CS561

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

PRESENTED BY

VISHAKHA, CHRIS, NANDANA

CRITIC

HSIANG EN LIU

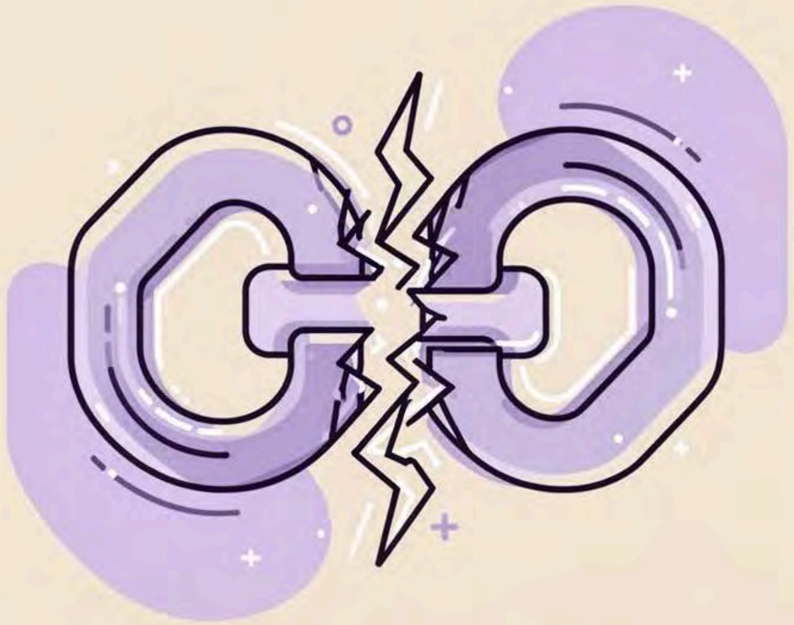
PROPONENT

MARCUS IZUMI

Why Query LLMs?

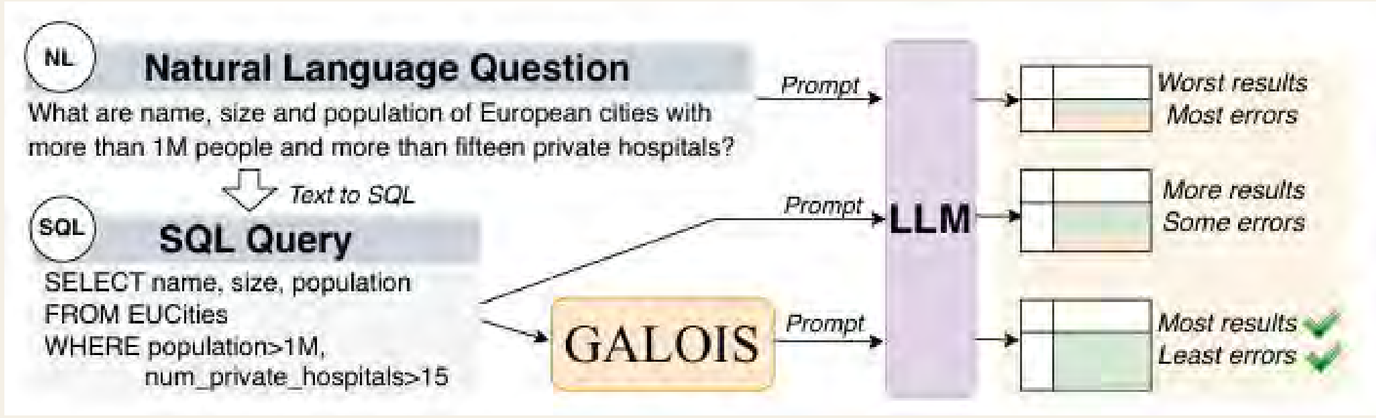
MOTIVATION

LLMs possess **vast unstructured knowledge**, which requires transformation into structured outputs like tables and attributes for effective use in QA, analytics, and data extraction.



The Problem

Natural language queries often lead to **inaccurate results** while SQL queries, though better, still struggle with reliability. Complex queries can easily overwhelm large language models, causing inefficiencies.



Key Tradeoff

COST CONSIDERATION

Fewer calls to LLMs can reduce costs significantly, but this often results in less accurate responses, impacting the overall utility of the data retrieved from the queries.

QUALITY FOCUS

Complex prompts enhance detail and improve accuracy but may introduce more errors, creating a balance between the depth of information and the reliability of outcomes in SQL execution.

INTRODUCING

GALOIS

Galois · GitHub

GALOIS

FRAMEWORK

1

Scan Operator

LLMScan / Filter-LLMScan

2

Dynamic Metadata Acquisition

confidence estimation

3

Cost + Quality Optimizer

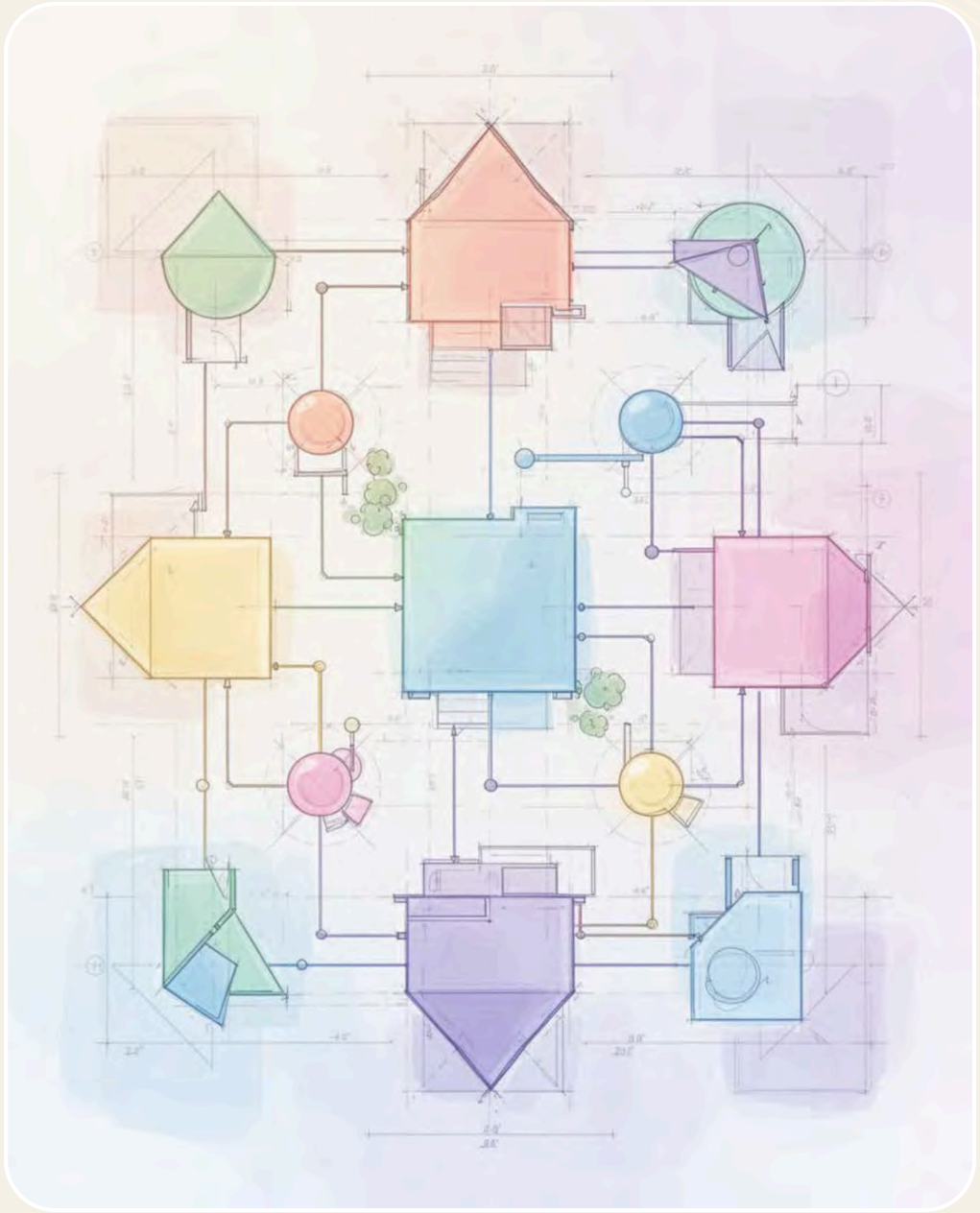
pushdown + physical scans

4

High-Quality Structured Results

pushdown + physical scans

Logical Plan



Logical Plan

Exploring Selection, Projection, and Join

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

Selection

Selection is the process of filtering data based on specific criteria, allowing for targeted insights and efficient data management in analytical processes.

Projection

Projection involves choosing specific fields or attributes from data sets, optimizing performance and clarity by focusing only on relevant information for analysis.

Join

Join is a technique used to combine data from multiple sources, enabling comprehensive analysis by correlating information across different datasets for richer insights.

Logical Plan

New Operators

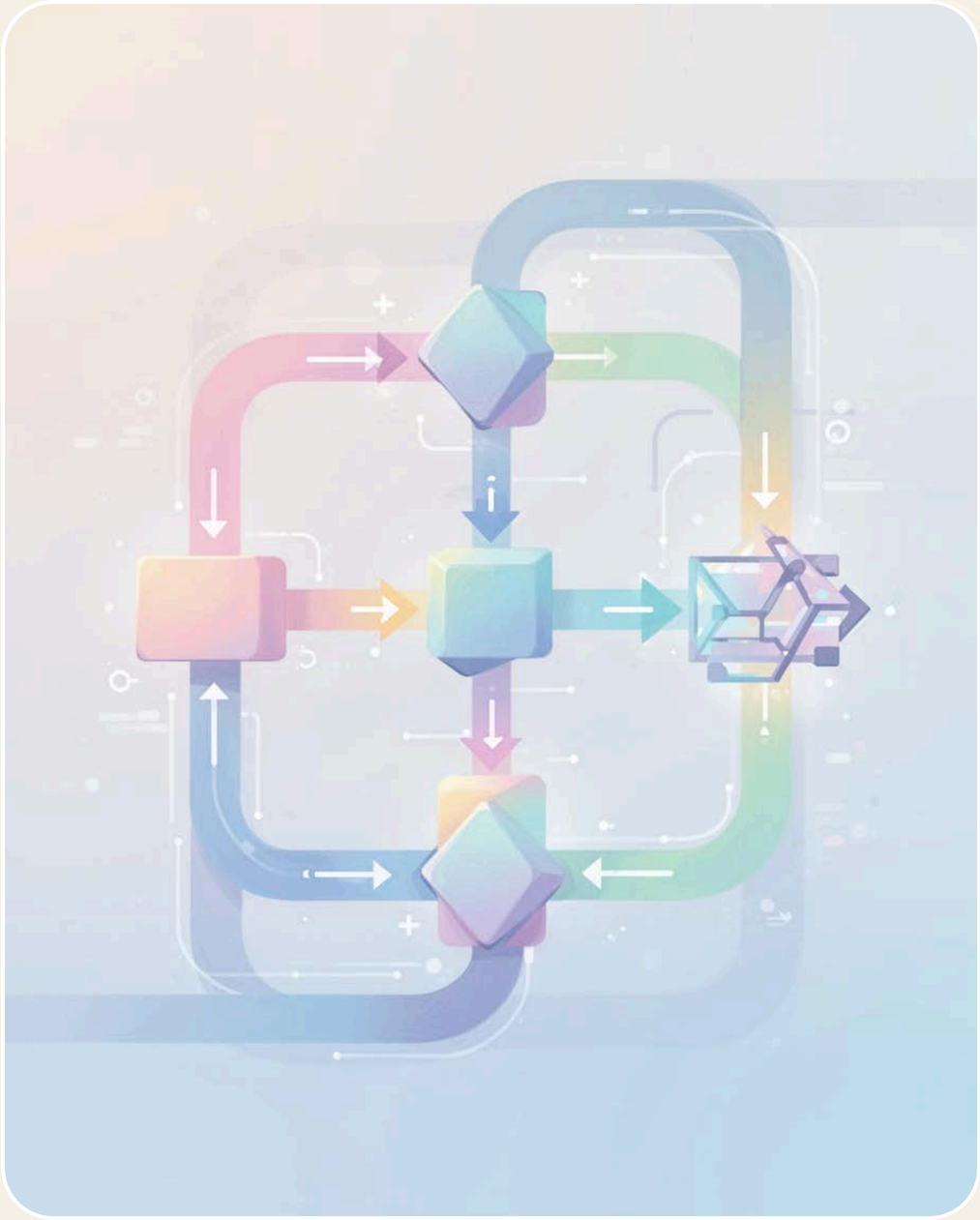
LLM Scan

LLMScan retrieves all candidate tuples without embedding predicates.

Filter-LLM Scan

Filter-LLMScan pushes one or more predicates into the prompt itself.

Pushdown Problem Overview



Pushdown Problem

Examining data push strategies and implications

No Push

Without any push, data management becomes **overwhelming**, leading to excessive resource use and decreased system efficiency, ultimately hindering performance and increasing operational costs.

Push All

Conversely, pushing all data may result in **errors** and bottlenecks, as the system struggles to handle excessive information, complicating the processing and analysis tasks.

Selective Push

The **selective push** strategy stands out as the optimal approach, as it balances data flow and system integrity, ensuring crucial data is prioritized while minimizing overload and errors.

Pushdown Problem

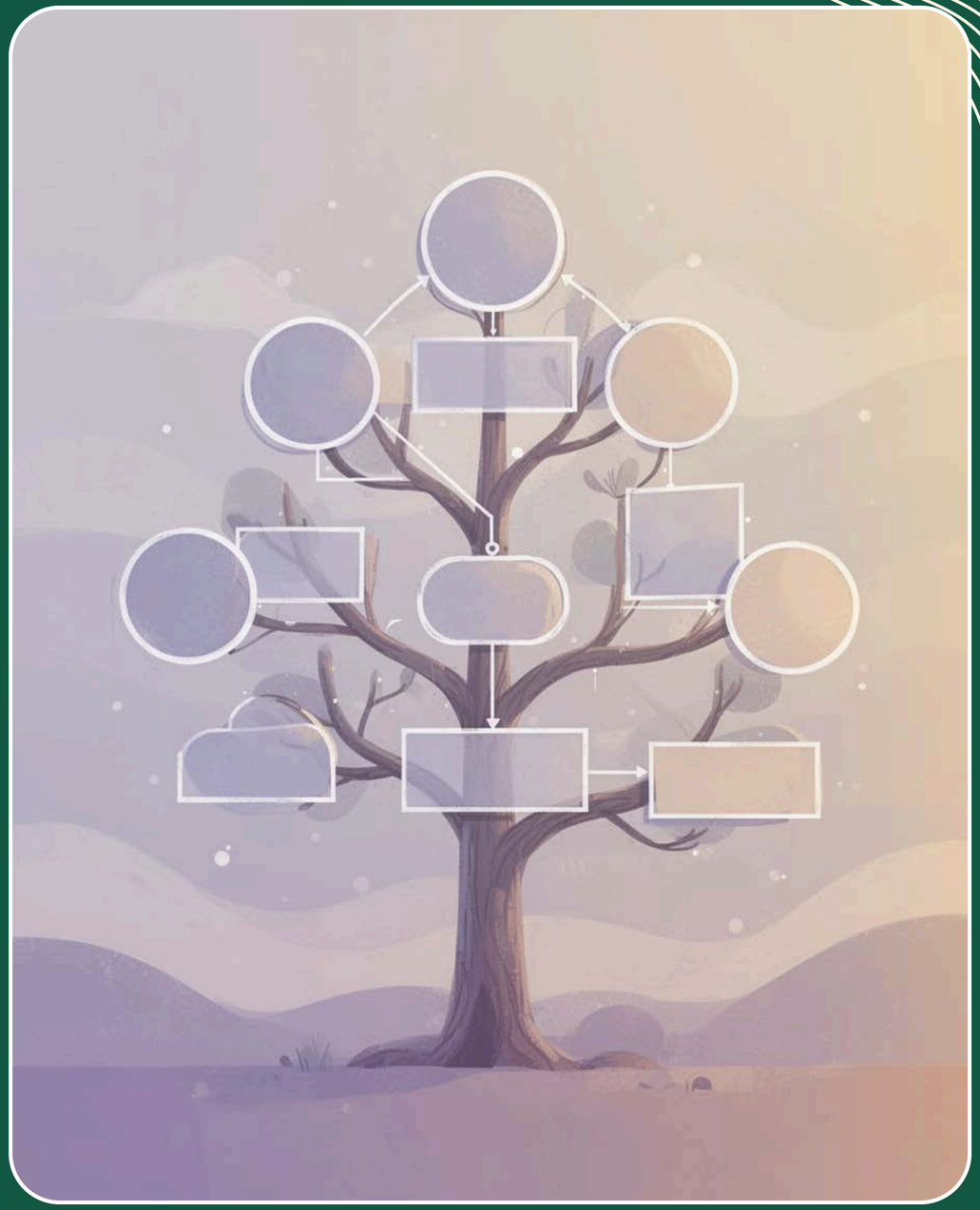
Seeing it through an example

SELECT city
FROM EUCities
WHERE population > 1M
AND private_hospitals > 15

Results
No push → Madrid only
Push all → empty
Push confident → Madrid + Berlin

Strategy	Prompt complexity	Output quality
No push	low	low recall
Push all	high	often empty
Push selective	medium	unstable
Push confident	medium	best balance

Logical Optimization



Logical Optimization

Choosing the best plan using LLMs

Multiple Plans

Exploring various strategies allows for **effective solutions** to complex problems. Each plan must be assessed for efficiency and potential success in reaching desired outcomes.

Use LLM Confidence

Leveraging the confidence of language models can significantly enhance decision-making. It helps prioritize options based on predicted success rates and operational feasibility.

Choose Best Plan

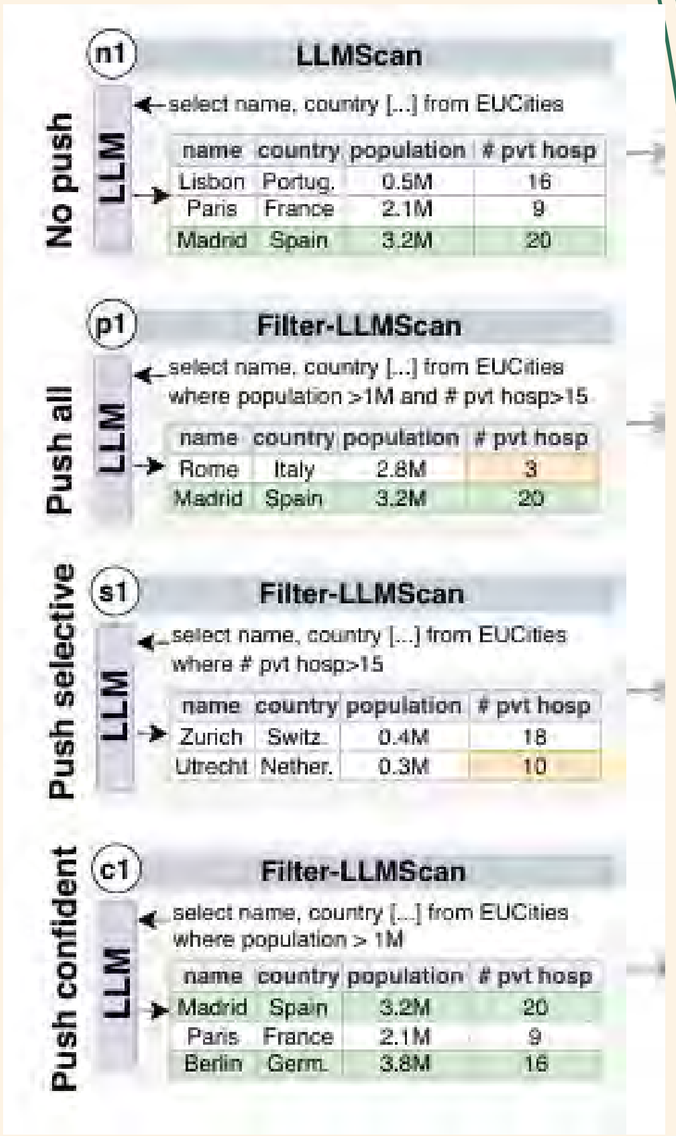
The ultimate choice hinges on analyzing results from multiple strategies. Selecting the most promising plan optimizes resources and maximizes overall performance and quality.

Logical Optimization

Choosing the best plan using LLMs

1. **Let LLM evaluate confidence of each predicate in WHERE clause**
   a. **If only 1 predicate has "high", pushdown to LLMScan**
   b. **If none, no pushdown**
   c. **If 2 ore more are "high", push down all predicate**

```
SELECT city
FROM EUCities
WHERE population > 1M

AND private_hospitals > 15
```



Takeaways

Key insights on logical planning

Logical Planning

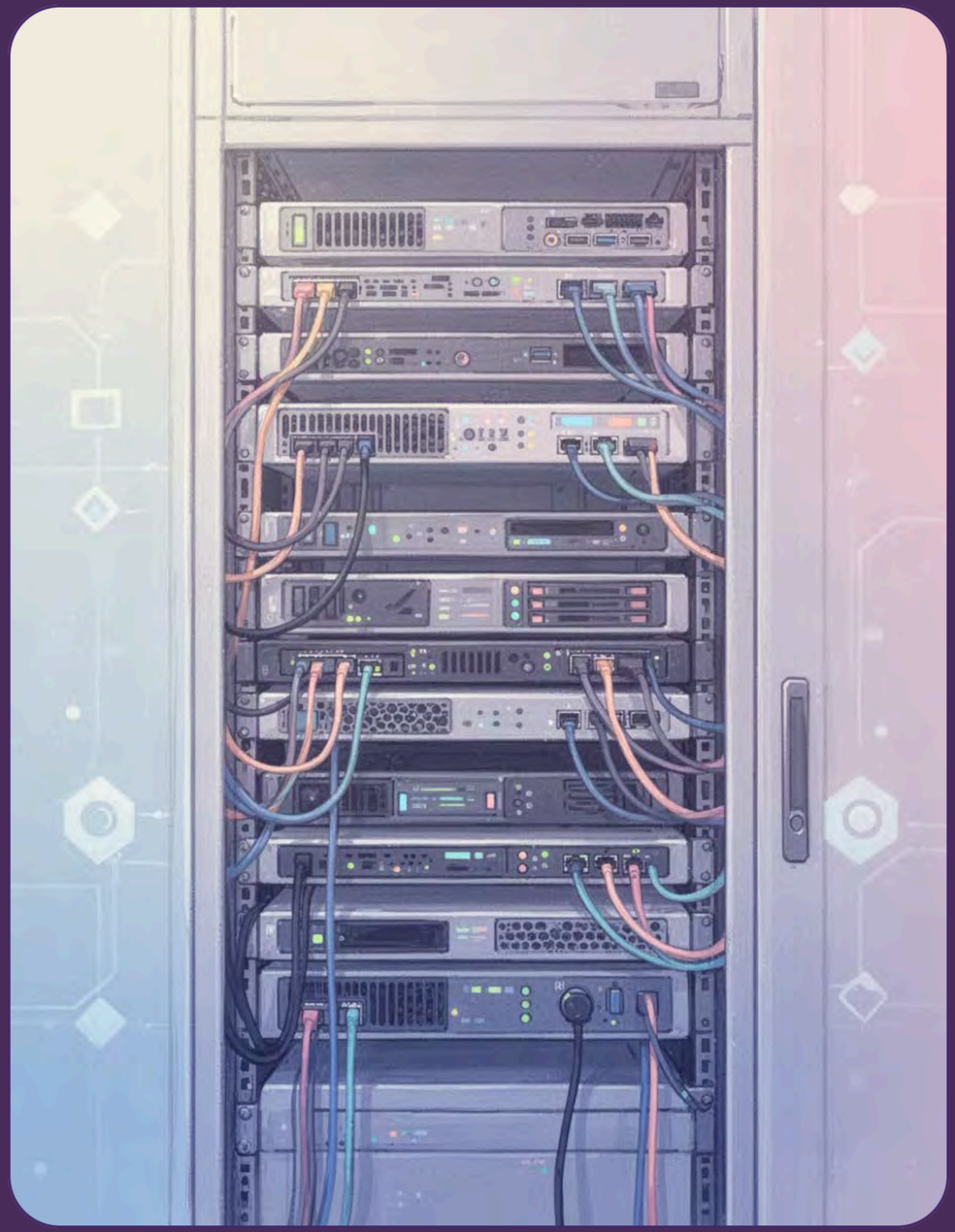
Effective **logical planning** is essential for optimizing both data retrieval and processing efficiency. It lays the groundwork for achieving superior performance in complex data environments.

Pushdown Effects

Understanding the **pushdown strategy** is crucial. The balance between data volume and error management directly influences overall system **quality and reliability**, leading to better operational outcomes.

Physical Query Execution

Adapting Query Execution Strategies to interact with the LLM as a Data Storage Layer

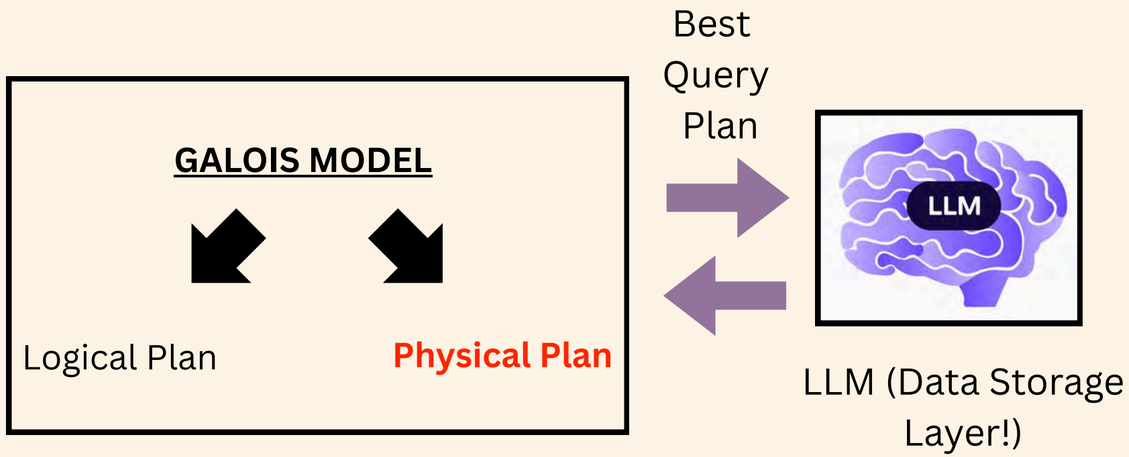


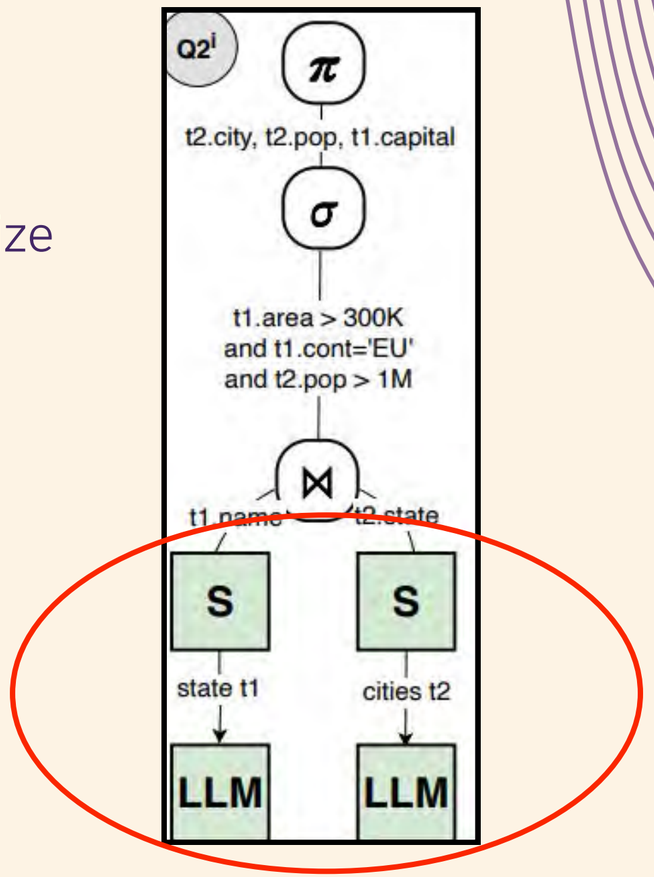
Physical Query Execution

<u>How</u> do we execute the plan?

Coming to LLMs…

- How can query execution be adapted when treating LLMs as the data storage layer, and what **new physical operators** are needed to optimize prompt generation for both data quality and retrieval cost?





Physical Query Execution

How do we execute the plan?

General Challenges faced with LLMs :

- LLMs generates the most likely next word based on training data, so common values appear easily, but rare or uncommon values may require multiple interactions to retrieve.
- LLMs can only generate a limited number of tokens in a single response, so a single prompt may not retrieve all relevant tuples.

Physical Query Execution

<u>How</u> do we execute the plan?

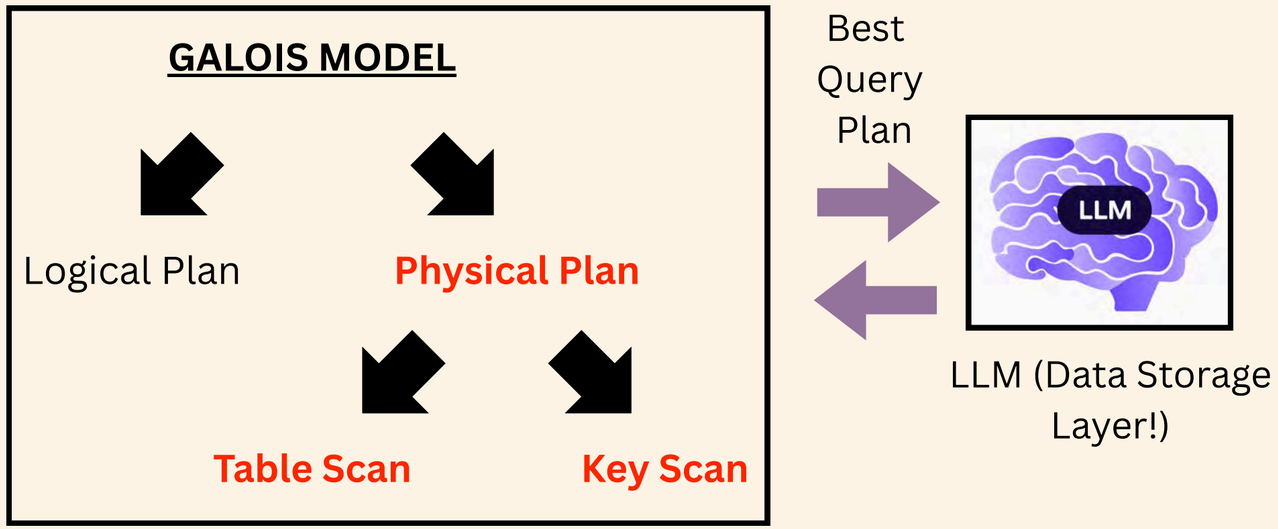
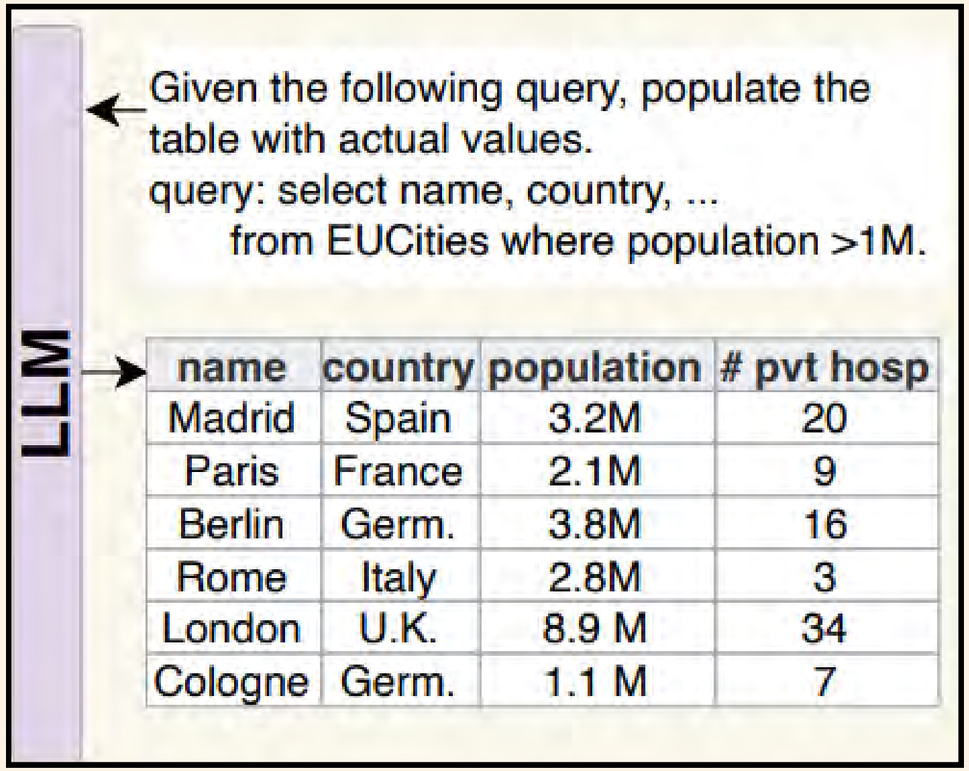


*Table Scan and Key Scan are the physical implementations of the LLMScan and FilterLLMScan operators.

Table Scan

- The LLM is used to retrieve all possible tuples from a table by **iteratively** generating, given the query, and refining results, **using previous outputs as context**, until no new data is found.



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

Input: SQL query q, table $tname$, db schema s, max iter. $maxIter$, language model LLM
Output: tuple set t

```
Input: SQL query q, table tname , db schema s, max iter. maxIter, language model LLM
while (i < maxIter) {
    if (i == 0) {
        prompt = genFirstPrompt(tname, s, q);
    } else {
        prompt = genIterativePrompt();
    }

    jsonResponse = LLM.request(prompt, context);
    parsedTuples = parse(jsonResponse, tname, s);

    if (noNewTuples(parsedTuples, t)) {
        break;
    } else {
        context.push_back(prompt);
        context.push_back(jsonResponse);
        t.insert(t.end(), parsedTuples.begin(), parsedTuples.end());
    }

    i++;
}

return t;
```

Table Scan Algorithm

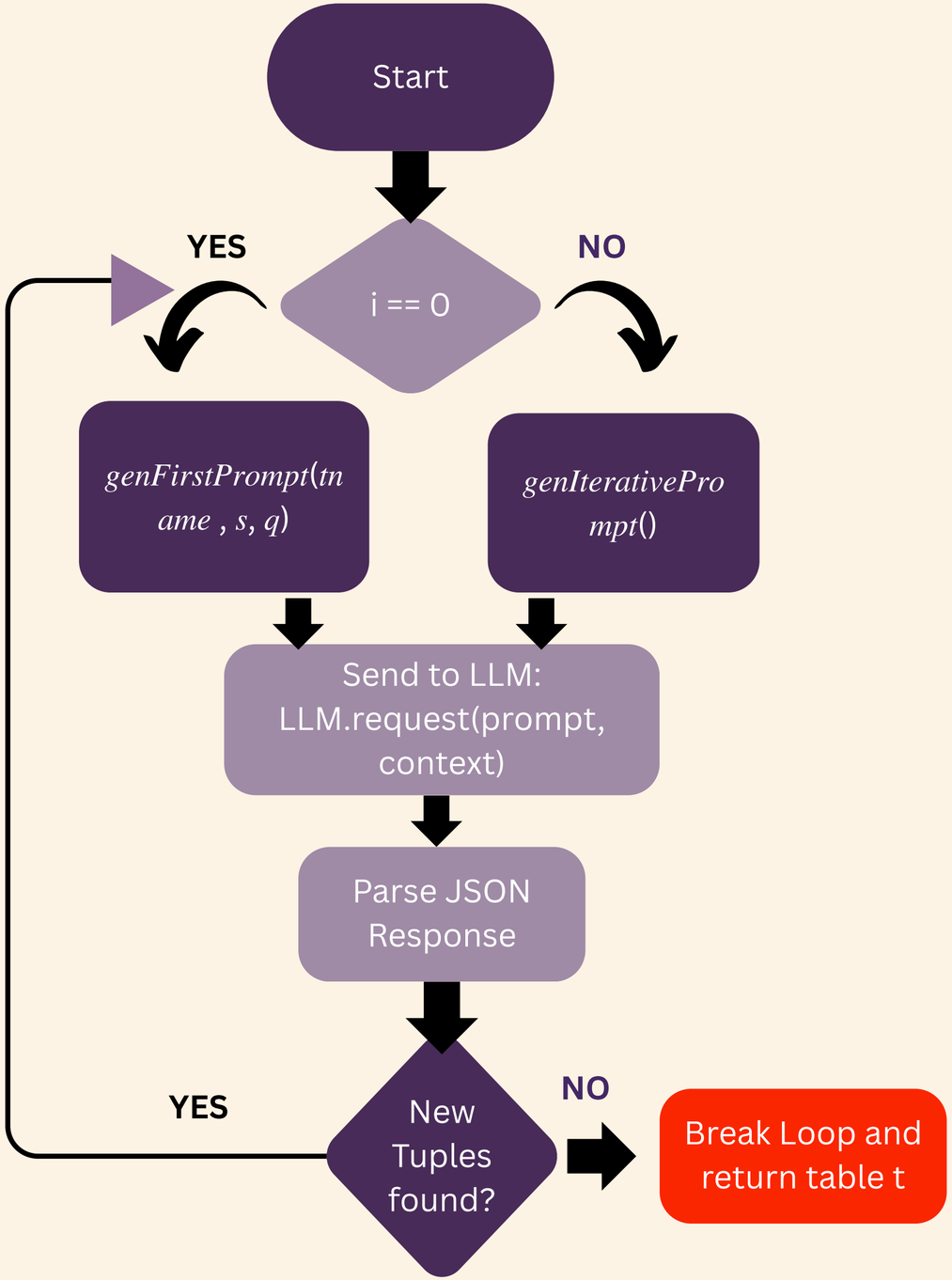


Table Scan

First Prompt to the LLM:

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.

Iterative Prompt to the LLM:

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

Table Scan

Is this a good enough approach for retrieving accurate results?

Mostly yes. Through iterative prompting, less frequent values can be retrieved (for the most part) and it also accommodates the LLM's token limitations by retrieving the data incrementally.

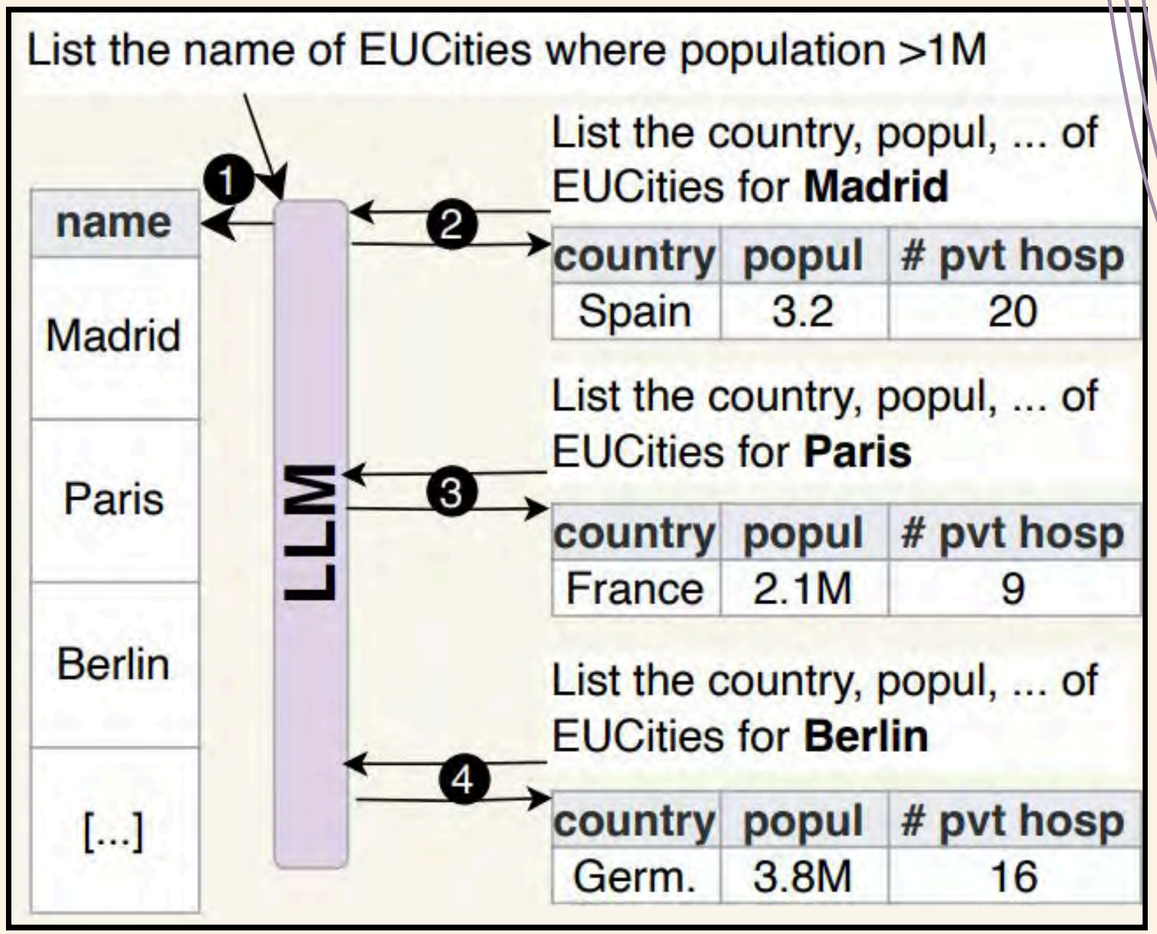
But...

The LLM can struggle with more complex reasoning, potentially missing relevant tuples.

Key Scan

1. Galois retrieves **all the key values** for the table involved in each LLMScan.

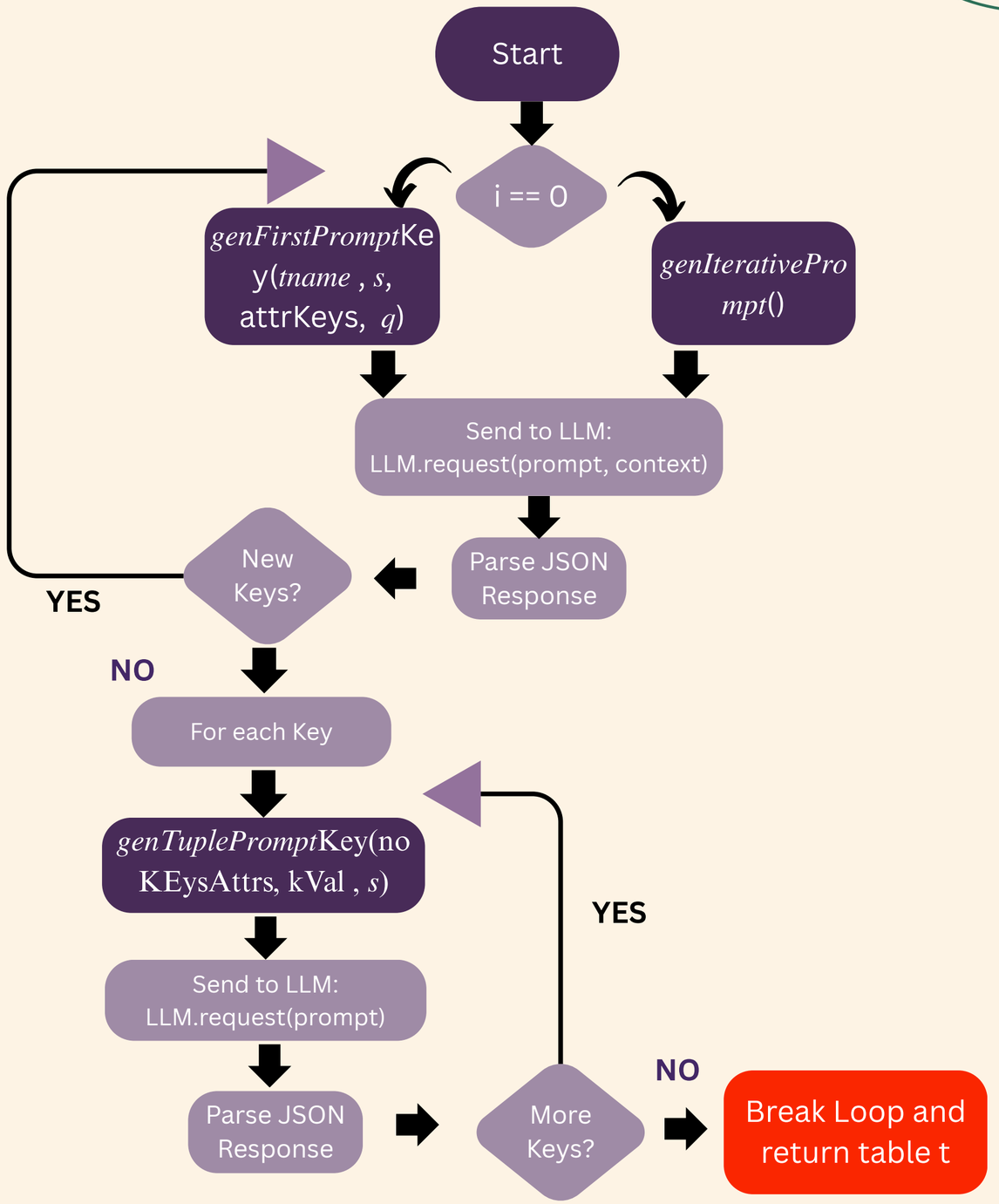
2. In the second step, for each key value retrieved, Galois obtains the values for the other attributes.



Input: SQL query q, table $tname$, db schema s, max iter. $maxIter$, language model LLM
Output: tuple set t

```
while (i < maxIter) {
    if (i == 0) {
        prompt = genFirstPromptKey(tname, s, attrKeys, q);
    } else {
        prompt = genIterativePromptKey();
    }

    jsonResponse = LLM.request(prompt, context);
    parsedKeys = parseKeys(jsonResponse, tname, s);

    if (noNewKeys(parsedKeys, keys)) {
        break;
    } else {
        context.push_back(prompt);
        context.push_back(jsonResponse);
        keys.insert(keys.end(), parsedKeys.begin(), parsedKeys.end());
    }

    i++;
}

noKeysAttrs = tname.attrs - tname.keys;

for (auto kVal : keys) {
    promptT = genTuplePrompt(noKeysAttrs, kVal, s);
    jsonResponse = LLM.request(promptT);
    parsedTuple = parse(jsonResponse, tname, s);
    t.insert(t.end(), parsedTuple.begin(), parsedTuple.end());
}

return t;
```

Key Scan Algorithm



Key Scan

First Prompt Key:

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

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

Key Scan

Is Key scan similar to Chain of prompting?

Yes! Key scan is similar to chain of prompting

- It breaks a complex query into multiple sequential prompts, where the output of one step becomes the input for the next
- Step-by-step decomposition mirrors how chain-of-thought or multi-step prompting structures reasoning into smaller, manageable parts.

Improved accuracy

Simpler prompts

Better handling of structured data

Parallelization

Reduced hallucination risk

Higher number of LLM calls

Incomplete key coverage

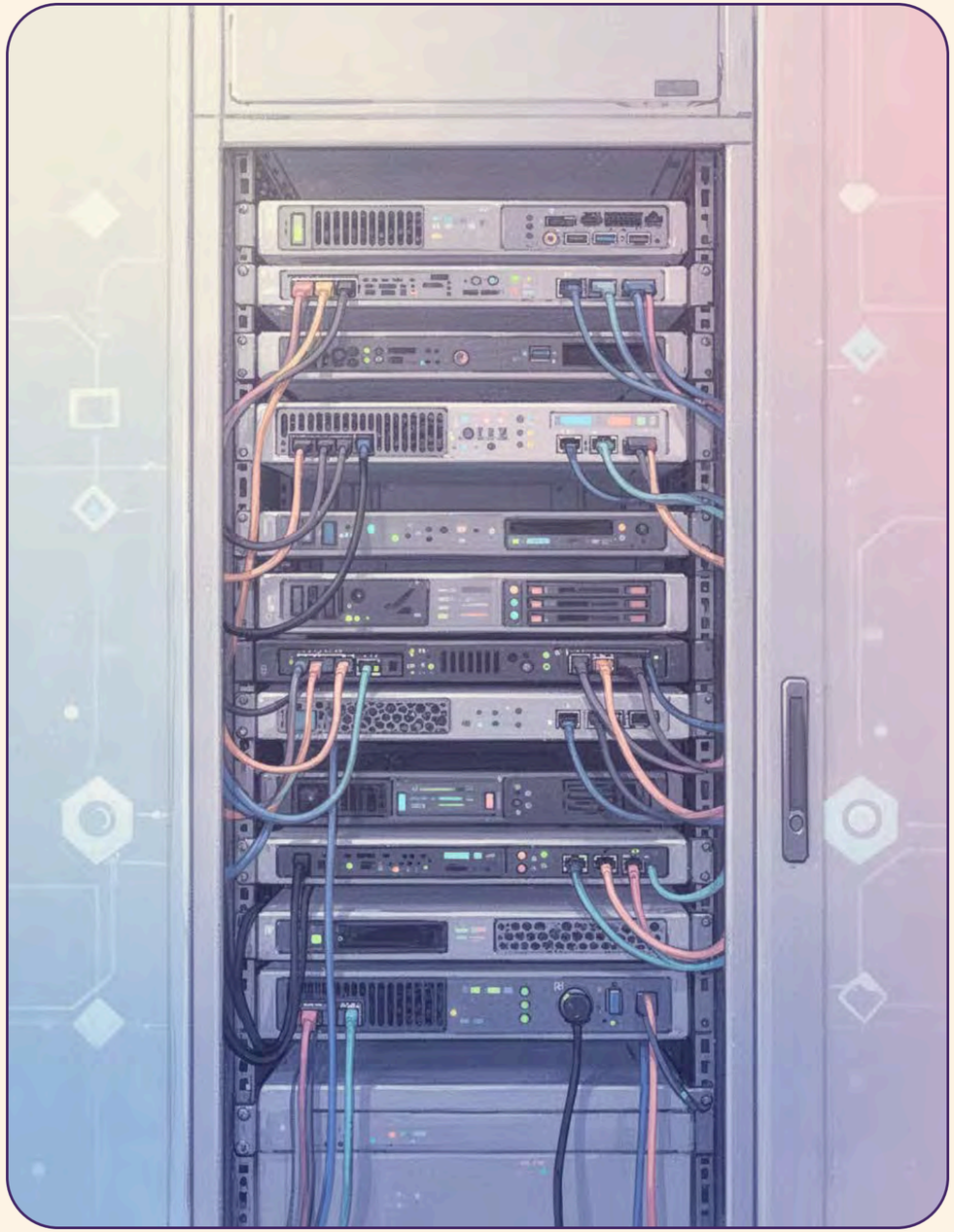
Error propagation

No strong guarantees

Extra overhead

Physical Optimization

Which Scan Strategy is the best to choose for Query Execution?



***Objective*: Which Scan Strategy is the best to choose for Query Execution?**

Table Scan

→ Iterative prompting makes it effective for general queries, and it can handle rare values better than a single prompt.

→ However, it can still struggle with complex reasoning or queries requiring step-by-step logic.

Key Scan

→ Key-Scan focuses on retrieving tuples one key (or small group of keys) at a time, which allows the LLM to apply more precise reasoning, like CoT prompting.

→ Reduces the risk of errors in complex queries.

Key Scan seems like the best choice here...but is it truly the optimal choice for all scenarios?

Solution: Use Confidence Based Selection

Query, *q*

$$conf\,(q) = LLMconf(keys\,|conds)^n$$

$$conf\,(q) \geqslant \tau \;\; \textbf{?}$$

YES → **Choose Key Scan (High Confidence)**

NO → **Choose Table Scan (Low Confidence)**

q → query
n → attributes in the query
keys → unique identifiers
conds → conditions in the query

"The rationale is that when the model's confidence is low, the accuracy of key retrieval in Key-Scan may be compromised".

Example

SELECT airline, abbreviation
FROM airlines
WHERE country = 'USA';

↓

Ask the LLM how confident it is in returning all the airline IDs having country = “USA”

↓

Assume LLM gives a value: $LLMconf\,(keys\,|conds)$ = 0.8

↓

n= 2, $conf\,(q) = 0.8^2 = 0.64$

↓

0.64 > 0.4

Assume $\tau = 0.4$

*Set at the point where using Key-Scan no longer increases AVG-Score, so it is applied only when it improves results.

↓

Key Scan!

Cost Optimization

Selective Attribute Retrieval

Many queries only require a subset of the attributes present in the output table.

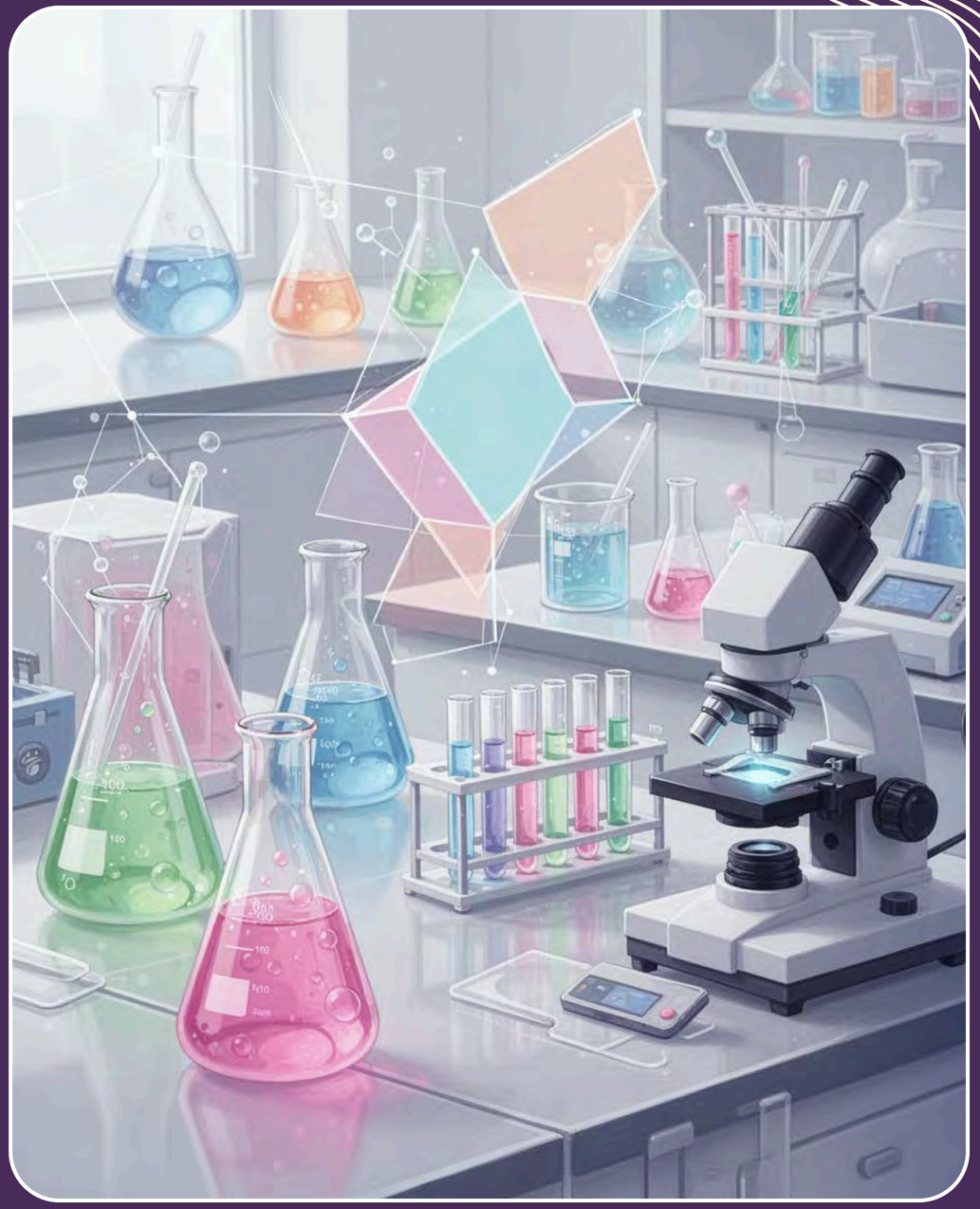
Maintain Context while Compacting the Output

Restrict the LLM's output to only the attributes we need while keeping the schema to provide context to the LLM.

Advantages w.r.t to Cost

This reduces the number of tokens generated by the LLM in its response, minimizing computational overhead and response time + advantageous for schemas with a lot of attributes.

Experiments & Results



Experiment Setup

Datasets

Internal Knowledge (Spider, IMDB, Presidents)

External Context (Premier, Fortune)

LLM

GPT-4o-mini

Llama-3.1 (8B, 70B)

Metrics

F1-Cell

Cardinality

Tuple Constraint

AVG-Score

BaseLines

NL

SQL

Galois (no opt)

Palimpzest

Cost

Tokens

+

Time

Galois Varients

Galois S (Selective Pushdown)

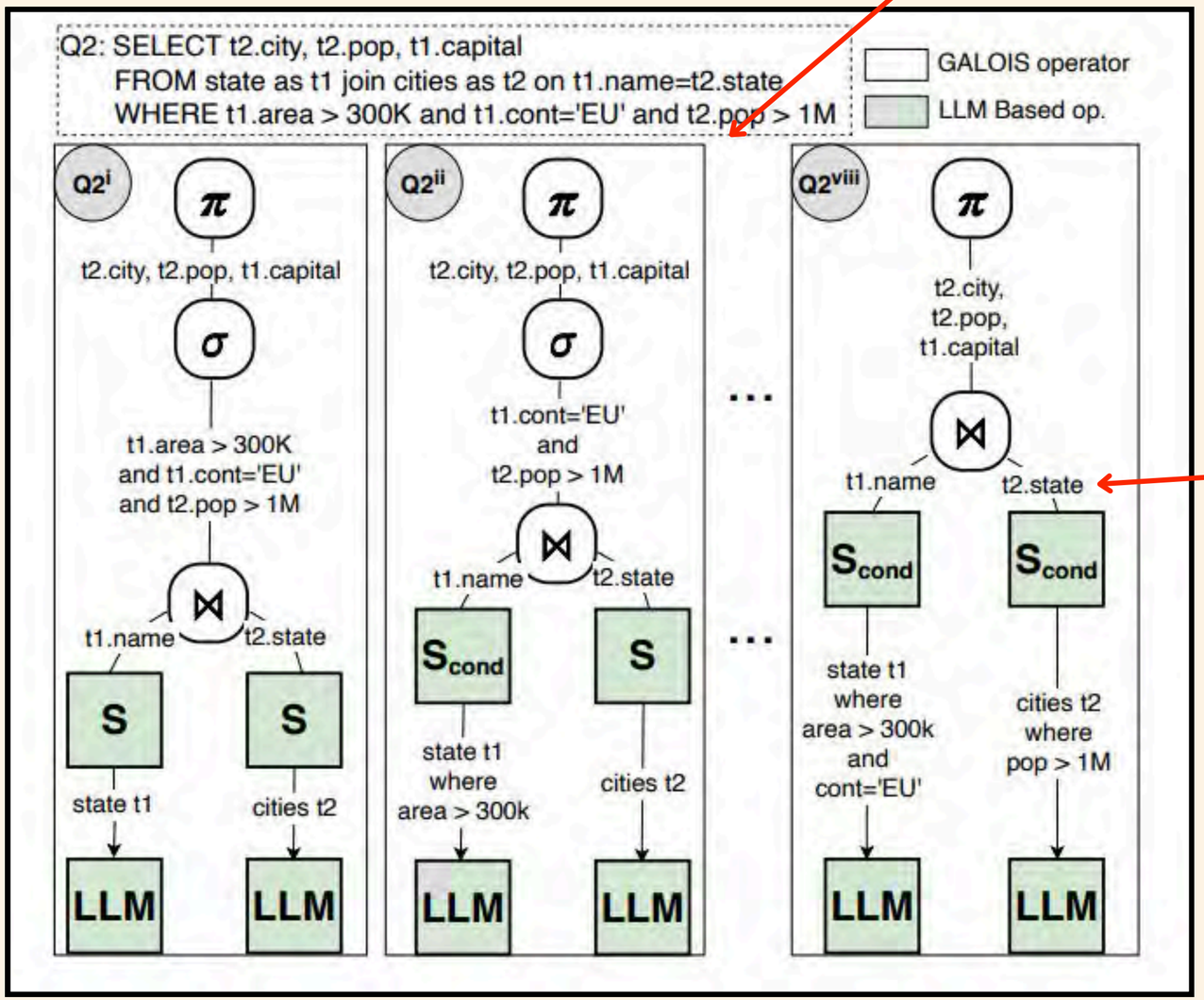
Galois A (All Attributes Pushdown)

Galois F (Full System) optimizations

Experiment Setup

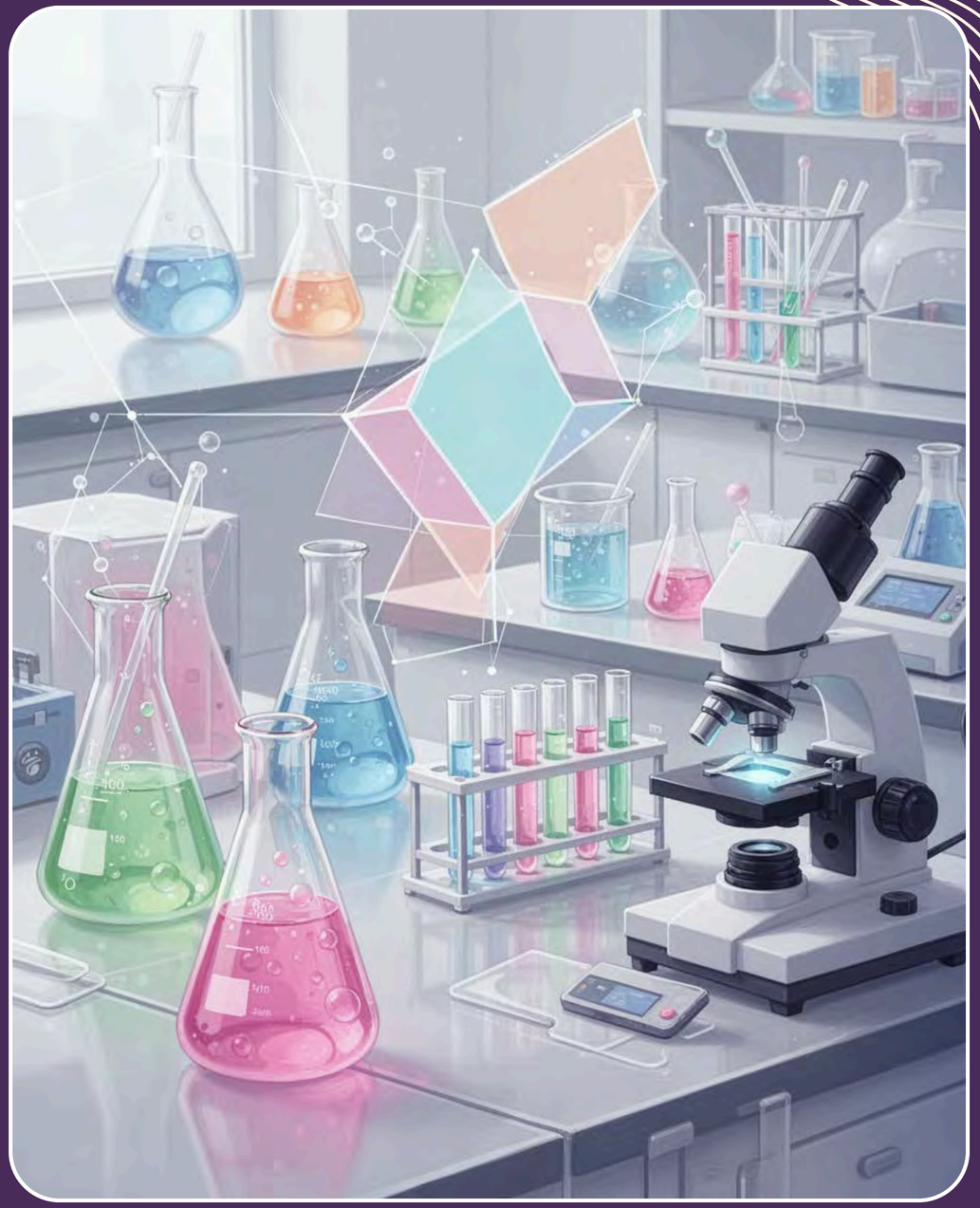
Galois S (Selective Pushdown)

Galois (No Pushdown)



Galois A (All Attributes Pushdown)

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?



Experiment 1

Comparing NL, SQL, and Galois

Took queries over internal datasets ran them using NL, SQL, Galois WO etc on LLaMA-3.1 70B, selected plans using a confidence threshold $\tau = 0.6$

NL was unreliable

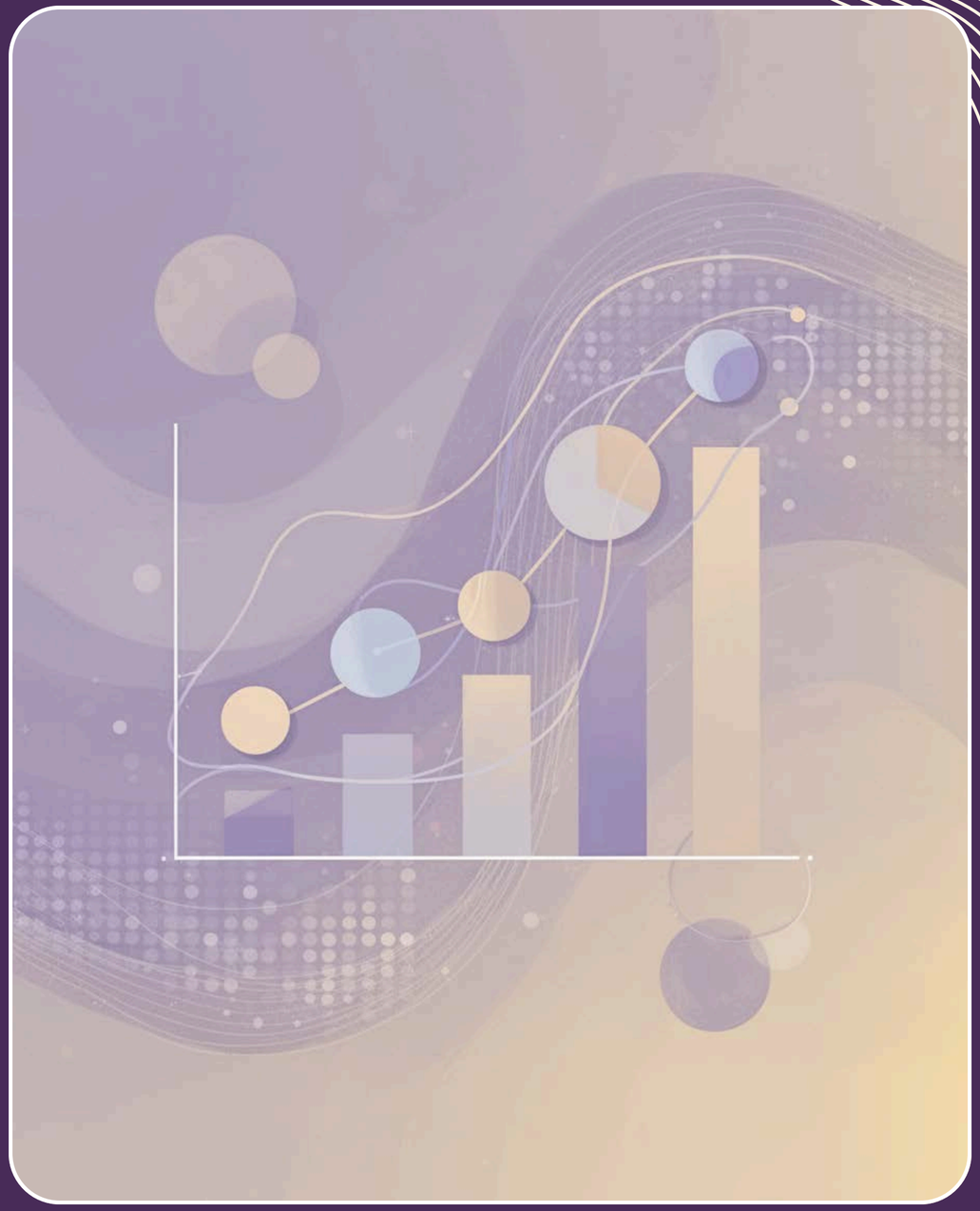
Struggled with reasoning

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**

TakeAway

Galois increases the accuracy and completeness of SQL query results compared to natural language and SQL baselines with 144% and 29% improvement, respectively

2) Are the proposed optimizations effective with respect to both the quality of the results and the associated costs?



Experiment 2

Comparing push strategies for optimization

Data overload occurs, leading to inefficiencies and slow processing times → hinder overall system performance
highest cost!

Often results in errors and excessive resource consumption, making data less reliable and harder to analyze

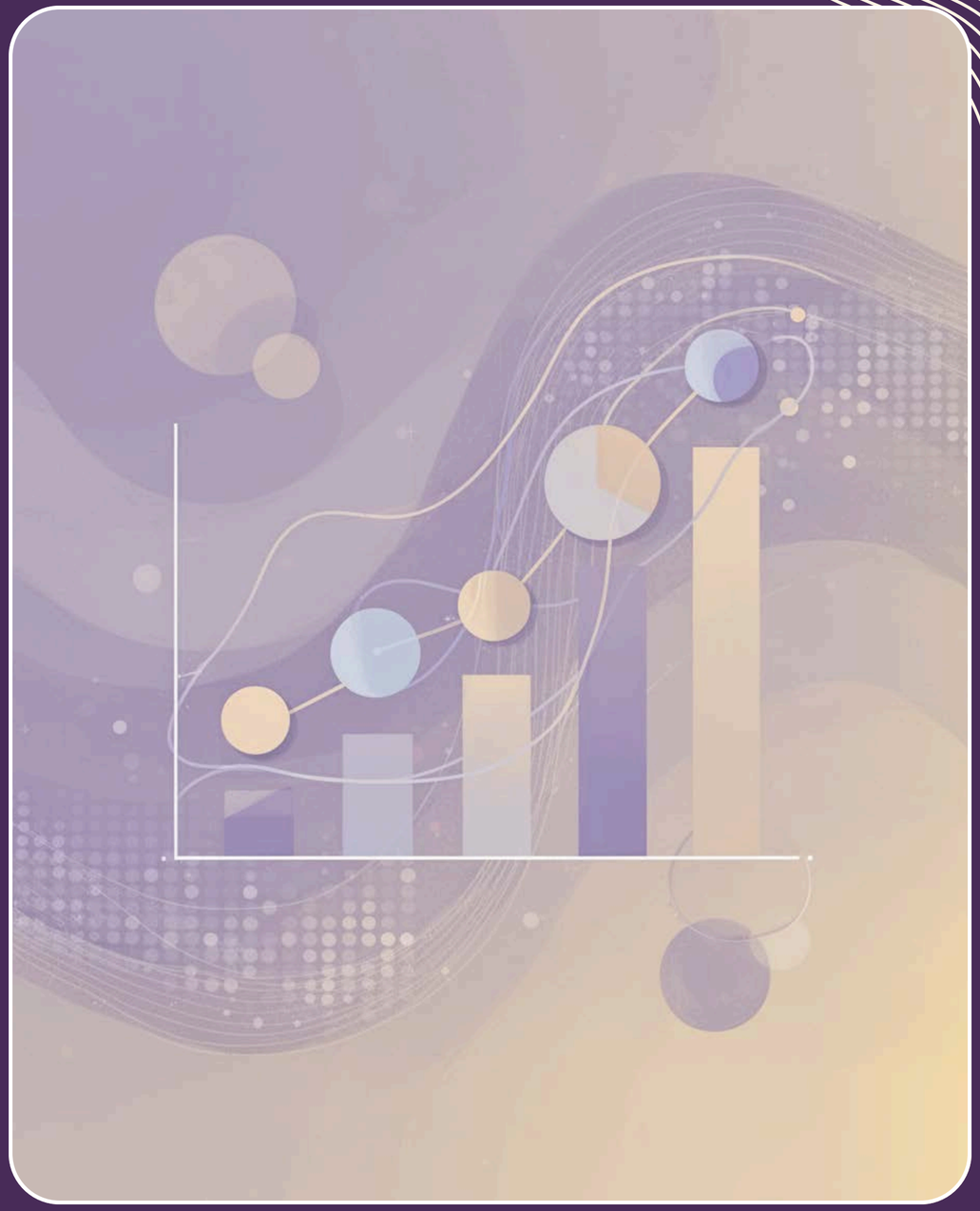
Good trade-off between the obtained quality and the number of generated tokens.

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

TakeAway

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.

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?



Impact of LLM parameters

Evaluation across multiple LLMs:

- NL, SQL, and all Galois variants are tested on three different LLM models.

Models used:

- GPT-4o Mini
- LLaMA 3.1 8B
- LLaMA 3.1 70B

Model parameter sizes:

- LLaMA 3.1 **8B** → 8 billion parameters
- LLaMA 3.1 **70B** → 70 billion parameters

How do the parameters influence the output?

Larger Models → More capacity to store a wider array of knowledge + also has more capacity for learning deeper relationships.

Smaller Models → Smaller memory, can struggle with complex queries

Impact of LLM parameters

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**

AVG-Score of different LLMs

- The LLAMA model with 70B can handle more factual data and complex queries.
- Galois F (pushdown of all conditions) has the best AVG score, implying that the LLM was able to give accurate results (the LLM has learnt complex patterns) and with the conditions pushed down, the LLM was able to decipher the complexity of the query.

Impact of Retrieved Values

Shows the impact of the values over the same queries, i.e., same logical plan except for the values in the conditions.

Quality in terms of AVG-Score

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**

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**

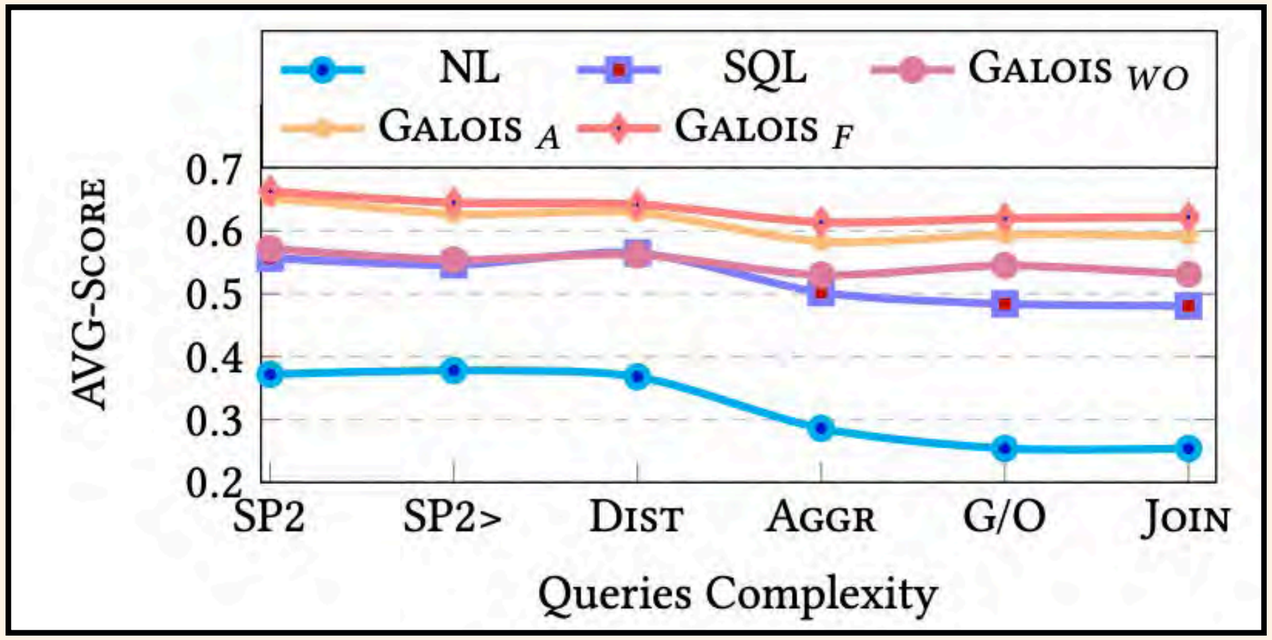
Impact of Complexity in the Query

Expected Result: Increasing the Complexity decreases the Quality.

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*

Queries by Type: Textual vs Numerical

Impact of Complexity in the Query



Result wrt Queries Complexity

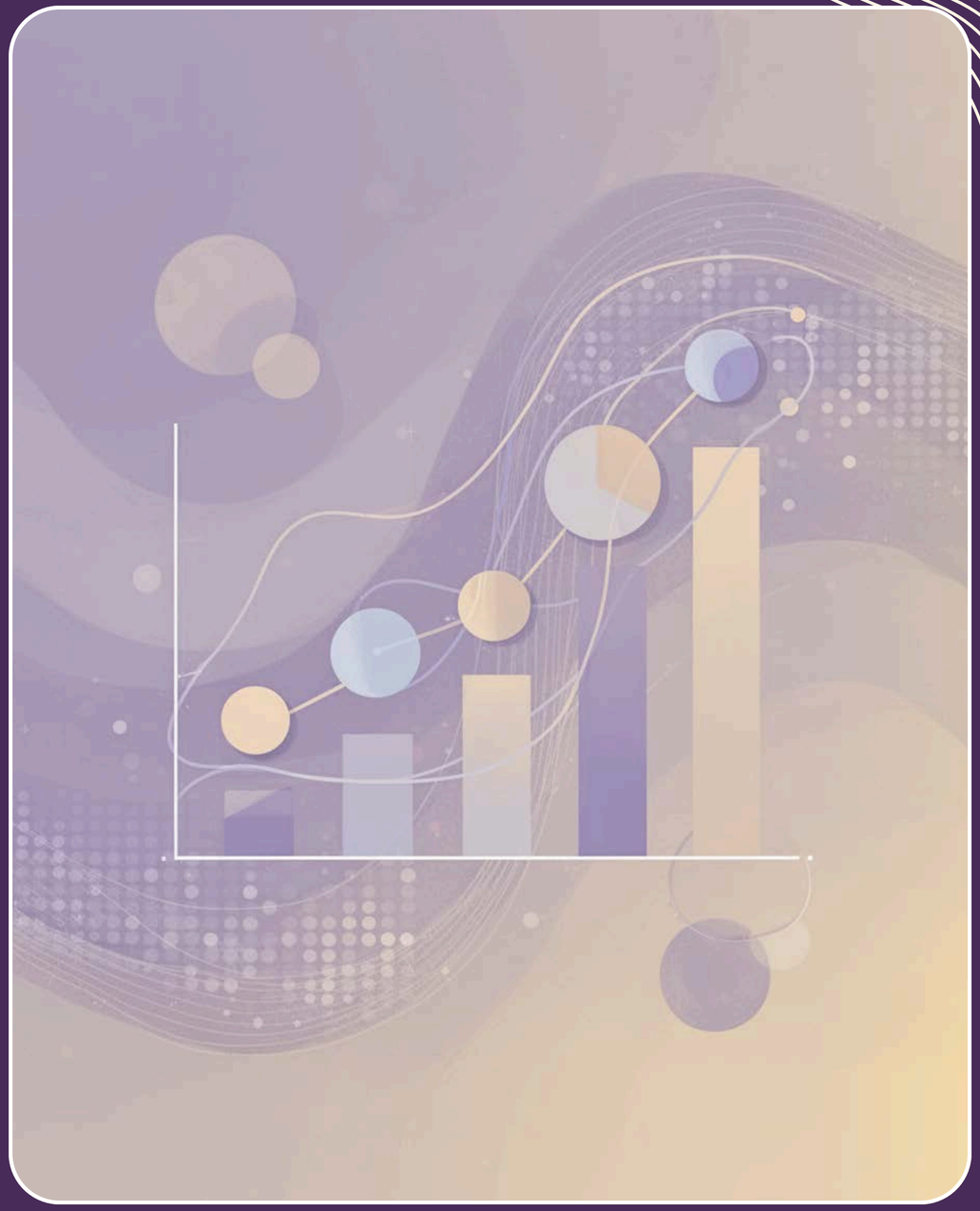
- **SP2** denotes queries with Selection and Projection with at most two conditions,
- **SP2>** those with more than two conditions
- **Dist** queries with DISTINCT
- **Aggr** queries with aggregate functions
- **G/O** queries with group by or order by, and Join queries with joins.

TakeAway

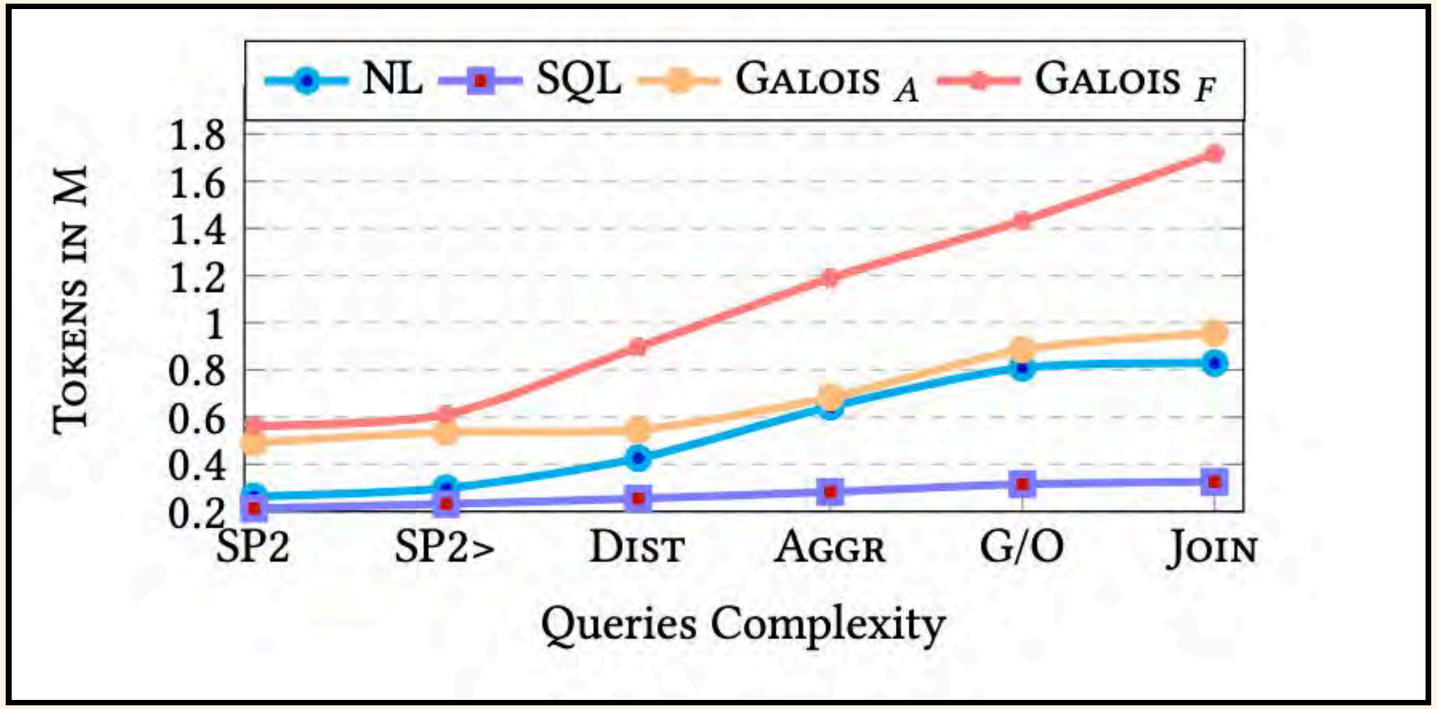
Galois on LLaMa 3.1 70B scores higher than on smaller models for its ability to store more factual data and execute complex pushdown operations.

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

4) What are the costs associated with using Galois, and what latency can be expected when employing our proposed approach?



Costs associated with the Galois Model



Costs w.r.t query complexity.

- Galois WO consumes **8.5M** to **19.7M** tokens (!)

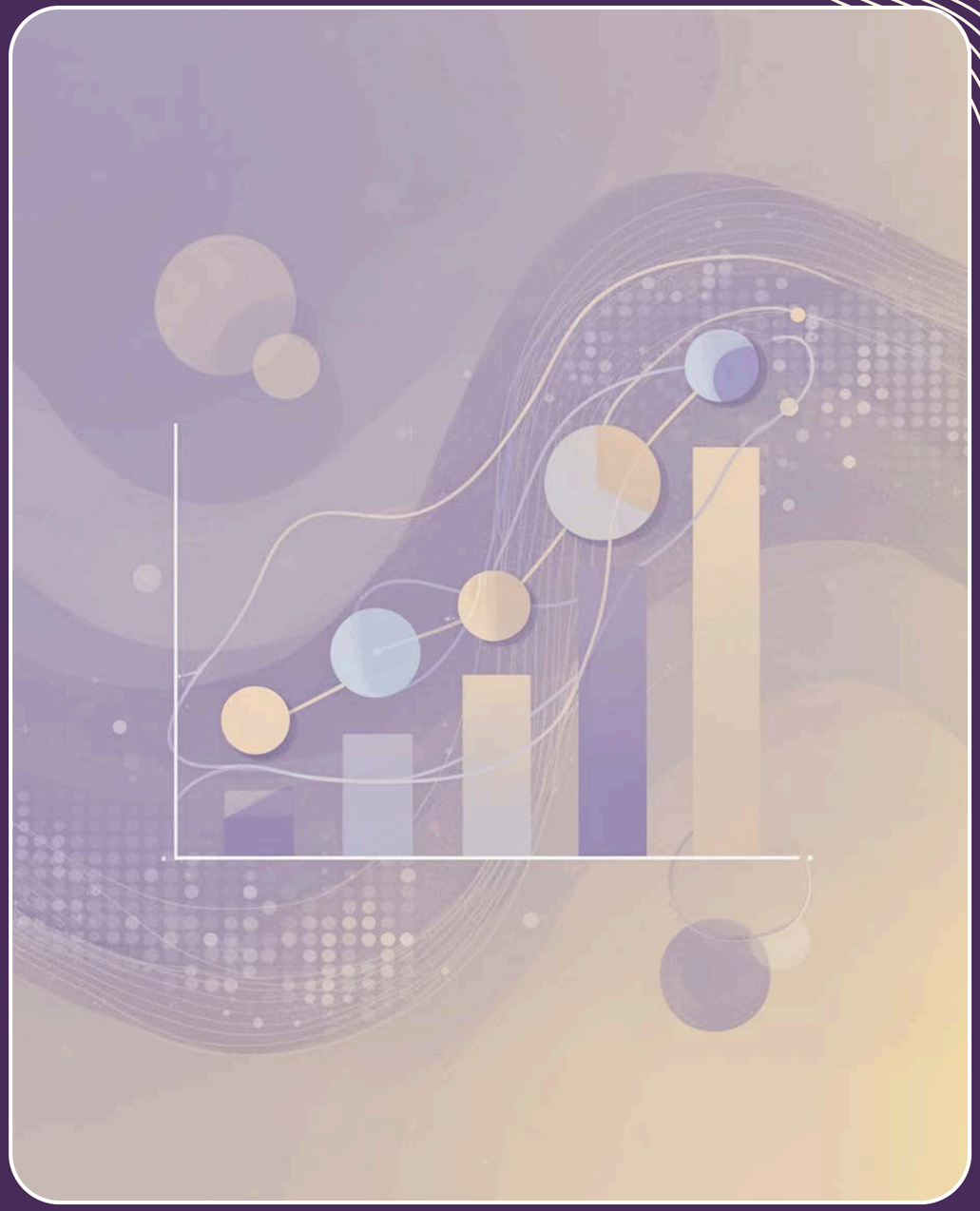
TakeAway

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

(5) Can the proposed framework be effectively combined with in-context learning techniques, such as those employed in RAG, to enhance performance?



In-Context Querying

- Uses RAG (Retrieval-Augmented Generation) to combine external knowledge with LLM generation.
- Handles queries on **novel** information from Premier (60 docs) and Fortune (500 docs) datasets.
- LLM (LLaMA 3.1 70B) generates answers using retrieved context.
- Compare accuracy (AVG-Score) and number of tokens) against baselines, including procedural **Palimpzest**.

In-Context Querying

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

Quality Results and Costs for RAG application

TakeAway

→ 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 vs Palimpzest where procedural steps are required (both give similar performance at the end, with Galois using less amount of tokens.

Conclusion

Galois bridges databases and LLMs by adding logical and physical query optimization

LLMs alone are not reliable for structured queries, but by treating them like a database and applying query optimization techniques, it can significantly improve both accuracy and efficiency.

Critiques & Proponents

#1: Can LLMs Replace Traditional Databases?

Critique

Using LLMs as replacement of traditional databases may:

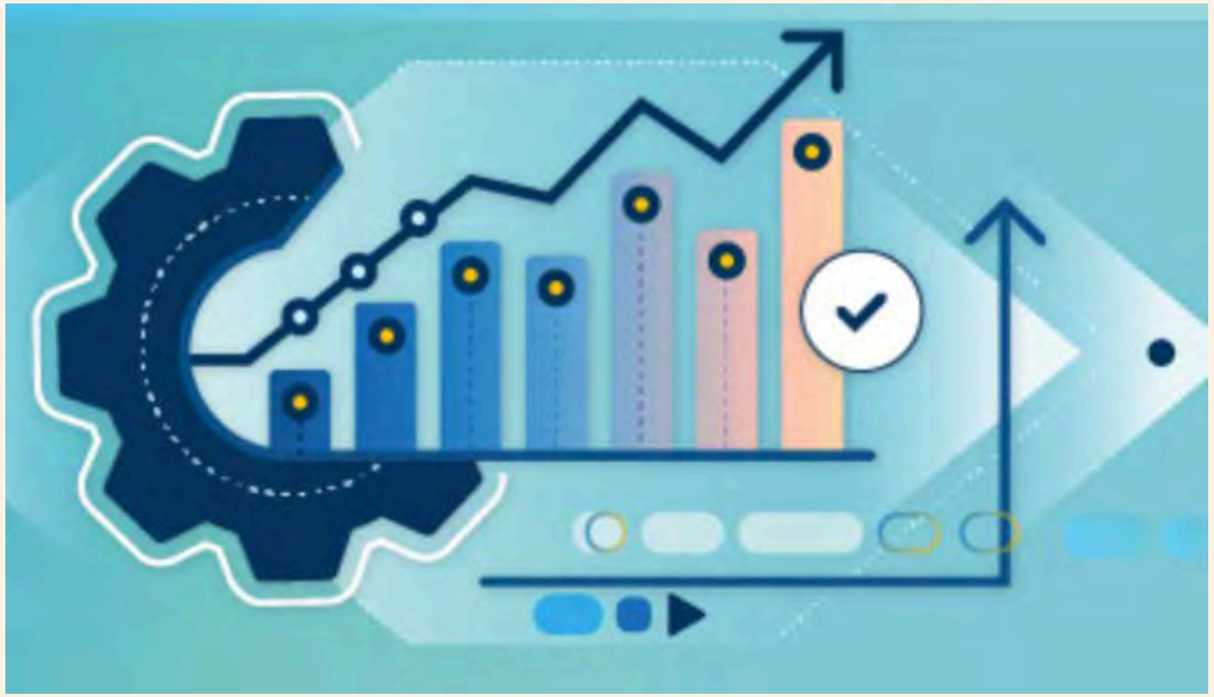
- omit entities
- hallucinate values
- vary with prompt structure

#1: No. But it can extend what we can query beyond traditional data

Proponent

Contribution of Paper Lies in:

- Enables users without SQL expertise to obtain structured result through DBMS techinques/optimization
- Allow declaritve query on unstructured data like reports



#2: Can we trust self-reported confidence to drive plan selection?

Critique

- **LLM is both data source and metadata source**
- **No independent signal for optimization**
- **Decisions rely on self-reported confidence**

```
String prompt = "Given the following relational schema ${relationalSchema} please give me your confidence,
based on your internal knowledge, for every attribute. give me a value between 0 and 1,
where 1 is perfect confidence and 0 is no confidence. Return an higher score if you are
able to return accurate and factual data. Return a lower score if you can't return accurate
and factual data. Return the results in JSON with the properties \"attribute\" and \"confidence\"";
```

#2: No Traditional Metadata to Exploit in LLM

Proponent

No histograms, no indexes, no exact stats under LLM engine

But some metadata is needed for optimizaiton

Results show that LLM-derived signals ≠ perfect, but useful

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**

#3: The Logical Plan is Over-Simplified

Critique

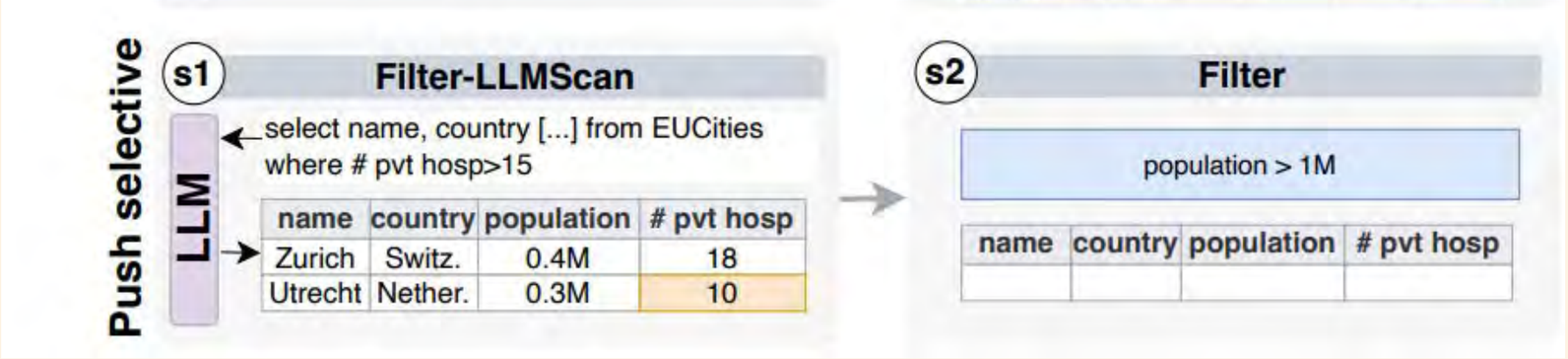
Aggressive Simplification?

Plan space is heavily pruned, and only explores:

- no pushdown
- single-predicate pushdown
- all-predicate pushdown

Does not explore arbitrary useful subsets of predicates

Misses useful predicate combinations



#3: It’s a scalability design choice!

Proponent

- **Plan space grows rapidly with predicates**
- **Full search is computationally expensive**
  - **Costly both monetary and latency wise**
- **Restricted plan space = practical optimization**

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

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

#4: The Quality Metrics are too Forgiving

Critique

- **Cell-level focus ignores structure**
  - correct values ≠ correct tuples
- **Attribute relationship not enforced**
  - values may be matched to wrong entities
- **AVG-Score can hide failures**
  - strong value scores mask structural errors

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**

- **F1-Cell** → correct individual values
- **Cardinality** → correct # of rows
- **Tuple Constraint** → entire rows are exactly correct (values + relationships)

#4: Metrics Handle LLM Variability

Proponent

- **Approximate matching handles variation**
  - avoids penalizing formatting differences
- **Multiple metrics capture different aspects**
  - values, completeness, and structure
- **AVG-Score provides a balanced summary**
  - combines strict and flexible evaluation

#5: Quality Improvements Come with higher Cost

Critique

- **Key-Scan increases LLM calls**
  - multiple prompts per query
- **Higher token usage and latency**
  - more expensive and slower
- **No clear cost-quality boundary**
  - unclear when extra cost is justified

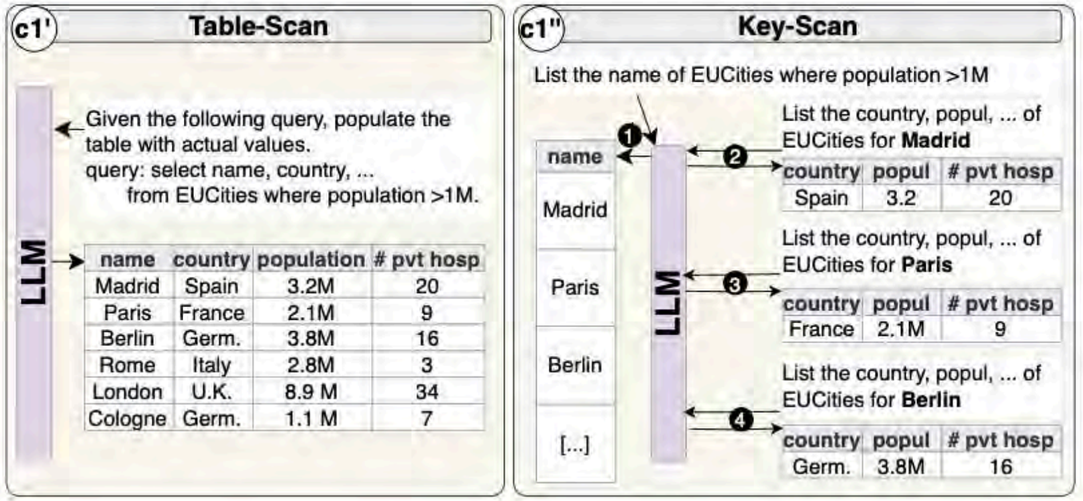


Fig. 3. Two alternative physical operators for implementing logical operator **c1** in Figure 2. Table-scan (**c1'**) gets entire tuples, while Key-Scan (**c1"**) gets key values first.

#5: Cost is Modeled & Flexible

Proponent

- **Cost is measured directly**
  - tokens and execution time reported
- **Trade-off is inherent to LLM systems**
  - better accuracy naturally costs more
- **Configurable strategies allow flexibility**
  - adapt to budget and latency constrants

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

Thank you!