# In-Context LLM Query Execution on Textual Documents

(Discussion Paper)

Dario Satriani[1], Enzo Veltri[1], Donatello Santoro[1] and Paolo Papotti[2]

[1]Università degli Studi della Basilicata (UNIBAS), Potenza, Italy
[2]EURECOM, Biot, France

**Abstract**

In the last few years, Large Language Models (LLMs) have been widely adopted for data science tasks such as data extraction. In this scenario, textual documents are fed in the context of the LLM - as in a RAG setting - and tabular data is extracted via declarative queries. However, LLMs often struggle with complex queries, resulting in low precision and recall. This challenge highlights the limitations of existing data processing assumptions when applied to LLM-based query execution. Traditional optimization techniques are exclusively cost-based, while LLMs also present challenges over the output quality, and rely on catalog metadata, unavailable in this context. Our results show that traditional query optimization principles fail to generate the best plans in terms of result quality. To address this issue, we present a novel approach for optimizing SQL query results by introducing techniques tailored for LLMs. We present Galois, an intermediary between SQL queries and LLMs, treating the latter as a storage layer. We extend traditional optimization strategies to incorporate alternative physical operators designed for LLM-based query execution. Given the absence of a conventional data catalog, we introduce confidence-based metadata collection techniques for query optimization. Our results demonstrate the effectiveness of using such metadata in logical and physical optimization, ultimately showing that Galois successfully balances the trade-off between result's quality and execution cost in querying textual documents.

**Keywords**

Large Language Models, SQL, Query Optimization, Data Extraction

## 1. Introduction

Large Language Models (LLMs) have proved to be a valid solution in direct question answering and data processing tasks [1]. Solely through the usage of Natural Language (NL) or SQL-like queries, users can extract meaningful and structured data exploiting both the internal representations within these models, in fact querying the *parametric knowledge* in the LLM, and external sources, via an *in-context learning* setting, e.g., RAG. Consequently, these models can be seamlessly integrated into mainstream data retrieval and elaboration pipelines, eliminating the need to resort to traditional user-centric extraction methods, which require either data annotation efforts [2] or manually crafted extraction pipelines [3].

**The Problem with Data Outputs.** Despite being able to process simple NL queries, more

*SEBD 2025: 33rd Symposium on Advanced Database System, June 16-19, 2025 - Ischia, Italy*
dario.satriani@unibas.it (D. Satriani); enzo.veltri@unibas.it (E. Veltri); donatello.santoro@unibas.it (D. Santoro); papotti@eurecom.fr (P. Papotti)
0009-0002-1668-7866 (D. Satriani); 0000-0001-9947-8909 (E. Veltri); 0000-0002-5651-8584 (D. Santoro); 0000-0003-0651-4128 (P. Papotti)

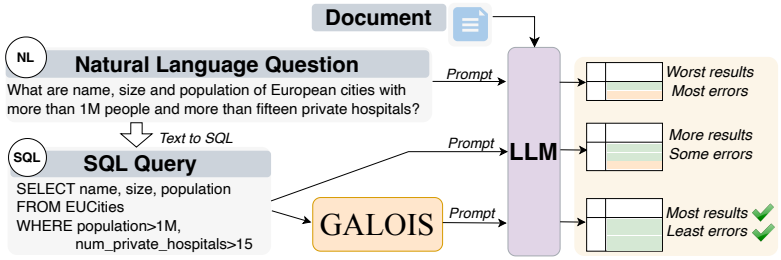


**Figure 1:** Methods for querying European city data from text documents as LLM input. GALOIS executes SQL queries over LLMs with logical and physical optimizations for more accurate and complete results than direct natural language or SQL prompts.

complex instances that require structured results as output often prove to be a challenge for those models [4]. An example is shown in Figure 1. Given as in-context input a textual document about private hospitals and the NL question: "What are names, size and population of European cities with more than 1M people and more than fifteen private hospitals?", the correctness and completeness of the output result are unsatisfactory[1]. Recent advancements allow LLMs to process SQL queries directly from prompts, also by removing natural language ambiguities [5, 6], obtaining more precise answers compared to the corresponding questions in NL. However, feeding the corresponding SQL query:

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

improves results w.r.t. natural language, but still lacks correctness and completeness. These limitations underscore the inherent design constraints of LLMs. If prompted with queries that require complex reasoning, such as with multiple conditions or aggregates, the models are not able to produce a satisfactory result. Therefore, to get the best possible results, we argue that a database management system should be used to handle the query execution while using the LLM exclusively as a storage layer, as shown in the bottom part of Figure 1.

**Challenges.** This approach comes with a series of challenges. First, when querying LLMs with SQL, a trade-off between the accuracy of the results and the execution costs must be considered. While direct execution of SQL queries comprises the most cost-effective approach, decomposing these queries into a sequence of small operator executions, akin to the plans in a DBMS, yields superior result quality. As a result the traditional optimization principles, primarily focused on cost, are only partially applicable. Additionally, LLMs do not provide access to crucial metadata such as schema details, column statistics or histograms, significantly complicating query optimization efforts.

**The Framework.** GALOIS [7, 8] is a query optimization framework designed to address the challenges of optimizing SQL queries for LLMs. It defines a novel physical "Scan" operator crafted for interfacing with LLMs, which balances the efficiency and accuracy of data retrieval. Furthermore, it introduces a cost/quality model and accompanying optimization techniques that balance execution cost with query accuracy, thus enhancing the overall retrieval quality. Finally, it exposes dynamic methods for acquiring essential metadata during query execution,

[1]We assume here that the entire document fits the LLM input context.


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

focusing on estimating the confidence of LLM output to bridge the gap typically filled by catalog information. Experimental evaluations show the effectiveness of the proposed approach, reporting improvements in result quality while maintaining a competitive edge in terms of resource efficiency. While Galois can also handle querying the LLMs over their parametric knowledge [8], in this work we focus on the in-context learning scenario, where the textual documents are fed as input to the LLM.

## 2. Problem Formulation and Challenges

The core challenge is executing SQL queries that cannot be answered by existing databases but can be resolved using textual documents. The goal is to feed the queries directly to the LLM that acts as an extractor, returning structured data from unstructured text. This setting combines traditional DBMS challenges (efficiency) with LLM-specific ones (output quality and token cost). Traditional optimizations focused on performance fail to ensure high-quality results with LLMs. A major limitation is the lack of a catalog. Traditional metadata (e.g., column stats) nor LLM-specific metadata is readily available, yet both are essential for optimization. Our solution enables dynamic data extraction from documents provided at runtime, leaving the broader case of accessing LLM pretraining knowledge to the full paper [8].

**Logical Level.** In traditional DBMSs, logical optimization focuses on minimizing resource usage, such as CPU and memory. With LLMs, however, output quality becomes a key metric.

Consider query *Q1* in the top part of Figure 2. The simplest logical plan uses a single prompt to retrieve all tuples (operator **n1**), followed by a filtering step outside the LLM (operator **n2**).

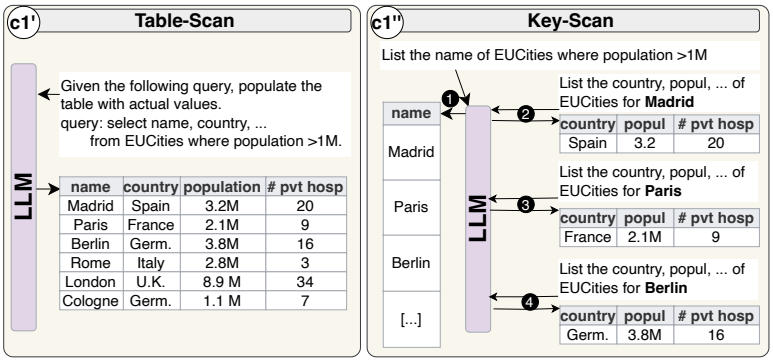


**Figure 3:** Two alternative physical operators for implementing logical operator **c1** in Figure 2. Table-scan (**c1'**) gets entire tuples, while Key-Scan (**c1"**) gets key values first.

This yields high precision but low recall as only one tuple is returned.

A key optimization is *condition pushdown*. While it reduces execution cost by limiting LLM interactions, it can create complex prompts that degrade quality. For instance, pushing both *population* and *hospital* conditions (operator **p1**) saves tokens but overwhelms the LLM's ability to handle multi-faceted queries. Even pushing only the most selective condition (operator **s1**) might lead to errors preventing the identification of relevant data and ultimately producing an empty result. Achieving a balance between cost and quality requires analyzing each query component's effect on the LLM. In the last example in Figure 2 (**c1**), pushing the condition for which the model is most confident leads to the best results.

**Physical Level.** The next challenge is adapting query execution to treat LLMs as the data storage layer. In the physical plan, this requires new operators, as data is accessed via natural language prompts. Prompt generation becomes critical, balancing variability in data quality and cost. Unlike traditional systems where a single command retrieves data, querying LLMs demands more flexible strategies.

For instance, a straightforward scanning technique retrieves the entire tuple set directly with a prompt, as in the *Table-Scan* operator **c1'** in Figure 3 that retrieves all tuples in one prompt, minimizing interactions but often producing low-quality results. 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 improves quality with more focused prompts but increases the number of LLM calls. These trade-offs highlight the complexity of designing prompt-based operators that balance execution cost with result quality.

Moreover, LLMs face two key limitations in knowledge extraction. First, they generate the most likely next word based on prior text, often favoring frequent values from their training data—making rare information harder to retrieve without multiple interactions. Second, their output length is constrained, so a single response often fails to capture all relevant data.

These challenges highlight the need for new operators and optimization strategies tailored to LLMs. While traditional methods provide a foundation, they must be extended to support LLM-based data processing. Crucially, dynamic metadata generation is essential, as it directly impacts both logical and physical query planning.

## 3. Methodology

In traditional DBMSs, query execution involves generating a logical plan to define the steps needed to produce the results, followed by a physical plan to determine how the query will be executed. However, querying LLMs requires different strategies to effectively manage logical and physical optimization.

| 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 | $\sigma_{cond}$ | Select tuples w.r.t. cond |
| Projection | $\pi_{attrs}$ | Extract attrs from tuples |
| Join | $\bowtie_{cond}$ | Join two table given cond |
| Distinct | $\delta$ | Removes duplicate tuples |
| Grouping | $\gamma_f$ | Groups tuples on common values and compute $f$ over groups |

**Table 1**
Logical Operators Supported by Galois

**Logical Level.** Given a user query $q$ and schema $s$, Galois decomposes $q$ into an initial unoptimized logical plan using the relational algebra operators shown in Table 1. All operators are executed in memory, except for *LLMScan* and its variant, *Filter-LLMScan*. Once data is retrieved using the Scan operator, all other operations proceed without involving LLM interaction.

Traditional DBMSs use Scan operators to convert stored tables into rows. In contrast, Galois's LLMScan fetches data on the fly by prompting an LLM, thus, the LLMScan does not have complete data readily available. This precludes the applicability of many traditional optimizations.

To address this, Galois generates multiple logical plans by using the pushdown of the selection conditions directly into a different scan operator (Filter-LLMScan). It supports three pushdown strategies: 1) *no-pushdown*, where the LLMScan retrieves all the tuples; 2) *all conditions pushdown*, where the Filter-LLMScan retrieves only tuples matching all the conditions; 3) *single pushdown*, where the Filter-LLMScan retrieves the tuples that match a single condition.

Despite the inherent limitations in the considered pushdown strategies, the number of possible logical plans for a given query can still be substantial. A natural strategy for reducing token consumption is to push down all filter conditions into the LLMScan operator. While it minimizes 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 because simpler prompts reduce complex reasoning and the risk of hallucination.

To guide logical optimization, Galois employs the LLM itself as a source of information for estimating the most efficient logical plan. Using a classification prompt (that uses the table schema $s$ and the query $q$), it estimates a "high" or "low" confidence score for each atom in the WHERE clause. All the atoms with "high" confidence are pushed down in the Filter-LLMScan.

**Physical Level.** The goal of *LLMScan* and *Filter-LLMScan* is to extract structured data from the LLM, which is then processed by the logical plan. Galois achieves this by generating natural language prompts tailored to the query.

The natural strategy, called *Table-Scan*, prompts the LLM to extract all relevant tuples for

a given query $q$. Given a logical plan $p$ and schema $s$, GALOIS uses an iterative prompting approach for each LLMScan operator in $p$. The initial prompt requests all table attributes and instructs the LLM to return results in JSON format, aligned with the schema derived from $s$. Due to space limits, a template for this prompt is shown in the full paper [8].

LLMs are known to benefit from techniques like Chain-of-Thought (CoT) prompting [9], which improve reasoning and output quality. Based on this, we introduce *Key-Scan*, an alternative scan operator designed to improve data extraction accuracy. Inspired by CoT, Key-Scan splits data collection into two steps: first, GALOIS retrieves all key values for the target table; then, for each retrieved key value, it fetches the corresponding attribute values. For some queries, this two-step approach improves quality results by asking simpler and more specific prompts to LLM. Due to space limits, the example prompt is shown in the full paper [8].

For each SQL query $q$, GALOIS must choose between two scan strategies: *Table-Scan* and *Key-Scan*. While *Key-Scan*, based on CoT prompting, generally offers higher accuracy, *Table-Scan* can outperform it in cases where additional context (i.e., the full table structure) helps the LLM generate more reliable 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 we estimate the *confidence* of the model in returning factual data in the Scan operation. To do so we prompt the LLM to gather a confidence value between 0 and 1 ($conf(q)$). To leverage the strengths of both *Key-Scan* and *Table-Scan*, we introduce a confidence threshold, $\tau$. If $conf(q) > \tau$, we use *Key-Scan*; otherwise, *Table-Scan* is selected. 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.

## 4. Experimental Results and Conclusions

GALOIS can be seamlessly integrated with frameworks like Retrieval Augmented Generation (RAG), which combine traditional information retrieval with the generative power of LLMs. This experiment demonstrates how GALOIS's optimizations enhance both the accuracy and efficiency of LLM-based data retrieval in in-context learning settings.

To evaluate the ability to handle novel information, we use the PREMIER and FORTUNE datasets, containing 60 and 500 documents respectively. These corpora are designed to include information not present in the LLM's training data. PREMIER includes reports from the first six match-days of the 2024–2025 Premier League season (scraped from BBC News), while FORTUNE comprises data on the 2024 Fortune 500 companies, collected from Kaggle. Both datasets are processed via a RAG engine built using LangChain4j. Each document is divided into text segments with 128 tokens for PREMIER and 400 tokens for FORTUNE. Segments are encoded using the "WhereIsAI/UAE-Large-V1" model [10] and stored in a vector database. At runtime, the prompt is augmented with the query schema and the 50 most relevant segments, retrieved via embedding similarity. All models use the same retrieved chunks for consistency. As LLM

we use LlaMa 3.1 70B.

We evaluate variants of Galois against three baselines: NL, SQL, and Palimpzest [11] (PZ). In NL, the LLM is directly prompted with a natural language question; in SQL, the input is an SQL query and both NL and SQL expect structured data as output. Palimpzest [11] shares our data extraction use case but 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 fair 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. 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 optimizations based on the LLM confidence.

Each experiment comprises a query $q$ and a database $d$. We compute the expected tuple set by executing $q$ over $d$ ($t_{exp} = q(d)$). We aim to compare $t_{exp}$ with the tuple set produced by executing $q$ on Galois ($t_{act}$). As quality metrics to compare those two sets of tuples, we adopt metrics used to benchmark SQL queries on LLMs [12, 13]:

- F1-Cell: we compute the F1 score among the set of cells in $t_{act}$ w.r.t those in $t_{exp}$. The rationale of this metric is to evaluate the results considering only the cell values.
- Cardinality: measures 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: measures the fraction of tuples in $t_{exp}$ present in $t_{act}$ comparing tuples as whole. Tuple Constraint is 1.0 if $t_{exp}$ and $t_{act}$ have the same schema, cardinality, and cell values, making it stricter than F1-Cell, as it requires not only that the same values appear but also within the same corresponding tuples.
- AVG-Score combines F1-Cell, Cardinality (that are soft metrics that do not consider tuple schema at all), and Tuple Constraint (that is an hard metric) into a single metric, averaging them to provide a comprehensive comparison score.

To prevent false negatives, we normalize cell values in $t_{act}$ and $t_{exp}$ before evaluation. We use string similarity (Edit Distance [14]) with a 10% threshold to match similar values, such as "Bill Clinton" and "Bill J. Clinton". For numerical values, we allow a 10% difference w.r.t. the expected value. We use simple and efficient comparisons, a more sophisticated implementation for matching tuples could resort to Entity Resolution methods [15, 16, 17] or tuples matching [18].

Results are shown in Table 2. NL and SQL perform poorly due to the challenges of 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 Galois versions outperform them, with Galois $_F$ scoring higher. Palimpzest achieves the highest quality, but at a cost 11 times higher than Galois $_F$. Palimpzest's high costs are due to its

| 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 |

**Table 2**
Quality Results and Costs for RAG application over Premier and Fortune datasets. In bold (italic) the best ($2^{nd}$) result for each metric.

multi-step processing, which involves repeated LLM interactions. In contrast, in Galois only the scan operator interacts with the LLM (once per query), thus reducing computational costs.

In summary, Galois effectively integrates with in-context learning (like RAG), achieving comparable quality to the best baseline while reducing token costs. Moreover, Galois only requires users to write SQL scripts. Experiments with queries executed over the parametric knowledge of the LLMs confirm these results and are reported in the full paper [8].

**Conclusions.** We study the problem of querying textual documents through SQL queries over LLMs. Our system 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 [19]. Alternatively, an LLM-first architecture poses intriguing possibilities with new challenges, e.g., 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 [20] and its issues with long inputs [21], recent advancements are overcoming this constraint [22, 23, 24].

Another promising research direction involves support for queries spanning multiple modalities, such as text, image, and structured data [25]. This integration could enable users to extract insights from diverse data formats in a unified querying framework [11].

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 [26].

Our work shows the increasing need to refine LLM confidence estimation mechanisms [27, 28]. Improving confidence estimation 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.

**Acknowledgments**. Veltri was partially supported by the TECH4YOU project.

## Declaration on Generative AI

The authors have not employed any Generative AI tools.

## References


[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: NeurIPS, 2020.

[2] L. Chiticariu, M. Danilevsky, Y. Li, F. Reiss, H. Zhu, SystemT: Declarative text understanding for enterprise, in: NAACL, Association for Computational Linguistics, 2018, pp. 76–83. doi:10.18653/v1/N18-3010.

[3] C. Zhang, C. Ré, M. J. Cafarella, J. Shin, F. Wang, S. Wu, Deepdive: declarative knowledge base construction, Commun. ACM 60 (2017) 93–102. doi:10.1145/3060586.

[4] F. Petroni, T. Rocktäschel, S. Riedel, P. Lewis, A. Bakhtin, Y. Wu, A. Miller, Language models as knowledge bases?, in: K. Inui, J. Jiang, V. Ng, X. Wan (Eds.), Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Association for Computational Linguistics, Hong Kong, China, 2019, pp. 2463–2473. doi:10.18653/v1/D19-1250.

[5] E. Veltri, G. Badaro, M. Saeed, P. Papotti, 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, 2023, pp. 450–463. doi:10.1109/ICDE55515.2023.00041.

[6] E. Veltri, D. Santoro, G. Badaro, M. Saeed, P. Papotti, Pythia: Unsupervised generation of ambiguous textual claims from relational data, in: SIGMOD, ACM, 2022, pp. 2409–2412. doi:10.1145/3514221.3520164.

[7] M. Saeed, N. D. Cao, P. Papotti, 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, 2024, pp. 365–372. doi:10.48786/EDBT.2024.32.

[8] D. Satriani, E. Veltri, D. Santoro, S. Rosato, S. Varriale, P. Papotti, Logical and physical optimizations for sql query execution over large language models, Proc. ACM Manag. Data 3 (2025). doi:10.1145/3725411.

[9] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Advances in Neural Information Processing Systems, volume 35, Curran Associates, Inc., 2022, pp. 24824–24837.

[10] X. Li, J. Li, Angle-optimized text embeddings, arXiv preprint arXiv:2309.12871 (2023).

[11] C. Liu, M. Russo, M. Cafarella, L. Cao, P. B. Chen, Z. Chen, M. Franklin, T. Kraska, S. Madden, G. Vitagliano, A declarative system for optimizing ai workloads, 2024. URL: https://arxiv.org/abs/2405.14696. arXiv:2405.14696.

[12] S. Papicchio, P. Papotti, L. Cagliero, Qatch: Benchmarking table representation learning models on your data, in: NeurIPS (Datasets and Benchmarks), 2023.

[13] A. Biswal, L. Patel, S. Jha, A. Kamsetty, S. Liu, J. E. Gonzalez, C. Guestrin, M. Zaharia, Text2sql is not enough: Unifying ai and databases with tag, 2024. URL: https://arxiv.org/

abs/2408.14717. arXiv:2408.14717.
[14] A. Marzal, E. Vidal, Computation of normalized edit distance and applications, IEEE transactions on pattern analysis and machine intelligence 15 (1993) 926–932.
[15] G. Papadakis, E. Ioannou, E. Thanos, T. Palpanas, The four generations of entity resolution, Springer, 2021.
[16] G. Simonini, L. Zecchini, S. Bergamaschi, F. Naumann, et al., Entity resolution on-demand, Proceedings of the VLDB Endowment 15 (2022) 1506–1518.
[17] M. Buoncristiano, G. Mecca, D. Santoro, E. Veltri, Detective gadget: Generic iterative entity resolution over dirty data, Data 9 (2024). doi:10.3390/data9120139.
[18] B. Glavic, G. Mecca, R. J. Miller, P. Papotti, D. Santoro, E. Veltri, Similarity measures for incomplete database instances, in: Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28, OpenProceedings.org, 2024.
[19] J. Yao, H. Li, Y. Liu, S. Ray, Y. Cheng, Q. Zhang, K. Du, S. Lu, J. Jiang, 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.
[20] G. Badaro, M. Saeed, P. Paolo, 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.
[21] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, P. Liang, 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.
[22] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, L. Qiu, 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, 2023, pp. 13358–13376. doi:10.18653/v1/2023.emnlp-main.825.
[23] Y. Liu, H. Li, Y. Cheng, S. Ray, Y. Huang, Q. Zhang, K. Du, J. Yao, S. Lu, G. Ananthanarayanan, M. Maire, H. Hoffmann, A. Holtzman, J. Jiang, Cachegen: Kv cache compression and streaming for fast large language model serving, in: Proceedings of the ACM SIGCOMM 2024 Conference, ACM SIGCOMM '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 38–56. doi:10.1145/3651890.3672274.
[24] G. Corallo, P. Papotti, FINCH: prompt-guided key-value cache compression for large language models, Trans. Assoc. Comput. Linguistics 12 (2024) 1517–1532. URL: https://doi.org/10.1162/tacl_a_00716. doi:10.1162/TACL_A_00716.
[25] M. Urban, C. Binnig, 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, 2024.
[26] A. Floratou, F. Psallidas, F. Zhao, S. Deep, G. Hagleither, W. Tan, J. Cahoon, R. Alotaibi, J. Henkel, A. Singla, A. V. Grootel, B. Chow, K. Deng, K. Lin, M. Campos, K. V. Emani, V. Pandit, V. Shnayder, W. Wang, C. Curino, 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, 2024.
[27] J. Geng, F. Cai, Y. Wang, H. Koeppl, P. Nakov, I. Gurevych, 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), 2024, pp. 6577–6595.
[28] L. Chen, A. Perez-Lebel, F. M. Suchanek, G. Varoquaux, Reconfidencing llms from the grouping loss perspective, arXiv preprint arXiv:2402.04957 (2024).