

DBMS-LLM Integration Strategies in Industrial and Business Applications: Current Status and Future Challenges

Zhengtong Yan
University of Helsinki
Helsinki, Finland
zhengtong.yan@helsinki.fi

Qingsong Guo
Hunan University of Technology
Zhuzhou, China
qingsongge@gmail.com

Gongsheng Yuan
Zhejiang University
Hangzhou, China
ygs@zju.edu.cn

Jiaheng Lu
University of Helsinki
Helsinki, Finland
jiaheng.lu@helsinki.fi

ABSTRACT

Modern enterprises are increasingly driven by the *DATA+AI* paradigm, in which Database Management Systems (DBMSs) and Large Language Models (LLMs) have become two foundational infrastructures powering a wide range of industrial and business applications, such as enterprise analytics, intelligent customer service, and data-driven decision-making. The efficient integration of DBMSs and LLMs within a unified system offers significant opportunities but also introduces new technical challenges. This paper surveys recent developments in DBMS-LLM integration and identifies key future challenges. Specifically, we categorize five representative architectural patterns based on their core design principles, strengths, and trade-offs. Based on this analysis, we further highlight several critical open challenges. We aim to provide a systematic understanding of the current integration landscape and to outline the unresolved issues that must be addressed to achieve scalable and efficient integration of traditional data management and advanced language reasoning in future intelligent applications.

VLDB Workshop Reference Format:

Zhengtong Yan, Gongsheng Yuan, Qingsong Guo, and Jiaheng Lu. DBMS-LLM Integration Strategies in Industrial and Business Applications: Current Status and Future Challenges. VLDB 2025 Workshop: Governance, Understanding and Integration of Data for Effective and Responsible AI (GUIDE-AI '25).

1 INTRODUCTION

Database Management Systems (DBMSs) have served as the foundational infrastructure of modern enterprises since the 1970s [15]. Over the past decades, DBMSs have evolved significantly across multiple dimensions, including system architecture (e.g., centralized, distributed, and cloud-native), data models (e.g., relational, graph, time-series, and vector), storage structures (e.g., row store, column store, LSM-tree, and B⁺-tree), and deployment environments (e.g., on-premises, clouds, and containers). These advances

enable DBMSs to support a wide variety of workloads such as Online Transaction Processing (OLTP), Online Analytical Processing (OLAP), Hybrid Transactional/Analytical Processing (HTAP), Business Intelligence (BI), and even Machine Learning (ML). Numerous DBMSs have been developed, including commercial systems like Oracle [43], IBM Db2 [30], Microsoft SQL Server [38], Amazon Aurora [3], Google BigQuery [10], and Snowflake [58], as well as open-source systems such as PostgreSQL [52], MySQL [40], DuckDB [18], ClickHouse [14], Neo4j [41], and Apache Spark [8]. These systems provide core functionalities for managing structured and semi-structured data, including data storage, querying, indexing, transactional guarantees (ACID), and analytical capabilities, thereby forming a foundational layer for modern industrial and business applications [59].

Since 2020, Large Language Models (LLMs) have emerged as new transformative technologies with the ability to understand, generate, and reason over unstructured and multimodal data [78]. Built on transformer architectures and trained on massive corpora, LLMs exhibit remarkable generalization abilities across diverse tasks with minimal supervision or fine-tuning. The LLM ecosystem is evolving very rapidly with diverse models. Early language models include GPT-3 [19], GPT-4 [1], PaLM [13], and Claude [5]. Open-source LLMs such as LLaMA [65], Mistral [39], DeepSeek [25, 37], and Qwen [9] have further enriched the LLM ecosystem. Beyond text-based LLMs, Vision-Language Models (VLMs) and Multi-modal LLMs (MLLMs) have also become increasingly prominent, such as Gemini [63, 64], GPT-4o [29], CLIP [42], Flamingo [2], ChatGLM [23], and Kosmos-2 [50]. Those models integrate visual, textual, and sometimes auditory modalities, enabling advanced reasoning over complex and multimodal inputs. LLMs can be deployed across various platforms, including cloud-based services, on-device components, or integrated agents, enabling a wide range of intelligent functionalities across industries. As their capabilities continue to expand, LLMs are increasingly regarded as a new layer of general-purpose infrastructure, particularly for intelligent applications that require natural language interaction, semantic reasoning, and multi-modal understanding.

Current industrial and business systems are increasingly driven by the *Data+AI* paradigm, in which DBMSs and LLMs are expected to function as complementary and coexisting *dual infrastructures* to support modern applications. This relationship can be expressed

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, ISSN 2150-8097.

Table 1: Comparison of DBMSs and LLMs in industrial and business applications.

Aspect	DBMSs	LLMs
Primary Purpose	Data Storage, Query, Management	Natural Language Understanding, Reasoning, Generation
Main Use	OLTP/OLAP/HTAP/BI, Indexing and Querying, Transaction Processing, etc.	Q&A , Semantic Search, Text Summarization, etc.
Strengths	ACID Guarantees, High Query Efficiency, Data Integrity, Joins and Aggregations, etc.	Human-like Interaction, Unstructured Data Processing, Fine-tuning, Generalization from Examples, etc.
Data Model/Modality	Structured and Semi-Structured (e.g., Relation, Graph, JSON, Time-Series)	Unstructured Data (e.g., Text, Documents, Images, Videos)
Interface	SQL, APIs	Natural Language Prompts, APIs
Interpretability	Transparent Outputs, Well-understood Plans	Opaque Reasoning, Black-box Behavior
Latency	Typically Low-latency, Optimized Query Plans	Often Higher Latency, Depends on Model and Context Size
Update	Tuple-level Updates, Transactions Support	No Native Data Update, Retraining or RAG Needed

as:

$$\text{LLMs} + \text{DBMSs} \rightarrow \text{Dual Infrastructures of Enterprises} \quad (1)$$

In this paradigm, both *data* and *models* are regarded as strategic assets. Table 1 summarizes and compares the core roles, capabilities, and characteristics of DBMSs and LLMs. While originating from distinct technical backgrounds and solving different problems, DBMSs and LLMs are not competing technologies. Instead, their integration offers the potential to unlock powerful synergies that surpass the capabilities of either system can achieve in isolation. There are several motivating factors for integrating DBMSs and LLMs into a unified system. For example, many modern applications require hybrid query processing that combines structured data retrieval (e.g., SQL) with unstructured data understanding, natural language interpretation, or semantic reasoning. Another key motivation is system-level synergy, such as leveraging DBMS features (e.g., indexing, caching, and transaction management) to boost the efficiency and consistency of LLM-based operations, particularly in scenarios involving large-scale or dynamic datasets. Lastly, the growing field of Industrial Large Models (ILMs) increasingly demands systems where DBMSs manage structured backends while LLMs provide interpretability, reasoning, and language interfaces [80, 81].

The choice of integration strategy significantly affects system performance, complexity, and maintainability. It determines how the data flows between DBMSs and LLMs, how the components interact, and how easily the system can adapt to changes. Integration is not merely about invoking LLM APIs from within a database or vice versa. Rather, it involves deeply embedding and aligning LLMs and DBMSs across different levels of the system stack, such as the system level, component level, and processing pipeline level. Achieving this requires a thorough understanding of the internal architectures, capabilities, and operational semantics of both DBMSs and LLMs. To meet diverse application needs, various DBMS–LLM integration architectures have been proposed in industry and business, ranging from simple pipeline connectors to deeply integrated strategies. For instance, a simple form of integration involves connecting DBMSs and LLMs via external data pipeline tools, where data flows between components without mutual understanding. A more advanced approach introduces a middleware layer to manage interactions and task decomposition. Deeper strategies embed LLMs directly into the DBMS execution engine as custom query

operators or user-defined functions (UDFs). Finally, cloud-native platforms increasingly offer end-to-end integration capabilities, with providers like Oracle, Google, Amazon, and Alibaba launching unified environments where both DBMSs and LLMs can be tightly coupled and jointly optimized.

In this paper, we aim to provide a comprehensive survey of existing DBMS–LLM integration architectures, along with an analysis of key research challenges and open directions in this evolving area. Our main contributions are summarized as follows:

- **Survey of the Current Status.** We systematically review and categorize existing DBMS–LLM integration approaches in industrial and business applications, covering a wide range of use cases and architectural patterns. We also provide recommendations for choosing the optimal integration strategies according to specific application requirements and system constraints.
- **Analysis of Future Research Challenges.** We also identify and discuss some key challenges in DBMS–LLM integration to enable more robust, scalable, and intelligent integrated systems in the future.

2 RELATED WORK

A growing body of surveys has explored the intersection between LLMs and DBMSs, reflecting the increasing interest in combining traditional data management with novel reasoning capabilities.

Interactions Between DBMSs and LLMs. Several recent studies investigate how LLMs can interact with traditional database systems. Pan et al. [45] present a roadmap for unifying Knowledge Graphs (KGs) and LLMs, outlining some design principles and use cases that benefit from their integration. Similarly, Khorashadizadeh et al. [33] provide a comprehensive survey of collaborative strategies that leverage LLMs and KGs to enable more advanced and interpretable reasoning. Kim et al. [34] categorize various DBMS–LLM interaction paradigms, where LLMs serve as data sources, data processors, or query translators between structured and unstructured data.

LLM-enhanced Data Management. LLMs have also been applied to enhance core data management tasks. Hong et al. [28] explore the use of LLMs for SQL generation and complex query interpretation, demonstrating the potential of language models in

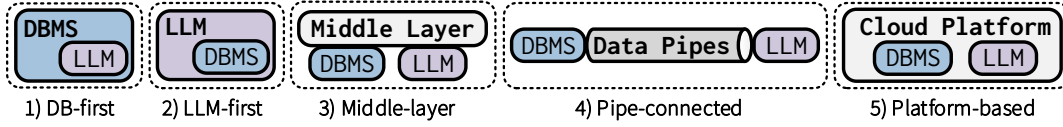


Figure 1: Illustrations of different integration strategies.

Table 2: Comparison of strengths and weaknesses of different DBMS-LLM integration strategies.

Strategy	Features	Strengths	Weaknesses
DB-First	• LLM embedded as a DB component (e.g., interface, optimizer) or operator (e.g., UDF into query execution plans) • DB is in full control	• Tight integration with query engines • Benefits from database indexing • Easier governance	• Limited LLM context window • Hard to manage model versions • DB constraints limit LLM potential
LLM-First	• DB acts as a machine (e.g., caching or indexing component) inside LLM • LLM is central to drives logic	• Flexible and user-friendly • Natural interface for non-experts • Easily handles unstructured data	• Hallucinations or incorrect SQL • Weak security/access control • DB optimization not utilized well
Middle-Layer	• Orchestration layer mediates between DB and LLM by tools like LangChain • Intermediary system coordinates communication, data flow, pipelines	• Modular design • Composable pipelines • Easier to evolve components • Rich interaction patterns	• Increased system complexity • Performance bottlenecks • Harder to debug and test
Pipe-Connected	• DB and LLMs run independently and are integrated via stream or batch data pipelines • Dataflow-centric architecture	• Highly decoupled and modularized • Well-suited for event-driven architectures • Separation of concerns	• High latency or eventual consistency • Complex to manage state and recovery • Debugging across stages can be hard
Platform-Based	• Cloud platforms offer LLM+DB as managed services (e.g., Snowflake Cortex, Google Cloud, and Oracle Cloud)	• Low setup cost (ready to use) • Full-stack scalability • Vendor ecosystem support	• Vendor lock-in • Limited transparency • Hard to customize deeply

bridging natural language interfaces with structured queries. Zhou et al. [84] examine the broader role of LLMs in data management tasks such as data cleaning and entity resolution. However, their discussion is largely high-level and lacks a technical taxonomy or system-level analysis of integration strategies.

Vector Databases. Vector databases have become a foundational infrastructure for enabling LLM-based retrieval tasks, particularly in retrieval-augmented generation (RAG) pipelines. Surveys by Pan et al. [44], Han et al. [26], and Jing et al. [32] provide comprehensive overviews of vector database architectures, indexing methods, and similarity search techniques. The integration of LLMs into these systems to enhance semantic retrieval and reasoning capabilities is also addressed in [32].

Databases Meet AI. In the broader context of database and AI, Zhou et al. [83] provide early insights into the bidirectional relationship of AI4DB and DB4AI. Cai et al. [12] survey the use of deep reinforcement learning (DRL) in data analytics and database operations. Yan et al. [75] focus specifically on DRL-based techniques for join order selection, a key challenge in query optimization.

Compared to these works, our paper presents a focused and in-depth survey of architectural-level integrations between LLMs and DBMSs. We categorize and analyze five representative integration patterns, highlight their trade-offs, and identify open challenges to guide future research in this evolving domain.

3 CURRENT STATUS OF LLM-DBMS INTEGRATION STRATEGIES

In this section, we present a detailed introduction to current strategies for integrating LLMs with DBMSs. We first provide a high-level

overview and comparison of integration strategies by summarizing their key features, advantages, and trade-offs. Then, we offer an in-depth discussion of each strategy.

3.1 Overview of Integration Strategies

During the early Deep Learning (DL) era (2010s to early 2020s), there emerged a growing demand for integrating DL pipelines with DBMSs. Lixi et al. [82] summarized three common architectural paradigms: *DL-centric*, *UDF-centric*, and *Relation-centric*. They also proposed a vision for an advanced system architecture that seamlessly integrates these paradigms, along with hybrid designs that bridge the gap between the three paradigms. Their works offer valuable insights for the LLM-DBMS integration in the era of LLMs.

Typically, the integration of LLMs and DBMSs can be characterized along several dimensions: 1) the purposes of integrations (e.g., query understanding, augmentation), 2) the system layer or endpoint where the integration occurs (e.g., inside the DBMS, at the interface, or externally), and 3) the degree and depth of coupling between the LLM and DBMS (e.g., tightly integrated or loosely coupled).

Based on these dimensions, we identify five major integration strategies, as illustrated in Figure 1. Each strategy reflects a different structural and functional relationship between the DBMS and the LLM. For example, the DB-first strategy embeds the LLM directly within the DBMS, treating it as an internal module or an operator. In contrast, the LLM-first strategy treats the DBMS as an external machine, possibly for executing structured sub-queries generated by the LLM or for accessing specific records during LLM-driven

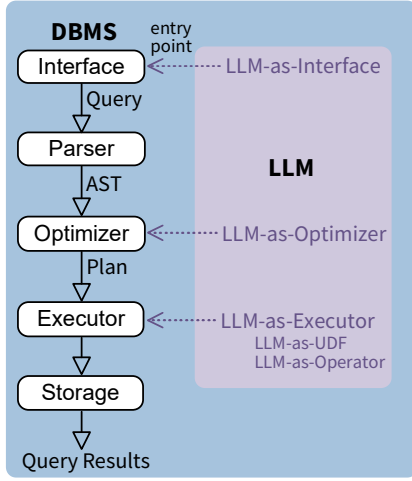


Figure 2: The architecture and processing pipelines of DBMSs, as well as the entry points of LLMs.

reasoning. These strategies vary in their architectural focus and operational trade-offs. Table 2 summarizes the comparative strengths and limitations of each strategy.

It is important to note that our categorization is **non-orthogonal**: some of the integration dimensions are not mutually exclusive, and hybrid combinations frequently occur in real-world industrial and business systems. For example, a system may simultaneously adopt a DB-first architecture to embed LLM-enhanced execution operators directly within the query engine, while also leveraging pipe-based tools (e.g., Apache Flink) to facilitate data exchange with external LLM services for tasks such as semantic enrichment or user query interpretation.

3.2 DB-first Strategy: Integration on the DB-side

The DB-first strategy positions a traditional DBMS as the core architecture of the system, with LLMs serving as auxiliary modules or external services. Figure 2 shows the typical DBMS architecture and processing pipelines that include the following stages: 1) User Interface: accepting user queries like SQL, 2) Query Parsing: converting the query into an Abstract Syntax Tree (AST) or other internal representations, 3) Query Optimization: performing logical and physical query optimization to generate execution plans, and 4) Query Execution: executing physical query plans that are typically represented as a tree or a Directed Acyclic Graph (DAG) of operators.

DB-first strategies can preserve the traditional DBMS processing guarantees while augmenting them with LLM-driven intelligence. In this architecture, the DBMS remains the core system, and LLMs are selectively integrated into specific components or stages of the processing pipeline to enhance functionality. As illustrated in Figure 2, different entry points exist for such integration, enabling different use cases like *LLM-as-Interface*, *LLM-as-Optimizer*, and *LLM-as-Executor*.

LLM-as-Interface. Structured query languages such as SQL can be overly complex and unintuitive, even for experienced database experts. One of the most natural and widely adopted ways is to leverage LLMs’ natural language understanding capabilities to translate user intents expressed in natural language into executable SQL queries. This line of research and development is commonly referred to as Text-to-SQL (Text2SQL or NL2SQL). Several recent works have contributed significant advances in this area. Jinyang et al. [35] introduce BIRD, a large-scale Text2SQL benchmark targeting complex real-world databases. BIRD includes task challenges such as noisy data, external knowledge dependencies, and efficiency constraints. Their findings indicate that even state-of-the-art models like GPT-4 still struggle with these real-world complexities, highlighting the gaps between academic performance and practical usability. Kaushikpresent et al. [53] propose NL2WeId, a system that can translate natural language directly into the Weld Intermediate Representation (IR) using GPT-4. This approach bypasses SQL entirely, enabling more optimized data analytics by mapping natural language to a lower-level, execution-friendly representation. Tianshu et al. [73] present DBCopilot, a schema routing framework designed to handle large and heterogeneous databases. It efficiently selects relevant schemas for a given natural language query, which can improve the scalability and accuracy of schema-agnostic Text2SQL generation. Simone et al. [46] investigate the impact of various training strategies—such as zero-shot prompting, supervised fine-tuning with reasoning traces, reinforcement learning, and their combinations—on the performance of LLMs in Text2SQL tasks. Their results show that combining fine-tuning and reinforcement learning achieves the best generalization and execution accuracy, allowing smaller models to rival the performance of much larger ones. Despite these advancements, it is important to note that Text2SQL primarily addresses the user interface layer, transforming how users interact with databases but not altering the core query processing, optimization, or execution logic of the DBMS itself. This is why some researchers argue that focusing solely on Text-to-SQL is insufficient for truly intelligent and adaptive database systems, and deeper integration strategies are needed to unlock the full potential of LLM-DBMS synergy [11].

LLM-as-Optimizer. The query optimizer is often referred to as the *brain* of a DBMS, as it plays a critical role in determining the most efficient execution plans for a given query. Given their strong reasoning, abstraction, and pattern recognition capabilities, LLMs are increasingly being explored as promising tools to enhance or even replace traditional optimization components, including tasks such as query rewriting, join order enumeration, and execution plan selection. Zhaodonghui et al. [36] propose LLM-R2, a hybrid system that integrates LLMs with traditional rule-based query rewriting to improve SQL execution efficiency while preserving query equivalence. The system employs a contrastive curriculum-trained representation model to select high-quality example rewrites, enabling the LLM to generate more effective and semantically valid query rewrites without relying on potentially inaccurate cost models. Building upon this idea, Zhaoyan et al. [61] design R-Bot, which incorporates multi-source rewrite evidence and a hybrid structure-semantic retrieval mechanism. This architecture grounds the LLM’s output in verified rule sets and high-quality Q&A pairs, thereby

reducing hallucinations and improving the robustness and reliability of the rewriting process. Jie et al. [62] present LLM-QO, a novel framework that directly generates SQL execution plans using LLM’s fine-tuning abilities. This approach bypasses traditional plan enumeration and cost modeling altogether. Experimental evaluations show that LLM-QO consistently outperforms both classical and learned optimizers across diverse datasets, demonstrating strong generalization to unseen queries. Zhiming et al. [76] introduce LLMOpt, a comprehensive query optimization framework based on fine-tuned LLMs. LLMOpt combines plan generation with global candidate selection, eliminating reliance on heuristic search or inaccurate cost estimators. Benchmarking results reveal that LLMOpt delivers superior performance compared to systems such as PostgreSQL and some machine learning based query optimizers. Nikita et al. [70] propose LLM-PM, a training-free, lightweight framework that leverages LLM-derived embeddings of execution plans. Rather than modifying the DBMS itself, LLM-PM augments existing cost-based optimizers by suggesting performance hints, such as join strategies or join tree shapes (e.g., left-deep or bushy trees). Through a nearest-neighbor search over plan embeddings and a two-neighborhood consistency check, LLM-PM achieves significant query runtime improvements on standard benchmarks, validating the utility of plan representation learning for real-world optimization. In summary, these works demonstrate that LLMs can play an increasingly active role in query optimization—either by generating execution plans directly, augmenting existing optimizers with semantic insight, or automating the traditionally manual process of rule-based rewriting. Unlike LLM-as-Interface approaches, these strategies target the core logic of query processing, pushing the boundaries of intelligent and adaptive DBMS design.

LLM-as-Executor. The execution engine of a DBMS is responsible for evaluating physical query plans and generating query results. Within this component, LLMs can be integrated in two primary modes: as User-Defined Functions (UDFs) and as native operators embedded in the query execution pipelines. Both modes enable the system to support advanced semantic and reasoning operations during query execution, such as classification, summarization, and multi-modal inference. **(1) LLM-as-UDF.** UDFs have long served as a powerful mechanism for injecting user-defined logic into SQL-based systems [20, 21, 57]. The LLM-as-UDF paradigm encapsulates LLM capabilities inside SQL-callable functions, allowing LLM inference to be invoked directly in SQL queries. This approach enables tight coupling between traditional database processing and the reasoning abilities of LLMs, without requiring complex external orchestration. For example, Parker et al. [22] propose BlendSQL, which integrates LLMs with SQLite by exposing specialized LLM-powered SQL functions. This allows users to perform semantic mapping and contextual reasoning over tabular data inside the database engine. Similarly, Fuheng et al. [77] introduce Hybrid Query User-Defined Functions (HQUDFs) that combine structured queries over relational data with semantic inference over unstructured knowledge via LLMs, enabling rich hybrid querying capabilities. Anas et al. [17] present FlockMTL, which integrates LLMs directly into the DuckDB query engine. FlockMTL exposes LLM functionality as scalar and aggregate SQL functions, allowing tuple-level or group-level semantic operations. These UDFs can perform language understanding tasks like classification or summarization as part of the query execution

process. **(2) LLM-as-Operator.** In contrast to the UDF mechanism, the LLM-as-Operator strategy incorporates LLMs as native operators within the physical query execution pipeline. This approach treats LLMs as first-class citizens of the DBMS, enabling them to participate directly in execution plans and interact with database operators [49]. Mohammed et al. [55, 56] introduce GALOIS, a system that executes SQL queries on top of pre-trained LLMs. GALOIS decomposes SQL queries into logical plans, where physical operators are implemented via structured LLM prompts. These prompts guide the LLM to retrieve or synthesize structured outputs from its latent knowledge or textual embeddings, supporting query execution directly over encoded unstructured data. Matthias et al. [69] present ELEET, which introduces multi-modal operators (MMOps) into the DBMS. These native operators—such as multi-modal scan, join, and union—use small language models (SLMs) as extractive decoders. ELEET allows structured tuples to be extracted from semi-structured or unstructured sources (e.g., text), enabling multi-row transformations during query execution. Expanding this concept, Matthias et al. [66, 67] develop CAESURA, a system that uses LLMs as multi-modal query planners. CAESURA defines four distinct multi-modal operators—*VisualQA*, *TextQA*, *Python UDF*, and *Image Select*—to support unified querying across diverse data modalities including text, images, and code. The LLM provides coordination and semantic interpretation within the physical query plan, enabling flexible and expressive multi-modal queries.

3.3 LLM-first Strategy: Integration on the LLM-side

In contrast to the DB-first strategy, which embeds LLM functionalities within the database system, the LLM-first strategy positions the LLM as the central orchestration layer. In this architecture, the database is treated as a supporting backend component, often used for storage, caching, or structured retrieval, while the LLM handles the primary logic, reasoning, and interaction tasks. It is especially suitable for applications where LLMs serve as the primary user interface or analytical engine, such as conversational agents or intelligent assistants. Compared with DB-first strategies, integration options of LLM-first strategies are more limited on the LLM side, since traditional DBMSs are not designed to be directly embedded within LLMs. Instead, DBMSs are often used as retrieval backends or cache layers to accelerate query responses or support Retrieval-Augmented Generation (RAG) pipelines.

Wenbo et al. [60] introduce TranSQL, a toolkit that integrates relational databases directly within LLMs by translating neural network operations into SQL queries and storing model weights as relational tables. This allows a relational database to efficiently manage LLM execution using built-in database features like disk-to-memory management and caching. TranSQL enables end-to-end transformer-based text generation entirely within a relational environment, eliminating the need for specialized deep learning infrastructure. Alekh et al. [31] present GOD Machine, an ambitious framework that turns relational databases into generative AI systems. It integrates LLMs with logical data models (LDMs) of databases to provide scalable, interpretable, and privacy-preserving analytics. This system automates data modeling, retrieval, and pipeline management through scalable retrieval mechanisms and introduces a framework

for building end-to-end AI-powered applications directly on relational databases. Xinyang et al. [79] develop Chat2Data, which is an interactive data analysis system that combines LLMs, vector databases, and RAG techniques. This system supports natural language querying across both structured and unstructured data. A key component of the architecture is a vector database used as a cache layer, storing embeddings of frequently asked questions and their corresponding answers to improve system responsiveness and reduce redundancy.

The LLM-first strategy emphasizes semantic abstraction, natural interaction, and integration simplicity, enabling rapid development of AI-powered applications without tightly coupling LLMs to low-level database internals. However, this approach often sacrifices fine-grained query control and data processing guarantees, making it less suitable for mission-critical transactional workloads or performance-sensitive analytical queries. In practice, LLM-first strategies are commonly used for user-facing interfaces, exploratory data analysis, and knowledge-driven reasoning tasks.

3.4 Middle-layer Strategy: Integration via a Middleware

Distinct from the DB-first and LLM-first paradigms that treat either the database or the LLM as the central component, the Middle-layer strategy introduces an intermediary orchestration layer that coordinates the two systems and processing pipelines. This middleware sits between the LLM and the DBMS, acting as a smart controller to manage, optimize, and unify the processing pipelines of both components.

The key motivation behind this strategy is to bridge two fundamentally different data processing paradigms: 1) the deterministic, rule-based query and retrieval pipeline of a traditional DBMS, and 2) the stochastic, probabilistic generation and reasoning pipeline of an LLM. Therefore, the essence of integration lies in building a middleware that can compose, schedule, and optimize hybrid tasks, allowing smooth and efficient interaction between structured queries and semantic reasoning.

Yu et al. [24] propose equipping LLMs with customized middleware tools that serve as an intermediate layer between LLMs and external systems such as knowledge bases or relational databases. These tools interpret user intent, generate executable queries (e.g., SQL), and return results to the LLM, which mitigates token-length limitations and improves task performance in complex environments. Sumedh et al. [54] presents a multi-LLM orchestration engine that integrates multiple LLMs with a temporal graph database and a vector database to enable personalized, context-rich AI assistance. The temporal graph database captures evolving conversational history and user preferences over time, while the vector database securely encodes private data for precise retrieval. The middleware coordinates these components to deliver personalized, privacy-preserving, and context-rich responses, without retraining the LLMs. This addresses major challenges such as long-term memory, hallucination, and private data integration. Jiayi et al. [72] propose AOP, a middleware framework that unifies LLM-based semantic reasoning with DBMS query processing. It dynamically composes execution pipelines containing both semantic operators (invoked via LLMs) and structured query operators (e.g., SQL or dataframe

operations). By integrating over heterogeneous data sources in data lakes, AOP orchestrates hybrid workflows across unstructured and structured data, achieving flexibility and performance gains.

3.5 Pipe-connected Strategy: Integration via Data Pipes

Pipe-connected strategies refer to integrating LLMs and DBMSs using data pipe connectors or tools, which is a pragmatic, loosely coupled approach that leverages mature streaming and ETL technologies to enable real-time or batch data exchange. Unlike DB-first or LLM-first strategies that embed functionality into the system core, the pipe-connected approach preserves system modularity by treating the LLM and DBMS as independent services connected via well-defined data flows. This integration is particularly effective in event-driven architectures or scenarios requiring scalable, real-time analytics, where tight coupling may not be necessary or practical. Modern data processing tools (e.g., Apache Kafka [7], Apache Flink [6], and Spark Streaming [8]) enable high-throughput, low-latency pipelines, facilitating seamless integration between components without requiring deep architectural fusion.

Moreh et al. [48] propose Dataverse, which is an open-source, user-friendly ETL platform designed specifically for preparing large-scale datasets for LLM training and inference. It features a block-based workflow interface and supports scalable data processing via integration with systems like Apache Spark and AWS EMR. Key features include data deduplication, decontamination, and bias mitigation. Dataverse can serve as a bridge between data stored in DBMSs and LLMs by enabling automated ingestion, transformation, and delivery of curated datasets for semantic tasks. Kai et al. [71] position data streaming as a transformative paradigm that enables real-time decision-making and event-driven innovation across industries. They emphasize organizational strategies (e.g., fostering internal streaming communities) and showcase emerging trends such as integrating LLMs into streaming pipelines for enhanced reasoning and semantic analytics. [71] also gives a case about an autonomous airport, which leverages data streaming, DBMSs, and LLMs in its digital transformation to enhance airport operations and passenger experiences.

Overall, the pipe-connected strategy offers several practical benefits like scalability, flexibility, and ease of integration. However, its loose coupling also introduces several challenges, such as latency overhead in complex pipelines, and increased engineering effort for managing failure recovery.

3.6 Platform-based Strategy: Integration within a Cloud Platform

The platform-based strategy focuses on integration modes enabled by cloud-native platforms, where both LLMs and DBMSs are deployed and managed as modular services in the cloud. This category emphasizes deployment architecture and interoperability capabilities provided by modern platforms, rather than the functional embedding of one system into another.

Over the past decades, cloud-native technologies have matured to support highly flexible service delivery modes, such as Software-as-a-Service (SaaS) for DBMSs, Model-as-a-Service (MaaS) for LLMs, and even Platform-as-a-Service (PaaS) for the whole application.

These services are typically built on microservices and serverless architectures, which now form the backbone of IT infrastructure in modern enterprises. According to Gartner, over half of enterprise databases are now cloud-hosted, with a growing percentage being fully cloud-native [16]. Leading cloud providers, including Amazon AWS, Google Cloud, Microsoft Azure, Oracle Cloud, IBM Cloud, Alibaba Cloud, Tencent Cloud, and Huawei Cloud, offer rich ecosystems that include both LLM and DBMS services. This creates significant opportunities for integrating LLMs and DBMSs within the same cloud platform, enabling unified resource management, coordinated data processing pipelines, standardized communications, and orchestrated multi-component workflows.

A unified cloud platform can act as a communication and control hub where both LLMs and DBMSs are able to exchange data and participate in joint pipelines. For instance, cloud-native services like AWS Lambda, Azure Logic Apps, or Google Cloud Workflows can coordinate the invocation of LLM models and SQL-based data processing in a seamless, event-driven fashion. Recent developments have also popularized workflow-based architectures that use LLMs as autonomous agents to drive decision-making and database interactions. These workflows support multi-step task execution, retrieval-augmented generation, data validation, and response generation in a unified loop. This is consistent with trends in cloud-native data mesh architectures [51] and agent-based LLM orchestration frameworks. For example, Snowflake Cortex integrates LLM capabilities directly into its data cloud, allowing users to perform LLM-powered analysis on structured data using SQL-like interfaces. Oracle Cloud offers integrated solutions where Oracle Autonomous Database and Oracle LLM Services work together under a shared compute and storage layer to support intelligent workflows.

In summary, platform-based strategies abstract away infrastructure concerns and allow developers to compose and orchestrate complex DBMS-LLM pipelines using standardized cloud-native tools and workflows.

3.7 What Strategies to Choose?

Table 3 provides a feature-based comparison of the five DBMS-LLM integration strategies, using qualitative scores (Strong/Moderate/Weak) across several dimensions. Based on this analysis, we argue that the optimal integration strategy should be selected according to the following key factors:

1) Task Requirements: The nature of the workloads (e.g., batch vs. streaming, offline processing vs. real-time interaction) strongly affects integration needs. Real-time and low-latency tasks are better served by DB-first, pip-connected, or platform-based integrated solutions, while offline analytics can afford more flexibility in architectural choice.

2) Functional Requirements: If an application demands complex multi-modal or cross-model interactions (e.g., querying both relational tables and unstructured text, or linking visual data with structured knowledge graphs), DB-first integration is advantageous, as it enables native access and processing capabilities over heterogeneous data sources.

3) LLM Dependencies: In scenarios where LLMs heavily rely on advanced LLM functionalities (e.g., code generation, reasoning,

instruction following), the LLM-first integration provides greater flexibility in function execution and adaptability. By contrast, the DB-first integration may be constrained by the predefined execution models of database engines.

4) Deployment Modes: When using cloud-native platforms or workflow-based orchestration systems, platform-based integration offers significant benefits, including unified management of both data and model services, simplified scaling, and standardized APIs for cross-component communications.

5) Hybrid Strategies: In some practical settings, hybrid integration strategies may offer the best trade-offs. For example, combining DB-first integration for data access efficiency with platform-based coordination for scalability and flexibility allows systems to balance performance with manageability.

4 FUTURE CHALLENGES

In this section, we summarize some key open challenges in the integration of DBMSs and LLMs, with a focus on architectural, algorithmic, and performance aspects.

4.1 Challenges in DB-First Strategies

In DB-first strategies, the DBMS acts as the primary control system, embedding LLMs as logical operators or external functions. Several challenges arise from this tightly coupled integration, such as:

- **Cross-Model Query Execution.** Systems like [68] propose extending query processing to include both relational and unstructured text data. A major challenge is designing unified execution models and query languages that can handle hybrid data types (e.g., structured tables, semi-structured graphs, unstructured text, and images) efficiently. For instance, HybridGraph [4] introduces graph-based models that mix relational and semantic representations, but efficient execution, indexing, and optimization remain underexplored.
- **Cost Modeling and Plan Optimization.** Injecting LLMs into DBMSs' original query processing pipelines will introduce new types of operators with non-deterministic behavior, high latency, and variable computational costs. Traditional cost-based optimizers of DBMSs are ill-equipped to handle these uncertainties. Therefore, how to estimate the cost of the injected LLM-based operators and integrate them into join ordering and plan enumeration frameworks is an open research direction [74].
- **Cross-Modal Operator Design.** As LLMs become increasingly multi-modal, there is a need to define new operator classes for sub-tasks across modalities (e.g., *Image Filter*, *Video Summary*, *Text Extraction* [66]) and design cross-modal join operators (e.g., joining table rows with images or videos). These operators can be seen as LLM-backed UDFs and must be systematically integrated into the DBMS pipeline. Similar to prior efforts on hybrid relational-graph operators such as in GRFusion [27], the challenge is to balance expressiveness, performance, and composability in these new operator designs.

Table 3: Feature-oriented comparison of DBMS-LLM integration strategies (✓ = Strong, ≈ = Moderate, ✗ = Weak).

Dimension	DB-First	LLM-First	Middle-Layer	Pipe-Connected	Platform-Based
Coupling Degree	✓	✗	≈	✗	≈
Real-Time	✓	✗	≈	≈	✓
Scalability	Limited by DB	Limited by LLM	✓	✓	✓
Extensibility	✗	✓	✓	✓	✓
Complexity	✗	✗	✓	≈	✗
Security	✓	✗	≈	≈	✓

4.2 Challenges in LLM-First Strategies

In LLM-first architectures, the LLM is the central component, and the DBMS is typically abstracted as a tool (e.g., KV store, SQL plugin, or retrieval engine). This design simplifies end-user interfaces but introduces unique concerns such as:

- **Limited Declarativity and Control.** Delegating control to the LLM sacrifices the optimization capabilities of DBMSs. It becomes difficult to ensure query correctness, efficiency, or even completeness, especially when dealing with complex queries or large data volumes.
- **Tool Use and Reliability.** Prompt-based access to DBMSs introduces tool-call fragility, dependency on context window limits, and inconsistency in tool invocation. Improving tool-calling reliability remains challenging.

4.3 Challenges in Middle-Layer Orchestration

Middleware-based orchestration strategies act as intermediaries between DBMSs and LLMs. These orchestration layers mediate data access, plan decomposition, operator execution, and caching. However, several challenges still remain, such as:

- **System Coordination.** Coordinating execution between DBMS engines and LLMs requires advanced runtime environments that can manage state, partial results, operator delegation, and errors.
- **Dynamic Adaptation.** Middleware systems need to adapt execution strategies based on workload characteristics, latency budgets, and application intent (e.g., information retrieval vs. transaction processing).

4.4 System Design and Application Suitability

Given the wide range of available integration strategies, a key research question is how to select the most suitable architecture for a given application and workload. As discussed in [47], this is a fundamental decision that depends on latency requirements, data volume, security needs, and user expertise.

- **Quantitative Evaluation Frameworks.** A standardized benchmarking framework is needed to evaluate the trade-offs among strategies in terms of latency, accuracy, cost, interpretability, and robustness.
- **Integration Selection Agent.** Future systems could benefit from meta-controllers or learning-based agents that can dynamically select and configure the integration architecture based on application goals and context.
- **Hybrid Architectures.** Designing and managing hybrid architectures that combine multiple integration strategies

poses unique challenges. This includes making principled trade-offs across dimensions such as performance, modularity, fault isolation, and development complexity.

5 CONCLUSION

The integration of LLMs and DBMSs represents a promising paradigm shift in the design of intelligent *Data+Model* driven applications. In this paper, we examine current integration strategies and categorize five representative architectural patterns, each offering unique trade-offs in terms of flexibility, performance, and deployment complexity. While these integration efforts unlock new possibilities by bridging structured data processing with unstructured language understanding, they also introduce several open challenges, such as performance optimization and system interoperability. Looking forward, future research should focus on developing adaptive and composable integration frameworks, establishing standardized benchmarks for evaluation, and designing robust architectures that can operate reliably at scale. We believe that addressing these challenges is essential for realizing the full potential of synergy between DBMSs and LLMs, which paves the way to the next generation of intelligent, scalable, and trustworthy systems.

REFERENCES

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023). <https://doi.org/10.48550/arXiv.2303.08774>
- [2] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. 2022. Flamingo: a Visual Language Model for Few-Shot Learning. *Advances in neural information processing systems* 35 (2022), 23716–23736.
- [3] Amazon Aurora. [n.d.]. <https://aws.amazon.com/cn/rds/aurora/>.
- [4] Mouna Ammar, Christopher Rost, Riccardo Tommasini, Shubhangi Agarwal, Angela Bonifati, Petra Selmer, and Erhard Rahm. 2025. Towards Hybrid Graphs: Unifying Property Graphs and Time Series. In *28th International Conference on Extending Database Technology (EDBT)*. 2483–2490. <https://openproceedings.org/2025/conf/edbt/paper-183.pdf>
- [5] Anthropic: Claude. [n.d.]. <https://www.anthropic.com/>.
- [6] Apache Flink. [n.d.]. <https://flink.apache.org/>.
- [7] Apache Kafka. [n.d.]. <https://kafka.apache.org/>.
- [8] Apache Spark. [n.d.]. <https://spark.apache.org/>.
- [9] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen Technical Report. *arXiv preprint arXiv:2309.16609* (2023).
- [10] BigQuery. [n.d.]. <https://cloud.google.com/bigquery>.
- [11] Asim Biswal, Liana Patel, Siddharth Jha, Amog Kamsetty, Shu Liu, Joseph E Gonzalez, Carlos Guestrin, and Matei Zaharia. 2025. Text2SQL is Not Enough: Unifying AI and Databases with TAG. In *CIDR*.
- [12] Qingpeng Cai, Can Cui, Yiyuan Xiong, Wei Wang, Zhongle Xie, and Meihui Zhang. 2022. A Survey on Deep Reinforcement Learning for Data Processing and Analytics. *IEEE Transactions on Knowledge and Data Engineering* 35, 5 (2022), 4446–4465.

- [13] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113. <http://jmlr.org/papers/v24/22-1144.html>
- [14] ClickHouse. [n.d.]. <https://clickhouse.com/>.
- [15] Edgar F Codd. 1970. A Relational Model of Data Large Shared Data Banks. *Commun. ACM* 13, 6 (1970), 377–387.
- [16] Haowen Dong, Chao Zhang, Guoliang Li, and Huanchen Zhang. 2024. Cloud-Native Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 36, 12 (2024), 7772–7791. <https://doi.org/10.1109/TKDE.2024.3397508>
- [17] Anas Dorbani, Sunny Yasser, Jimmy Lin, and Amine Mhedhbi. 2025. Beyond Quackling: Deep Integration of Language Models and RAG into DuckDB. *arXiv preprint arXiv:2504.01157* (2025).
- [18] DuckDB. [n.d.]. <https://duckdb.org/>.
- [19] Luciano Floridi and Massimo Chiriatti. 2020. GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds and Machines* 30, 4 (2020), 681–694.
- [20] Yannis Foufoulas and Alkis Simitis. 2023. Efficient Execution of User-Defined Functions in SQL Queries. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3874–3877.
- [21] Kai Franz, Samuel Arch, Denis Hirn, Torsten Grust, Todd C Mowry, and Andrew Pavlo. 2024. Dear User-Defined Functions, Inlining isn’t working out so great for us. Let’s try batching to make our relationship work. Sincerely, SQL. In *Conference on Innovative Data Systems Research (CIDR)*.
- [22] Parker Glenn, Parag Dakle, Liang Wang, and Preethi Raghavan. 2024. BlendSQL: A Scalable Dialect for Unifying Hybrid Question Answering in Relational Algebra. In *Findings of the Association for Computational Linguistics ACL* 2024, 453–466.
- [23] Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. 2024. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. *arXiv preprint arXiv:2406.12793* (2024).
- [24] Yu Gu, Yiheng Shu, Hao Yu, Xiao Liu, Yuxiao Dong, Jie Tang, Jayanth Srinivasa, Hugo Latapie, and Yu Su. 2024. Middleware for LLMs: Tools Are Instrumental for Language Agents in Complex Environments. *arXiv preprint arXiv:2402.14672* (2024).
- [25] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025).
- [26] Yikun Han, Chunjiang Liu, and Pengfei Wang. 2023. A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge. *arXiv preprint arXiv:2310.11703* (2023).
- [27] Mohamed S Hassan, Tatiana Kuznetsova, Hyun Chai Jeong, Walid G Aref, and Mohammad Sadoghi. 2018. GRFusion: Graphs as First-Class Citizens in Main-Memory Relational Database Systems. In *Proceedings of the 2018 International Conference on Management of Data*. 1789–1792. <https://doi.org/10.1145/3183713.3193541>
- [28] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. *arXiv preprint arXiv:2406.08426* (2024).
- [29] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. GPT-4o System Card. *arXiv preprint arXiv:2410.21276* (2024). <https://doi.org/10.48550/arXiv.2410.21276>
- [30] IBM Db2. [n.d.]. <https://www.ibm.com/products/db2>.
- [31] Alekh Jindal, Shi Qiao, Sathwik Madhula, Kanupriya Raheja, and Sandhya Jain. 2024. Turning Databases Into Generative AI Machines. In *Conference on Innovative Data Systems Research (CIDR)*.
- [32] Zhi Jing, Yongye Su, Yikun Han, Bo Yuan, Haiyun Xu, Chunjiang Liu, Kehai Chen, and Min Zhang. 2024. When Large Language Models Meet Vector Databases: A Survey. *arXiv preprint arXiv:2402.01763* (2024). <https://doi.org/10.48550/arXiv.2402.01763>
- [33] Hanieh Khorashadizadeh, Fatima Zahra Amara, Morteza Ezzabady, Frédéric Ieng, Sanju Tiwari, Nandana Mihindukulasooriya, Jinghua Groppe, Soror Sahri, Farah Benamara, and Sven Groppe. 2024. Research Trends for the Interplay between Large Language Models and Knowledge Graphs. *arXiv preprint arXiv:2406.08223* (2024). <https://doi.org/10.48550/arXiv.2406.08223>
- [34] Kyoungmin Kim and Anastasia Ailamaki. 2024. Trustworthy and Efficient LLMs Meet Databases. *arXiv preprint arXiv:2412.18022* (2024). <https://doi.org/10.48550/arXiv.2412.18022>
- [35] Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2023. Can LLM Already Serve as A Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs. *Advances in Neural Information Processing Systems* 36 (2023), 42330–42357.
- [36] Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2024. LLM-R2: A Large Language Model Enhanced Rule-Based Rewrite System for Boosting Query Efficiency. *Proceedings of the VLDB Endowment* 18, 1 (2024), 53–65.
- [37] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. DeepSeek-V3 Technical Report. *arXiv preprint arXiv:2412.19437* (2024).
- [38] Microsoft SQL Server. [n.d.]. <https://www.microsoft.com/en-us/sql-server>.
- [39] Mistral. [n.d.]. <https://mistral.ai/>.
- [40] MySQL. [n.d.]. <https://www.mysql.com/>.
- [41] Neo4j. [n.d.]. <https://neo4j.com/>.
- [42] OpenAI: CLIP. [n.d.]. <https://openai.com/index/clip/>.
- [43] Oracle Database. [n.d.]. <https://www.oracle.com/database/>.
- [44] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Survey of Vector Database Management Systems. *The VLDB Journal* 33, 5 (2024), 1591–1615.
- [45] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering* 36, 7 (2024), 3580–3599. <https://doi.org/10.1109/TKDE.2024.3352100>
- [46] Simone Papicchio, Simone Rossi, Luca Cagliero, and Paolo Papotti. 2025. Think2SQL: Reinforce LLM Reasoning Capabilities for Text2SQL. *arXiv preprint arXiv:2504.15077* (2025). <https://doi.org/10.48550/arXiv.2504.15077>
- [47] Paolo Papotti. 2024. Querying Structured and Unstructured Data: LLM-first or DB-first? Keynote Talk of 29th International Conference on Database Systems for Advanced Applications (DASEAA).
- [48] Hyunbyung Park, Sukyung Lee, Gyoungjin Gim, Yungi Kim, Dahyun Kim, and Chanjun Park. 2024. Dataverse: Open-Source ETL (Extract, Transform, Load) Pipeline for Large Language Models. *arXiv preprint arXiv:2403.19340* (2024).
- [49] Linnea Passing, Manuel Then, Nina C Hubig, Harald Lang, Michael Schreier, Stephan Günnemann, Alfons Kemper, and Thomas Neumann. 2017. SQL-and Operator-centric Data Analytics in Relational Main-Memory Databases. In *20th International Conference on Extending Database Technology (EDBT)*. 84–95. <https://www.openproceedings.org/2017/conf/edbt/paper-36.pdf>
- [50] Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, and Furu Wei. 2023. Kosmos-2: Grounding Multimodal Large Language Models to the World. *arXiv preprint arXiv:2306.14824* (2023).
- [51] Maximilian Plazotta and Meike Klettke. 2024. Data Architectures in Cloud Environments. *Datenbank-Spektrum* (2024), 243–247. <https://doi.org/10.1007/s13222-024-00490-5>
- [52] PostgreSQL. [n.d.]. <https://www.postgresql.org/>.
- [53] Kaushik Rajan, Aseem Rastogi, Akash Lal, Sampath Rajendra, Krithika Subramanian, and Krut Patel. 2024. Welding Natural Language Queries to Analytics IRs with LLMs. In *CIDR*.
- [54] Sumedh Rasal. 2024. A Multi-LLM Orchestration Engine for Personalized, Context-Rich Assistance. *arXiv preprint arXiv:2410.10039* (2024).
- [55] Mohammed Saeed, Nicola De Cao, and Paolo Papotti. 2023. A DB-First Approach to Query Factual Information in LLMs. In *NeurIPS 2023 Second Table Representation Learning Workshop*. <https://openreview.net/forum?id=R8VFPafOcN>
- [56] Mohammed Saeed, Nicola De Cao, and Paolo Papotti. 2024. Querying Large Language Models with SQL. In *27th International Conference on Extending Database Technology (EDBT)*. 365–372. <https://openproceedings.org/2024/conf/edbt/paper-61.pdf>
- [57] Moritz Sichert and Thomas Neumann. 2022. User-Defined Operators: Efficiently Integrating Custom Algorithms into Modern Databases. *Proceedings of the VLDB Endowment* 15, 5 (2022), 1119–1131. <https://doi.org/10.14778/3510397.3510408>
- [58] Snowflake. [n.d.]. <https://www.snowflake.com/en/>.
- [59] Michael Stonebraker and Andrew Pavlo. 2024. What Goes Around Comes Around... And Around... *ACM Sigmod Record* 53, 2 (2024), 21–37.
- [60] Wenbo Sun, Ziyu Li, Vaishnav Srinidhi, and Rihan Hai. 2025. Database is All You Need: Serving LLMs with Relational Queries. In *International Conference on Extending Database Technology (EDBT)*. 1118–1121. <https://openproceedings.org/2025/conf/edbt/paper-326.pdf>
- [61] Zhaoyan Sun, Xuanhe Zhou, and Guoliang Li. 2024. R-Bot: An LLM-based Query Rewrite System. *arXiv preprint arXiv:2412.01661* (2024).
- [62] Jie Tan, Kangfei Zhao, Rui Li, Jeffrey Xu Yu, Chengzhi Piao, Hong Cheng, Helen Meng, Deli Zhao, and Yu Rong. 2025. Can Large Language Models Be Query Optimizer for Relational Databases? *arXiv preprint arXiv:2502.05562* (2025).
- [63] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: A Family of Highly Capable Multimodal Models. *arXiv preprint arXiv:2312.11805* (2023).
- [64] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530* (2024).
- [65] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971* (2023). <https://doi.org/10.48550/arXiv.2302.13971>
- [66] Matthias Urban and Carsten Binnig. 2024. CAESURA: Language Models as Multi-Modal Query Planners. In *Conference on Innovative Data Systems Research (CIDR)*. <https://vldb.org/cidrdb/papers/2024/p14-urban.pdf>

- [67] Matthias Urban and Carsten Binnig. 2024. Demonstrating CAESURA: Language Models as Multi-Modal Query Planners. In *Companion of the 2024 International Conference on Management of Data*. 472–475. <https://doi.org/10.1145/3626246.3654732>
- [68] Matthias Urban and Carsten Binnig. 2024. Efficient Learned Query Execution over Text and Tables [Technical Report]. *arXiv preprint arXiv:2410.22522* (2024). <https://doi.org/10.48550/arXiv.2410.22522>
- [69] Matthias Urban and Carsten Binnig. 2024. ELEET: Efficient Learned Query Execution over Text and Tables. *Proceedings of the VLDB Endowment* 17, 13 (2024), 4867–4880. <https://doi.org/10.14778/3704965.3704989>
- [70] Nikita Vasilenko, Alexander Demin, and Vladimir Boorlakov. 2025. Training-Free Query Optimization via LLM-Based Plan Similarity. *arXiv preprint arXiv:2506.05853* (2025).
- [71] Kai Waehner. 2025. *The Ultimate Data Streaming Guide: Concepts, Use Cases, Industry Stories*. Confluent.
- [72] Jiayi Wang and Guoliang Li. 2025. AOP: Automated and Interactive LLM Pipeline Orchestration for Answering Complex Queries. In *Conference on Innovative Data Systems Research (CIDR)*. <https://vldb.org/cidrb/papers/2025/p32-wang.pdf>
- [73] Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. 2025. DBCopilot: Natural Language Querying over Massive Databases via Schema Routing. In *28th International Conference on Extending Database Technology (EDBT)*. 707–721. <https://openproceedings.org/2025/conf/edbt/paper-209.pdf>
- [74] Wentao Wu. 2020. A Note On Operator-Level Query Execution Cost Modeling. *arXiv preprint arXiv:2003.04410* (2020). <https://doi.org/10.48550/arXiv.2003.04410>
- [75] Zhengtong Yan, Valter Uotila, and Jiaheng Lu. 2023. Join Order Selection with Deep Reinforcement Learning: Fundamentals, Techniques, and Challenges. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3882–3885.
- [76] Zhiming Yao, Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2025. A Query Optimization Method Utilizing Large Language Models. *arXiv preprint arXiv:2503.06902* (2025).
- [77] Fuheng Zhao, Divyakant Agrawal, and Amr El Abbadi. 2025. Hybrid Querying Over Relational Databases and Large Language Models. In *Conference on Innovative Data Systems Research (CIDR)*. <https://vldb.org/cidrb/papers/2025/p10-zhao.pdf>
- [78] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A Survey of Large Language Models. *arXiv preprint arXiv:2303.18223* (2023). <https://doi.org/10.48550/arXiv.2303.18223>
- [79] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2024. Chat2Data: An Interactive Data Analysis System with RAG, Vector Databases and LLMs. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4481–4484. <https://doi.org/10.14778/3685800.3685905>
- [80] Jiehan Zhou, Yang Cao, Quanbo Lu, Weishan Zhang, Xin Liu, and Weijian Ni. 2024. Industrial Large Model: Toward A Generative AI for Industry. In *2024 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 80–81.
- [81] Jiehan Zhou, Yang Cao, Quanbo Lu, Yan Zhang, Cong Liu, Shouhua Zhang, and Junsuo Qu. 2024. Industrial Large Model: A Survey. In *MATEC Web of Conferences*, Vol. 401. EDP Sciences, 10009. <https://doi.org/10.1051/mateconf/202440110009>
- [82] Lixi Zhou, Qi Lin, Kanchan Chowdhury, Saif Masood, Alexandre Eichenberger, Hong Min, Alexander Sim, Jie Wang, Yida Wang, Kesheng Wu, et al. 2024. Serving Deep Learning Models from Relational Databases. In *27th International Conference on Extending Database Technology (EDBT)*. 717–724. <https://openproceedings.org/2024/conf/edbt/paper-174.pdf>
- [83] Xuanhe Zhou, Chengliang Chai, Guoliang Li, and Ji Sun. 2020. Database Meets Artificial Intelligence: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 34, 3 (2020), 1096–1116.
- [84] Xuanhe Zhou, Xinyang Zhao, and Guoliang Li. 2024. LLM-Enhanced Data Management. *arXiv preprint arXiv:2402.02643* (2024).