

LLM-Enhanced Data Management [Vision]

Xuanhe Zhou

Tsinghua University

zhouxuan19@mails.tsinghua.edu.cn

Xinyang Zhao

Tsinghua University

xy-zhao20@mails.tsinghua.edu.cn

Guoliang Li

Tsinghua University

liuguoliang@tsinghua.edu.cn

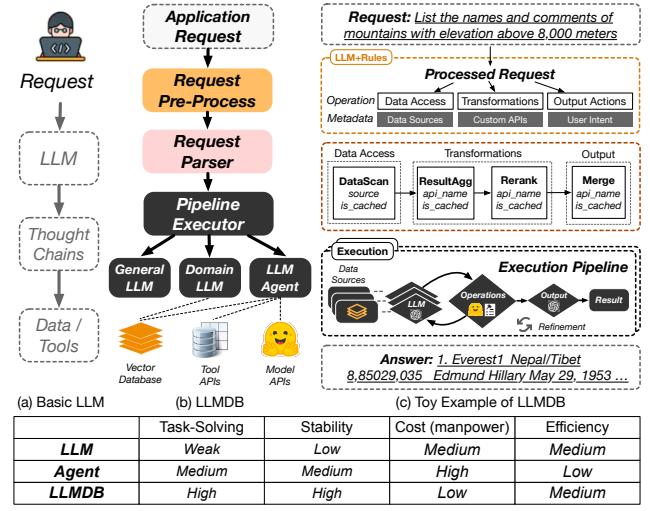
ABSTRACT

Machine learning (ML) techniques for optimizing data management problems have been extensively studied and widely deployed in recent five years. However traditional ML methods have limitations on generalizability (adapting to different scenarios) and inference ability (understanding the context). Fortunately, large language models (LLMs) have shown high generalizability and human-competitive abilities in understanding context, which are promising for data management tasks (e.g., database diagnosis and data analytics). However, existing LLMs have several limitations: hallucination, high cost, and low accuracy for complicated tasks. To address these challenges, we design LLMDB, an LLM-enhanced data management paradigm which has good generalizability and high inference ability while avoiding hallucination, reducing LLM cost, and achieving high accuracy. LLMDB embeds domain-specific knowledge to avoid hallucination by LLM fine-tuning and prompt engineering. LLMDB reduces the high cost of LLMs by vector databases which provide semantic search and caching abilities. LLMDB improves the task accuracy by LLM agent which provides multiple-round inference and pipeline executions. We showcase three real-world data management scenarios that LLMDB can well support, including query rewrite, database diagnosis and data analytics. We also summarize the open research challenges of LLMDB.

1 INTRODUCTION

Machine learning algorithms have been widely adopted to optimize data management problems, e.g., data cleaning [22], data analytics [5], query rewrite [29], database diagnosis [14]. However, traditional machine learning algorithms cannot address the generalizability and inference problem. For example, existing machine algorithms are hard to adapt to different databases, different query workloads, and different hardware environments. Moreover, existing machine algorithms cannot understand the context and do multi-step reasoning, e.g., database diagnosis for problem detection and root-cause analysis. Fortunately, large language models (LLMs) can address these limitations and offer great opportunities for data management [8–10]. For example, LLMs can be used to diagnose the database problems and help DBAs to find the root causes of slow SQL queries. LLMs can also enable natural language (NL) based data analytics such that users can use NL to analyze their datasets.

However LLMs have three limitations: hallucination, high cost, and low accuracy for complicated tasks. These shortcomings pose significant challenges, even risks, especially when LLMs are used in critical data management steps (e.g., price trend analysis in financial forecasting). Although there are some LLM agent based methods to overcome these challenges, like chains of thought [20, 25–27] and tool-calling functionalities [19]. These works, while impressive, reveal several limitations. First, they heavily rely on LLMs to support nearly every tasks (e.g., designing, coding, and testing in LLM-based software development [13, 16]), leading to instability



(d) Comparison of Three LLM-Enhanced Data Management Paradigms

Figure 1: LLM for Data Management

and a high error rate. For instance, GPT-4 might fail to extract text from slides due to misunderstandings of intent, despite having this capability. Second, for complex tasks like tool calling, extensive training data for specific APIs are required to fine-tune the LLM. This approach is vulnerable to API changes and can result in significant cost inefficiencies [18]. Third, LLM agents still lack the capability to fully utilize knowledge from multiple sources, which are vital to mitigate hallucination issues in LLMs (e.g., serving as evaluation criteria). Thus it calls for a new LLM-enhanced data management paradigm to address these limitations.

Vision of LLM-Enhanced Data Management. Developing a robust LLM-enhanced data management system is essential to harness the full potential of LLMs. There are three main challenges. First, *how to effectively utilize data sources* (e.g., tabular data and various document formats) to reduce LLM hallucination problems (e.g., through knowledge-augmented answering). Second, *how to reduce the LLM overhead?* It is rather expensive to call LLMs for every request. It is important to accurately interpret the intent behind user requests and capture the domain knowledge in order to reduce the iterations with LLMs. Third, the execution of complex data management tasks may involve multiple operations. *How to efficiently manage these operations and pipelines* to enhance both execution effectiveness and efficiency.

Idea of LL MDB. To address these challenges, we propose LL MDB, a general framework for designing and implementing LLM-enhanced data management applications. As shown in Figure 1, LL MDB improves the basic LLM by incorporating a series of modular components. LL MDB embeds domain-specific knowledge to avoid hallucination by LLM fine-tuning and prompt engineering. LL MDB reduces the high cost of LLMs by vector databases which provide

semantic search and caching abilities. LLMDB improves the task accuracy by LLM agent which provides multiple-round inference and pipeline executions. Specifically, LLMDB leverages a combination of *general LLMs*, *domain-specific LLMs*, *LLM agent*, *vector databases*, and *data sources* to handle data management tasks. LLMs provide understanding and inference ability. Users provide domain-specific knowledge and data sources. Domain-specific LLMs fine-tune the general LLMs based on the user-provided data sources, and provide the domain-specific knowledge. Vector databases generate embeddings of user-provided data sources, retrieve top- k document segments, take them as prompts and input them to LLMs. With the domain-knowledge prompts, LLMs can provide better answers. Moreover, for complicated tasks, LLM agent generates a pipeline with multiple operations to process the task.

Contributions. We make the following contributions.

- (1) We design an LLM-enhanced data management system, which has generalizability and high inference ability while avoiding hallucination, reducing LLM cost, and achieving high accuracy.
- (2) We propose a domain-knowledge embedding and semantic search method to address the hallucination problem. We adopt vector databases to reduce the LLM overhead. We design a pipeline generation and execution framework to optimize complicated tasks.
- (3) We conduct case study on three typical data management applications using LLMDB, and discuss the system advantages.
- (4) We summarize research challenges of LLM-enhanced data management, including LLMDB operation design, pipeline executor optimization, embedding selection, and knowledge augmentation.

2 LLMDB PARADIGM

2.1 LLMDB Architecture

LLMDB has five important components, including *general LLMs*, *domain-specific LLMs*, *LLM executor agent*, *vector databases*, and *data source manager*. Specifically, general LLMs provide understanding and inference ability. Data source manager provides domain-specific data and knowledge. Domain-specific LLMs fine-tune the general LLMs based on the user-provided data sources, which are used for request pre-processor and parser that capture the user intent and generate execution pipeline. Vector databases generate embeddings of user-provided data sources, retrieve top- k document segments for user requests, take them as prompts and input them to LLMs. With the domain-knowledge prompts, LLMs can provide high-quality results. Moreover, for complex tasks, LLM agent generates a pipeline with multiple operations to optimize the complex task. LLMDB includes offline preparation and online inference.

Offline Preparation. Given an application requirement, LLMDB first collects and analyzes the information from various data sources. Then LLMDB derives detailed specifications that outline the application objectives, traditional tools (e.g., various database plugins), and the usage scenarios of domain AI models (e.g., Marcoroni-7B model for sentiment analysis [2]).

Online Inference. For a user request, LLMDB first parses the request to detect the intent and operations. Following this, a pipeline of the operations is generated, which is then executed in a graph-style format to ensure efficient processing. The final step involves an iterative process of result evaluation and pipeline re-generation, ensuring the accuracy and relevance of final outcome.

2.2 Offline Preparation

Offline preparation is to initialize LLM-based data management applications before online usage. This involves five main modules.

(1) *Data Source Collection*: This module gathers a variety of data sources. For instance, in diagnosis task, the data can come from maintenance documents, historical logs, and operational records.

(2) *Tool/Model Collection*: Based on the collected data, this module gathers and sets up essential tools and small-scaled AI models. For instance, in database diagnosis, this might include the collection of monitoring tools, analytics tools, and image-to-text (e.g., for program running screenshot) models like vit-gpt2 [1].

(3) *Keyword-API Mapping*: This module creates a mapping table (metadata) that links keywords or phrases to relevant tool or model APIs. For example, the keyword “image classification” might be mapped to an API of a convolutional neural network (CNN) model. This mapping enables the automated selection of appropriate tools or models based on the specific requirements of the input request.

(4) *Domain LLM Training*: This module fine-tunes small-scale domain models using the updated data sources, which ensures that the models are well-adapted to the application’s needs, leading to improved accuracy and performance in the target scenarios.

(5) *LLM Alignment*: This module adapts LLM workers to stuffs like interface language updates. For instance, an LLM worker may need to transition from SQL to a knowledge graph language, emphasizing semantic relationships over structured queries for richer, more complex data interactions.

(6) *Vector Data Generation*: This module generates embedding for each data source and inserts the embeddings into vector databases. The vector database provides the vector search ability.

2.3 Online Inference

Given the prepared *tool and model APIs/data sources/metadata*, next we explain how to handle various online requests.

Request Pre-processor. This module prepares the input request so as to ease the following parsing and execution procedures.

(1) *Keyword Recognition*: With the keyword corpus, it emphasizes keywords occurring within the input request, where we can use techniques like synonym matching (e.g., “examination” → “analysis”) and stem matching (e.g., “computing” → “comput-.”) [23].

(2) *Intent Identification*: Apart from basic keyword recognition, it identifies the underlying purpose of a request using semantic analysis. For instance, consider the request “*Identify similar property listings and compare their features*”. A semantic model will not only focus on keywords like “compare” and “features” but also understand the contextual meaning of “similar property listings”. This involves identifying the implicit criteria for similarity (e.g., location, price range) and relevant features for comparison (e.g., number of bedrooms). Thus, the request is transformed into a more actionable form, such as “*Retrieve property listings based on [criteria for similarity] from . . . Compare their [feature 1], [feature 2], . . .*”.

(3) *Dependency Identification* analyzes the grammatical structure to understand how keywords relate to each other, which is vital to the following *semantic-based request segmentation*.

(4) *Interactive Inquiry* allows the application to ask follow-up questions so as to obtain more information that may clarify the user’s intent and required operations (e.g., more entity attributes).

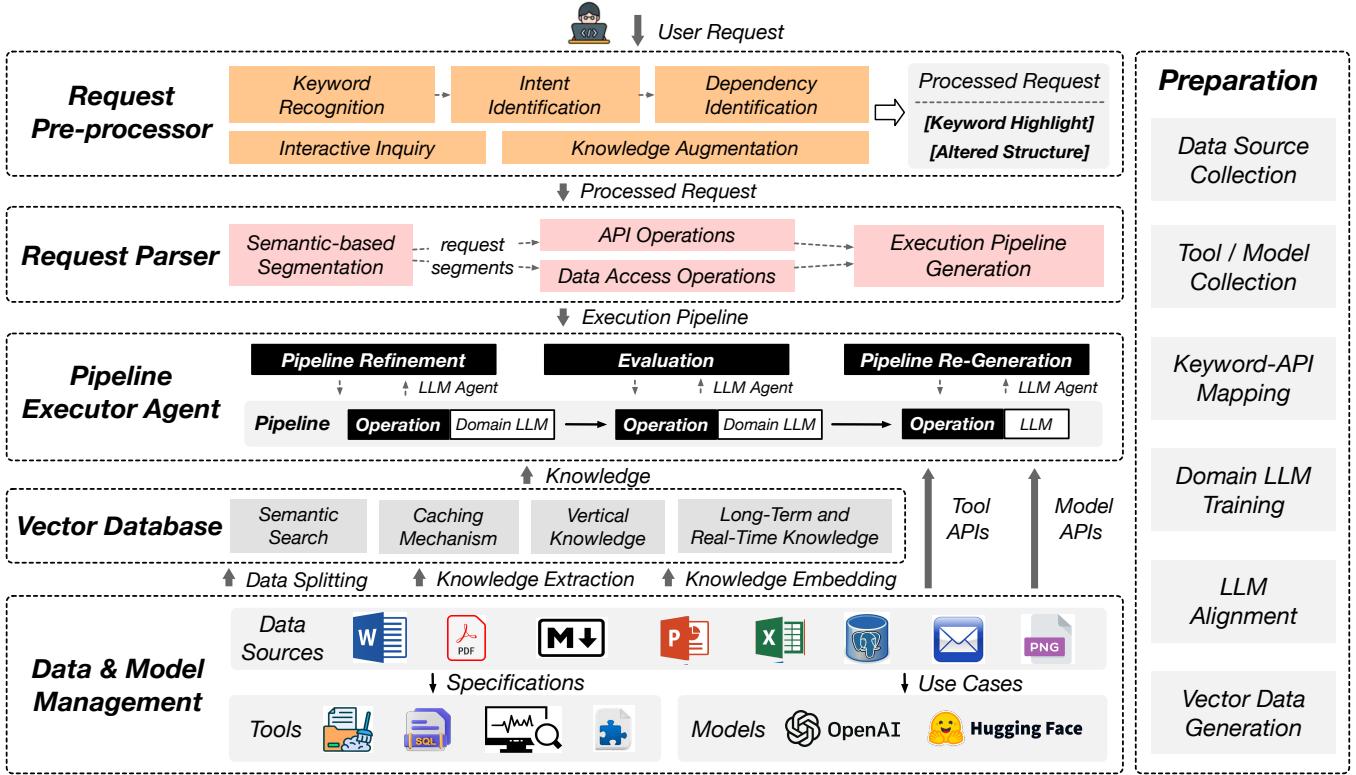


Figure 2: LLMDB Overview

(5) *Knowledge Augmentation* enriches the request with valuable knowledge (e.g., the meaning of *Prometheus* in diagnosis task) or empirical rules (e.g., the criteria of entity matching).

Request Parser. This module interprets the user’s request into an executable operation pipeline.

(1) *Semantic-based Request Segmentation:* Different from basic techniques like *Tokenization*, it splits the input request into meaningful segments. For example, with the Semantic Role Labeling (SRL) technique [15], it assigns roles to words in the pre-processed request to capture their semantic relationships. For instance, in the request “*Find flights from New York to London departing tomorrow*”, it identifies segments such as “*Find flights*”, “*from New York to London*”, and “*departing tomorrow*”, each conveying a specific operation or parameters for the flight search request.

(2) *Operation Mapping:* Based on the keyword-API mapping table, we convert the segments into functional operations (using tool or model APIs) and data access operations (using data sources).

(3) *Execution Pipeline Generation:* These operations form a basic execution pipeline based on their structural logic of the request. For example, for a segmented request $R = [“Identify\ missing\ value”, “in\ dataset\ column”]$, R_0 is designated as the root, helping arrange other operations based on both the order and semantic relations.

Executor Agent. Given the basic pipeline, this module refines the pipeline with underlying information, evaluates the outcomes, and re-generates more effective pipeline (arriving time or cost limit).

(1) *Pipeline Refinement:* For each operation in the pipeline, it enriches with information like (i) which data source to use (e.g., the same document can be stored in both vector database and

elastic search engine [11]) and (ii) which operation orders are most execution-efficient (using methods like genetic algorithm).

(2) *Evaluation:* It assesses the quality of results by executing the current pipeline. For instance, in a data analytics task, we can utilize an LLM to analyze user feedback about the usability and relevance of the analytics results.

(3) *Pipeline Re-Generation:* If the evaluation result is poor, it generates a new execution pipeline. For example, we can use decision trees and neural networks to identify and add any intermediate operations that are potential to derive more reliable results.

Vector Databases. Vector databases are used to improve the execution effectiveness and efficiency from multiple aspects.

(1) *Semantic Argumentation and Search:* When a knowledge query is issued by LLM worker, the vector database (i) integrates context and intent analysis to enrich the query; and (ii) utilizes advanced similarity search algorithms (e.g., graph-network-based embedding for relational knowledge patterns) to ensure high relevance in search results by understanding the semantic difference of queries.

(2) *Caching Mechanism:* We can cache hot user requests and their answers, and when a similar request is posted, we can use vector database to directly answer the request, without involving LLMs.

(3) *Vertical Knowledge:* It serves as a metadata container, enabling unified access and management of multiple data sources.

(4) *Long-term and Real-time Knowledge:* It transforms time-series data into vector embeddings, facilitating the storage and retrieval of real-time and long-term data. This way, it enhances the system’s ability to perform complex analysis over extended periods.

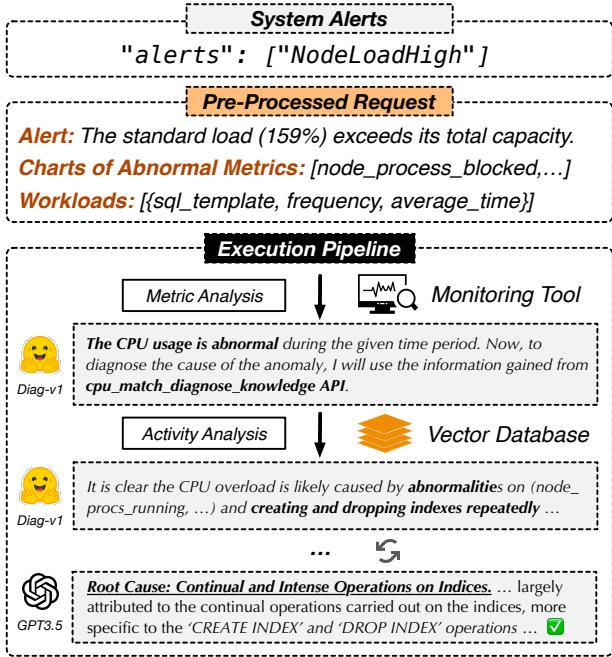


Figure 3: LLMDB for Database System Diagnosis

Data & Model Management. This module functions as the intermediary between *Pipeline Executor* and data sources, tools, and AI models. It includes LLM agents (used during pipeline scheduling and execution), which serve as interfaces to external *tool or model APIs*. This module (i) aligns the system’s internal understanding with external knowledge and (ii) mitigates over-reliance on LLMs.

2.4 Challenges

There are several research challenges in this LLMDB paradigm. First, how to effectively understand the user request and generate the execution pipeline? Second, how to select well-designed execution operations with completeness and minimalism such that they can be combined to generate high-quality execution pipeline? Third, how to design high-quality executor agent that can utilize multiple operations to effectively answer a complicated task? Fourth, how to select effective embedding method that can capture the domain-specific proximity such that can enrich the query with semantic augmentation? Fifth, how to balance LLM fine-tuning and prompt engineering? Fine-tuning can optimize LLM with domain-specific knowledge, but it requires a large volume of high-quality training data; while prompt does not require training data, but it requires to add the prompts for every request. Sixth, how to utilize the LLMDB feedback to iteratively optimize our pipeline.

3 LLMDB FOR SYSTEM DIAGNOSIS

Database system diagnosis focuses on identifying the root causes of anomalies, which inputs various monitoring information and outputs root causes and solutions to fix the root causes. Traditional machine learning methods often treat system diagnosis as a classification problem [7, 14, 28]. These methods (i) rely heavily on large sets of high-quality labeled data, limiting their generalizability to new scenarios (e.g., with different monitoring metrics) and (ii)

cannot generate detailed analysis reports like human DBAs. While there are prototypes leveraging LLMs to solve this problem [30, 31], their implementation requires substantial expertise. In contrast, we explore how to automate the development of LLM-enhanced database system diagnosis.

Domain-Specific Data Preparation. First, LLMDB extracts knowledge from extensive documents, including technical manuals, logs, and case studies. This extracted knowledge is then systematically structured and stored in vector database, facilitating efficient retrieval and analysis. Second, LLMDB sets up a suite of diagnosis tools, each with (i) defined APIs, (ii) text specifications, and (iii) demonstration examples, which are tailored scenarios illustrating the application of these tools in real-world system diagnosis. They not only serve as a guide for understanding the practical application of the tools, but also assist in fine-tuning LLMs and domain models.

User Request Pre-Processing. LLMDB enriches original system alerts to create a comprehensive context for the following analysis. For instance, if an alert indicates high CPU usage, LLMDB further decides relevant factors like *the duration of high usage*, *related processes consuming excessive CPU resources*, and *any concurrent system activities* that might contribute to the anomaly. Additionally, this stage includes workload or slow queries from the system (e.g., logs or system views) during the anomaly period.

Request Parsing. Since the execution pipeline of database system diagnosis is relatively fixed, LLMDB directly maps the enriched alert into a pipeline with operations in the order of (i) high-level metric analysis, (ii) finer-grained metric analysis, (iii) activity analysis, (iv) workload analysis, and (v) solution recommendation. Each operation is accordingly matched with relevant tool APIs (e.g., *index advisor API* for solution recommendation) and knowledge.

Execution Pipeline. The execution pipeline for database diagnosis is primarily focused on two areas: tool callings for gaining specific abnormal status and knowledge-matching-based root cause analysis. First, tool calling sequences perform automated checks on database performance and integrity. For example, one operation may use a performance monitoring tool to identify slow queries, followed by an integrity checker for detecting data corruption. Second, knowledge matching in one operation leverages a vast repository of documented issues and solutions, comparing current anomalies with historical incidents to identify matches or similarities, which is addressed by vector databases. Moreover, there are also operations that need to implement selected solutions. For instance, a pattern of locking conflicts could lead to actions like (i) adjusting transaction isolation levels or (ii) redesigning the database schema.

Model Management. Database diagnosis seldomly has an open corpus. Thus, it is vital to train LLM to learn anomaly root causes from real cases. This involves unsupervised LLM training to recognize and interpret complex patterns from sampled time-series data. For example, an LLM needs to identify that recurring divergences in transactional data could suggest concurrency conflicts. Similarly, LLM needs to analyze log files showing intermittent database access failures and correlate this with issues like network instability.

Research Challenges. First, how to involve human experts in the diagnosis loop. Human experts may provide a vast number of valuable advice when using the diagnosis application in real

cases, which should be learned by caching in vector database or fine-tuning the models. Second, how to integrate multi-modal information sources like text, figures, and videos in diagnosis process. This could provide a more comprehensive context and improve the accuracy of root cause identification.

4 LLMDB FOR DATA ANALYTICS

In many data analytical scenarios, users are unable to write SQL queries and prefer to use natural language (NL) to analyze the data. A challenge here is how to generate analytical results from NL queries. Current LLM works [6, 17] depend on LLMs for crafting these queries, which often leads to errors (e.g., non-executable queries or results that do not meet user requirements). Thus, we showcase how to accurately conduct data analysis with LL MDB.

Domain-Specific Data Preparation. For a data analytic task like “*You are given a table on · · · . Filter the table to show only facilities located in Boston.*”, it mainly aims to generate executable analysis programs with visualizations as outputs. Thus, we prepare (i) data sources like existing analysis requests and example programs in different languages (e.g., SQL, Python, R); and (ii) models that are good at translating corresponding languages. For the data source preparation, we can adopt active learning to refine the LLM’s understanding, focusing on generating diverse examples that cover a wide array of analytics scenarios. For model preparation, we can fine-tune LLMs with training examples that involve Natural Language (NL) instructions, sequences of API calls (code lines), and detailed explanations.

Request Pre-Processing. When handling an incoming analytics request, we first prepare and load the raw tabular data. For example, when processing a dataset with attributes such as sales records, we standardize date formats by writing a program, typically in Python, that utilizes a series of Pandas functions like *drop_duplicates()* and *pd.to_datetime()*.

Request Parsing. The pipelines in data analytics are sequences of APIs tailored for the specific analytics task. Each chain in the pipeline represents a workflow, integrating multiple APIs to perform complex data analysis (e.g., data loading, transformation, and visualization). The operations of these chains utilizes the fine-tuned LLM to predict the most relevant API calls based on the input NL queries and data context (e.g., column correlations within the tabular data), ensuring accurate data processing flow. The following code demonstrates how to translate the natural language query into a data analytics workflow using Pandas. It involves loading the hospital data, filtering it by the condition “located at Toronto”, and computing the average capacity:

```

1 import pandas as pd
2 # Load data
3 data = pd.read_csv('hospital_data.csv')
4 # Filtering data on the city column
5 filter_data = data[data['city'] == 'Toronto']
6 # Calculate average capacity
7 average_capacity = filter_data['capacity'].mean()
8
9 print(f"The average capacity for hospitals located in
    Toronto is: {average_capacity}")

```

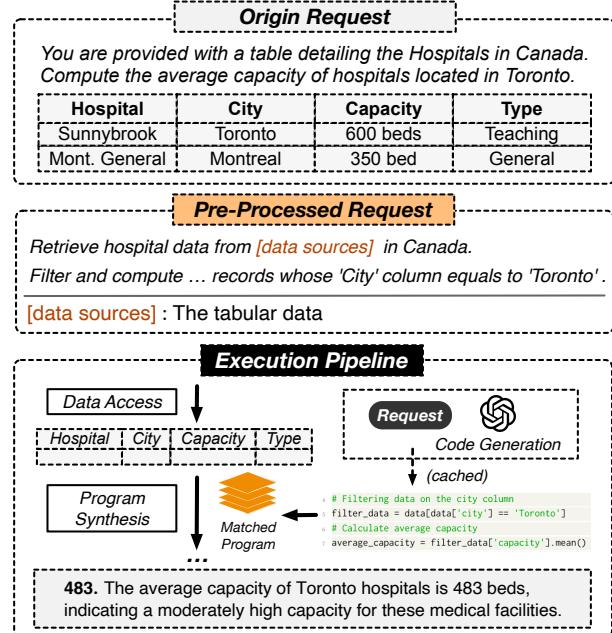


Figure 4: LL MDB for NL-based Data Analytics

Execution Pipeline. Apart from directly running the program derived by the above execution pipeline, we can utilize the vector database to function as a dynamic caching mechanism. That is, the vector database stores and manages the embeddings of natural language (NL) queries. These embeddings are effectively mapped to corresponding program or a sub-sequence of APIs that are essential for executing the analytics task at hand. For instance, when a query related to trend analysis within a given dataset is received, the vector database strategically caches embeddings. These cached embeddings are then mapped to a series of specialized APIs that are proficient in time-series analysis. This approach also enables the system to adaptively learn from incoming queries, thereby progressively optimizing the caching mechanism for future requests.

Research Challenges. First, how to develop algorithms and models that can accurately generate analytical results from natural language queries. Second, how to automatically generate analysis programs, handling diverse data formats, and creating models capable of translating different programming languages to facilitate seamless data analytics. Third, how to efficiently optimize the sequence of API calls in data analytics workflows, i.e., developing algorithms to predict the most effective sequence of API calls.

5 LL MDB FOR QUERY REWRITE

Query rewrite [12] aims to transfer a SQL query to an equivalent but more efficient SQL query. It includes two main types of work. The first type [3, 24], aims to create new rewrite rules from existing SQL pairs [24] or to develop new domain-specific languages (DSLs) to make it easier to implement these rules [3]. However, these methods are limited because they can only handle simple rules (e.g., with up to three operators); but they don’t make use of valuable rewrite experiences in textual documents and DBA experiences. The second type of work [29, 32] tries to improve rewrite performance by changing the order of rule application. Yet, these methods also

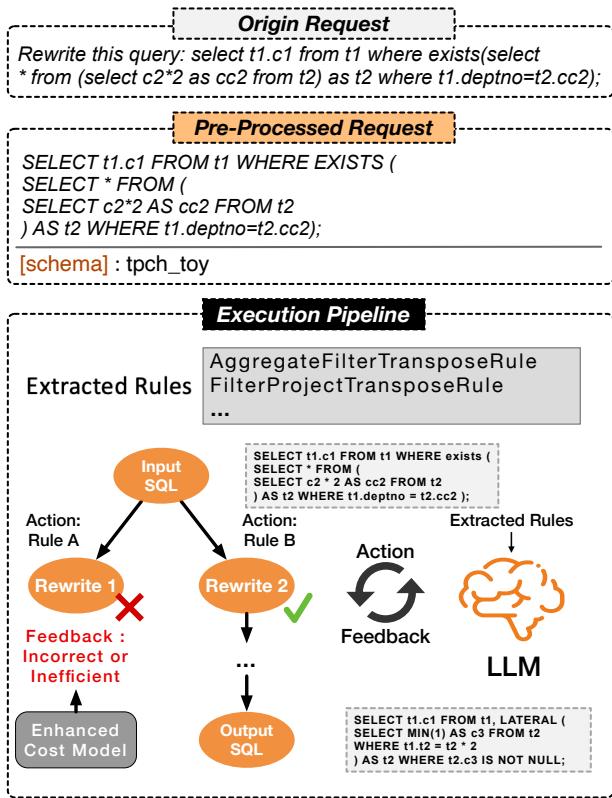


Figure 5: LLMDB for Query Rewrite
do not use structural information to (i) reduce the search space or to (ii) effectively judge the quality of rewrites at the logical query level. In this section, we showcase the methods and techniques used in LLMDB for query rewrite.

Domain-Specific Data Preparation – Rewrite Rules. Given documents like database operation manuals or academic research papers, we can employ LLMs to automatically *extract, verify, and correct* optimization rules. These rules are formulated in a formal language, either as a logical DSL rule or as an executable function (e.g., Java). For each rule, we also create a natural language description. This description includes details like the rule’s purpose, steps, application scenarios, and examples. These descriptions aid LLMs in the executor agent to effectively rewrite queries.

Request Pre-Processing. In most cases, extracting the target query from the rewrite requests is straightforward. This can be done by writing basic regular expression (regex) code to capture complete SQL statements. For example, we can use the regex pattern $r^{(select|\cdot\cdot\cdot|delete).*?}$ to retrieve queries ranging from “select” to “delete” commands.

Request Parsing. We match the extracted SQL queries with the corresponding table schema and applicable rules. These rules may be well-formatted (executed by engines like Calcite [4] or CockroachDB [21]), or require LLM in *Execution Agent* to interpret and follow. Each rule is specifically designed to address different aspects of query rewrite. For example, one rule might aim at optimizing subqueries, while another might be focused on making aggregation processes more efficient. For each extracted query, we organize the sequence of matched rules as a basic execution pipeline.

Executor Agent. Within the execution pipeline, the *Pipeline Refinement* module begins by selecting the most effective strategy of applying these rules. This involves training an order assignment LLM, which learns from experiences such as: (i) rules involving different operators that can be applied simultaneously, and (ii) removing aggregations within a subquery may enable the application of more effective rules that were previously inapplicable. Next, we evaluate the rewritten query by criteria like *whether it has effectively reduced the complexity of operators*. The final step is to adopt the rewritten query that ensures (i) semantically equivalent results and (ii) the lowest estimated cost.

Caching-based Execution. We also support an advanced caching mechanism to enhance the rewrite procedure. After rewriting an SQL query, the vector database caches its embedding together with the rewrite strategies (i.e., rule sequence). This cached embedding is then leveraged to rapidly rewrite the following queries with similar rewrite preferences. For instance, if a cached embedding indicates a query with complex grouping, the vector database guides the system to employ the strategy optimized for grouping operations, thereby saving the rewrite time.

Model and Embedding Management. During online usage, there can be SQL queries where it has low confidence or makes incorrect predictions. For these queries, generate representative samples to (i) train the LLM in *Executor Agent* and (ii) cache in vector databases, where each sample is composed of the original SQL query, rule, rewritten query, and explanatory annotations. Given the scarcity of high-quality samples for query rewriting [33], we can apply methods like active learning to proactively collect such samples.

Research Challenges. First, how to evaluate the quality and overlap of generated rules. Some rules may be redundant or represent similar conditions, which existing LLMs may fail to identify. Second, how to utilize LLM to accurately verify semantic equivalence between the original and rewritten queries, which is a complex problem and lacks a general solution. Third, how to make a light-weight implementation within database kernels. This involves issues like minimizing the computational overhead of LLM inference and reducing the number of applied rules.

6 CONCLUSION

In this paper, we propose an LLM-enhanced data management framework for addressing the limitations of LLMs: hallucination, high cost, and low accuracy. LLMDB can well cover diversified data management applications. By integrating LLMs, vector database, and LLM agent, LLMDB offers enhanced accuracy, relevance, and reliability of data processing. The framework’s potential in areas such as query rewriting, data analytics, and diagnosis is vast, promising significant advancements in various sectors. Part of the source code is open-sourced in <https://github.com/TsinghuaDatabaseGroup/DB-GPT> (and will soon be split off into an independent repository).

ACKNOWLEDGMENTS

We thank Wei Zhou, Zhaoyan Sun, Zhiyuan Liu, and AgentVerse Team for their valuable advice on this vision. Zui Chen and Lei Cao have assisted in designing Figure 5.

REFERENCES

- [1] <https://huggingface.co/nlpconnect/vit-gpt2-image-captioning>. Last accessed on 2024-1.
- [2] <https://huggingface.co/thebloke/marcoroni-7b-v3-gguf>. Last accessed on 2024-1.
- [3] Qiuishi Bai, Sadeem Alsudais, and Chen Li. Querybooster: Improving SQL performance using middleware services for human-centered query rewriting. *Proc. VLDB Endow.*, 16(11):2911–2924, 2023.
- [4] Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J. Mior, and Daniel Lemire. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *SIGMOD*, pages 221–230. ACM, 2018.
- [5] Chengliang Chai, Guoliang Li, Ju Fan, and Yuyu Luo. Crowdchart: Crowd-sourced data extraction from visualization charts. *IEEE Trans. Knowl. Data Eng.*, 33(11):3537–3549, 2021.
- [6] Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. Seed: Domain-specific data curation with large language models. *arXiv e-prints*, pages arXiv-2310, 2023.
- [7] Karl Dias, Mark Ramacher, Uri Shaft, Venkateshwaran Venkataramani, and Graham Wood. Automatic performance diagnosis and tuning in oracle. In *Second Biennial Conference on Innovative Data Systems Research, CIDR 2005, Asilomar, CA, USA, January 4-7, 2005, Online Proceedings*, pages 84–94. www.cidrdb.org, 2005.
- [8] Fatih Kadir Akin et al. Awesome chatgpt prompts. <https://github.com/f/awesome-chatgpt-prompts>, 2023.
- [9] Jules White et al. A prompt pattern catalog to enhance prompt engineering with chatgpt. 2023.
- [10] Pengfei Liu et al. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 2023.
- [11] Clinton Gormley and Zachary Tong. *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. " O'Reilly Media, Inc.", 2015.
- [12] Goetz Graefe. Volcano - an extensible and parallel query evaluation system. *IEEE Trans. Knowl. Data Eng.*, 6(1):120–135, 1994.
- [13] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.
- [14] Minghua Ma, Zheng Yin, Shenglin Zhang, and et al. Diagnosing root causes of intermittent slow queries in large-scale cloud databases. *Proc. VLDB Endow.*, 13(8):1176–1189, 2020.
- [15] Lluís Márquez, Xavier Carreras, Kenneth C Litkowski, and Suzanne Stevenson. Semantic role labeling: an introduction to the special issue, 2008.
- [16] Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, and et al. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.
- [17] Yujia Qin, Shengding Hu, Yankai Lin, and et al. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*, 2023.
- [18] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toollm: Facilitating large language models to master 16000+ real-world apis, 2023.
- [19] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.
- [20] Noah Shimn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection, 2023.
- [21] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, and et al. Cockroachdb: The resilient geo-distributed SQL database. In *SIGMOD*, pages 1493–1509. ACM, 2020.
- [22] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Sam Madden, and Mourad Ouzzani. Rpt: relational pre-trained transformer is almost all you need towards democratizing data preparation. *arXiv preprint arXiv:2012.02469*, 2020.
- [23] Peter D. Turney. A uniform approach to analogies, synonyms, antonyms, and associations. In Donia Scott and Hans Uszkoreit, editors, *COLING*, pages 905–912, 2008.
- [24] Zhaoguo Wang, Zhou Zhou, Yicun Yang, Haoran Ding, Gansen Hu, Ding Ding, Chuzhe Tang, Haibo Chen, and Jinyang Li. Wetune: Automatic discovery and verification of query rewrite rules. In *SIGMOD*, pages 94–107. ACM, 2022.
- [25] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.
- [26] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.
- [27] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izahk Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.
- [28] Dong Young Yoon, Ning Niu, and Barzan Mozafari. Dbsherlock: A performance diagnostic tool for transactional databases. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 – July 01, 2016*, pages 1599–1614. ACM, 2016.
- [29] Xuanhe Zhou, Guoliang Li, Chengliang Chai, and Jianhua Feng. A learned query rewrite system using monte carlo tree search. *Proc. VLDB Endow.*, 15(1):46–58, 2021.
- [30] Xuanhe Zhou, Guoliang Li, and Zhiyuan Liu. Llm as dba. *arXiv preprint arXiv:2308.05481*, 2023.
- [31] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. D-bot: Database diagnosis system using large language models. *CoRR*, abs/2312.01454, 2023.
- [32] Xuanhe Zhou, Guoliang Li, Jianming Wu, Jiesi Liu, Zhaoyan Sun, and Xinning Zhang. A learned query rewrite system. *Proc. VLDB Endow.*, 16(12):4110–4113, 2023.
- [33] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. Db-gpt: Large language model meets database.