Logical and Physical Optimizations for SQL Query Execution over Large Language Models

DARIO SATRIANI, University of Basilicata, Italy
ENZO VELTRI, University of Basilicata, Italy
DONATELLO SANTORO, University of Basilicata, Italy
SARA ROSATO, EURECOM, France
SIMONE VARRIALE, EURECOM, France
PAOLO PAPOTTI, EURECOM, France

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 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.

CCS Concepts: • **Information systems** → **Data management systems**; **Middleware for databases**; • **Computing methodologies** → *Natural language processing*.

Additional Key Words and Phrases: Large Language Models, SQL, Query Optimization

Authors' Contact Information: Dario Satriani, dario.satriani@unibas.it, University of Basilicata, Potenza, Italy; Enzo Veltri, enzo.veltri@unibas.it, University of Basilicata, Potenza, Italy; Donatello Santoro, donatello.santoro@unibas.it, University of Basilicata, Potenza, Italy; Sara Rosato, rosato@eurecom.fr, EURECOM, Biot, France; Simone Varriale, varriale@eurecom.fr, EURECOM, Biot, France; Paolo Papotti, paolo.papotti@eurecom.fr, EURECOM, Biot, France.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 2836-6573/2025/6-ART181
https://doi.org/10.1145/3725411

ACM Reference Format:
Dario Satriani, Enzo Veltri, Donatello Santoro, Sara Rosato, Simone Varriale, and Paolo Papotti. 2025. Logical and Physical Optimizations for SQL Query Execution over Large Language Models. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 181 (June 2025), 28 pages. https://doi.org/10.1145/3725411

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 [3, 36, 37, 53, 60]. 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 [7]. These models offer a sophisticated alternative to traditional extraction methods, which either require data annotation efforts [15] or manually crafted extraction pipelines [69]. These solutions are typically static and can only extract fixed sets of attributes, leaving unsolved the automatic extraction of relational data [1].

By directly querying the internal representations within LLMs, users can extract meaningful data without the need for complex, manual parsing processes [3]. This has opened avenues where developers issue natural language (NL) or SQL-like queries, thus integrating LLMs into mainstream data retrieval processes [30, 58]. We distinguish two use cases for querying LLMs.

The first use case focuses on the ability to obtain structured data from the *parametric knowledge* in the LLMs [53]. One application is to obtain structured data from the LLM for auditing its biases [5], e.g., a query that extracts a table with the "best" city and "most important" person in all countries can be used for analysis of the LLM cultural bias [42]. Precisely measuring query results over known facts also enables systematic benchmarking of new LLMs for their factuality [12]. Finally, queries can populate questionnaires to reflect the human behavior as modeled by the LLM from input documents [47].

The second use case focuses on structured data extraction from text documents fed in the LLM input, the *in-context learning* setting [36]. For example, tables are derived from the financial reports or health records that are passed as input in the LLM's context entirely [37], or in smaller chunks, as in a RAG setting [32].

In both scenarios, LLMs are becoming indispensable tools in tasks where interpreting and extracting data from extensive text corpora are essential, streamlining access to information captured within the neural fabric of these models [60, 70].

The Problem with Data Outputs. While NL question answering can gather the knowledge embedded within pre-trained language models or in documents provided at runtime, complex NL questions that aim at obtaining data outputs often challenge these models, leading to inaccuracies and missing data in the response [50].

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

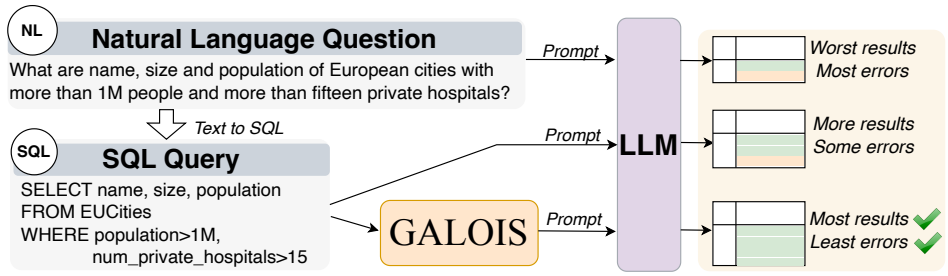

Fig. 1. Comparison of query methods for obtaining data on European cites with specific criteria from the information embedded in the LLM. Galois executes SQL queries over LLMs with logical and physical optimizations, achieving superior accuracy and completeness over direct natural language and SQL prompts.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

queries over several topics, both over the LLM internal knowledge and over documents passed in the prompt (in-context setting). Direct NL prompting yields the least accurate results. Since NL questions can be translated into SQL queries, either manually or with the assistance of text-to-SQL techniques [30], we assume SQL queries as input in the following. However, while direct SQL query prompting improves the quality, it still falls 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 reduces 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 derived 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 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 for interfacing with LLMs, which balances the efficiency and

[1]Évariste Galois (rhymes with French word *voilà*) was a 19^{th} century mathematician.

accuracy of data retrieval. Second, we introduce dynamic methods for acquiring essential metadata during query execution, focusing on estimating 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, while maintaining a competitive edge in terms of resource efficiency and execution speed[2]. Our approach achieves up to 144% quality improvement over NL questions and 29% over direct SQL, while the more competitive baselines are 11 times more expensive in terms of tokens consumed, both in querying the LLM's parametric knowledge and in the in-context learning setting. Results also show that the quality of the answers is bounded by the limitations of the underlying LLM, i.e., its ability in understanding complex input text. Given the evolving capabilities of LLMs, we believe our findings underscore the potential of integrating database management solutions with LLM technology, paving the way for sophisticated and efficient data querying practices.

2 Problem Formulation and Challenges

The core problem involves executing SQL queries that cannot be answered on the existing 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 new setting includes both the challenges in a traditional DBMS (efficiency, here in terms of consumed tokens) and those that are specific to LLMs (quality of the generated output).

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

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 pushdown*. 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. 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**).

[2]Code and data available at https://github.com/dbunibas/galois

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

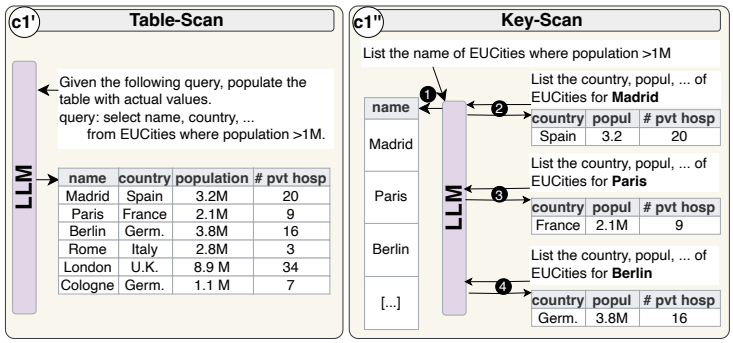

Fig. 3. Two alternative physical operators for implementing logical operator **c1** in Figure 2. Table-scan (**c1'**) gets entire tuples, while Key-Scan (**c1"**) gets key values first.

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.

Preliminaries. We focus on queries with the following structure:

```
SELECT ((attr | agg(attr))+)
FROM ST+
[WHERE predicate+]
[GROUP BY (attr+)]
[HAVING (attr+)]
[ORDER BY (attr+)]
[LIMIT X]
```

Where `attr` is an attribute, `agg` is an aggregative function (min, max, avg, sum, count), `ST` refers to one or more tables that could be joined, `predicate` is a conjunction or disjunction of atoms with comparators =, >(=), <(=). Our system does not use fixed templates and it does not make assumptions on the number of attributes in the Select and Where clauses. While the system can be extended to support more operators, we believe the current subset suffices to show the potential of the approach.

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 with the LLM are the *LLMScan* operator, and its variant *Filter-LLMScan*. Once the data for a relation is retrieved with a Scan operator, the other operators are optimized and executed without involving the LLM.

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 LLMScan operator fetches data on the fly

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

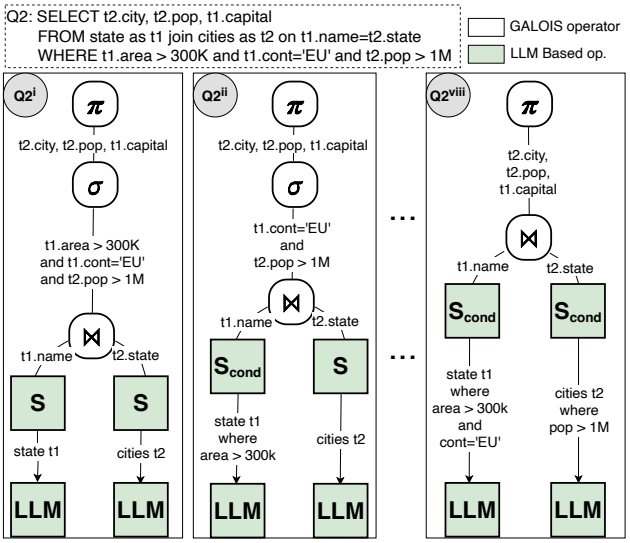

Fig. 4. Different Logical Plans for query $Q2$. The first plan does not use any pushdown of the conditions in $\mathcal{S}$(LLM). The second and the third plans uses $\mathcal{S}_{cond}$(LLM) with a single condition, while the last uses $\mathcal{S}_{cond}$(LLM) with both conditions.

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.

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 getting into the optimization strategies to select the most efficient query plan, we first elaborate on the implementation of the scan operators.

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 LLMScan operator p. In the initial interaction, we prompt the LLM with 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. In case of an incomplete JSON response (i.e., missing closing parenthesis), we use a best-effort approach to repair the JSON structure and parse it; otherwise, we use LLM prompts to improve model responses using feedback [49]. We skip further invalid JSON responses. 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. Prompt with feedback is not reported due to space limit.

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 the generated questions and responses). We then prompt the LLM to provide additional data, if any exists, using the iterative prompt 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

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;
```

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.

Fig. 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.

are valid against the JSON schema of t_{name}. $noNewTuples$ 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 [63] 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.

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 ∈ 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;
```

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 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. We use the same JSON response management as Table-Scan. 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.

Key-Scan may be seen as an approximation of an index scan in traditional databases. However, there are several differences. First, the LLM-based Key-Scan does not rely on predefined index structures, but it dynamically retrieves key and tuples at inference time based only on the contextual understanding of the LLM. Second, given the LLM context, Key-Scan does not provide the guarantees

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.

Fig. 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.

offered by deterministic index scans. Finally, rather than improving efficiency, the Key-Scan aims at improving results quality.

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 $keys$.

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 [37]. 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 LLMScan. 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 processing complexity for 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 [25]. 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.

With open LLMs, an alternative approach to optimize logical plans involves post-execution analysis that estimates the confidence of the model from its final layer [61]. This method entails executing all potential plans to analyze the outputs and determine the best pushdown. While post-execution may give better confidence estimates, it is not reasonable in terms of cost to execute every possible plan. Consequently, we prioritize pre-execution confidence estimation over attributes during catalog construction. This approach allows us to preemptively select the most promising logical plans based on the LLM's confidence feedback, thus achieving an effective balance between cost and result 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 again 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 [66]. 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.

Cost 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 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 [44] and LLaMa3 [18]. 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.

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

Dataset name	Dataset source	# of queries	Avg. expected cells	Type
Flight	Spider [68]	6	267.5	IK
Geo	Spider [68]	32	22.8	IK
World	Spider [68]	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 [68]	10	24.1	IK

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.

We divide the datasets into those that query the *internal parametric knowledge* (IK) 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* (MC).

For the first group, the IK scenario, we selected five datasets containing factual information, with pre-existing ground truth. Three come from the Spider [68] corpus. Each example consists of an NL question, the expected SQL query, and the tables with data. We exclude from our evaluation queries related to data contained only in Spider. Movies is 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 NL questions and SQL queries.

In the second group (MC), 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 in evaluating Galois's performance within the context of applications such as RAG, where information external to the LLM's knowledge is retrieved.

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 [6, 46]:

- 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 [43] 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 [9, 45, 55] or tuples matching [24]. 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.

Both values are part of Galois's output for every executed query.

Baselines. We consider four baselines :

- **NL**. Directly prompting the LLM with a natural language question and getting structured data as a response.
- **SQL**. Directly prompting the LLM using a SQL query and getting structured data as a response.
- **Galois**$_{WO}$ [53] (Without Optimizations). Querying LLMs with a reasoning multi-step approach with a traditional DBMS and specialized LLM operators. Galois$_{WO}$ first retrieves all the keys (without any push-down optimization) and then it populates the tuple cell by cell.
- **Palimpzest** (PZ) [37] is a declarative system designed to execute AI workloads; it is used for experiments involving in-context querying.

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

Exp-1. Overall Evaluation. In this experiment, we obtain structured data from the internal parametric knowledge of the LLM. We evaluate the performance of the variants of Galois against the NL, SQL, and Galois 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.

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

Metric	NL	SQL	Galois $_{WO}$	Galois $_{S}$	Galois $_{A}$	Galois $_{F}$
F1-Cell	0.237	0.431	0.518	0.480	*0.543*	**0.563**
Cardinality	0.462	0.659	0.691	0.655	*0.799*	**0.835**
Tuple Constr.	0.065	0.351	0.389	0.365	*0.448*	**0.464**
AVG-Score	0.254	0.481	0.531	0.500	*0.592*	**0.622**
#Tokens (M)	*0.83*	**0.33**	19.71	0.96	0.95	1.72
Avg Time	120	*61.4*	1460	130	120.5	**47.4**

In the results for the quality metrics shown in Table 3, NL exhibits lower performance due to the inherent ambiguity of natural language, which can lead to misinterpretation by the LLM. This is evidenced by the higher rate of hallucinations and data repetitions observed in the NL approach, as reflected in the low Cardinality. Even when the LLM gets the query's intent, with the NL approach it struggles to express the extracted information in a structured format, leading to incomplete or poorly structured tuples, as indicated by the low Tuple Constraints.

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 logical operators with Galois $_{WO}$, Galois $_{S}$ and Galois $_{A}$ improves the quality w.r.t. NL and SQL. However, traditional DBMS optimizations (represented by Galois $_{S}$ and Galois $_{A}$) do not lead to the best quality results. i.e., pushing down the most selective attribute (Galois $_{S}$) leads to suboptimal performance. By prioritizing the most selective attribute, the LLM may operate on 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%, 29%, and 17% AVG-Score improvement w.r.t. NL, SQL, and Galois $_{WO}$, respectively.

From the point of view of the total costs expressed in # TOKENS, Galois $_{WO}$ has the highest cost due to the retrieval of the values for the tuples that requires an LLM request for every cell. The cheapest solution in terms of produced tokens is SQL, while our system requires more tokens in the multiple steps to execute its plans. 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 Galois $_{F}$ obtains the highest quality.

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, we do not report results for Galois $_{S}$ since it shows lower quality than 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

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 parameters. We execute Galois over the same datasets used for Exp-1. To compare the different models we use the AVG-Score.

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

LLM	NL	SQL	Galois $_{WO}$	Galois $_A$	Galois $_F$
GPT-4o mini	0.258	0.240	0.456	0.457	**0.468**
LLaMa 3.1 8B	0.230	0.372	0.375	0.520	**0.528**
LLaMa 3.1 70B	0.254	0.481	0.531	0.592	**0.622**

Results in Table 5 show that GPT-4o Mini and LLaMa 3.1 8B have comparable results. Increasing the size of the parameters to 70B improves the overall quality for two reasons. The first is that

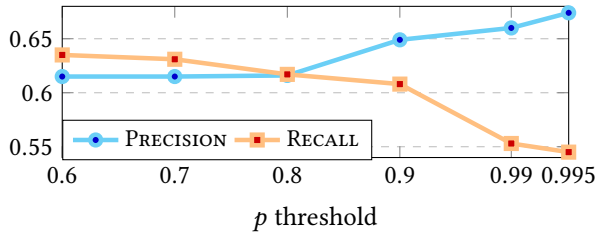

Fig. 7. Precision and Recall for Galois when filtering out results varying the threshold on the LLM output logprob.

models with fewer parameters store less factual data than larger models. The second is related to the complexity of the prompt, when Galois variants push down multiple conditions, smaller models 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. As LLaMa 3.1 70B returns the best results, we use it in the following experiments.

Open LLMs also enable the analysis of the output of the model to estimate its confidence over the results. A possible approach is to exploit the log probabilities (logprobs) returned by the LLM for each generated token, i.e., the natural logarithm of the softmax output of the model's final layer [61]. For each tuple, we compute the mean of the probability of the tokens associated with each cell [21]. We then set a threshold p to filter out the tuples with mean logprob below it. Figure 7 reports the average Precision and Recall metrics (as in the F1-Cell) for the datasets from Exp-1. As expected, increasing p improves the precision while the recall decreases.

Exp-4. Impact of retrieved values. While in a DBMS the quality does not depend on the queries, the values involved in the queries play a role with LLMs. We show the impact of the values over the same queries, i.e., same logical plan except for the values in the conditions. Table Presidents contains data about world presidents. We collect 13 queries about USA's presidents and the same 13 queries about Venezuela's 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 for USA goes from 1789 till today. We split the queries into three categories: All-Time covering all dates, Recent for queries that cover data from late 1900 to today, and Past for queries for 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 $_{WO}$	Galois $_{A}$	Galois $_{F}$
Presidents-Usa	0.263	0.546	0.733	0.782	**0.862**
Presidents-Venezuela	0.203	0.285	0.411	0.425	**0.482**

Table 6 reports the quality in terms of AVG-Score. 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 in terms of AVG-Score in comparing the temporal values. In bold best results.

Dataset	NL	SQL	Galois $_{WO}$	Galois $_{A}$	Galois $_{F}$
Recent	0.209	0.398	0.531	0.584	**0.623**
Past	0.171	0.305	0.469	0.518	**0.548**
All-Time	0.325	0.562	0.703	0.722	**0.857**

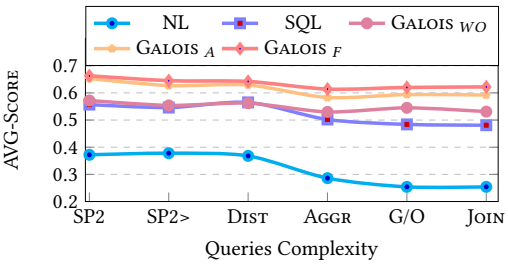

Fig. 8. Result quality w.r.t query complexity.

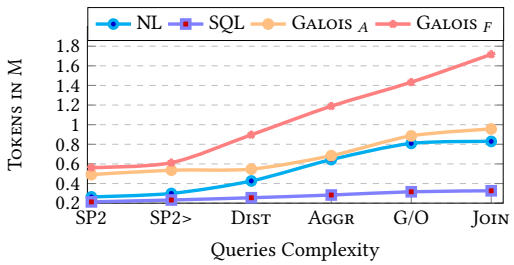

Fig. 9. Costs w.r.t query complexity. Galois $_{WO}$ (not reported) consumes 8.5M to 19.7M tokens.

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 denotes queries with Selection and Projection with at most two conditions, SP2> those with more than two conditions, Dist queries with DISTINCT, Aggr queries with aggregate functions, G/O queries with group by or order by, and Join queries with joins. Figure 8 reports how AVG-Score is impacted by the complexity of the queries for all systems. Increasing the complexity of the queries decreases the quality. With aggregate functions, group/order by and joins, the LLMScan must retrieve all the relevant data to get the correct results. This classes of queries are harder because even a single incorrect or missing value leads to a mismatch with the ground truth.

Table 8. Quality results in terms of AVG-Score with different selection conditions in the WHERE clause. In bold (italic) the best (2^{nd} best) result for each scenario.

Condition(s)	NL	SQL	Galois $_{WO}$	Galois $_{A}$	Galois $_{F}$
1 textual	0.319	0.600	0.566	*0.674*	**0.699**
>1 textual	0.283	0.565	0.577	*0.647*	**0.695**
>1 text., 1 numer.	0.264	0.527	0.528	*0.619*	**0.633**
>1 text., >1 numer.	0.222	0.384	0.479	*0.530*	**0.539**
1 numerical	0.260	**0.545**	0.486	0.500	*0.517*
>1 numerical	0.223	0.455	**0.532**	0.459	*0.512*

Results in Fig. 8 show that Galois is very robust w.r.t. the complexity of the queries. This is expected as its output quality depends only on the retrieved values in the LLMScan. To surface what

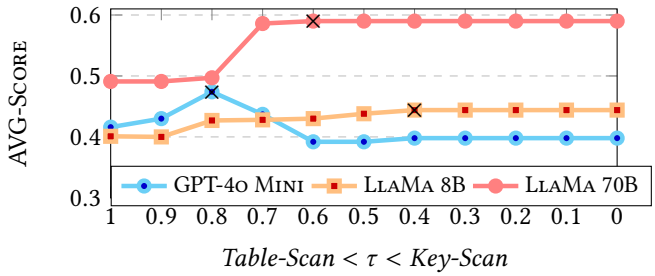

Fig. 10. τ selection, $\times$ marks the optimal τ for each LLM.

affects the quality of the scan operator, we focus on the impact of the conditions in the WHERE clause. We categorize the queries by the type (textual vs numerical) and number of conditions. Results in Table 8 show that textual conditions are easier for all approaches. Galois variants outperform all baselines when a textual condition is present and Galois $_F$ has the second highest quality in cases with numerical conditions only.

Takeaways for question (3): Galois on LLaMa 3.1 70B scores higher than on smaller models for its ability to store more factual data and execute complex pushdown operations. This suggests Galois's potential to further improve outcomes alongside the ongoing release of new LLMs. The quality of results is impacted by both the popularity and the temporal relevance of query values, while results are stable w.r.t. the complexity of SQL scripts.

Galois's top performance in terms of quality come with a cost in terms of tokens consumed by the LLM. Results in Figure 9 show that Galois collects more data from the LLM to get an answer compared to NL and SQL. However, we do not report Galois consumption in the figure, as it consumes ≈11x the amount of tokens of Galois.

Takeaways for question (4): Galois has higher token costs due to its multi-step execution plans compared to the straightforward SQL approach, while is cheaper w.r.t. the multi-step baseline Galois. 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. Evolution of Prompt Size. The goal of this experiment is to measure the impact of the increasing size of the prompt across iterations in Galois. Our iterative Scan operator incrementally increases the prompt passed in the LLM context during execution, as at each iteration we pass the tuples (ids) previously retrieved. To measure the benefit of our proposal, we include a Galois's unoptimized variant, Galois $_\emptyset$, which uses Table-Scan without push-down optimizations. As metrics, we measure the average total number of input tokens for all iterations per query, the average number of iteration per query, and the number of queries that go over 10 iterations to collect the data.

Table 9. Prompt size without push-down (Galois $_\emptyset$) and with Galois's optimizations.

	Galois $_\emptyset$	Galois $_A$	Galois $_F$
Avg # Input Tokens All Iters	63405	5137	2930
Avg # Iters per query	6.82	4.22	3.92
# Queries with 10+ Iters	37	1	4

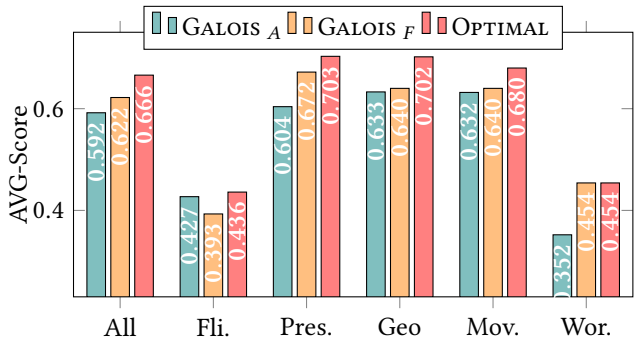

Fig. 11. AVG-Score Comparison for Galois $_A$ and Galois $_F$ against the optimal plans across all datasets.

Table 9 report the results for all queries from Exp-1 with the three Galois variants. Results show that push-downs significantly reduce the size of the prompt across iterations. Both Galois $_A$ and Galois $_F$ require a much smaller context compared to Galois $_\emptyset$. Notably, optimized version of Galois reach the number of iterations set by the *maxIter* parameter (10 in prior experiments) for only five queries, in contrast with the 37 ones of Galois $_\emptyset$. For these five queries, we experimentally evaluate higher *maxIter* values and observe diminishing returns, with an improvement in AVG-Score of 0.06% at $maxIter = 20$ and an increment of consumed #TOKENS of 24%.

Exp-7. 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 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 - it does not contain any query in the test datasets. Figure 10 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. Such thresholds depend only on the LLM and are consistent across various tables, datasets, and submitted queries.

Exp-8. Errors w.r.t. the optimal plan. For each dataset and for each query, we generate all the possible logical and execution plans to identify the Optimal plan, i.e., the one with the highest AVG-Score. Results in Figure 11 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 if we consider 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 the World dataset, Galois produced the optimal plan for all its queries.

5.3 In-Context Querying

In the previous experiments, we used the LLM to answer queries based on its inherent internal knowledge. However, our system extends beyond this. It can be seamlessly integrated with other frameworks, such as Retrieval Augmented Generation (RAG), which combines the strengths of traditional information retrieval systems with the generative capabilities of LLMs. This experiment

aims to demonstrate that Galois's optimizations significantly improve the accuracy and efficiency of LLM-driven data retrieval also in the setting of in-context learning.

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.

Each document is divided into text segments with a size of 128 tokens for Premier and 400 tokens for Fortune. Segments are encoded using the "WhereIsAI/UAE-Large-V1" model [34] and stored in a vector database. The prompt is then fed at runtime with the query schema and 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. For these experiments, we compare our system also against Palimpzest [37], which shares our data extraction use case, but it differs significantly in design and execution. While our system uses a declarative SQL-based interface to define operations, Palimpzest adopts an ETL-like, procedural approach, requiring users to explicitly define data transformations and model invocations as a sequence of function calls in Python. This distinction has practical implications. Palimpzest provides granular control over individual processing steps, which can be advantageous in specialized use cases but comes at the cost of increased user effort and complexity. In addition, it also offers advanced features such as support for extracting and processing diverse documents. For the purpose of this comparison, all queries in our experiments have been rewritten using Palimpzest's API, using functions such as *filter*, *convert*, and *execute*, to implement the SQL execution; we use its optimization policy *MaxQualityAtFixedCost*, as suggested by the authors.

Table 10. Quality Results and Costs for RAG application over Premier and Fortune datasets. In bold the best results.

Metric	NL	SQL	Galois $_A$	Galois $_F$	PZ
AVG-Score	0.389	0.520	0.628	0.711	**0.720**
# TOKENS in M	**1.448**	1.625	1.478	1.598	13.818

Results are presented in Table 10. NL and SQL exhibit again 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 perform better than NL and SQL, with Galois $_F$ achieving higher quality scores. Palimpzest achieves the highest quality performance, although very close to Galois $_F$, but with a cost that is approximately 11 times higher.

Analyzing the token count reveals that Palimpzest exhibits the highest costs because each document is processed through multiple steps, requiring repeated interactions with LLMs. In contrast, in Galois only the scan operator processes text (once per query execution) and scan is the only operator that interacts with the LLM - this design significantly reduces the overall computational cost.

Takeaways for question (5): Galois integrates effectively with in-context learning like RAG, delivering comparable quality performance to the best baseline while achieving significant token savings. Moreover, Galois only requires users to write SQL scripts.

6 Related Work

From NL Questions to SQL Scripts. NL questions are popular as users seek intuitive ways to interact with systems [33, 51]. With the rapid advancements in text-to-SQL technologies, these NL queries can be transformed into SQL scripts [6, 30, 62]. Despite these advancements, Galois assumes SQL scripts as input, recognizing their superior expressiveness and precision in specifying user needs.

Structured Data Extraction. A common approach to manage unstructured data involves extracting it into tabular form for subsequent querying. This method is employed by IE extractors [15, 69] and text-to-table systems [65]. 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 extract a table where each document corresponds to a single row [3]. Other systems derive extraction scripts to enable effective and efficient table population [36]. Our proposal can be adopted in these solutions.

Unlike IE methods, LLMs can synthesize information not explicitly covered in the input documents [8]. For example, an LLM might deduce that A is the grandchild of C from texts outlining parent relationships between (A, B) and (B, C). However, this ability also brings a risk of inaccuracies as LLMs do not have a mechanism for recognizing when they lack the necessary information [5].

LLMs and RAG. LLMs are gaining significant attention for their use in data querying and retrieval tasks [27]. RAG techniques enhance LLM capabilities by integrating retrieval mechanisms to pinpoint relevant data segments, better accommodating them within LLM constraints in terms of context windows [19, 32]. 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 or proprietary content. Our method enhances the LLM's 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 [13, 17, 31, 38, 59, 71], 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 [14, 29, 40, 53, 58, 60, 70]. As these systems use declarative querying primitives to process data with LLMs, our contributions are orthogonal to these solutions. The optimization techniques in Galois can be integrated with such frameworks, providing improved efficiency in query execution.

Other recent declarative systems optimize AI-powered analytical queries by balancing runtime, cost, and data quality [37, 48, 54]. While they optimizes a range of tasks involving unstructured and structured data, most of them take an ETL-like approach, therefore our work can integrate into such frameworks to enhance their efficiency, e.g., by dynamically acquiring metadata during query execution for cost-effective query optimization.

Factual Knowledge in LLMs. While LLMs encapsulate extensive information, their accuracy, particularly concerning less predominant facts, is inconsistent [57]. Additionally, LLMs can present overconfident responses even when uncertain, a challenge partly addressed through recalibrating confidence scores [10]. Question answering over the LLM pre-training information encounters the same challenges and it also benefits from advancements in confidence alignment and model factuality.

Query Optimization. Our approach can be extended to adopt techniques in optimizing plans in the presence of opaque user-defined function predicates (UDFs), e.g., when a predicate cardinality is unknown [26]. More ideas can be taken from the optimization of complex dataflows with UDFs [2, 52]. In the lens of optimizing filters over sampling operators, LLM inference can be seen as sampling from an infinite database, with parallels in sampling-based query optimization techniques [64]. These connections present an opportunity to refine our operators and optimization strategies further, ensuring more robust and efficient query processing.

Crowdsourcing. In querying LLMs, there are analogies with DBMS extensions involving crowd workers to answer open world questions [22]. Techniques proposed for crowd databases, such as redundancy and validation questions, are also studied to improve the quality of LLM's responses and could be adopted in our setting [35].

7 Conclusion

In this work, we study the problem of querying LLMs through SQL queries. Our system, Galois, acts as an intermediary between the user and the LLM, extending traditional 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. This direction opens several problems, such as the design of mechanisms to simulate index-like efficiency using LLMs, e.g., through caching techniques based on prior interactions [67]. Alternatively, an LLM-first architecture poses intriguing possibilities with new challenges. One question is whether LLMs can replace DBMSs by ingesting structured data during training or in-context. While research in tabular language models indicates that such a scenario is not yet feasible, primarily due to context size limitations [4] and its issues with long inputs [39], recent advancements are overcoming this constraint [16, 28, 41].

Another promising research direction involves support for queries spanning multiple modalities, such as text, image, and structured data [60]. This integration could enable users to extract insights from diverse data formats in a unified querying framework [37].

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 "private hospitals", the LLM may return the list of US hospitals first. 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 [20]. This issue also suggests that extracting structured data from LLMs offers promising applications, such as auditing biases and benchmarking factual accuracy, by enabling systematic analysis of cultural perspectives and factual consistency.

Our work also shows the increasing need to refine LLM confidence estimation mechanisms [11, 23]. Enhanced confidence assessments can lead to more reliable outputs, informing users of the certainty associated with query responses by investigating how to incorporate the estimated confidence from open LLMs. Finally, allowing LLMs to dynamically allocate test-time compute could significantly enhance their performance on challenging prompts [56].

Acknowledgments

Veltri was partially supported by the TECH4YOU project (CUP: C43C22000400006). Papotti was partially supported by the ANR project ATTENTION (ANR-21-CE23-0037).

References

[1] Daniel Abadi, Anastasia Ailamaki, David Andersen, Peter Bailis, Magdalena Balazinska, Philip A. Bernstein, Peter Boncz, Surajit Chaudhuri, Alvin Cheung, Anhai Doan, Luna Dong, Michael J. Franklin, Juliana Freire, Alon Halevy, Joseph M. Hellerstein, Stratos Idreos, Donald Kossmann, Tim Kraska, Sailesh Krishnamurthy, Volker Markl, Sergey

Melnik, Tova Milo, C. Mohan, Thomas Neumann, Beng Chin Ooi, Fatma Ozcan, Jignesh Patel, Andrew Pavlo, Raluca Popa, Raghu Ramakrishnan, Christopher Re, Michael Stonebraker, and Dan Suciu. 2022. The Seattle Report on Database Research. *Commun. ACM* 65, 8 (jul 2022), 72–79. doi:10.1145/3524284

[2] Divy Agrawal, Sanjay Chawla, Bertty Contreras-Rojas, Ahmed K. Elmagarmid, Yasser Idris, Zoi Kaoudi, Sebastian Kruse, Ji Lucas, Essam Mansour, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, Saravanan Thirumuruganathan, and Anis Troudi. 2018. RHEEM: Enabling Cross-Platform Data Processing - May The Big Data Be With You! -. *Proc. VLDB Endow.* 11, 11 (2018), 1414–1427. doi:10.14778/3236187.3236195

[3] 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

[4] 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. doi:doi.org/10.1162/tacl_a_00544

[5] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?. In *FAccT*. ACM, 610–623. doi:10.1145/3442188.3445922

[6] 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. arXiv:2408.14717 [cs.DB] https://arxiv.org/abs/2408.14717

[7] 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

[8] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712* (2023).

[9] Marcello Buoncristiano, Giansalvatore Mecca, Donatello Santoro, and Enzo Veltri. 2024. Detective Gadget: Generic Iterative Entity Resolution over Dirty Data. *Data* (2024). doi:10.3390/data9120139

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

[11] 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).

[12] shiqi chen, Yiran Zhao, Jinghan Zhang, I-Chun Chern, Siyang Gao, Pengfei Liu, and Junxian He. 2023. FELM: Benchmarking Factuality Evaluation of Large Language Models. In *Advances in Neural Information Processing Systems*, Vol. 36. Curran Associates, Inc., 44502–44523. https://proceedings.neurips.cc/paper_files/paper/2023/file/8b8a7960d343e023a6a0afe37eee6022-Paper-Datasets_and_Benchmarks.pdf

[13] 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. arXiv:2310.00749 [cs.DB] https://arxiv.org/abs/2310.00749

[14] 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

[15] 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. doi:10.18653/v1/N18-3010

[16] Giulio Corallo and Paolo Papotti. 2024. FINCH: Prompt-guided Key-Value Cache Compression for Large Language Models. *Trans. Assoc. Comput. Linguistics* 12 (2024), 1517–1532. doi:10.1162/TACL_A_00716

[17] 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. doi:10.5555/3430915.3442430

[18] 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).

[19] 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.

[20] 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

[21] Marina Fomicheva, Shuo Sun, Lisa Yankovskaya, Frédéric Blain, Francisco Guzmán, Mark Fishel, Nikolaos Aletras, Vishrav Chaudhary, and Lucia Specia. 2020. Unsupervised Quality Estimation for Neural Machine Translation. *Transactions of the Association for Computational Linguistics* 8 (09 2020), 539–555. doi:10.1162/tacl_a_00330

[22] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. 2011. CrowdDB: Answering Queries with Crowdsourcing. In *ACM SIGMOD*. 61–72. doi:10.1145/1989323.1989331

[23] 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.

[24] Boris Glavic, Giansalvatore Mecca, Renée J. Miller, Paolo Papotti, Donatello Santoro, and Enzo Veltri. 2024. Similarity Measures For Incomplete Database Instances. 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, 461–473. doi:10.48786/EDBT.2024.40

[25] 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. doi:10.18653/v1/2023.findings-emnlp.647

[26] Wenjia He, Michael R. Anderson, Maxwell Strome, and Michael Cafarella. 2020. A Method for Optimizing Opaque Filter Queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1257–1272. doi:10.1145/3318464.3389766

[27] Xingyu Ji, Aditya Parameswaran, and Madelon Hulsebos. [n. d.]. TARGET: Benchmarking Table Retrieval for Generative Tasks. In *NeurIPS 2024 Third Table Representation Learning Workshop*.

[28] 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. doi:10.18653/v1/2023.emnlp-main.825

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

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

[31] 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

[32] 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

[33] 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

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

[35] Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring How Models Mimic Human Falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Dublin, Ireland, 3214–3252. doi:10.18653/v1/2022.acl-long.229

[36] 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. arXiv:2405.04674 [cs.DB] https://arxiv.org/abs/2405.04674

[37] 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. arXiv:2405.14696 [cs.CL] https://arxiv.org/abs/2405.14696

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

[39] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173. doi:10.1162/tacl_a_00638

[40] 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). doi:10.48550/ARXIV.2403.05821 arXiv:2403.05821

[41] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2024. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. In *Proceedings of the ACM SIGCOMM 2024 Conference* (Sydney, NSW, Australia) *(ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 38–56. doi:10.1145/3651890.3672274

[42] Zhaoming Liu. 2024. Cultural Bias in Large Language Models: A Comprehensive Analysis and Mitigation Strategies. *Journal of Transcultural Communication* (2024). doi:doi:10.1515/jtc-2023-0019

[43] 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.

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

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

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

[47] Joon Sung Park, Carolyn Q Zou, Aaron Shaw, Benjamin Mako Hill, Carrie Cai, Meredith Ringel Morris, Robb Willer, Percy Liang, and Michael S Bernstein. 2024. Generative agent simulations of 1,000 people. *arXiv preprint arXiv:2411.10109* (2024).

[48] 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). doi:10.48550/ARXIV.2407.11418 arXiv:2407.11418

[49] Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and Jianfeng Gao. 2023. Check Your Facts and Try Again: Improving Large Language Models with External Knowledge and Automated Feedback. *CoRR* abs/2302.12813 (2023). doi:10.48550/ARXIV.2302.12813 arXiv:2302.12813

[50] 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. doi:10.18653/v1/D19-1250

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

[52] Astrid Rheinländer, Ulf Leser, and Goetz Graefe. 2017. Optimization of Complex Dataflows with User-Defined Functions. *ACM Comput. Surv.* 50, 3, Article 38 (May 2017), 39 pages. doi:10.1145/3078752

[53] 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*. OpenProceedings.org, 365–372. doi:10.48786/EDBT.2024.32

[54] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2024. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. arXiv:2410.12189 [cs.DB] https://arxiv.org/abs/2410.12189

[55] 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.

[56] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314* (2024).

[57] 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? arXiv:2308.10168 [cs.CL] https://arxiv.org/abs/2308.10168

[58] 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.

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

[60] 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

[61] Roman Vashurin, Ekaterina Fadeeva, Artem Vazhentsev, Akim Tsvigun, Daniil Vasilev, Rui Xing, Abdelrahman Boda Sadallah, Lyudmila Rvanova, Sergey Petrakov, Alexander Panchenko, Timothy Baldwin, Preslav Nakov, Maxim Panov, and Artem Shelmanov. 2024. Benchmarking Uncertainty Quantification Methods for Large Language Models with LM-Polygraph. *CoRR* abs/2406.15627 (2024). doi:10.48550/ARXIV.2406.15627 arXiv:2406.15627

[62] Enzo Veltri, Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Data Ambiguity Profiling for the Generation of Training Examples. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 450–463. doi:10.1109/ICDE55515.2023.00041

[63] 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

[64] Wentao Wu, Jeffrey F. Naughton, and Harneet Singh. 2016. Sampling-Based Query Re-Optimization. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 1721–1736. doi:10.1145/2882903.2882914

[65] 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. doi:10.18653/V1/2022.ACL-LONG.180

[66] 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.

[67] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2024. CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion. *CoRR* abs/2405.16444 (2024). doi:10.48550/ARXIV.2405.16444 arXiv:2405.16444

[68] 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. doi:10.18653/v1/D18-1425

[69] 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. doi:10.1145/3060586

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

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

Received October 2024; revised January 2025; accepted February 2025