The Duality between Large Language Models and Data Management Systems

M. Tamer Özsu and Kerem Akillioglu and Xiangru Jian

Abstract

The interaction between large language models (LLMs) and data management systems (DMSs) has emerged as one of the most consequential intersections in contemporary computer science. This interaction is bidirectional. In one direction (LLM4DB), LLMs assist, augment, or replace components of a DMS. Cooperation with LLMs facilitates tasks such as natural language querying, data cleaning and transformation, source integration, and semantic annotation. In the other direction (DB4LLM), DMS architectures and techniques support the management and preparation of managing large-scale multimodal training datasets for LLMs. They enable low-latency approximate nearest neighbour search over high-dimensional vector data for retrieval-augmented generation at inference time. These two directions exhibit a duality in which advances on one side create demand and opportunity on the other. In this paper, we report on research on both sides of this duality. In this paper, we discuss two projects in this space that we are engaged in, one in each direction.

1 Introduction

There is significant interest, both in academia and in industry, in investigating the alignment between large language models (LLMs) and data management systems (DMS)[1]. Some share the view that LLMs call into question the very reasons for the existence of the DMSs that centre around how to answer questions over data accurately and efficiently [17]. Others argue that LLMs can address the limitations of "traditional" machine learning (ML) models in data management and claim that LLMs offer opportunities throughout the data pipeline [71]. These discussions are addressing one direction of the interaction: LLMs for DMSs (LLM4DB), but the interaction is bidirectional, and there is a strong case to be made for data management technology assisting LLMs (DB4LLM). These two directions exhibit a duality where each strengthens (or can strengthen) the other.

The first direction (DB4LLM) addresses how DMS architectures and techniques can be used for the training, serving, and retrieval requirements of the LLMs. Modern LLMs are data-intensive systems, and their performance (in terms of quality) is tightly bound by the training dataset. The quality, coverage, and organization of this dataset are, therefore, a significant concern. Data engineering provides a systematic way of ensuring that the dataset is of appropriate quality through appropriate data management techniques and the data preparation pipeline. The datasets that LLMs use are large-scale, multimodal data, and the management of this data is what the data management community has expertise in. The argument is not to use the existing technologies (e.g., relational DBMSs) but the technologies that have been developed over the years (e.g., parallel data management, geo-distributed data management, NoSQL system technologies, and stream data processing). The second component of this direction is data preparation, which is the process of data acquisition, data integration, dataset

[1]We use the term "data management system" (DMS) to include systems that may not have full database management system (DBMS) functionality, such as the NoSQL systems

selection, and data quality enforcement. Data preparation ensures that the training datasets that LLMs use are of high quality [3]. A recent challenge in this direction is the incorporation into the training dataset of very high-dimensional vector data through embeddings with a new workload: approximate nearest neighbour (ANN) search. On the serving side, retrieval-augmented generation (RAG) combines LLM inference with an external knowledge store that is queried at runtime[31]. More specifically, for reasonable latency, RAG systems demand low-latency approximate nearest neighbour (ANN) search over billions of indexed vectors from multimodal data[46]. The quality of LLM-generated responses is fundamentally determined by the performance characteristics of this retrieval layer.

The second direction (LLM4DB) reverses the connection: it concerns the use of LLMs to assist, augment, or replace components of a DMS. The central point of this direction is that LLMs can assist across a wide range of DMS tasks: translating natural language into executable queries, cleaning and transforming messy data, integrating heterogeneous sources, generating semantic annotations, and supporting decision-making. In these ways, they lower the expertise barrier for interacting with complex DMSs. One of these tasks, namely the access of databases in natural language, is a long-term ambition of the database community, with early work going back to Codd [11] and with repeated attempts subsequently (e.g., [21, 32, 50, 63, 70]). This objective proved generally elusive until LLMs arrived with sufficient language understanding to make it practically viable. Today, LLM4DB initiatives are broader, and a number of major vendors embed LLM-based operators directly within SQL. This allows queries to invoke model inference on data as a first-class query operation [2].

These two directions represent the duality between them. This paper reports on our research situated on both sides. In DB4LLM direction, we describe our project that targets designing a data management stack that can support AI workloads. In LLM4DB direction, the project involves developing a multi-agent system to support natural language access to federated data sources.

2 Designing an AI-Native Data Management Stack

Providing a tighter integration of LLMs and database systems represents a new frontier in data infrastructure, one that requires solving problems beyond the scope of traditional database research. LLMs are trained on large amounts of data and are highly adaptable across downstream data management tasks such as natural language to SQL translation [58], data cleaning and integration [42], and semantic data processing [12, 25, 36, 47, 54], thereby enabling capabilities that are difficult or impossible to realize with conventional methods alone. This adaptability has already motivated direct integration of LLMs into SQL queries, also known as AI-SQL, as seen in PostgreSQL and DuckDB extensions [15, 61], as well as in offerings from major data platforms such as Google [18], Snowflake [35], and Databricks [13]. Another increasingly important use case is the emergence of LLM-based data agents that autonomously query and interact with data systems to automate complex, multi-step workflows [41]. This marks a shift from using LLMs merely as components within data processing pipelines to treating LLM agents as users of data systems [38].

However, current data systems are not yet designed for emerging LLM-agent users. The existing DMS stack, from query languages and optimizers to indexes and resource managers, continues to assume a predictable user in the background, whereas LLMs and LLM-based data agents often exhibit longer and more variable execution times, irregular invocation patterns, and inherently non-deterministic behaviour. This is important because the mismatch is fundamental rather than incidental: it challenges the assumptions under which existing data systems expose abstractions, optimize execution, allocate resources, and maintain correctness. If data systems continue to be built around assumptions that no longer hold, they risk becoming a bottleneck for emerging AI-native applications, leading to inefficiency, higher cost, increased latency, and weaker reliability. Addressing this mismatch is therefore necessary to

ensure that data systems remain an effective foundation for LLM-agent workloads. We therefore aim to redesign data systems to better support the workloads generated by LLM agents.

The problem with most of the existing LLM-database integrations is that they use the existing DMS backends and make the optimizations to accommodate LLM or LLM-based agent workloads, which leads to further issues later on. In practice, this means current solutions use existing DMS and build *scaffolding* around the models of the moment through carefully engineered prompts, model-specific retrieval strategies, and pipeline configurations tuned to the strengths and weaknesses of a particular LLM. Yet such scaffolding can be fragile, and its optimizations do not reliably transfer across model upgrades or even across model families. Empirical studies on prompt robustness show that even meaning-preserving changes such as prompt formatting, instruction paraphrases, or the presentation of few-shot examples that can induce large performance swings, and prompt variants that work best for one model may correlate only weakly with those that work best for another [53]. Overall, model-tailored workflows face even greater generalization risks when the underlying model is replaced by a newer, stronger, or higher-reasoning one.

By contrast, the literature increasingly suggests that improvements in the *systems layer* and in the design of tools for LLMs [64] are more likely to remain useful as new models emerge. Recent work has explored large-scale tool-use infrastructure [48], context management and validation mechanisms for agentic workflows [8]. These contributions do not depend on a single prompt format or on compensating for the weaknesses of one particular model; instead, they provide better interfaces, better execution strategies, and better mechanisms for supplying and organizing information. When such tools are available, LLMs can reason about which tool is most appropriate for the task at hand. As a result, building better DMSs and tools for LLMs and LLM agents is more likely to complement stronger future models rather than be invalidated by them. This distinction motivates our focus: rather than investing primarily in better scaffolding around today's models, we argue for building better systems and tools *for* LLMs and LLM agents, so that the infrastructure becomes more valuable as the underlying models improve.

2.1 Workload Characteristics

Modern use cases increasingly require querying across heterogeneous systems and data sources [29]. A single workflow may need to access both structured and unstructured stores, such as relational databases, document stores, APIs, knowledge graphs, and unstructured file collections.

2.1.1 Read Operations

Read and write operations expose the mismatch between traditional data-system assumptions and LLM-agent behaviour in fundamentally different ways. On the read side, the mismatch lies between the exact structural assumptions of traditional data systems and the greater tolerance of LLM agents for ambiguity. In conventional systems, schema serves two main purposes: to preserve integrity under update (cf. normal forms), and to support efficient query processing and integrity enforcement. When the consumer is an LLM performing read-and-synthesize workloads, rather than executing precise analytical queries or maintaining referential integrity, both justifications weaken. LLMs are trained over enormous corpora of messy, weakly aligned, and heterogeneous data, and thus have strong priors for resolving ambiguity across loosely structured sources. This resembles classical schema integration, but with a key difference: the consumer itself is semantically flexible, shifting part of the reconciliation burden from middleware to the model. As a result, systems may be able to relax some conventional schema assumptions for read operations. At the same time, this flexibility complicates optimization, since cost models must account not only for data access cost, but also for semantic interpretation, token

expenditure, and the reliability of model-mediated retrieval.

2.1.2 Write Operations

Write operations are less forgiving. When multiple LLM agents operate concurrently over shared mutable state, coordination and consistency cannot be left to the agents themselves. Unlike read-side ambiguity, where the model can often recover a plausible interpretation, concurrent writes expose the system to classical anomalies: lost updates when two agents overwrite the same artifact without awareness of one another, stale or dirty reads when one agent acts on partially updated state, and violations of causal ordering when logically dependent actions execute against inconsistent snapshots. These are familiar problems in database systems, but they become novel in agentic settings because agent outputs are non-deterministic, write intentions are harder to predict in advance, and multi-step actions may span both structured and unstructured states.

The central design question is therefore not whether coordination is necessary, but when it is necessary and what form it should take. Some classes of agent-written state admit coordination-free operation: append-only logs, monotonically growing sets, or last-writer-wins registers for non-critical metadata may tolerate weak consistency or well-defined merge semantics. Other classes of state require stronger guarantees, but the required strength depends on the property of interest: local constraint preservation, causal consistency, or full transactional isolation. This suggests that agent-oriented systems should expose a richer space of write semantics than the traditional binary choice between strict transactions and weak eventual consistency.

One promising point in this design space is optimistic concurrency control (OCC). Rather than synchronizing all agent actions eagerly, the system may allow speculative progress and validate updates at commit time. This is attractive in agentic settings because many concurrent actions will not conflict, while eager coordination may impose unnecessary latency and serialization. Yet OCC alone may be insufficient when writes are uncertain, semantically ambiguous, or difficult to validate automatically. In such cases, branching becomes a useful generalization of optimism: instead of forcing uncertain or potentially conflicting updates directly into shared state, the system can isolate them into separate branches for later validation, comparison, merging, or approval. In this sense, branching can be viewed as an extension of OCC for agentic environments, one that accommodates not only conflict detection, but also deferred semantic reconciliation and human oversight where needed.

For state that does not require strict isolation, but for which coordination-free convergence is too weak, probabilistic consistency models offer a principled middle ground. Probabilistically Bounded Staleness (PBS) [5] showed that eventually consistent partial-quorum systems often return consistent data within milliseconds of a write, and that staleness can be quantified continuously along both version and time axes rather than treated as a binary property. This perspective is especially appealing for agentic workloads, where some reads—for example, over context summaries, cached tool outputs, or non-critical metadata—may tolerate bounded staleness in exchange for lower latency. PBS therefore provides a useful foundation for reasoning about freshness-latency trade-offs in LLM-facing systems. However, the original PBS analysis focused on single-key staleness. Extending such reasoning to the multi-key, multi-agent setting required by agentic architectures, where causal relationships and atomicity requirements span multiple objects and actions, remains an open problem.

2.2 Deep Dive into the Stack

Redesigning data systems for LLM-agent users brings challenges with little precedent in traditional database research and calls for a fundamental rethinking of the data stack. This requires not only revisiting existing components, but also introducing new ones tailored to the demands of these emerging

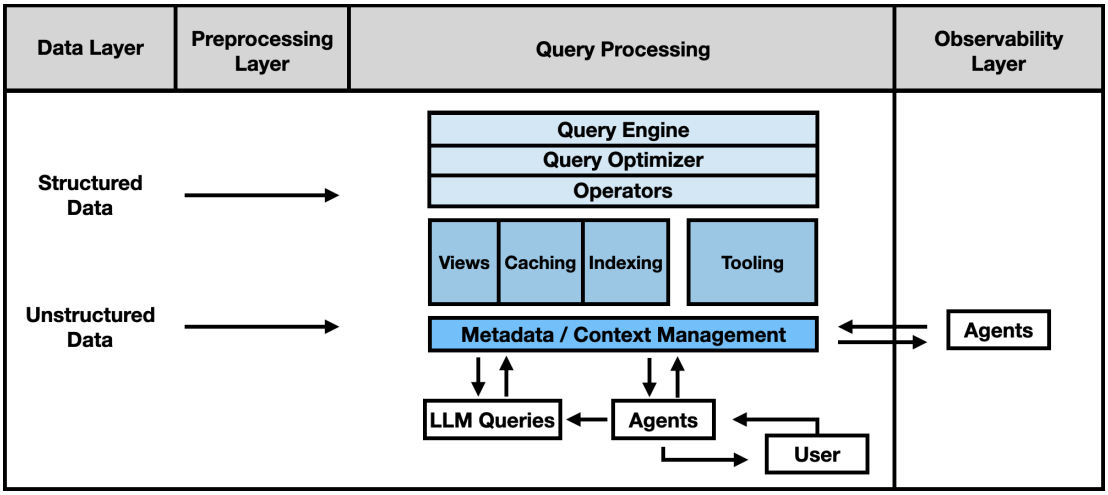

Figure 1: AI Native Stack Architecture.

workloads. Across these design choices, three key dimensions are worth optimizing for: accuracy, cost, and system efficiency. Figure 1 illustrates our overall architecture. In this section, we discuss the key aspects of workload characteristics, system's querying interface, the adaptations needed in the existing DMS architecture, and the new components required to support the LLM-agent workload.

2.2.1 Querying Interface

Heterogeneous querying naturally raises the question of how query interfaces should be reconsidered for the new system. A structured declarative interface is particularly helpful when large or complex queries are difficult to express reliably in natural language alone. At the same time, even as workloads change, much of the world's valuable data remains stored in relational DBMSs. For that reason, we emphasize that nearly any system developed in this space should continue to support SQL, given its longevity and central role in data management [59]. A natural direction, then, is to build on top of SQL rather than discard it entirely. AI-SQL syntax can be viewed in this light, as a more expressive interface that extends SQL to better support LLM functionality, heterogeneous data access, and agent-driven workloads while preserving a declarative style. However, the implications of these new workloads for query languages and interfaces go beyond simply extending SQL. Prior critiques have already pointed out that SQL's irregular syntax, implicit ordering between clauses, and limited abstraction mechanisms create usability challenges even for human users [43]. For LLM agents, these issues become even more pronounced: irregular grammar increases the error surface for query generation, verbosity inflates token costs at scale, and fragmentation into incompatible dialects across systems makes querying heterogeneous sources especially fragile. This suggests value not only in supporting SQL and exploring extensions such as AI-SQL, but also in investigating alternative representations and interfaces that may reduce token cost, improve robustness, and better fit multi-source, agent-oriented workloads.

2.2.2 Rethinking Existing Components and Building New Ones

Building views, caching, and indexing will remain central, but they need to be reconsidered for LLM-agent workloads. Classical work has already shown how powerful views can be for query answering, source integration, and reusing expensive computation across queries [20]; in the same spirit, an AI-native stack should revisit these mechanisms under the assumptions of semantic, multi-step, and heterogeneous workloads. A view may no longer be just a stored relational subquery, but also a reusable

intermediate result such as a filtered context set, an extracted schema summary, or a semantically enriched representation prepared for downstream reasoning. Caching similarly extends beyond tuples and pages to model responses, prompt-conditioned retrieval results, and intermediate outputs whose reuse depends not only on key equality, but also on task context, approximation tolerance, and model choice. Indexing, in turn, must support not only exact predicate evaluation, but also semantic access paths over embeddings, metadata, and cross-source references. The point is that the role of traditional components expands: they must now help the system reduce repeated model calls, control token and compute cost, and support more efficient and reliable execution over heterogeneous data and agent-driven workflows.

Managing context becomes a first-class systems problem in our setting. Unlike a traditional query processor, an agent repeatedly invokes an LLM across multiple steps, and each step must decide what information from prior interactions, retrieved data, tool outputs, and task instructions should be carried forward into the next call. This makes context construction far more than a prompt-writing issue: it requires dynamically selecting, compressing, and structuring the right state for the current decision. Prior work already shows that seemingly small changes in prompt wording or example selection can materially affect performance [42, 49], while irrelevant information in the input can degrade quality even when the relevant information is present [57]. For agentic systems, this implies that context should not be treated as a raw transcript or a full dump of available state, but as a curated representation of only the information necessary for the next action. A system that exposes too little context leaves the agent under-informed; a system that exposes too much increases token cost, dilutes attention, and raises the likelihood of error.

Metadata management is equally important because agents often rely on auxiliary information to interpret data, select tools, and decide what to do next. Schema descriptions, data dictionaries, tool specifications, retrieved examples, execution history, and summaries of prior intermediate results can all improve performance when they are relevant, but they also introduce overhead and can interfere with reasoning when included indiscriminately. This tension becomes more severe in multi-step workflows, where mistakes in early context selection can compound over time. Studies of agentic search, for example, suggest that once an agent is nudged toward the wrong local path, it may continue refining that path rather than broadening its search, amplifying early errors across subsequent steps [44]. From this perspective, metadata is not just supporting information; it is part of the control surface of the system. The key challenge is therefore not merely to store metadata, but to determine what metadata should be surfaced, in what form, and at which step, so that the agent remains both well-grounded and efficient.

Tools are equally important in this setting because they shape how agents plan, decompose, and execute tasks over data systems. Examples include schema inspection tools that reveal available tables and attributes, query execution tools that allow the agent to test and refine candidate queries, vector retrieval tools that fetch relevant documents or rows, validation tools that check whether outputs satisfy constraints, and cost-estimation tools that help compare alternative plans. Consider an agent asked to "find customers whose recent support tickets mention billing issues and summarize the common causes." Without tools, the agent would have to guess the schema, write a query from memory, and reason about the results entirely within the model. With tools, it can first inspect the schema to discover the ticket and customer tables, issue a query to retrieve recent billing-related tickets, use retrieval or semantic filtering to identify relevant text, validate that the join keys and predicates are correct, and then summarize only the filtered results. In this way, tools do not merely assist execution; they help the agent form better plans by grounding each step in system feedback, reducing unnecessary reasoning, and making multi-step interaction with heterogeneous data more reliable.

Taken together, these considerations have direct implications for the design of operators, the query optimizer, and the execution engine. The system can no longer rely solely on traditional relational operators and cost assumptions, but must introduce operators that account for semantic retrieval, model invocation, validation, tool use, and interaction across heterogeneous sources. The optimizer, in turn,

must reason not only about compute and I/O, but also about token cost, latency, model selection, approximation, and the uncertainty introduced by non-deterministic components. The execution engine must support adaptive, multi-step execution in which later decisions may depend on intermediate model outputs, tool feedback, and runtime conditions. Supporting LLM-agent workloads is therefore not simply a matter of attaching models to an existing engine; it requires rethinking the core machinery that determines what operations are available, how plans are chosen, and how those plans are executed.

2.3 Query Processing and Optimization

For the systems layer beneath AI agents, the central query-processing challenge is not merely supporting larger datasets, but supporting repeated and expensive interaction with LLM-based users. Each agent request may trigger multiple rounds of retrieval, reasoning, validation, and tool invocation, making execution costlier, less regular, and harder to predict than in traditional data systems. As a result, the system is no longer optimizing only for fast execution of a well-specified query, but for dependable support of requests that unfold through uncertain and adaptive steps. This tightly couples cost, latency, and accuracy: improving accuracy may require richer context, additional verification, or stronger models, each of which increases both response time and system cost. The role of the system, then, is not merely to execute requests efficiently, but to manage these tradeoffs in a principled way.

A major source of this difficulty is that query processing is increasingly dominated by the AI layer rather than by conventional database execution. Retrieval, filtering, joins, and other database operations may still complete relatively quickly, while the surrounding LLM calls dominate end-to-end latency and monetary cost. One practical observation is that not every interaction requires the same level of reasoning power: simpler cases may be handled by cheaper models, while only harder cases justify the use of frontier models. This motivates mechanisms such as *model cascades*, which have been explored in adjacent settings and, more recently, in AI-SQL and LLM-powered data processing systems [4, 9, 27, 39, 47]. From the perspective of the underlying system, however, cascades are only one part of the solution. They reduce the cost of individual model decisions, but do not change the fact that the system may still need to support too many model invocations overall. Running models locally shifts this burden from API spending to infrastructure provisioning, but does not eliminate it; the bottleneck simply moves from the billing layer to the GPU layer [2]. Thus, the core problem is not only where inference runs, but how the system determines when inference is necessary at all.

This has important implications for jointly considering logical and physical optimizations [1]. Existing work has explored physical optimizations such as batching, prefix caching, KV-cache compression, and model selection [37, 52, 55], as well as logical optimizations such as semantic operators and rewrite strategies that alter what computation is performed and in what order [15, 25, 47, 54]. However, these examples are largely drawn from AI-SQL and LLM-powered data-processing systems rather than from systems designed natively around AI-agent users. For a system serving AI agents, these two levels cannot be treated independently. Decisions about rewriting, decomposition, or tool selection affect how many model calls are made and what physical optimization opportunities remain; conversely, knowledge of model cost, cacheability, and latency should influence which logical alternatives the system prefers. Query optimization therefore becomes inherently multi-objective: the system must balance monetary cost, end-to-end latency, and overall system efficiency, while also accounting for approximation and uncertainty. The rewrite space becomes correspondingly richer. Optimization is no longer limited to algebraic equivalences inside a query plan, but may involve rewriting an entire execution pipeline, changing the order of retrieval and filtering, deciding when to rely on symbolic execution instead of a model, selecting among cascades of different cost and capability, or replacing repeated reasoning with cached or precomputed results. In this sense, the design challenge is broader than extending an existing engine with LLM support. It requires a system that can jointly reason about logical and physical choices

for AI-agent workloads. To our knowledge, this agent-native systems perspective remains largely open in the current literature.

3 A Multi-Agent System for Analytics over Heterogeneous Federated Systems

The preceding section addresses the cost of invoking LLMs at the granularity of individual data points. A complementary problem arises in *analytical* workloads: complex queries that combine evidence from multiple autonomous data sources to support decision-making. We define this problem, characterize what makes it hard, explain why existing approaches do not solve it, and present a multi-agent architecture that addresses the gap.

3.1 Problem Definition

Modern data science projects involve multimodal data from many different sources. Structured facts reside in relational DBMSs, entities and relationships required for linkage and reasoning are maintained in knowledge graphs, and narrative context and business details are recorded in document corpora. These data are held and managed by autonomous systems, each retaining its own data model, query interface, and operational semantics. In the database literature, such a setting is a *heterogeneous federated system*: a collection of autonomous data sources that must be queried together without physical centralization [45, 56]. The defining properties are source autonomy, heterogeneity, and virtual rather than materialized integration [56]. Traditionally, such federations are accessed through dedicated wrappers that expose each source's capabilities, coordinated by a mediator that decomposes queries across them [45, 66]. We extend that model beyond database sources to include knowledge graphs and document collections alongside relational DBMSs. Two structural features of this federated setting are central. First, the data is *multimodal*. Second, the federation is *heterogeneous* not only across modalities but within them: two relational DBMSs may expose different SQL dialects and schemas; two document collections may differ in whether layout information is indexed. No global schema governs the federation, and the system must discover what each source can do at query time [45].

We consider *analytical workloads*: mostly read-only, decision-support queries over historical data that combine operators such as filtering, aggregation, comparison, and linkage to identify trends, generate reports, and derive insights. We extend traditional relational OLAP workloads to those that may draw on semi-structured and unstructured data and require reasoning that spans modality boundaries. Consider the query: *"Assess the credit risk exposure across our top 20 suppliers over the past fiscal year: correlate payment delays and outstanding receivables with corporate ownership structures to identify concentrated risk, and flag any supplier whose audited financial statements show deteriorating liquidity ratios or covenant breaches."* Answering this requires relational aggregation over procurement and accounts-payable records (payment timeliness, invoice aging, outstanding receivables); graph traversal over corporate ownership and cross-guarantee relationships, discovering, for instance, that three apparently independent suppliers are subsidiaries of the same parent entity, which concentrates risk; and layout-aware extraction from audited financial statements and credit rating reports, visually complex PDF documents containing multi-column financial tables and trend charts whose liquidity ratios, debt-to-equity figures, and auditor qualification notes are not available in any structured source.

No single source or modality suffices, and the difficulty is not merely additive. Four properties distinguish such complex analytical tasks from simpler cross-source lookups:

1. **Source involvement is query-dependent**: which sources are relevant cannot be determined before the query is issued, i.e., data localization is required for each query. In the example, the

supplier ownership graph becomes necessary only after the relational aggregation reveals which suppliers qualify.

2. **Reasoning trajectories are correlated**: each step may condition on the output of previous steps, so that errors or omissions at any stage compound rather than average out. The ownership resolution, for instance, depends on the set of suppliers identified by the initial aggregation.

3. **Sources may require iterative access**: intermediate results may reveal new information needs that require returning to a previously consulted source, or accessing a source that was not part of the original plan. The initial graph traversal may surface a parent entity whose financial statements must then be retrieved, triggering a second round of document extraction not foreseeable at planning time.

4. **Evidence must be synthesized across modalities with provenance**: the final answer must compose results whose representations, semantics, and granularity differ across sources, and present the composite result with a trace sufficient for a domain expert to verify each claim against its origin. The credit risk assessment must reconcile a numeric total from a relational table, an ownership path from a knowledge graph, and an auditor qualification from a PDF, each traceable to its source.

These properties make it impractical for users to manually formulate and execute such queries. Domain experts understand what they need to know, but cannot efficiently execute the multi-step, multi-paradigm procedures their questions require [26, 28]. Analytics teams face backlogs that span weeks or months, and many questions go unasked because the cost of formulating them exceeds the perceived payoff.

LLMs offer a path forward: their natural language understanding, code generation, and tool invocation capabilities make it possible to interact with diverse data sources on the user's behalf, bridging the gap between what domain experts can ask and what federated systems can execute. This work falls within the direction of using LLMs for data management, specifically using LLM-based agents to perform complex analytical tasks over heterogeneous federated systems that were previously infeasible without deep technical expertise across multiple query paradigms.

3.2 Why Existing Approaches Fall Short

Several existing approaches address parts of this problem, but none solves it fully.

Single-modality natural language interfaces. Text-to-SQL systems translate natural language into SQL [23, 40], KGQA systems generate SPARQL queries [30], and RAG systems retrieve passages and condition a language model on them [19, 31]. Each is confined to its modality: when an analytical task spans multiple modalities, no single-modality interface can serve as the sole solution.

Polystore and data federation systems. Polystore systems such as BigDAWG [16] and query frameworks such as Apache Calcite [6] provide execution infrastructure across multiple storage engines. However, cross-engine decomposition remains a user responsibility, they require formal query languages (which do not help domain experts who lack the technical skills to write them), they rely on upfront schema mediation that contradicts the virtual integration model of federated systems [56], and they primarily target structured data with limited support for unstructured documents.

LLM-based agents. LLM-based agents [69] can accept natural language and invoke external tools in sequence, but the four complexity properties expose fundamental limitations. Because sources differ in what they can do and the relevant sources are not known in advance, the agent must reason about each source's characteristics; in practice, single-agent systems treat all APIs as interchangeable [62]. Because reasoning trajectories are correlated, an error in an early step propagates through all subsequent steps

with no mechanism for intermediate validation. When combined with RAG, all data access is reduced to retrieving passages and then generating text, which cannot replace the structured computation that different sources provide [19]. The agent also typically produces a final answer without maintaining a record of which source contributed which claim.

General-purpose multi-agent frameworks. AutoGen [68], Camel [33] and MetaGPT [22] allow multiple agents to collaborate, but they are general-purpose coordination tools, not designed for heterogeneous data analytics. Agents decide what to do next through conversation without an explicit plan that can be validated before execution. The frameworks do not model what each data source can do, so source-level reasoning limitations carry over from single-agent designs. Intermediate results are exchanged as unstructured text without provenance records.

In sum, none of these approaches fully addresses the problem. To enable complex analytical tasks over heterogeneous federated systems, we need a dedicated multi-agent architecture in which each data source is served by a specialized agent that understands that source's specific characteristics, coordinated by a planning and execution layer that can reason about what each source can do, validate intermediate results, and assemble a traceable final answer. We refer to this coordination as *compositional orchestration*.

3.3 A Multi-Agent Architecture

The traditional mediator/wrapper approach [45, 66] works when queries are expressed in a formal language, the relevant sources can be identified from the schema, and decomposition follows deterministic rewriting rules. The four complexity properties require capabilities beyond static query processing: interpreting natural language input, determining which sources are relevant as intermediate results reveal new needs, evaluating partial results to decide whether to proceed or re-plan, and composing evidence across representations with no shared data model. These decisions depend on the content and quality of intermediate results at runtime and cannot be specified as rewriting rules before execution begins. What might work better is an *agentic* approach: system components that can autonomously perceive their operating context, reason about what action to take, and adapt based on observed outcomes [67].

The heterogeneity of sources, not just across modalities but within them, means that interacting with a given source requires understanding that source's specific interface, schema, and operational constraints. A single agent cannot maintain this depth across all sources in the federation. This motivates a *multi-agent* design in which each source is served by a dedicated specialist. Figure 2 shows the resulting architecture, which comprises five components and a provenance-by-construction design. The upper layer contains three *general agents* (User Agent, Orchestrator, Executor) that handle query refinement, planning, and execution coordination, all grounded in a shared Metadata Repository (MR). The lower layer contains *Specialized Data Agents* (SDAs), each dedicated to a specific data source and interacting with it via native query execution. Provenance records flow through the entire pipeline, from individual SDA results back through the Executor to the final answer presented by the User Agent.

Specialized Data Agents (SDAs). Each data source is served by a dedicated agent operating within a purpose-built *execution environment* that determines which operations the agent can perform. We call the complete set of these operations the source's *action space*. The action space includes all operations exposed by the source's native query interface as well as any prebuilt operations for frequently needed tasks. For this prototype, we develop an SDA for relational DBMSs whose action space covers the full SQL interface and schema exploration [34], one for knowledge graphs covering SPARQL and ontology navigation with tool-augmented entity resolution [24], and one for document collections covering retrieval and layout-aware extraction [51]. The execution environment may additionally include higher-level operations such as parameterized query templates or domain-specific extraction routines that extend

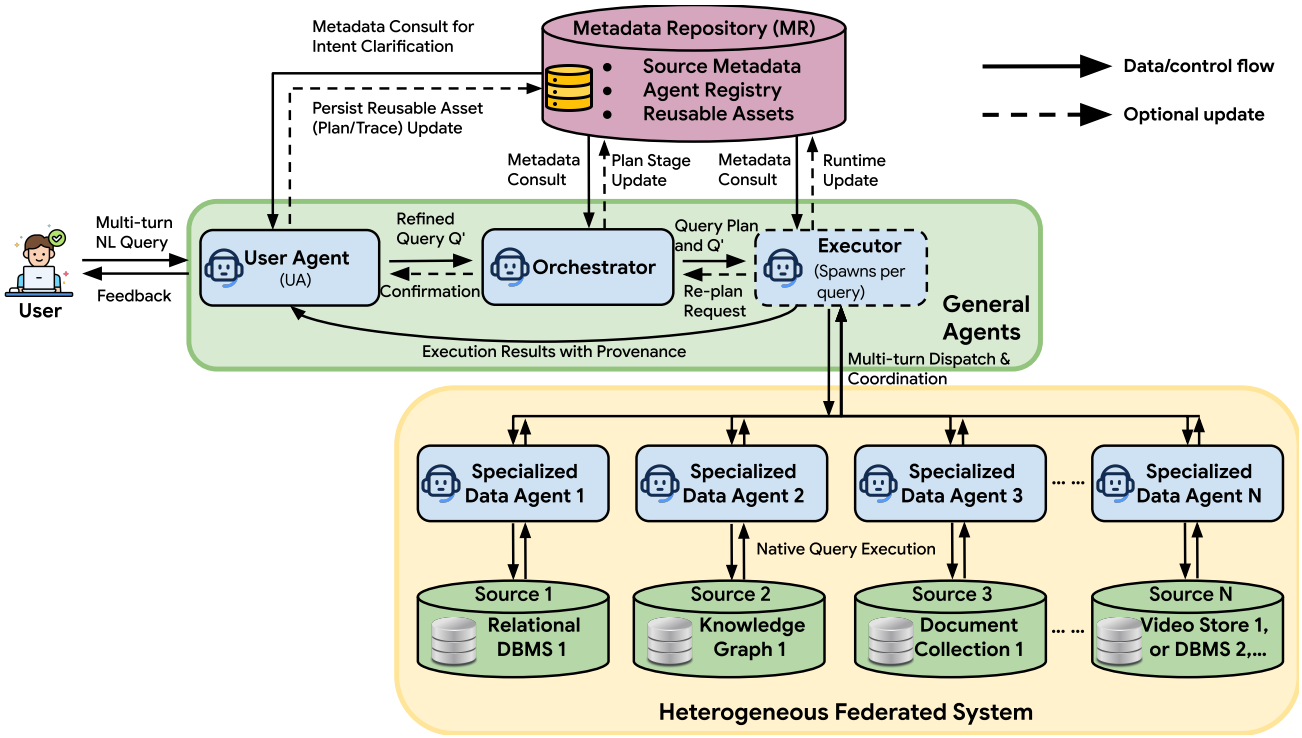

Figure 2: Multi-agent architecture for analytics over heterogeneous federated systems. The User Agent refines queries through multi-turn dialogue grounded in MR; the Orchestrator consults MR to produce an explicit query plan; the ephemeral Executor dispatches subtasks to source-level SDAs via multi-turn coordination, validates intermediate results, and can request re-planning. Each SDA operates within a purpose-built execution environment tailored to its source's native interface. Specialization is at the source level, not the modality level: Source N illustrates that additional sources of any type (e.g., a second relational DBMS or a video store) are each served by their own SDA instance.

the action space beyond raw query execution. Although each agent targets a particular modality, specialization occurs at the source level: two relational DBMSs with different schemas are served by two distinct agent instances.

Metadata Repository (MR). The system maintains a central record of what each source contains and what each agent can do, including source-level metadata, an agent registry, and previously successful query plans. This allows the system to operate without a global schema: it looks up the characteristics of each source in MR when planning a query. When a source is added or changes, only MR is updated. As shown in Figure 2, MR is consulted by all components: the UA for user intent clarification, the Orchestrator for plan-stage metadata lookup, and the Executor for runtime re-routing decisions.

Orchestrator. Given a refined query Q' from the UA (Figure 2), the Orchestrator consults MR and produces a query plan consisting of subtasks assigned to specific sources with explicit dependencies. It does not execute the plan itself but passes the plan and Q' to an Executor. Separating planning from execution prevents any single component from having to both decide what to do and carry it out, which becomes unmanageable as the number of sources and the depth of reasoning grow. The separation also enables parallelism: the Orchestrator can continue planning new queries while multiple Executors concurrently carry out previously issued plans. The explicit plan can be validated before any source is

queried, catching problems such as assigning a subtask to a source whose action space does not include the required operation.

Executor. The Executor carries out the plan by dispatching subtasks to SDAs through multi-turn coordination (Figure 2), managing the flow of intermediate results, and handling parallel and sequential execution. Each Executor is spawned per query and discarded afterward. Before passing any result to the next step, the Executor validates it and can re-route to an alternative source or reformulate the subtask if something goes wrong. When more fundamental issues arise, it sends a re-plan request back to the Orchestrator This intermediate validation is the primary mechanism for preventing per-step errors from compounding through the plan. The Executor's execution trace and successful patterns can be recorded in MR as runtime updates for future reuse.

User Agent (UA). A conversational entry point that receives natural language queries and engages in multi-turn dialogue to clarify ambiguous references, grounded in MR. A reference to "sales data" may correspond to a CRM database or an analytics warehouse; the UA consults MR to determine which sources exist and what they contain, enabling disambiguation grounded in the actual data landscape. On the output side, the UA presents the final answer with provenance information, allowing the user to verify each claim, and can persist successful query plans and execution traces back to MR as reusable assets.

Provenance-by-construction. Every intermediate result carries a record of which source produced it and what evidence supports it. As results are combined across sources, these records are preserved and composed. The final answer includes a complete trace, binding each claim to its source evidence. When sources disagree or evidence is insufficient, the system surfaces the conflict rather than silently choosing one version. We call this provenance-by-construction: claim-evidence bindings are built incrementally as results flow through the pipeline, rather than attached after the fact.

3.4 Open Challenges

This architecture raises several research challenges that the community has not yet addressed.

Query understanding and disambiguation. The UA receives natural language queries that may contain ambiguous references, implicit constraints, or domain-specific terminology. Before any planning can begin, the system must resolve what the user is asking. A reference to "sales data" may correspond to a CRM database, an analytics warehouse, or both; a request for "recent" results requires a specific time range. The UA consults MR to determine which sources exist and what they contain, enabling it to ground the disambiguation in the actual data landscape rather than guessing. When the query remains ambiguous after one pass, the UA engages in multi-turn dialogue to refine the intent before forwarding a resolved query to the Orchestrator. This challenge is a prerequisite for all subsequent steps: an incorrectly interpreted query yields a correct plan for the wrong question.

Dynamic decomposition under unknown source requirements. The Orchestrator must decompose a query into subtasks and assign each to a source, but these decisions are interdependent. The best decomposition depends on which sources are available, and which sources are relevant may only become clear as intermediate results arrive. In the credit risk example, the supplier ownership graph becomes necessary only after the relational aggregation identifies qualifying suppliers. This demands iterative or conditional planning, not just a single upfront decomposition. Existing plan-generation methods do not account for this kind of dynamic source discovery [62].

Routing and execution with intermediate validation. Even with a sound plan, executing it correctly requires matching each subtask to a source whose action space includes the required operation and validating intermediate results before they are consumed downstream. This matching must account for differences both across and within modalities: a subtask requiring graph traversal cannot be sent to a relational DBMS, but equally, a subtask requiring a specific table cannot be sent to an arbitrary DBMS. Without this intermediate validation, per-step errors compound multiplicatively through the plan. Designing validation policies that detect subtle type mismatches or incomplete results without access to ground truth remains an open problem.

Cross-source entity resolution. Entities are represented differently across modalities: a supplier appears as a row with a vendor ID in a relational table, as a named entity in a knowledge graph, and as a name mention in a PDF. Before results from different sources can be correctly combined, these representations must be aligned. This resolution is not a one-time preprocessing step but arises dynamically during execution, because the entities to be aligned depend on intermediate results produced by earlier subtasks.

Cross-modality provenance composition. Database provenance [7] and truth discovery [14] are well-studied individually, but composing provenance across modality boundaries in LLM-based pipelines is largely open. When a claim depends on a reasoning chain that spans all three modalities, the provenance record must compose across these boundaries while remaining interpretable to a human user. The challenge is twofold: reconciling different forms of evidence into a unified trace that a domain expert can follow, and detecting when sources conflict or when the available evidence is insufficient to support a claim.

Source registration and metadata evolution. The architecture depends on MR maintaining an accurate record of each source's action space, metadata, and access constraints. In practice, these records are inherently incomplete even for a static federation, because fully exploring every source upfront is infeasible: a relational DBMS may contain thousands of tables with undocumented columns. The system must therefore operate under partial knowledge and refine its records incrementally as queries reveal new information about what each source contains and can do. Beyond this initial incompleteness, sources also evolve over time: a relational DBMS may add new tables or deprecate old ones, a document collection may be re-indexed with richer metadata, and a knowledge graph may incorporate a large batch of new entities. If MR's records are stale or incomplete, the Orchestrator may route subtasks to sources that cannot perform the required operation, or miss sources that could have contributed. Bootstrapping records for new sources, refining them through query experience, and detecting when existing records have become outdated are all open problems that affect the reliability of planning and routing.

Evaluation gaps. Existing cross-source benchmarks such as HybridQA [10] and MultiModalQA [60] combine a small number of modalities and do not require source discovery, within-modality routing, or provenance assembly. More recent efforts such as FDA-Bench [65] expand modality coverage but do not isolate the contributions of the orchestration layer or evaluate provenance quality. Building benchmarks that jointly test capability-aware routing, provenance assembly, and adaptive re-planning, including gold decomposition plans and gold provenance records, is itself a significant research challenge.

4 Conclusion

In this paper we presented two research projects that are placed on the opposite sides of the LLM-DMS duality. On the DB4LLM side, the project aims to support LLM-agent workloads that go beyond attaching a model to an existing infrastructure; it requires a new design of the data stack. The mismatch between traditional DMS assumptions and the non-deterministic, adaptive behaviour of LLM agents is structural. Therefore, addressing these workloads requires building systems whose features are designed to support agent workloads. On the LLM4DB direction, we describe a project that explores a multi-agent approach for analytical queries over heterogeneous federated systems. In this system specialized data agents, a metadata repository, and a distinct planning-and-execution layer interact to enable complex, multi-step queries expressed in natural language. The queries execute over multiple data sources including relational databases, knowledge graphs and document collections.

Taken together, these two projects illustrate the some of the issues central to the LLM-DMS duality: advances in data management infrastructure make LLM-based analytical systems more reliable and efficient, while the demands of LLM-agent workloads expose limitations in existing DMS designs and motivate new DMS designs. The challenges highlighted by these projects are largely open problems. We hope that framing these problems in terms of the broader duality will encourage the community to engage with both directions together, recognizing that progress on one side creates both demand and opportunity on the other.

References

[1] Kerem Akillioglu. Optimizing data systems for LLM workloads. In *PhD Workshop at VLDB 2025*, 2025. Co-located with VLDB 2025.

[2] Kerem Akillioglu, Anurag Chakraborty, Sairaj Voruganti, and M. Tamer Özsu. Research challenges in relational database management systems for LLM queries. In *Proc. 6th Intl. Wkshp on Applied AI for Database Syst. and Appl.*, 2025. Co-located with VLDB 2025.

[3] Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, Colin Raffel, Shiyu Chang, Tatsunori Hashimoto, and William Yang Wang. A survey on data selection for language models. *Trans. Machine Learning Res.*, 2024. Survey Certification.

[4] Michael R. Anderson, Michael Cafarella, German Ros, and Thomas F. Wenisch. Physical representation-based predicate optimization for a visual analytics database. In *Proc. 35th IEEE Intl. Conf. on Data Engineering*, pages 1466–1477, 2019.

[5] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Quantifying eventual consistency with PBS. *Commun. ACM*, 57(8):93–102, August 2014.

[6] 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 *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 221–230, 2018.

[7] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In *Proc. 8th Intl. Conf. on Database Theory*, pages 316–330. Springer Berlin Heidelberg, 2001.

[8] Edward Y. Chang and Longling Geng. Sagallm: Context management, validation, and transaction guarantees for multi-agent LLM planning. *Proc. VLDB Endowment*, 18(12):4874–4886, August 2025.

[9] Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: how to use large language models while reducing cost and improving performance. arXiv 2305.05176, 2023.

[10] Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036, Online, November 2020. Association for Computational Linguistics.

[11] E. F. Codd. Seven steps to rendezvous with the casual user. In J. W. Klimbie and K. L. Koffeman, editors, *Proc. IFIP Working Conf. Data Base Management*, pages 179–200, 1974.

[12] Hanjun Dai, Bethany Y Wang, Xingchen Wan, Bo Dai, Sherry Yang, Azade Nova, Pengcheng Yin, Phitchaya M Phothilimthana, Charles Sutton, and Dale Schuurmans. UQE: A query engine for unstructured databases. *Advances in Neural Information Processing Syst., Proc. 37th Annual Conf. on Neural Information Processing Syst.*, 37:29807–29838, 2024.

[13] Databricks Inc. Apply AI on data using Databricks AI functions. Databricks Documentation, 2025. Last updated Feb 5, 2026. Accessed: 2026-03-01.

[14] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Data fusion: resolving conflicts from multiple sources. In *Proc. 14th Intl. Conf. on Web-Age Information Management*, page 64–76, Berlin, Heidelberg, 2013. Springer-Verlag.

[15] Anas Dorbani, Sunny Yasser, Jimmy Lin, and Amine Mhedhbi. Beyond quacking: Deep integration of language models and rag into duckdb. *Proc. VLDB Endow.*, 18(12):5415–5418, August 2025.

[16] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magdalena Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stanley B. Zdonik. The BigDAWG polystore system. *ACM SIGMOD Rec.*, 44(2):11–16, 2015.

[17] Raul Castro Fernandez, Aaron J. Elmore, Michael J. Franklin, Sanjay Krishnan, and Chenhao Tan. How large language models will disrupt data management. *Proc. VLDB Endowment*, 16(11):3302–3309, 2023.

[18] Tianxiang Gao and Derrick Li. BigQuery AI supports Gemini 3.0, simplified embedding generation and new similarity function. Google Cloud Blog, January 2026. Accessed: 2026-02-24.

[19] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. arXiv 2312.10997, 2024.

[20] Alon Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, December 2001.

[21] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2):105–147, 1978.

[22] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *Proc. 12th Intl. Conf. on Learning Representations*, 2024.

[23] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-Generation Database Interfaces: A Survey of LLM-Based Text-to-SQL. *IEEE Trans. Knowl. and Data Eng.*, 37(12):7328–7345, December 2025.

[24] Xiangru Jian, Zhengyuan Dong, and M. Tamer Özsu. InteracSPARQL: An interactive system for SPARQL query refinement using natural language explanations. arXiv 2511.02002, 2025.

[25] Saehan Jo and Immanuel Trummer. Thalamusdb: Approximate query processing on multi-modal data. *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 2(3), May 2024.

[26] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2917–2926, December 2012.

[27] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *Proc. VLDB Endow.*, 10(11):1586–1597, August 2017.

[28] Georgia Koutrika. Natural Language Data Interfaces: A Data Access Odyssey. In Graham Cormode and Michael Shekelyan, editors, *Proc. 27TH Intl. Conf. on Database Theory*, volume 290 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:22, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[29] Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, Sivaprasad Sudhir, Anna Zeng, Anton A. Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, Jun Chen, Markos Markakis, Matthew Russo, Weiyang Wang, Ziniu Wu, Mike Cafarella, Lei Cao, Samuel Madden, and Tim Kraska. KRAMABENCH: A benchmark for AI systems on data-to-insight pipelines over data lakes. In *Proc. 14th Intl. Conf. on Learning Representations*, 2026.

[30] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. Complex knowledge base question answering: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 35(11):11196–11215, November 2023.

[31] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Syst., Proc. 34th Annual Conf. on Neural Information Processing Syst.*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020.

[32] Fei Li and H. V. Jagadish. Constructing an Interactive Natural Language Interface for Relational Databases. *Proc. VLDB Endowment*, 8(1):73–84, 2014.

[33] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Syst., Proc. 36th Annual Conf. on Neural Information Processing Syst.*, 2023.

[34] Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. OmniSQL: Synthesizing high-quality text-to-sql data at scale. *Proc. VLDB Endow.*, 18(11):4695–4709, July 2025.

[35] Paweł Liskowski, Benjamin Han, Paritosh Aggarwal, Bowei Chen, Boxin Jiang, Nitish Jindal, Zihan Li, Aaron Lin, Kyle Schmaus, Jay Tayade, Weicheng Zhao, Anupam Datta, Nathan Wiegand, and Dimitris Tsirogiannis. Cortex AISQL: A production SQL engine for unstructured data. arXiv 2511.07663, 2025.

[36] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, and Gerardo Vitagliano. Palimpzest: Optimizing AI-powered analytics with declarative query processing. In *Proc. 15th Biennial Conf. on Innovative Data Systems Research*, 2025.

[37] Shu Liu, Asim Biswal, Amog Kamsetty, Audrey Cheng, Luis Gaspar Schroeder, Liana Patel, Shiyi Cao, Xiangxi Mo, Ion Stoica, Joseph E. Gonzalez, and Matei Zaharia. Optimizing llm queries in relational data analytics workloads. In M. Zaharia, G. Joshi, and Y. Lin, editors, *Proc. 8th Conf. on Machine Learning and Syst.*, volume 7. MLSys, 2025.

[38] Shu Liu, Soujanya Ponnapalli, Shreya Shankar, Sepanta Zeighami, Alan Zhu, Shubham Agarwal, Ruiqi Chen, Samion Suwito, Shuo Yuan, Ion Stoica, et al. Supporting our AI overlords: Redesigning data systems to be agent-first. arXiv 2509.00997, 2025.

[39] Yao Lu, Aakanksha Chowdhery, Srikanth Kandula, and Surajit Chaudhuri. Accelerating machine learning inference with probabilistic predicates. In *Proceedings of the 2018 International Conference on Management of Data*, page 1493–1508, New York, NY, USA, 2018. Association for Computing Machinery.

[40] Yuyu Luo, Guoliang Li, Ju Fan, Chengliang Chai, and Nan Tang. Natural language to SQL: State of the art and open problems. *Proc. VLDB Endow.*, 18(12):5466–5471, August 2025.

[41] Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Raj Sinha, Jinwoo Shin, and Tomas Pfister. DS-STAR: Data science agent for solving diverse tasks across heterogeneous formats and open-ended queries. arXiv 2509.21825, 2026.

[42] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. Can foundation models wrangle your data? *Proc. VLDB Endow.*, 16(4):738–746, December 2022.

[43] Thomas Neumann and Viktor Leis. A critique of modern sql and a proposal towards a simple and expressive query language. In *Proc. 14th Biennial Conf. on Innovative Data Systems Research*, 2024.

[44] Jingjie Ning, João Coelho, Yibo Kong, Yunfan Long, Bruno Martins, João Magalhães, Jamie Callan, and Chenyan Xiong. Agentic search in the wild: Intents and trajectory dynamics from 14m+ real search requests. arXiv 2601.17617, 2026.

[45] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer, 4th edition, 2020.

[46] James Jie Pan, Jianguo Wang, and Guoliang Li. Survey of vector database management systems. *VLDB J.*, 33(5):1591–1615, 2024.

[47] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. Semantic operators and their optimization: Enabling llm-based data processing with accuracy guarantees in lotus. *Proc. VLDB Endowment*, 18(11):4171–4184, July 2025.

[48] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565. Curran Associates, Inc., 2024.

[49] Ralph Peeters, Aaron Steiner, and Christian Bizer. Entity matching using large language models. arXiv 2310.11244, 2024.

[50] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proc. 8th Intl. Conf. on Intelligent User Interfaces*, pages 149–157, New York, NY, USA, 2003. Association for Computing Machinery.

[51] Juan A. Rodriguez, Xiangru Jian, Siba Smarak Panigrahi, Tianyu Zhang, Aarash Feizi, Abhay Puri, Akshay Kalkunte Suresh, François Savard, Ahmed Masry, Shravan Nayak, Rabiul Awal, Mahsa Massoud, Amirhossein Abaskohi, Zichao Li, Suyuchen Wang, Pierre-Andre Noel, Mats Leon Richter, Saverio Vadacchino, Shubham Agarwal, Sanket Biswas, Sara Shanian, Ying Zhang, Sathwik Tejaswi Madhusudhan, Joao Monteiro, Krishnamurthy Dj Dvijotham, Torsten Scholak, Nicolas Chapados, Sepideh Kharaghani, Sean Hughes, M. Özsu, Siva Reddy, Marco Pedersoli, Yoshua Bengio, Christopher Pal, Issam H. Laradji, Spandana Gella, Perouz Taslakian, David Vazquez, and Sai Rajeswar. Bigdocs: An open dataset for training multimodal models on document and code tasks. In *Proc. 13th Intl. Conf. on Learning Representations*, 2025.

[52] Gabriele Sanmartino, Matthias Urban, Paolo Papotti, and Carsten Binnig. The stretto execution engine for LLM-augmented data systems. arXiv 2602.04430, 2026.

[53] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. In *Proc. 12th Intl. Conf. on Learning Representations*, 2024.

[54] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. Docetl: Agentic query rewriting and evaluation for complex document processing. *Proc. VLDB Endowment*, 18(9):3035–3048, May 2025.

[55] Junyi Shen, Noppanat Wadlom, and Yao Lu. Batch query processing and optimization for agentic workflows. arXiv 2509.02121, 2026.

[56] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[57] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proc. 41st Intl. Conf. on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 31210–31227. PMLR, 23–29 Jul 2023.

[58] Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. A survey on employing large language models for text-to-sql tasks. *ACM Comput. Surv.*, 58(2), September 2025.

[59] Michael Stonebraker and Andrew Pavlo. What goes around comes around... and around... *ACM SIGMOD Rec.*, 53(2):21–37, July 2024.

[60] Alon Talmor, Ori Yoran, Amnon Catav, Dan Lahav, Yizhong Wang, Akari Asai, Gabriel Ilharco, Hannaneh Hajishirzi, and Jonathan Berant. MultimodalQA: complex question answering over text, tables and images. In *Proc. 9th Intl. Conf. on Learning Representations*, 2021.

[61] Timescale. pgai. Available at: `https://github.com/timescale/pgai`, 2026. Accessed: 2026-02-25.

[62] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models - A critical investigation. In *Advances in Neural Information Processing Syst., Proc. 36th Annual Conf. on Neural Information Processing Syst.*, 2023.

[63] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. In *Proc. 58th Annual Meeting Assoc. for Computational Linguistics*, pages 7567–7578, 2020.

[64] Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. What are tools anyway? A survey from the language model perspective. In *Proc. 1st Conf. on Language Modeling*, 2024.

[65] Ziting Wang, Shize Zhang, Haitao Yuan, Jinwei Zhu, Shifu Li, Wei Dong, and Gao Cong. FDABench: A benchmark for data agents on analytical queries over heterogeneous data. arXiv 2509.02473, 2025.

[66] G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.

[67] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[68] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *Proc. 1st Conf. on Language Modeling*, 2024.

[69] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proc. 11th Intl. Conf. on Learning Representations*, 2023.

[70] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: generating structured queries from natural language using reinforcement learning. arXiv 1709.00103, 2017.

[71] Xuanhe Zhou, Xinyang Zhao, and Guoliang Li. LLM-enhanced data management. arXiv 2402.02643, 2024.