Optimizing Data Systems for LLM Workloads

Kerem Akillioglu
supervised by M. Tamer Özsu
University of Waterloo
Waterloo, Canada
k2akilli@uwaterloo.ca

ABSTRACT

Large language models (LLMs) are becoming increasingly popular in data analytics because of the enhanced capabilities they provide. This shift introduces LLM-powered queries that rely heavily on model inference. Such workloads pose new challenges for database management systems, which must still deliver low-latency performance for complex queries; and current solutions fail to meet the performance demands of LLM-powered queries. My PhD research aims to fill this gap by proposing a query-aware, hardware-conscious optimization framework to efficiently process LLM-powered queries.

VLDB Workshop Reference Format:
Kerem Akillioglu. Optimizing Data Systems for LLM Workloads. VLDB 2025 Workshop: PhD Workshop.

1 INTRODUCTION

LLMs have emerged as powerful tools for data management [11], offering processing opportunities for unstructured (text-centric) [6, 25] and structured (relational) [17] workloads, both separately and in combination [18, 22, 26]. For unstructured data processing, LLMs show promising opportunities to be used for interpreting and synthesizing information from raw text that enables complex document processing [25], and multi-modal data analytics [6]. LLMs also leverage their broad knowledge to infer or fill in gaps in data: for example, fine-tuned LLMs have been used to impute missing values in sparse datasets [7], and prompting foundation models on data cleaning and integration tasks has achieved state-of-the-art results without task specific training [21]. For structured tables, LLMs have proven extremely useful for "fuzzy" row-level inference because they can interpret context and semantics beyond exact string matching. It is practically relevant for many hard data management tasks. This semantic capability has been shown to significantly improve some of the most challenging data management tasks, such as entity and schema matching [27, 28], and data cleaning [19] by overcoming the limitations of earlier methods that relied solely on exact string matching.

The promise of LLM-driven inference has led to their integration in DBMSs. Major cloud warehouses (e.g. Databricks [1], Amazon Redshift [2], Google BigQuery [3]) now offer LLM-based UDFs for row-wise operations in SQL. As an example, the following query in Figure 1 performs a (i) join operation for movies and their Rotten Tomatoes reviews, (ii) a filter operation to obtain "Fresh reviews", (iii) an LLM call to confirm that the movie is kid-friendly; and (iv) another LLM call to generate a personalized list of recommended movies using the movie metadata and the review text. The order in which these queries execute influences the overall query execution time.

```
SELECT LLM("Recommend movies for the user based on
     {movie information} and {user review}", m.
    movie_info, r.review_content) AS
    recommendations
FROM Movies m
JOIN Reviews r ON r.rotten_tomatoes_link =
m.rotten_tomatoes_link
WHERE
LLM("Analyze whether this movie would be suitable
    for kids based on {movie information} and {
    user review}", m.movie_info, r.review_content)
     == "Yes"
AND r.review_type == "Fresh"
```

Figure 1: Example LLM Query.

Even though LLMs provide enhanced data processing capabilities, scaling to more data points remains the biggest problem. For instance, running the query in Figure 1 on only 17,000 rows takes more than 5 hours to complete using local inference on a single A100 GPU combined with an open-source DBMS. Even if enterprise inference solutions are used, such as Open AI's lightweight model GPT4o-mini [4], it takes 16 minutes to process the whole dataset. In contrast, it takes milliseconds to process a million data rows for relational DBMS. This inefficiency stems from insufficient optimizations for such queries and the need to invoke LLM inference individually for every row.

Although the data-management community excels at large-scale processing, integration between LLMs and DBMSs remains unexplored, and when LLMs are involved, additional concerns arise. For instance, enterprise inference solutions require sending data to third-party providers. For sensitive data, this process creates legitimate privacy issues. Organizations that want the benefits of LLMs without compromising confidentiality, therefore, need to host the model locally. In such settings, LLM+DBMS integrations must be far more heavily optimized to achieve acceptable performance on the available data that has already been stored. LLM workloads are different from conventional SQL queries and diverge from the conventional OLTP/OLAP scenarios that databases traditionally optimize for in three fundamental ways:

Proceedings of the VLDB Endowment. ISSN 2150-8097.

(1) **Inference computation:** Whereas conventional SQL operations are dominated by CPU-friendly scans and joins, LLM inference is FLOP-heavy; each transformer layer reduces to heavy matrix multiplications and memory-intensive attention kernels that achieve near-peak throughput only on highly parallel GPUs or AI accelerators. Consequently, inference computations dominate the query latency.
(2) **Inference-state reuse:** They must manage and cache large key–value attention states or "prompt modules," whose reuse can decrease the time to generate outputs.
(3) **Hybrid structured + unstructured access:** LLM workloads require routinely fetching of external passages or embeddings during generation.

Prior ETL-centric pipelines such as LOTUS, Palimpzest[16], and DocETL[25] operate outside the DBMS, focusing on semantic rewriting or agentic extraction of unstructured documents. They neither optimize GPU scheduling nor focus on query-plan selectivity. My research, in contrast, aims to keep inference inside the data processing engine, including the co-design of queries with new GPU-centric operators for inference operations. The proposed PhD work aims to develop multiple systems optimizations for to efficiently process LLM workloads at scale.

2 RESEARCH GOAL

The designed architecture proposes an efficient approach to optimize LLM workloads for relational DBMSs and integrate them efficiently to boost analytics capabilities. Currently, implementations of LLM+DBMS integrations require the invocation of an LLM for every row, and existing implementations do not adopt systems optimizations to alleviate the high computational overhead, which makes them difficult to apply to large datasets [8, 12, 23]. Moreover, LLM inference is resource intensive and typically orders of magnitude slower than standard query operations, so executing an LLM on each row of a large table can bottleneck the entire query. Further complicating matters are memory constraints (such as the context-window limits within LLMs). Since current implementations are not designed for such workloads, a naive integration of UDFs often leads to timeouts or excessive costs.

This work aims designing an efficient data processing system to handle SQL queries embedded with LLM calls as in the example given in Figure 1. Some of the issues can be addressed by careful engineering designs, but others require new approaches that require research. We aim to tackle these research challenges through the following contributions:

(1) **Query-aware optimization**. We design a query-aware optimization model that scores each query's complexity and routes simple cases to fast, fine-tuned small models while sending harder ones to larger models, thereby balancing accuracy, cost, and throughput.
(2) **Resource-aware query optimization**. We present a resource-aware query optimizer that coordinates CPU–GPU scheduling, batches and shares prompts via KV-cache reuse, and pipelines multi-query LLM inference to transform local model calls into a high-throughput operator.

3 CHALLENGES

LLM inference requires too much time and resources to fit smoothly into the low-latency, high-throughput execution model of a traditional database query, so new strategies are needed to make LLM-powered queries practical and efficient. We propose two techniques: (i) query-aware model optimization to reduce latency by sending simple queries to small, cheap models and reserving large models for hard cases. Resource-aware pipelining keeps the large model but speeds it up through batching, KV-cache reuse, and shared GPU pipelines. The first design reduces the amount of computation the system must perform, while the second accelerates the operations that remain while highlighting the need to control both *what* gets executed and *how* it is executed across CPU–GPU resources.

3.1 Query-aware Optimization

Conventional relational query optimizers are designed to handle operators such as scans, joins, aggregations whose per tuple latencies lie in the microsecond to millisecond range. In contrast, an operator that invokes LLM inference can run for minutes. LLM inference cost is dominated by GPU-intensive matrix multiplies, attention-memory traffic, and cross-device scheduling. Even when the model is accessed through enterprise services such as OpenAI [4] or Anthropic [5], conventional heuristics like join reordering or index selection have little impact because overall latency is still governed by inference. This raises an immediate design question: is a full-scale LLM truly necessary for the call, or could a smaller, faster model deliver adequate quality?

Recent research shows that fine-tuned small language models can perform on par with large language models, offering high accuracy while maintaining low deployment costs for specified tasks [29]. Therefore, an interesting line of research is pushing the LLM's capabilities inside the database system to minimize data movement and leverage the strengths of the DBMSs. The key problem is to understand when to use a smaller model, when to use a larger model, and when to use LLMs, also when LLMs are used, which one to route to.

We propose to adopt query-aware model scaling to address the challenges. In our design, when the system receives a query, a computational complexity score is assigned. For lower complexity difficulty queries, easy queries, a lightweight local model will be used to generate an output. Then, the output will be assessed to check if it matches the predefined quality requirements. If it does, it will be used as the final response. Otherwise, the query will be routed to a more computationally intensive but higher-quality model to produce the final result. The rationale is that certain queries are inherently simpler and can be processed by small models with no or minor quality degradation.

Adaptive query routing approach can choose between small cheap models and larger, accurate ones and considers quality for cost. The approach is referred to as query-aware as it routes queries based on the complexities of each query. The main goal is to address the computational burden of models by allowing easy queries to be processed exclusively by light models that execute faster. This time saving from handling easy queries enables the system to achieve higher serving throughput. We visualize our workflow in Figure 2.

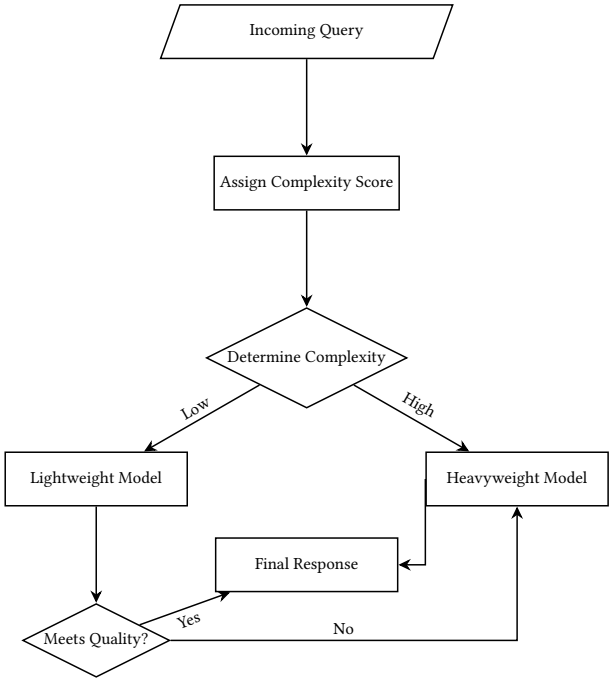

Figure 2: Query-aware routing workflow.

3.2 Resource-aware Query Optimization

When leveraging LLM-based operations in privacy-sensitive settings, local inference is essential. This requires GPU resources to serve models, and given the computational characteristics of LLM workloads, careful coordination between CPU and GPU operations becomes critical. As shown in Figure 1, the query consists of four distinct operations, yet existing systems lack mechanisms to determine their optimal execution order. To address this gap, we propose a query optimization framework that jointly considers CPU-GPU resource allocation and execution ordering, aiming to improve both resource utilization and overall query efficiency.

In these scenarios, the optimizer must consider explicitly about batching, key-value reuse, and prompt overlap, elevating latency-cost trade-offs to first-class planning primitives. Promising directions revolve around exposing both data-flow selectivity and model-level economics to the planner. In this setting, a GPU-aware cost model can estimate latency and cost for every plan alternative by accounting for matrix-multiply FLOPs, KV-cache hit probabilities, and GPU utilization. Then, the optimizer can apply prompt-sharing and batching rewrites that coalesces tuples with overlapping text segments, enabling reuse of attention states and reducing the prefill cost.

This design not only maximizes throughput but also unlocks deeper multi-query optimization: duplicate prompts from separate SQL statements share the same prefill stage. Shared KV caches aim to avoid recomputation, and speculative partial decodes aim to satisfy multiple queries with the same prefix while eliminating redundant matrix multiplications. Also, sequence-pipeline parallelism can be used to process several queries concurrently. Each call is split into (i) token prefill, (ii) attention-based decode steps, and (iii) lightweight post-processing. With this design, a scheduler interleaves micro-batches from many queries on the same GPU, similar to vectorized CPU engine processes tuples through its pipelines.

4 LONG TERM RESEARCH DIRECTIONS

Our design presents an efficient method for optimizing LLM workloads in relational DBMSs, and its integration into full-scale systems opens several promising avenues for future research. In this section, we will describe the open research problems that we have identified and how we plan on addressing them in the long term.

ML systems community proposed various caching optimizations to accelerate LLM inference. Relevant work includes Prompt Cache, which reuses modular attention states across prompts that share overlapping text segments (such as system prompts or document context) [13]. Also, semantic caching is proposed to save the embedding and answer of each past LLM query, then serves that cached answer whenever a new query's embedding is semantically similar enough, eliminating redundant model calls and thus reducing latency and cost [24].

In the long term, we plan to investigate how semantic and prompt caching can further accelerate the execution of LLM-powered queries. By returning results directly from cache whenever a new query's embedding matches a stored one, semantic caching can avoid unnecessary LLM calls or round-trips to base tables while complementing our optimizer. Exploring this direction also encourages us to re-examine classic data-management tools such as indexing, materialized views, and view-maintenance policies to adapt them for the workloads dominated by LLM inference.

5 RELATED WORK

Recently retrieval-augmented generation (RAG) has been widely used to overcome the narrow context windows of language models. By fetching only the most relevant data from large datasets, RAG allows LLMs to remain within prompt-length limits while still grounding their output in the necessary information [14, 15]. We are not concerned with using LLMs for RAG operations. Instead, we aim to optimize the DBMS+LLM interaction by enabling LLM-powered SQL queries to be performed efficiently.

LLMs are also gaining significant attention for being applied to structured tabular data tasks [10]. Recent studies have proposed LLM-driven approaches for automatically generating new table columns or features [20], for inferring and enriching existing data in tables [9] and data cleaning [19]. We are not concerned with using LLMs for such tasks, instead, our proposal aims to execute LLM-powered queries performantly.

To efficiently process LLM-powered SQL queries at scale, researchers introduce optimizations like prompt prefix-sharing and row-reordering [17]. Also, declarative systems have been proposed to optimize LLM-powered analytical queries, they process unstructured and structured data together [22, 26], and use declarative querying primitives to process data with LLMs. Our work diverges by embedding model calls inside the relational optimizer and proposing GPU-aware plan rewrites. Whereas the above systems treat inference latency as fixed and external, we make it variable and optimizable, introducing novel state-sharing and batch-scheduling mechanisms. Our design complements these systems to efficiently

support their workloads as it is orthogonal to these solutions, and can be adopted with such frameworks.

6 CONCLUSION

This paper outlines the key research challenges in integrating large-model inference into relational DBMSs. We present a potential query-aware optimization strategy that can toggle between small and large models under user-defined quality bounds and discuss how GPU-level batching, KV-cache reuse, and prompt sharing can improve end-to-end latency without sacrificing accuracy. Our immediate next step is to build a working prototype of the query-aware optimizer, integrate it with an open-source engine, and evaluate how far hardware-aware LLM-query optimization can extend the performance of LLM-powered SQL workloads.

REFERENCES

[1] 2025. AI Functions on Databricks. https://docs.databricks.com/aws/en/large-language-models/ai-functions. Databricks Documentation. Last updated Feb 10, 2025.

[2] 2025. Large Language Models for Sentiment Analysis with Amazon Redshift ML Preview. https://aws.amazon.com/blogs/big-data/large-language-models-for-sentiment-analysis-with-amazon-redshift-ml-preview/. AWS Big Data Blog.

[3] 2025. LLM with Vertex AI: Only Using SQL Queries in BigQuery. https://cloud.google.com/blog/products/ai-machine-learning/llm-with-vertex-ai-only-using-sql-queries-in-bigquery. Google Cloud Blog.

[4] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[5] Anthropic. 2024. Model Card: Claude 3. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf. Model card for Anthropic's Claude 3. Accessed: March 14, 2025.

[6] Hanjun Dai, Bethany Yixin Wang, Xingchen Wan, Bo Dai, Sherry Yang, Azade Nova, Pengcheng Yin, Phitchaya Mangpo Phothilimthana, Charles Sutton, and Dale Schuurmans. 2024. UQE: A Query Engine for Unstructured Databases. arXiv:2407.09522 [cs.DB] https://arxiv.org/abs/2407.09522

[7] Zhicheng Ding, Jiahao Tian, Zhenkai Wang, Jinman Zhao, and Siyang Li. 2024. Data Imputation using Large Language Model to Accelerate Recommendation System. https://api.semanticscholar.org/CorpusID:271213203

[8] Anas Dorbani, Sunny Yasser, Jimmy Lin, and Amine Mhedhbi. 2025. Beyond Quacking: Deep Integration of Language Models and RAG into DuckDB. arXiv:2504.01157 [cs.DB] https://arxiv.org/abs/2504.01157

[9] Yael Einy, Tova Milo, and Slava Novgorodov. 2024. Cost-Effective LLM Utilization for Machine Learning Tasks over Tabular Data. In *Proceedings of the Conference on Governance, Understanding and Integration of Data for Effective and Responsible AI* (Santiago, AA, Chile) *(GUIDE-AI '24)*. Association for Computing Machinery, New York, NY, USA, 45–49. https://doi.org/10.1145/3665601.3669848

[10] Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun (Jane) Qi, Scott Nickleach, Diego Socolinsky, "SHS" Srinivasan Sengamedu, and Christos Faloutsos. 2024. Large language models (LLMs) on tabular data: Prediction, generation, and understanding — a survey. *Transactions on Machine Learning Research* (2024). https://www.amazon.science/publications/large-language-models-llms-on-tabular-data-prediction-generation-and-understanding-a-survey

[11] Raul Castro Fernandez, Aaron J. Elmore, Michael J. Franklin, Sanjay Krishnan, and Chenhao Tan. 2023. How Large Language Models Will Disrupt Data Management. *Proc. VLDB Endow.* 16, 11 (July 2023), 3302–3309. https://doi.org/10.14778/3611479.3611527

[12] FlockMTL. 2024. FlockMTL. http://github.com/dsg-polymtl/flockmtl/releases.

[13] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems* 6 (2024), 325–338.

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

[15] 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 *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 793, 16 pages.

[16] 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]

[17] Shu Liu, Asim Biswal, Amog Kamsetty, Audrey Cheng, Luis Gaspar Schroeder, Liana Patel, Shiyi Cao, Xiangxi Mo, Ion Stoica, Joseph E. Gonzalez, and Matei Zaharia. 2025. Optimizing LLM Queries in Relational Data Analytics Workloads. In *Eighth Conference on Machine Learning and Systems*. https://openreview.net/forum?id=R7bK9yycHp

[18] Shicheng Liu, Jialiang Xu, Wesley Tjangnaka, Sina J Semnani, Chen Jie Yu, and Monica S Lam. 2023. SUQL: Conversational Search over Structured and Unstructured Data with Large Language Models. *arXiv preprint arXiv:2311.09818* (2023).

[19] Zan Ahmad Naeem, Mohammad Shahmeer Ahmad, Mohamed Eltabakh, Mourad Ouzzani, and Nan Tang. 2024. RetClean: Retrieval-Based Data Cleaning Using LLMs and Data Lakes. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 4421–4424. https://doi.org/10.14778/3685800.3685890

[20] Jaehyun Nam, Kyuyoung Kim, Seunghyuk Oh, Jihoon Tack, Jaehyung Kim, and Jinwoo Shin. 2024. Optimized Feature Generation for Tabular Data via LLMs with Decision Tree Reasoning. arXiv:2406.08527 [cs.LG] https://arxiv.org/abs/2406.08527

[21] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911* (2022).

[22] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. 2024. LOTUS: Enabling Semantic Queries with LLMs Over Tables of Unstructured and Structured Data. arXiv:2407.11418 [cs.DB]

[23] pgAI. 2024. pgAI. https://github.com/timescale/pgai.

[24] Sajal Regmi and Chetan Phakami Pun. 2024. GPT Semantic Cache: Reducing LLM Costs and Latency via Semantic Embedding Caching. arXiv:2411.05276 [cs.LG] https://arxiv.org/abs/2411.05276

[25] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2025. DocETL: Agentic Query Rewriting and Evaluation for Complex Document Processing. arXiv:2410.12189 [cs.DB] https://arxiv.org/abs/2410.12189

[26] Matthias Urban and Carsten Binnig. 2024. ELEET: Efficient Learned Query Execution over Text and Tables. *Proc. VLDB Endow.* 17, 13 (Sept. 2024), 4867–4880. https://doi.org/10.14778/3704965.3704989

[27] Yu Wang, Luyao Zhou, Yuan Wang, Zhenwan Peng, and Surya Prakash. 2024. Leveraging Pretrained Language Models for Enhanced Entity Matching: A Comprehensive Study of Fine-Tuning and Prompt Learning Paradigms. *Int. J. Intell. Syst.* 2024 (Jan. 2024), 14. https://doi.org/10.1155/2024/1941221

[28] Yunjia Zhang, Avrilia Floratou, Joyce Cahoon, Subru Krishnan, Andreas C. Müller, Dalitso Banda, Fotis Psallidas, and Jignesh M. Patel. 2023. Schema Matching using Pre-Trained Language Models. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. 1558–1571. https://doi.org/10.1109/ICDE55515.2023.00123

[29] Zeyu Zhang, Paul Groth, Iacer Calixto, and Sebastian Schelter. 2025. A Deep Dive Into Cross-Dataset Entity Matching with Large and Small Language Models.. In *EDBT*, Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic (Eds.). OpenProceedings.org, 922–934. http://dblp.uni-trier.de/db/conf/edbt/edbt2025.html#ZhangGCS25