call this strategy *Table-Scan*.

Table-Scan. Given a logical plan p and the relational database schema s, we employ an iterative prompting strategy for each LLM-Scan operator p. In the initial interaction, we prompt the LLM with

First Prompt: Given the following query, populate the table with actual values. query: select *attributes* from *table* (where *condition*). Respond with JSON only. Don't add any comment. Use the following JSON schema: *jsonSchema*.

attributes: is the set of attribute names of the table *table*
table: is the table name
condition: the condition if passed
jsonSchema: is the schema of the table translated in JSON schema

Iterative Prompt: List more values if there are more, otherwise return an empty JSON. Respond with JSON only.

Figure 5: Table Prompt Syntax. Text in *italic* is injected from the given SQL query. Values between parenthesis are populated only if the condition(s) is given.

a request that involves all the attributes of the table at hand. To facilitate structured data extraction, the prompt instructs the LLM to return the response in JSON format conforming to a schema derived from s. The template prompt for this task is presented in Figure 5. This enforces a structured response that the system can readily parse and store as tuples for the LLMScan operation.

Since the initial iteration typically does not retrieve all relevant data, in the following iterations we report the previous interactions in the context of the LLM (with all the generated questions and responses). We then prompt the LLM to provide additional data, if any exists, using the iterative prompt shown in Figure 5. Any non-empty JSON response is parsed, and the extracted tuples are appended to the existing result set. This iterative process terminates when the LLM returns an empty response or when an iteration yields no new tuples. Ultimately, the LLMScan operator accumulates the union of all extracted tuples, which are used in subsequent operations within the algebra tree. Algorithm 1 reports the pseudocode for the Table-Scan approach for a given table. *genFirstPrompt* and *genIterativePrompt* are auxiliary functions that implement the prompt generation illustrated in Figure 5 and enables the condition pushdown strategies. *parse* is an auxiliary function that, according to the table schema t_{name}, parses the produced response and returns a set of tuples that are valid against the JSON schema of t_{name}. $noNewTuples$ function checks if the current iteration returns no new tuples. Notice that at each iteration multiple tuples could be extracted.

This strategy addresses both the challenges of retrieving less common values and accommodating the LLM's token limitations. However, while intuitive and efficient, retrieving all data within a single prompt may not yield the most accurate results.

It is well known that LLMs benefit from techniques like Chain-of-Thought (CoT) prompting [40] to improve reasoning and output quality. Based on this observation, we introduce an alternative approach for scan, *Key-Scan*, designed to enhance the accuracy of data extraction from LLMs. In essence, CoT prompting leverages the power of step-by-step reasoning to improve the results of LLMs.

Key-Scan. This operators splits the data collection process into two steps. In the first step, Galois retrieves all the key values for the table involved in each LLMScan. In the second step, for

Algorithm 2: Key-Scan

```
Input: SQL query $q$, table $t_{name}$ , db schema $s$, max iter.
       $maxIter$, language model $LLM$
Output: tuple set $t$
1  $i = 0; prompt = ""; attrKeys = t_{name}.keys$;
2  $context = []; keys = \{\}; t = \{\}$;
3  while $i < maxITer$ do
4      if $i == 0$ then
5          $prompt = genFirstPromptKey(t_{name}, s, attrKeys, q)$;
6      else
7          $prompt = genIterativePromptKey()$;
8      $jsonResponse = LLM.request(prompt, context)$;
9      $parsedKeys = parseKeys(jsonResponse, t_{name}, s)$;
10     if $noNewKeys(parsedKeys, keys)$ then
11         break;
12     else
13         $context.add(prompt)$;
14         $context.add(jsonResponse)$;
15         $keys.addAll(parsedKeys)$;
16     $i$++;
17 $noKeysAttrs = t_{name}.attrs - t_{name}.keys$;
18 for $kVal \in keys$ do
19     $promptT = genTuplePrompt(noKeysAttrs, kVal, s)$;
20     $jsonResponse = LLM.request(promptT)$;
21     $parsedTuple = parse(jsonResponse, t_{name}, s)$;
22     $t.add(parsedTuple)$;
23 return $t$;
```

each key value retrieved, Galois obtains the values for the other attributes. For some queries, this two-step approach improves the quality of the result by asking simpler and more specific prompts to LLM. Algorithm 2 describes the Key-Scan operator. The keys are obtained with an iterative approach like the one discussed above. All prompts are reported in Figure 6. When all the keys are collected, i.e. the LLM returns no new keys or Galois reaches the maximum number of iterations, then for each key-value Galois asks the LLM to populate the remaining attribute values for the tuple. The function *genTuplePrompt* uses the template prompt (Tuple Prompt by Key) reported in Figure 6 to query the LLM and get the other attributes for the tuple with the given key value. Notice how, in this second iteration step, Galois does not use any context to query the LLM; thus, the operations in the second loop (lines 18-22) are parallelized.

In the experiments, we limit the number of iterations over the LLMs in both algorithms to reduce the costs in practice. However, both algorithms can iterate until no more new values are produced. For Table-Scan, it suffices to check if the tuple set t already contains all the new *parsedTuples*, while for Key-Scan it suffices to check if all *parsedKeys* are included in the keys set *keys*.

First Prompt Key: List the *key* of *table* (where the following condition holds: *condition*). Respond with JSON only. Use the following JSON schema: *jsonSchema*.

key: is the set of attributes that are keys for table *table*
table: is the table name
condition: the condition if passed
jsonSchema: is the schema of the table translated in JSON schema

Iterative Prompt Key: List more unique values if there are more, otherwise return an empty response. Don't repeat the previous values.

Tuple Prompt by Key: List the *attributes* of the *table* for *keyValue*. Respond with JSON only. Use the following JSON schema: *jsonSchema*

attributes: is the set of attributes of table *table* except the key attributes
keyValue: is the value for the attributes key for which we want to populate *attributes*
jsonSchema: is the schema of *table* translated in JSON schema.

Figure 6: Key-Scan Prompt Syntax. Text in *italic* is injected from the given SQL query. Values between parenthesis are populated only if the condition(s) is given.

4 LOGICAL AND PHYSICAL PLAN OPTIMIZATIONS

Given a SQL query q and schema s, Galois produces multiple plans considering both the application of condition pushdown and the choice between *Table-Scan* and *Key-Scan*. Traditional DBMSs select the optimal query execution plan by using metadata, such as value frequencies, to minimize I/O costs and query latency – factors that must be considered also when integrating LLMs in the query process [24]. However, when querying an LLM we focus our attention mainly on two aspects: *i*) the query results should be complete and accurate, avoiding hallucinations and returning factual data; and *ii*) reducing the I/O costs measured in the total tokens produced during the request and response iterations. Both those aspects depend on Logical and Physical optimizations.

Logical Optimizations. The first optimization involves the selection of the conditions that should be pushed down into the LLM-Scan. A key distinction from traditional DBMS lies in the absence of indexes and histograms, which typically guide the selection of optimal filter conditions to minimize data retrieval and processing costs. In our LLM-driven context, this translates to a lack of guidance in determining the most effective conditions to include in Filter-LLMScan operator for minimizing token usage. One natural strategy for minimizing token consumption is to push down all filter conditions into the LLMScan operator. While this approach effectively reduces the required tokens, it does not always produce the most accurate results. In certain cases, pushing down only a subset of conditions improves data quality. This is due to the tendency of LLMs to hallucinate when answering elaborate questions that demand complex reasoning. Conversely, simpler tasks, such as filtering on a single condition, reduce the cognitive load on the LLM and generally lead to more reliable results.

Unlike traditional optimization techniques, we leverage the LLM itself as a source of information for estimating the most efficient logical plan: given the schema tables s and the query q, we use the LLM in a classification task [15]. We prompt the LLM to return two *confidence* levels ("high" or "low") for each of the atoms present in the WHERE clause. All the atoms with confidence equal to high are pushed down into the LLMScan. To manage the potentially exponential growth in the number of generated plans, which scales with the number of predicates in the WHERE clause, our implementation considers only pushing down single predicates into the LLMScan operator. In particular, if the LLM returns "high" only on a single atom, Galois produces a single condition pushdown. If the LLM returns "high" for more conditions, we push down all the conditions in the WHERE of q. Otherwise, Galois produces a logical plan without any pushdown. The same process leads to LLM-driven estimates of the traditional *selectivity* for a condition; however, we show in the experiments that decisions driven by confidence lead to better results in terms of output quality.

Physical Optimizations. For a SQL query q, the next main step in Galois is the choice of the Scan strategy to execute. As we discussed in Section 3.2, we can execute *Table-Scan* or *Key-Scan*. Since *Key-Scan* uses the chain of thoughts, it is supposed to be the more accurate way to execute the Scan operation. However, in certain cases, providing the LLM with additional context regarding the table's structure and content, as in *Table-Scan*, enhances the quality of the retrieved data. By choosing the right physical scan operator is therefore possible to improve the result data quality. For this goal we rely again on metadata generated by the LLM itself.

Given a query q we estimate the *confidence* of the model in returning factual data in the Scan operation. To do so we prompt the LLM to gather a confidence value between 0 and 1.

LLMs have been recognized for overestimating their confidence in returning their knowledge [42]. To overcome this internal bias, we use the following approach: firstly, we ask the LLM to return its confidence $LLMconf(keys|conds)$ in retrieving all the key values, by providing as context the query q, the schema s, and the set of conditions *conds* for the pushdown. Then we compute how this confidence is propagated to the involved attributes in the SELECT of q. We compute this confidence, $conf(q)$ as:

$$conf(q) = LLMconf(keys|conds)^n \tag{3}$$

where n is equal to the number of attributes in the SELECT of q.

This metric aims to assess how errors, caused by the model's lack of confidence in retrieving key values, propagate through the final attributes involved in the query q.

To leverage the strengths of both *Key-Scan* and *Table-Scan*, we introduce a confidence threshold, τ. If the LLM's confidence score, $conf(q)$, for a given query q exceeds τ, we opt for the *Key-Scan* approach. Conversely, if $conf(q)$ falls below τ, we employ the *Table-Scan* operator. The rationale is that when the model's confidence is low, the accuracy of key retrieval in *Key-Scan* may be compromised. In such scenarios, *Table-Scan* provides a more reliable alternative by incorporating additional context from other attributes, potentially improving the quality of data extraction. The drawback of this approach is that it requires an extra interaction with the LLM, increasing the total costs measured in the number of tokens.

Costs Optimizations. To further optimize token usage and enhance the efficiency of LLM interactions, we introduce a mechanism for selective attribute retrieval. This optimization stems from the observation that many queries only require a subset of the available attributes in a table. By analyzing the query structure and identifying the specific attributes needed, we can restrict the LLM's response to include only the relevant attributes, while retaining the schema s within the prompts to provide comprehensive context to the LLM. However, we explicitly request to output data only for the relevant attributes, ensuring focused and efficient retrieval. This retrieval strategy directly reduces the number of tokens generated by the LLM in its response, minimizing computational overhead and response time. The optimization is particularly valuable when dealing with wide tables containing numerous attributes, where the cost for irrelevant data retrieval is higher.

5 EXPERIMENTS

We organize our evaluation around five main questions.

(1) Does GALOIS generate higher quality results when processing SQL queries compared to directly prompting the LLM with a natural language question or simply executing the SQL query?
(2) Are the proposed optimizations effective with respect to both the quality of the results and the associated costs?
(3) What affects the quality of the results? The size of the LLM parameters? The topic of the query? Or is it the complexity in terms of SQL constructs?
(4) What are the costs associated with using GALOIS, and what latency can be expected when employing our proposed approach?
(5) Can the proposed framework be effectively combined with in-context learning techniques, such as those employed in RAG, to enhance performance?

Before answering such questions, we present the experimental setup and the proposed baselines.

Experimental Setup. GALOIS is implemented in Java and uses as underlying LLM two different families: GPT [28] and LLaMa3 [11]. In particular, for the former, we used the GPT 4O-MINI model, and for the latter LLAMA-3.1-8B and LLAMA-3.1-70B, hosted on the Together AI platform (https://together.ai). We set the temperature parameter of the LLMs to zero for deterministic results.

For our evaluation, we use seven datasets, varying in the number of attributes, tables, and cardinality. Table 2 provides an overview of these datasets, including the number of queries in each and the expected average number of output cells per query.

To effectively evaluate our SQL-to-LLM approach, we selected datasets containing factual information, reflecting the LLM's ability to access general knowledge rather than transient, real-time data. This led us to use datasets with pre-existing ground truth and to further refine them by manually selecting queries most likely to have answers within the LLM's knowledge domain.

We divide the datasets into those that query the *internal knowledge* in the model and those that are crafted to contain information that cannot be in the LLM and therefore the relevant documents are passed as prompt in the *model's input context.*

Table 2: Statistics for the datasets in the experiments. *IK* stands for querying the *Internal Knowledge* and *MC* for querying the documents passed in the *Model Context.*

Dataset name	Dataset source	# of queries	Avg. expected cells	Type
FLIGHT	Spider [43]	6	267.5	IK
GEO	Spider [43]	32	22.8	IK
WORLD	Spider [43]	4	33.2	IK
MOVIES	IMDB	9	54.7	IK
PRESIDENTS	Wiki	26	42.2	IK
PREMIER	BBC	5	57.8	MC
FORTUNE	Kaggle	10	7.9	MC
GEO-TEST	Spider [43]	10	24.1	IK

For the first group, there are five datasets. Three come from the Spider [43] corpus, widely used in the NLP community for SQL-related tasks. Each training example consists of an NL question, the expected SQL query, and the tables with data. It is important to note that while the Spider datasets contain a larger number of queries, our evaluation focuses on a subset of these queries that are factual in nature and have answers that are likely to be contained within the LLMs' knowledge base, excluding from our evaluation any queries related to data contained only in Spider. This selection process ensures that the evaluation focuses on the core capabilities of GALOIS and its ability to accurately extract and process information already present within the LLMs. MOVIES is a dataset extracted from IMDB for which we manually write nine NL questions and SQL queries. PRESIDENTS is a web-scraped dataset from Wikipedia about government presidents, for which we manually write eighteen NL questions and SQL queries.

In the second group, there are two datasets that we feed to the models in their context at query execution time. We crafted PREMIER and FORTUNE to be certain that their information cannot be stored in the LLMs at the time we run the experiments, since all events in these datasets occurred in 2024 and the LLMs used in our evaluation were trained up to December 2023. PREMIER contains data from the first six match-days of the 2024-2025 Premier League season, scraped from BBC News. FORTUNE, downloaded from Kaggle, includes information about the 2024 Fortune 500 companies. These datasets serve a crucial role in evaluating GALOIS's performance within the context of applications such as RAG, where retrieving information external to the LLM's knowledge base is essential.

Finally, GEO-TEST is a dataset that we use for calibrating threshold τ for physical optimization (Section 4).

Metrics. Each experiment comprises a query q and a database d. We can compute the expected tuple set by executing q over d. Our goal is to compare the expected tuple set ($t_{exp} = q(d)$) with the tuple set produced executing the same query q on GALOIS (t_{act}).

As quality metrics to compare those two sets of tuples, we adopt metrics used to benchmark SQL queries on LLMs [3, 30]:

- F1-CELL: we compute the F1 score among the set of cells in t_{act} w.r.t the set of the cells in t_{exp}. The rationale of this metric is to evaluate the results considering only the cell values.

- CARDINALITY: we compute the ratio between the size of t_{act} w.r.t the size of t_{exp}. The rationale of this metric is to evaluate the capability of GALOIS in returning the right cardinality of the results. In particular, to report a value between 0 and 1, the cardinality quality measure is measured as: $min(size(t_{exp}), size(t_{act}))/max(size(t_{exp}), size(t_{act}))$.
- TUPLE CONSTRAINT: we measure the fraction of the tuples in t_{exp} that is present in t_{act}, where the tuple comparison is a tuple level. TUPLE CONSTRAINT is equal to 1.0 if t_{exp} and t_{act} have the same schema, the same cardinality, and the same values in the cells. This metric is stricter than F1-CELL, as it requires not only that the same values appear but also within the same corresponding tuples.
- AVG-SCORE: the F1-CELL and CARDINALITY are soft metrics, that do not consider the tuple schema at all, while TUPLE CONSTRAINT is a hard metric that considers only the tuples with the exact schema. To combine these aspects in a single metric, which is easier to compare, we introduce the AVG-SCORE that is the avg. of the previous three metrics and can be used as a proxy to summarize all the other metrics in a single number.

As F1-CELL and TUPLE CONSTRAINT rely on exact equality comparisons, we normalize cell values in both t_{act} and t_{exp} before evaluation. This normalization step helps prevent false negatives that might arise from variations in data representation, such as "1K" versus "1000". In addition, as the LLM could generate values that are similar to the expected but not the same even though they represent the same real entity, we use similarity for comparisons. This allows us to match cells like "Bill Clinton" with "Bill J. Clinton". For the string similarity, we use the Edit Distance [27] using a threshold of the 10% w.r.t. the expected cell value. For short strings, we are restrictive, allowing only a few characters of difference, while for long strings we are more permissive in the number of different characters. We use simple and efficient comparisons, a more sophisticated implementation for matching tuples could resort to Entity Resolution methods [29, 35]. For the actual numerical values, we allow a 10% difference w.r.t. the expected numerical value.

As cost metrics we use:

- # TOKENS: the total number of tokens used to prompt the LLM and generated by the LLM for a query q.
- TIME: the total time in seconds spent from sending the query q to GALOIS and getting the result.

Baselines. We consider two natural baselines:

- **NL**. This strategy involves directly prompting the LLM with a natural language question and getting structured data as a response.
- **SQL**. This strategy consists of prompting the LLM using a SQL query and getting structured data as a response.

After obtaining the initial response from each baseline model, we prompt it to generate additional values, iterating until no new tuples are produced. To ensure a fair comparison with our system, we use the same prompt structure, providing the natural language sentence or SQL query and requesting a JSON formatted response adhering to the generated JSON schema. These baselines, employed to address our first research question, aim to demonstrate the limitations of existing approaches when handling complex queries.

To demonstrate the effectiveness of decomposing queries operator and the impact of the optimizations detailed in Section 4, we introduce three variants of our system:

- GALOIS $_S$ simulates a database optimizations that pushes down the most Selective condition and employs a *Table-Scan* strategy. To pick the condition, we prompt the LLM to assess the selectivity of the conditions based on its internal knowledge, returning a value of "lower" or "higher". If only one condition is classified as "higher", we push it down; if more than one conditions are classified as "higher", we push down all the conditions; otherwise we do not push any condition.
- GALOIS $_A$ simulates a database optimization that pushes down All attributes and employs a *Table-Scan* strategy.
- GALOIS $_F$ represents the Full system with all the presented optimizations in Section 4, with both logical and physical optimizations based on the LLM confidence.

5.1 Quality of the Results

Table 3: Results for GALOIS variants and the two baselines. In bold (italic) the best (2^{nd}) result for each metric.

METRIC	NL	SQL	GALOIS $_S$	GALOIS $_A$	GALOIS $_F$
F1-CELL	0.237	0.431	0.480	*0.543*	**0.563**
CARDINALITY	0.462	0.659	0.655	*0.799*	**0.835**
TUPLE CONSTRAINTS	0.065	0.351	0.365	*0.448*	**0.464**
AVG-SCORE	0.254	0.481	0.500	*0.592*	**0.622**
# TOKENS IN M	*0.83*	**0.33**	0.96	0.95	1.72
AVG TIME	120	*61.4*	130	120.5	**47.4**

Exp-1. Overall Evaluation. In this experiment, we evaluate the performance of the variants of GALOIS against the NL and SQL baselines, using LLAMA 3.1 70B. All systems are tested on the datasets described in Table 2, excluding PREMIER and FORTUNE, which are analyzed separately in Section 5.3. A threshold $\tau = 0.6$ is used for Physical Plan selection, with the calibration process detailed in a dedicated experiment.

The results for the quality metrics are shown in Table 3, with the best results for each metric highlighted in bold and the second best in italic. NL exhibits lower performance due to the inherent ambiguity of natural language, which can lead to misinterpretation by the LLM and hinder its ability to generate the desired output. This is evidenced by the higher rate of hallucinations and data repetitions observed in the NL approach, as reflected in the CARDINALITY metric. Furthermore, the unstructured nature of natural language presents challenges in producing the required structured response. Even when the LLM gets the query's intent, it may struggle to express the extracted information in a structured format, leading to incomplete or poorly structured tuples, as indicated by the low TUPLE CONSTRAINTS metric.

SQL's declarative nature removes such ambiguities, but the approach fails with more complex queries, for example, aggregate queries, where the reasoning becomes more complex. Splitting the reasoning using the logical operators with Galois $_S$ and Galois $_A$ improves the quality w.r.t the baselines. However, as we discussed in Section 2 the classical DBMS optimizations (represented by Galois $_S$ and Galois $_A$) do not lead to the best results in terms of quality. As anticipated, the strategy of pushing down the most selective attribute (Galois $_S$) leads to suboptimal performance. By prioritizing the most selective attribute, the LLM is forced to operate on less frequent and potentially less reliable information. This can result in the omission of crucial data, as evidenced by the lower Cardinality metric achieved by Galois $_S$. Galois $_F$ reports the highest performance, with up to 144% and 29% AVG-Score improvement w.r.t NL and SQL, respectively.

From the point of view of the total costs expressed in # TOKENS, there is an increase in the number of generated tokens with Galois, the reason is the multiple steps to execute the plans. The cheapest solution in terms of produced tokens is SQL. However, overall Galois presents a good trade-off between the quality of the produced results and the costs of retrieving the data. Indeed, the cheapest solution with high quality is Galois $_A$ (always second in quality metrics), while the highest quality is obtained with Galois $_F$.

Finally, despite its high cost in terms of tokens, Galois $_F$ demonstrates the fastest result retrieval, primarily due to the use of the *Key-Scan* operator. When Galois employs Key-Scan, retrieving the keys requires less time than fetching an entire tuple. Additionally, once the keys are obtained, the remaining values of the tuples can be retrieved in parallel.

In the remaining, for the sake of presentation, we do not report results for Galois $_S$ since it shows lower quality that the other variants.

Takeaway for question (1): Galois increases the accuracy and completeness of SQL query results compared to natural language and SQL baselines with 144% and 29% improvement, respectively.

Exp-2. Effectiveness of the Optimizations. We adopt a controlled approach to analyze the effectiveness of the proposed optimizations. We fix one optimization choice and investigate the impact of varying the others. This allows us to isolate the effects of individual optimizations and understand their contributions to the overall performance. We use the GEO dataset as it has the most queries. We use LLaMa 3.1 70B as LLM to query.

Physical Optimization. We start fixing the pushdown and, for each query q in GEO, we execute it with both physical operators, i.e., *Table-Scan* and *Key-Scan.* We then measure how many times Galois $_F$ returns the optimal physical plan, i.e. the one with the highest AVG-Score between the two Scan strategies. The estimation of the correct physical plan is correct in 75% of the cases.

Logical Optimization. Since the number of combinations with all the possible pushdowns can be large, to save costs in querying the LLMs, we fix the physical plan by choosing to run only the Table-Scan, also as it is cheaper in terms of tokens. Then, for each query q with at least two conditions in the WHERE clause, we execute three different strategies involving the pushdown: 1) NO-PUSH, i.e. we execute the query without pushdown of any conditions into the LLMScan operator. This strategy represents the strategy with the highest cost since it continues to query the LLM until no new tuples are generated; 2) Galois $_A$, i.e., we push down all the conditions in q into the LLMScan. This strategy represents the strategy with the lowest cost since it should reduce the number of produced results and represents a classical DBMS strategy; and 3) Galois $_F$, that uses the Logical Optimization presented in Section 4.

Table 4: Impact of the Logical Optimization. In bold the best result per metric.

Metric	NO-PUSH	Galois $_A$	Galois $_F$
AVG-Score	0.637	0.598	**0.708**
# TOKENS in M	0.175	0.097	**0.092**

Results with the AVG-Score and number of tokens are reported in Table 4. Without any logical optimization is possible to reach a good level of quality, but it costs in terms of tokens, NO-PUSH is the one with the highest costs. Galois $_A$ reduces the number of generated tokens, but it shows an impact in terms of quality since some tuples that should be in the expected output are filtered out directly by the LLM due to complex reasoning. Finally, Galois $_F$ represents the good trade-off between the obtained quality and the number of generated tokens.

Takeaway for question (2): Galois's optimizer selects the best physical plan in 75% of cases and logical optimization identifies the plan with best quality results and lowest token cost.

5.2 Ablation Study

Table 5: AVG-Score of different LLMs. In bold the best results.

LLM	NL	SQL	Galois $_A$	Galois $_F$
GPT-4o mini	0.258	0.240	0.457	**0.468**
LLaMa 3.1 8B	0.230	0.372	0.520	**0.528**
LLaMa 3.1 70B	0.254	0.481	0.592	**0.622**

Exp-3. Impact of LLMs parameters. We execute Galois on different LLMs. We use GPT-4o Mini and LLaMa 3.1 with 8B and 70B of parameters. We execute Galois over the same datasets used for Exp-1. To compare the different models we use as metric the AVG-Score.

Results in Table 5 show that GPT-4o Mini and LLaMa 3.1 8B have comparable results. By increasing the size of the parameters, i.e. using LLaMa 3.1 70B we improve the overall quality. There are two main reasons for this difference: the first is that models with fewer parameters store less factual data than larger models; the second is related to the complexity of the pushdown, when Galois variants push down many conditions, the model may fail in retrieving the data. Variants of Galois show close results with smaller models, while the differences in terms of quality between the two variants increases with the bigger LLM.

Exp-4. Impact of retrieved values. Another aspect that is different in querying an LLM instead of a DBMS depends on the values involved in the queries. While in DBMS the quality does not depend on the queries, this is not the case for the LLM. We show the impact of the values over the same queries that have the same logical plan except for the values involved in the conditions. PRESIDENTS contains data about Presidents from the world. We collected thirteen queries about USA Presidents and the same thirteen queries about Venezuela Presidents, i.e., we change the country from "United States" to "Venezuela". Moreover, queries involve the temporal aspects, with some queries asking about historical data and other involving more recent information. Data about "Venezuela" cover the years from 1830 till today, while the data about "United States" cover the years from 1789 till today. We split the queries into three categories: ALL-TIME that covers all dates, RECENT for queries that cover data from late 1900 till today, and PAST for queries that cover data till the late 1900.

Table 6: Quality Results of in terms of AVG-SCORE in comparing the rarity of the values. In bold the best results.

DATASET	NL	SQL	GALOIS $_A$	GALOIS $_F$
PRESIDENTS-USA	0.263	0.546	0.782	**0.862**
PRESIDENTS-VENEZUELA	0.203	0.285	0.425	**0.482**

Table 6 reports the quality in terms of AVG-SCORE over LLAMA 3.1 70B. PRESIDENTS-USA contains the queries for the "United States" and PRESIDENTS-VENEZUELA the queries for the "Venezuela". All approaches suffer from the "popularity" of the data in the training set. It is more likely to obtain high-quality results when asking for more popular data, i.e., data that probably has been seen many times in the LLM's training. However, even for non-popular information, GALOIS significantly outperforms the baselines.

Table 7: Quality Results of in terms of AVG-SCORE in comparing the temporal values. In bold best results.

DATASET	NL	SQL	GALOIS $_A$	GALOIS $_F$
RECENT	0.209	0.398	0.584	**0.623**
PAST	0.171	0.305	0.518	**0.548**
ALL-TIME	0.325	0.562	0.722	**0.857**

Finally, Table 7 presents the quality results of the split datasets over time. We see how querying values about recent data impacts the quality of the results. This is because the LLMs have been trained on much more data about recent events w.r.t past events. Also in this case, GALOIS outperforms the baseline in all cases.

Exp-5. Impact of Complexity in the Query. We categorize the queries executed in Exp-1 into six categories with incremental complexity. SP2 for queries with Selection and Projection with at most two conditions, SP2> for those with more than two conditions, DIST for queries with DISTINCT, AGGR for queries with aggregate functions, G/O for queries with group by or order by, and JOIN for queries with joins. Figure 7 reports how AVG-SCORE is impacted by the complexity of the queries for the baselines and GALOIS. Increasing the complexity of the queries decreases the quality. In

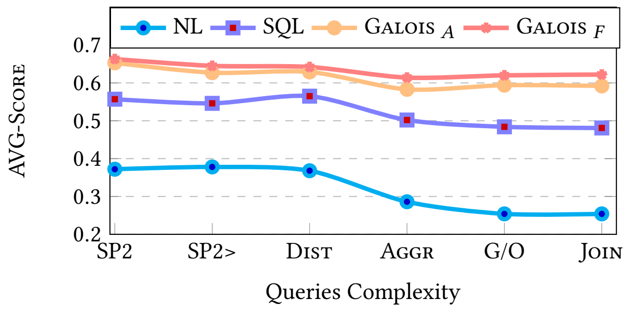


Figure 7: Result quality w.r.t query complexity.

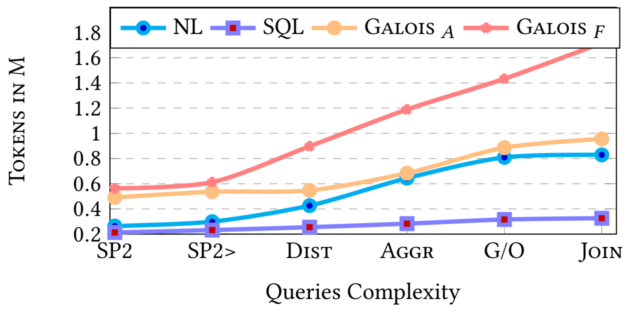


Figure 8: Costs w.r.t query complexity.

the cases of aggregate functions, group/order by and joins, the LLMScan needs to retrieve all the possible data to get the correct results. However, this is hard because even a single incorrect or missing value leads to a mismatch with the ground truth.

Takeaways for question (3): GALOIS's quality improves with larger LLMs. LLAMA 3.1 70B scores higher than smaller models (GPT-4o MINI, LLAMA 3.1 8B), for its ability to store more factual data and execute complex pushdown operations. The quality of results from LLMs is impacted by both the popularity and the temporal relevance of query values, while results are more stable w.r.t. the complexity of SQL scripts - GALOIS outperforms the baselines also in challenging cases.

The issue with the number of calls to the model is also visible in the results in terms of costs in Figure 8. When GALOIS works with complex queries, it collects as much as possible data from the LLM to get an answer, thus the number of tokens increases. But even retrieving a lot of data, the LLM could still miss something that produces incomplete results w.r.t the ground truth.

Takeaways for question (4): GALOIS has higher token costs due to its multi-step execution plans compared to the straightforward SQL approach. While GALOIS $_F$ provides the highest quality results and the fastest retrieval time, it results in increased token usage. GALOIS $_A$ offers a better trade-off between cost and quality.

Exp-6. Threshold Setup. To calibrate the threshold τ for the Plan Selection, i.e. to choose when using *Table-Scan* or *Key-Scan*, we start with the highest value for τ, i.e. we execute always *Table-Scan*. Then we lower the values for τ. We stop when the AVG-SCORE does not increase anymore. The rationale of this approach is to find the best τ that helps in using the *Key-Scan* only when we

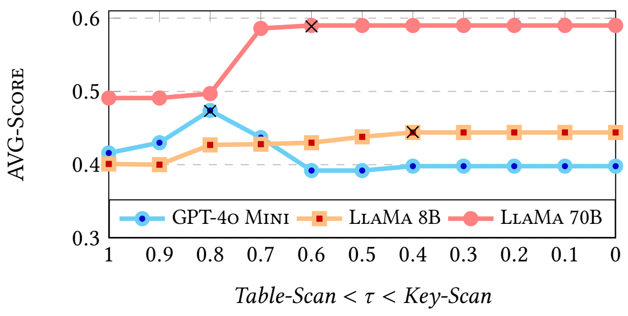


Figure 9: τ selection, $\times$ marks the optimal τ for each LLM.

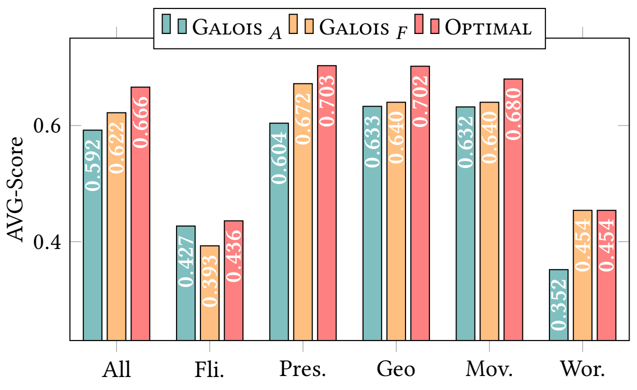


Figure 10: AVG-Score Comparison for Galois $_A$ and Galois $_F$ against the optimal plans across all datasets.

can improve the quality, otherwise, use *Table-Scan* to reduce the costs. The calibration procedure is executed for each LLM. We use a golden dataset to calibrate the threshold τ, where we know the expected query results. To not bias the chosen τ, we use GEO-Test that contains queries over different topics (mountains and states) and different typologies of queries and does not contain any query in the selected datasets used for the evaluation. Figure 9 reports how the AVG-Score changes with different thresholds. For LLaMa 3.1 70B the best τ is 0.6, and this is the value used for it on the experimental evaluation. We got 0.4 and 0.8 for Llama 3.1 8B and GPT-4o mini, respectively.

Exp-7. Errors w.r.t. the optimal plan. For each dataset and for each query in it, we generate all the possible logical and execution plans and we select the Optimal plan, i.e., the one with the highest AVG-Score. We compare Galois $_A$ and Galois $_F$ with Optimal. Results in Figure 10 report the AVG-Score on All the queries in the dataset, and we also break the AVG-Score for each dataset. We observe how Galois $_F$ overall is the closest to the Optimal, even though it is far from it with 0.44 points on All datasets. Only for Flight, Galois $_F$ has a lower AVG-Score w.r.t Galois $_A$. Analyzing the errors of Galois, we discover that are essentially due to the error in estimating the right Physical Plan, indeed in *Exp-2* we report an accuracy of 75%. For one dataset World, Galois produced the optimal plan for all the queries in it.

5.3 In-Context Querying

In the previous experiments, we used the LLM to answer queries based on its inherent internal knowledge, demonstrating how Galois enhances the quality of results. However, our framework extends beyond this. It can be seamlessly integrated with other state-of-the-art AI frameworks, such as Retrieval Augmented Generation (RAG), which combines the strengths of traditional information retrieval systems with the generative capabilities of LLMs.

While this experiment does not focus on optimizing the content enrichment phase of RAG, it aims to demonstrate that even with enriched context, converting natural language queries into SQL and applying our proposed optimizations can significantly improve the accuracy and efficiency of LLM-driven data retrieval.

To evaluate our framework's effectiveness in handling novel information, we use the Premier and Fortune datasets, comprising 60 and 500 textual documents respectively, specifically designed to contain information unseen by the LLM models. These datasets are processed using a RAG engine implemented with LangChain4j [2].

Each document is divided into text segments with a maximum size of 128 tokens for Premier and 400 tokens for Fortune, ensuring each segment contains as many paragraphs as possible. These segments are then encoded using the "WhereIsAI/UAE-Large-V1" model [22] and stored in a vector database. The prompt is then enriched at runtime with the 50 most relevant segments retrieved from the vector store based on embedding distance. All models are fed with the same chunks from the retriever. As LLM we use LLaMa 3.1 70B. We compare Galois with the other baselines and we measure the AVG-Score for the quality and the # Tokens for the costs.

Table 8: Quality Results and Costs for RAG application over Premier and Fortune datasets. In bold the best results.

Metric	NL	SQL	Galois $_A$	Galois $_F$
AVG-Score	0.348	0.387	0.477	**0.494**
# TOKENS in M	**0.661**	0.838	0.691	0.811

Results are presented in Table 8. Also in this scenario, NL and SQL exhibit the lowest performance due to the inherent challenges posed by complex queries. The unstructured nature of natural language in NL and the rigid structure of SQL restrict the LLM's ability to interpret and respond to intricate queries accurately. Both versions of Galois demonstrate similar performance, with Galois $_F$ achieving slightly higher quality scores. This similarity in performance stems from the use of new documents in the context. While Key-Scan aimed to identify key fields within the LLM's internal knowledge in previous experiments, the in-context querying shifts the focus to the contextual information provided by retrieved documents. This modification effectively replaces the need for internal key discovery with an external search mechanism, leading Galois $_F$ to rely on most queries to do a *Table-Scan* like Galois $_A$. The observed difference in quality scores can be attributed to the logical optimization layer in our system. By selectively avoiding the pushdown of all conditions for certain queries, Galois $_F$

[2]https://docs.langchain4j.dev/

achieves marginal performance gains over GALOIS $_A$, demonstrating the continued importance of logical optimization even in RAG applications.

Analyzing the token count reveals that NL incurs the lowest cost, but also delivers the lowest quality. Conversely, our system exhibits the highest cost. This is attributed to the behavior of KEY-SCAN, which, when employed, necessitates the repetition of contextual information within the prompt for each tuple retrieval, thereby increasing the overall token count.

Takeaways for question (5): GALOIS integrates effectively with in-context learning like RAG, achieving improved average performance of ten absolute points compared to the best baseline.

6 RELATED WORK

From NL Questions to SQL Scripts. NL questions are popular as users seek intuitive ways to interact with systems [21, 33]. With the rapid advancements in text-to-SQL technologies, these NL queries can be transformed into SQL scripts [3, 18]. Despite these advancements, our work in GALOIS pivots towards using SQL scripts as input, recognizing their superior expressiveness and precision in specifying user demands. GALOIS optimizes query outcomes, ensuring a robust and precise interaction with databases that leverages the strengths of SQL.

Structured Data Extraction. A common approach to managing unstructured data involves extracting it into tabular form for subsequent querying. This method is employed by IE extractors [9, 44] and text-to-table systems [41]. However, using an LLM to create complete tables upfront can be costly and prone to errors, especially with large and complex datasets. Some works assume a set of documents, passed in the model's context, and extracts a structured table where each document corresponds to a single row [1]. Also other systems show that in this setting extraction scripts can be derived to enable effective and efficient table population [23]. However, our proposal can be adopted in these solutions. Moreover, we enable querying also on the documents consumed by the LLM in the pre-training phase (no in-context).

LLMs and RAG. LLMs are gaining significant attention for their use in data querying and retrieval tasks. RAG techniques enhance LLM capabilities by integrating retrieval mechanisms to pinpoint relevant data segments, better accommodating them within LLM output constraints in terms of context windows [12, 20]. In our setting, a notable advantage of passing documents in the LLM input context is the ability to query them, even if they were not part of the LLM's pre-training data, allowing to query up-to-date content. Our approach in GALOIS uses the DBMS query structure to guide the process, thereby improving LLM query precision and recall when dealing with both fresh documents and pre-existing data sources. Our method enhances the LLM's ability to produce accurate results even when querying new and previously unseen documents.

Databases and LLMs. While significant work explores the integration of LLMs into various aspects of data management, such as query rewriting, data cleaning and database tuning [7, 10, 19, 25, 38, 46], our focus diverges. We are not concerned with using LLMs for such tasks, instead, GALOIS is designed to optimize the execution of SQL scripts over LLMs.

The evolution of DBMSs to support multiple modalities –such as text, images, and videos– is leading to the development of SQL-like interfaces across diverse data types [8, 17, 26, 34, 37, 39, 45]. As these systems use or expose declarative querying primitives to process data stored or processed by LLMs, our contributions are orthogonal to these solutions. The optimization techniques in GALOIS can be integrated with existing frameworks, providing a layer of improved efficiency and accuracy in query execution.

Other recent declarative systems optimize AI-powered analytical queries by balancing runtime, cost, and data quality [24, 31]. While they optimizes a range of tasks involving unstructured and structured data, our work can integrate into such frameworks to enhance them, e.g., by dynamically acquiring metadata during query execution for LLM-adapted and cost-effective query optimization.

In summary, considering the diverse frameworks leveraging LLM capabilities in the data management sphere, GALOIS is distinctive in integrating novel database optimization techniques with LLM functionalities for enriched query execution.

Factual Knowledge in LLMs. While LLMs have shown an ability to encapsulate extensive information, their accuracy, particularly concerning less predominant or nuanced facts, is inconsistent [36]. Additionally, LLMs can present overconfident responses even when uncertain, a challenge that has been partly addressed through recalibrating confidence scores [5]. Question answering over the LLM pre-training information also encounters these challenges; however, it stands to benefit from advancements in more factual models and solutions to improve confidence alignment.

7 CONCLUSION

In this work, we study the problem of querying LLMs through SQL queries. The system proposed, GALOIS, acts as an intermediary between the user and the LLM, adapting traditional database query optimization techniques to improve the precision and recall of query results from LLMs.

GALOIS adopts a DB-first architecture, integrating the LLM directly within the database operators to optimize structured data processing. Alternatively, an LLM-first architecture poses intriguing possibilities, though accompanied by distinct challenges. One key question is whether LLMs can replace DBMSs by ingesting structured data during training or via input context. While research in tabular language models indicates that such a scenario is not yet feasible, primarily due to context size limitations [2], recent advancements are overcoming this limitation [16].

Another promising research direction involves developing architectures that support declarative queries spanning multiple modalities, such as text, image, and structured data [39]. This integration could enable more complex and flexible querying capabilities, allowing users to extract insights from diverse data formats in a unified framework [24].

Beyond iterative refinement, another open challenge arises from the inherent biases within LLMs. These models may not return rare values unless explicitly prompted. For example, when asking for a list of "private hospitals", the LLM may return the list of US hospitals first, which is acceptable only if we are looking for private US hospitals. Instead, if we are interested in querying EU hospitals, our approach relies on the users specifying precisely their intent, which may not always be the case in practice [13].

Finally, our work also shows the increasing need to refine LLM confidence estimation mechanisms [6, 14]. Enhanced confidence assessments can lead to more reliable outputs, informing users of the certainty associated with query responses and guiding query refinements.

REFERENCES

[1] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *Proc. VLDB Endow.* 17, 2 (2023), 92–105. https://www.vldb.org/pvldb/vol17/p92-arora.pdf

[2] Gilbert Badaro, Mohammed Saeed, and Papotti Paolo. 2023. Transformers for Tabular Data Representation: A Survey of Models and Applications. *Transactions of the Association for Computational Linguistics* 11 (2023), 227–249. https://doi.org/doi.org/10.1162/tacl_a_00544

[3] Asim Biswal, Liana Patel, Siddarth Jha, Amog Kamsetty, Shu Liu, Joseph E. Gonzalez, Carlos Guestrin, and Matei Zaharia. 2024. Text2SQL is Not Enough: Unifying AI and Databases with TAG. (2024). arXiv:cs.DB/2408.14717 https://arxiv.org/abs/2408.14717

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *NeurIPS*. https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[5] Lihu Chen, Alexandre Perez-Lebel, Fabian M. Suchanek, and Gaël Varoquaux. 2024. Reconfidencing LLMs from the Grouping Loss Perspective. (2024). arXiv:cs.CL/2402.04957 https://arxiv.org/abs/2402.04957

[6] Lihu Chen, Alexandre Perez-Lebel, Fabian M Suchanek, and Gaël Varoquaux. 2024. Reconfidencing LLMs from the Grouping Loss Perspective. *arXiv preprint arXiv:2402.04957* (2024).

[7] Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. 2024. SEED: Domain-Specific Data Curation With Large Language Models. (2024). arXiv:cs.DB/2310.00749 https://arxiv.org/abs/2310.00749

[8] Zui Chen, Zihui Gu, Lei Cao, Ju Fan, Samuel Madden, and Nan Tang. 2023. Symphony: Towards Natural Language Query Answering over Multi-modal Data Lakes. In *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023.* www.cidrdb.org. https://www.cidrdb.org/cidr2023/papers/p51-chen.pdf

[9] Laura Chiticariu, Marina Danilevsky, Yunyao Li, Frederick Reiss, and Huaiyu Zhu. 2018. SystemT: Declarative Text Understanding for Enterprise. In *NAACL*. Association for Computational Linguistics, 76–83. https://doi.org/10.18653/v1/N18-3010

[10] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *Proc. VLDB Endow.* 14, 3 (2020), 307–319. https://doi.org/10.5555/3430915.3442430

[11] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[12] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* 6491–6501.

[13] Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, Alex Van Grootel, Brandon Chow, Kai Deng, Katherine Lin, Marcos Campos, K. Venkatesh Emani, Vivek Pandit, Victor Shnayder, Wenjing Wang, and Carlo Curino. 2024. NL2SQL is a solved problem... Not!. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024.* www.cidrdb.org. https://www.cidrdb.org/cidr2024/papers/p74-floratou.pdf

[14] Jiahui Geng, Fengyu Cai, Yuxia Wang, Heinz Koeppl, Preslav Nakov, and Iryna Gurevych. 2024. A Survey of Confidence Estimation and Calibration in Large Language Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers).* 6577–6595.

[15] Shai Gretz, Alon Halfon, Ilya Shnayderman, Orith Toledo-Ronen, Artem Spector, Lena Dankin, Yannis Katsis, Ofir Arviv, Yoav Katz, Noam Slonim, and Liat Ein-Dor. 2023. Zero-shot Topical Text Classification with LLMs - an Experimental Study. In *Findings of the Association for Computational Linguistics: EMNLP 2023.* Association for Computational Linguistics, 9647–9676. https://doi.org/10.18653/v1/2023.findings-emnlp.647

[16] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing.* Association for Computational Linguistics, Singapore, 13358–13376. https://doi.org/10.18653/v1/2023.emnlp-main.825

[17] Saehan Jo and Immanuel Trummer. 2024. ThalamusDB: Approximate Query Processing on Multi-Modal Data. *Proc. ACM Manag. Data* 2, 3 (2024), 186. https://doi.org/10.1145/3654989

[18] George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *VLDB J.* 32, 4 (2023), 905–936. https://doi.org/10.1007/S00778-022-00776-8

[19] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. 2024. CHORUS: Foundation Models for Unified Data Discovery and Exploration. *Proc. VLDB Endow.* 17, 8 (2024), 2104–2114. https://www.vldb.org/pvldb/vol17/p2104-kayali.pdf

[20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*, Vol. 33. 9459–9474. https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[21] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023.* http://papers.nips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html

[22] Xianming Li and Jing Li. 2023. AnglE-optimized Text Embeddings. *arXiv preprint arXiv:2309.12871* (2023).

[23] Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, Sepanta Zeigham, Aditya G. Parameswaran, and Eugene Wu. 2024. Towards Accurate and Efficient Document Analytics with Large Language Models. (2024). arXiv:cs.DB/2405.04674 https://arxiv.org/abs/2405.04674

[24] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A Declarative System for Optimizing AI Workloads. (2024). arXiv:cs.CL/2405.14696 https://arxiv.org/abs/2405.14696

[25] Jie Liu and Barzan Mozafari. 2024. Query Rewriting via Large Language Models. (2024). arXiv:cs.DB/2403.09060 https://arxiv.org/abs/2403.09060

[26] Shu Liu, Asim Biswal, Audrey Cheng, Xiangxi Mo, Shiyi Cao, Joseph E. Gonzalez, Ion Stoica, and Matei Zaharia. 2024. Optimizing LLM Queries in Relational Workloads. *CoRR* abs/2403.05821 (2024). https://doi.org/10.48550/ARXIV.2403.05821 arXiv:2403.05821

[27] Andres Marzal and Enrique Vidal. 1993. Computation of normalized edit distance and applications. *IEEE transactions on pattern analysis and machine intelligence* 15, 9 (1993), 926–932.

[28] OpenAI. 2024. GPT-4 Technical Report. (2024). arXiv:cs.CL/2303.08774 https://arxiv.org/abs/2303.08774

[29] George Papadakis, Ekaterini Ioannou, Emanouil Thanos, and Themis Palpanas. 2021. *The four generations of entity resolution.* Springer.

[30] Simone Papicchio, Paolo Papotti, and Luca Cagliero. 2023. QATCH: Benchmarking Table Representation Learning Models on Your Data. In *NeurIPS (Datasets and Benchmarks).*

[31] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. 2024. LOTUS: Enabling Semantic Queries with LLMs Over Tables of Unstructured and Structured Data. *CoRR* abs/2407.11418 (2024). https://doi.org/10.48550/ARXIV.2407.11418 arXiv:2407.11418

[32] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language Models as Knowledge Bases?. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 2463–2473. https://doi.org/10.18653/v1/D19-1250

[33] Abdul Quamar, Vasilis Efthymiou, Chuan Lei, and Fatma Özcan. 2022. Natural Language Interfaces to Data. *Found. Trends Databases* 11, 4 (2022), 319–414. https://doi.org/10.1561/1900000078

[34] Mohammed Saeed, Nicola De Cao, and Paolo Papotti. 2024. Querying Large Language Models with SQL. In *Proceedings 27th International Conference on*

Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28, Letizia Tanca, Qiong Luo, Giuseppe Polese, Loredana Caruccio, Xavier Oriol, and Donatella Firmani (Eds.). OpenProceedings.org, 365–372. https://doi.org/10.48786/EDBT.2024.32

[35] Giovanni Simonini, Luca Zecchini, Sonia Bergamaschi, Felix Naumann, et al. 2022. Entity resolution on-demand. *Proceedings of the VLDB Endowment* 15, 7 (2022), 1506–1518.

[36] Kai Sun, Yifan Ethan Xu, Hanwen Zha, Yue Liu, and Xin Luna Dong. 2024. Head-to-Tail: How Knowledgeable are Large Language Models (LLMs)? A.K.A. Will LLMs Replace Knowledge Graphs? (2024). arXiv:cs.CL/2308.10168 https://arxiv.org/abs/2308.10168

[37] James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Y. Levy. 2021. From Natural Language Processing to Neural Databases. *Proc. VLDB Endow.* 14, 6 (2021), 1033–1039.

[38] Immanuel Trummer. 2024. DB-BERT: making database tuning tools "read" the manual. *VLDB J.* 33, 4 (2024), 1085–1104. https://doi.org/10.1007/S00778-023-00831-Y

[39] Matthias Urban and Carsten Binnig. 2024. CAESURA: Language Models as Multi-Modal Query Planners. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*. www.cidrdb.org. https://www.cidrdb.org/cidr2024/papers/p14-urban.pdf

[40] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 24824–24837. https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[41] Xueqing Wu, Jiacheng Zhang, and Hang Li. 2022. Text-to-Table: A New Way of Information Extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, 2518–2533. https://doi.org/10.18653/V1/2022.ACL-LONG.180

[42] Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. 2024. Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

[43] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP*. Association for Computational Linguistics, 3911–3921. https://doi.org/10.18653/v1/D18-1425

[44] Ce Zhang, Christopher Ré, Michael J. Cafarella, Jaeho Shin, Feiran Wang, and Sen Wu. 2017. DeepDive: declarative knowledge base construction. *Commun. ACM* 60, 5 (2017), 93–102. https://doi.org/10.1145/3060586

[45] Fuheng Zhao, Divyakant Agrawal, and Amr El Abbadi. 2024. Hybrid Querying Over Relational Databases and Large Language Models. (2024). arXiv:cs.DB/2408.00884 https://arxiv.org/abs/2408.00884

[46] Fuheng Zhao, Lawrence Lim, Ishtiyaque Ahmad, Divyakant Agrawal, and Amr El Abbadi. 2024. LLM-SQL-Solver: Can LLMs Determine SQL Equivalence? (2024). arXiv:cs.DB/2312.10321 https://arxiv.org/abs/2312.10321