Bulletin of the Technical Community on

Data Engineering

Sept 2025 Vol. 49 No. 3 IEEE Computer Society

Letters

Special Issue on Bridging Modalities: Innovations in Multi-Modal Data Management and Querying in Data Lakes

Conference and Journal Notices

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Community on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modeling, theory and application of database systems and technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of TC on Data Engineering, IEEE Computer Society, or authors' organizations.

The Data Engineering Bulletin web site is at
http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

Across industries – healthcare, law, finance, government, and beyond – vast troves of critical information remain locked inside unstructured documents. Hospitals, insurers, and public-health agencies, for example, are awash in paperwork yet struggle to answer even the simplest questions: How many hip replacements did we reimburse last year? Which clinics drove the sharp rise in imaging cost? These queries sound like ordinary SQL, but the source material is a welter of discharge notes, scanned claim forms, physician dictations, and radiology narratives that know no tables, keys, or datatypes.

For a time, Retrieval-Augmented Generation or RAG seemed like a salvation: extract a handful of relevant pages, feed them to a large language model, and let it produce answers. This method excels at generating summaries, but it falters when accuracy, completeness, and traceability are critical. In analytical contexts, we need to aggregate information with precision. Aggregation cannot tolerate overlooked notes, duplicated charges, or fabricated procedure codes. To move beyond these limitations, we must look deeper than surface-level retrieval and summarization. Within every clinical document lies an implicit structure: a latent schema that organizes the apparent chaos of narrative text. Diagnoses, drugs, quantities, prices, insurance plan codes, and many other details are embedded in free-form language, each following recurring patterns. The challenge is to automatically surface this hidden schema, promote each relevant fragment to a well-typed attribute, and keep the catalogue up to date as templates evolve, organizations merge, or reimbursement rules shift.

Once these attributes are identified, they must be connected to broader systems. For example, provider identifiers need to be cross-referenced with national registries to determine specialty and location. Drug names must be matched against external drug databases to obtain standardized codes and strengths. Procedure terms require mapping to official vocabularies to ensure that records from different sources can be meaningfully joined. Only by linking the discovered schema to external reference databases can we create a coherent, queryable view that spans institutions, payers, and time. Even with today's best language models, the journey from narrative text to structured, meaningful data is fraught with challenges. It is not simply about extracting numbers; it is about uncovering the underlying structure and semantics within unstructured language, transforming free-form narratives into well-typed fields and relationships. Only then can we aggregate, analyze, and ultimately derive actionable insights from the data. The sheer volume of information is a major hurdle, as billions of tokens must be processed to ensure that answers are both complete and reliable. The inferred schema is constantly shifting as forms evolve, new abbreviations appear, and vocabularies are updated. Record linkage remains fragile, with even minor variations in provider names disrupting connections. Economic constraints add further complexity, making it impractical to process every document with the largest models on a nightly basis. To address these issues, we need intelligent filtering, cascaded models, and planning systems that reserve costly inference for the most ambiguous or critical fragments, ensuring that the path from narrative to structure, through aggregation and analysis, leads to trustworthy insights.

The papers in this issue converge on those very gaps. One team describes a full software stack in which conversational agents design pipelines, cost-based optimisers decide which model cascade to run, and a serving layer keeps GPU memory from thrashing. Another argues that data discovery is as much social as technical—human analysts must express what they seek before any system can find it, and that expression itself hints at the latent schema waiting in the text. We meet databases that embed small language models in their storage engines, skimming over irrelevant paragraphs to slash I/O; a model-context protocol that lets billing notes, clinical narratives, and national registries live behind their own specialised servers, coordinated at query time; join-aware indices that pre-compute cross-database linkages so the model never attempts a billion-row match at runtime; and log-augmented inference that remembers last week's successful entity mappings instead of rediscovering them tomorrow. Individually these contributions advance extraction, schema induction, external linkage, storage, optimisation, and

serving; together they sketch a railway from raw narrative to verifiable totals.

Yet challenges remain. We still lack benchmarks that score faithful counting as rigorously as they score fluent summaries, and open-source ontologies that expose every revision of every code set. Privacy-preserving hosts are needed so models can train on protected health information without exfiltrating it. Above all, schema discovery and external linkage must be treated as first-class steps in the analytics stack. If we mis-type a field or mis-join a record, no brilliance in downstream analytics will salvage the insight. Should we succeed, the next time a policymaker asks, "What did we spend on lung-cancer imaging last quarter?" the answer will arrive in seconds, backed by measurable recall, transparent lineage, and a reproducible chain from prose to number. I invite you to read the pages that follow, challenge the ideas, and help turn narrative chaos into structured, dependable insight.

Haixun Wang
EvenUp

Letter from the Special Issue Editor

The ever-growing abundance of data in diverse formats–tables, text, images, audio, video–has led to the rise of **multimodal data lakes**, which promise unified storage and querying capabilities across heterogeneous modalities. With the advent of large language models (LLMs), new opportunities and challenges emerge in managing, understanding, and extracting value from such multimodal ecosystems.

This special issue brings together a collection of papers that address these challenges from multiple angles: foundational architectures, LLM-powered querying, optimized storage, agent-based orchestration, and socio-technical integration.

We begin with the article by **Castro Fernandez** from University of Chicago, which frames ***data discovery*** as a socio-technical process. He presents a reference architecture that separates data identification and retrieval and argues for a broader agenda called *data ecology* to bridge organizational knowledge and automated systems. Next, **Binnig et al.** from the Technical University of Darmstadt introduce ***LLM-augmented databases***, extending the relational model with natural language interfaces, multimodal query plans, and RAG-style execution pipelines. Their systems, ***CAESURA*** and ***ELEET***, showcase how traditional DBMS optimizations can coexist with AI-native capabilities. Complementing system-level design, **Sirin et al.** from Harvard University propose the ***Image Calculator***, a novel storage system that automatically tailors image storage formats to inference and training workloads. By breaking images into frequency components, they achieve substantial improvements in performance and space efficiency for image AI tasks. The MIT Data Systems and AI Lab's contribution articulates a vision for ***AI-enabled data-to-insights systems***, covering interactive pipeline orchestration, model routing, metadata-aware retrieval, and explainability. Their ***Sunroom*** prototype and ***KRAMABENCH*** benchmark push the boundaries of collaborative human-AI systems for data analysis. The system ***Taiji***, presented by **Zhang et al**. from Renmin University, adopts a Model Context Protocol (MCP) to orchestrate multimodal queries across specialized servers. This architecture supports query decomposition, feedback-driven refinement, and machine unlearning to keep both data and LLMs fresh and trustworthy. Finally, **Sun et al.** from Tsinghua University propose the concept of the ***Data Agent***, a holistic multi-agent architecture for orchestrating Data+AI pipelines. Their system, ***iDataScience***, dynamically selects agents using data skill embeddings and benchmark profiling, enabling robust and extensible analytics over complex tasks.

Together, these papers represent the state of the art in building scalable, intelligent, and trustworthy systems for multimodal data lakes. We hope this special issue will inspire further research in creating AI-native infrastructures for data-centric discovery.

Nan Tang, Yuyu Luo
The Hong Kong University of Science and Technology (Guangzhou)

Data Discovery is a Socio-Technical Problem: the Path from Document Identification and Retrieval to Data Ecology

Raul Castro Fernandez
Department of Computer Science, The University of Chicago
raulcf@uchicago.edu

Abstract

Data discovery is the task of identifying and retrieving documents that satisfy an information need—a need that may support decision-making, scientific inquiry, or operational insight. While the problem has existed since data began to be recorded, its complexity has grown with the scale and diversity of data, particularly tabular data in databases, spreadsheets, and cloud stores. Unlike well-defined tasks in data management, data discovery is ambiguous and socio-technical in nature, requiring both algorithmic and human-centered solutions. In this perspective, we conceptualize data discovery as distinct from but related to data search, introduce a reference architecture that separates identification from retrieval, and situate recent developments—including LLMs and retrieval-augmented generation—within this framework. We argue that solving data discovery demands embracing its socio-technical character and outline an emerging research direction, data ecology, that addresses this broader challenge.

1 Introduction

Data discovery is the process of identifying and retrieving documents that satisfy a specific information need. An *information need* refers to any data that supports a task—for example, a paragraph that answers a question, a row in a spreadsheet that informs a decision, or a feature that improves a predictor's performance beyond a given threshold. The task of *identifying* involves articulating the information need in a way that makes it possible to find relevant data. The task of *retrieval* involves gathering documents that fulfill this need. Defined broadly, the data discovery problem has existed since humans first began representing knowledge in the form of documents. Library science—concerned with how to organize and retrieve documents based on information needs—emerged after the printing press made books widely available. Later, database and information systems developed rapidly as organizations accumulated large document collections in the postwar era. Information retrieval and web search arose in response to the explosion of online (web) documents. Today, prompt engineering serves as a heuristic for retrieving relevant outputs from large language models (LLMs), given an information need expressed as a prompt. In this way, data discovery has evolved in step with both the scale of the problem and the technologies available to address it.

Data discovery is a critical problem in the context of tabular data. A significant portion of high-value data[1] is stored in tabular formats across databases, spreadsheets, and cloud storage systems. Data discovery for tabular data has been explored by the data management community and, more recently,

[1]We use high-value data informally to refer to documents that directly relate to an organization's operations, including those that contribute to value creation such as revenue, user engagement, and similar metrics.

by the machine learning community. It is also closely connected to fields like information retrieval and human-computer interaction (HCI), reflecting the diverse challenges involved in tackling this complex problem. Data discovery over tabular data has numerous applications. It supports scientific advancement by identifying documents that either validate a hypothesis or help formulate new ones (hypothesis-driven discovery). It also assists organizations in data-intensive tasks such as decision-making, product and service development, and more.

Some problems in data management resist precise definitions and are inherently ambiguous. This class includes tasks such as data cleaning, data wrangling, and data integration; data discovery belongs to this group too. When SQL is viewed as a language for retrieving documents that satisfy an information need—as specified by a query—the results are typically unambiguous, thanks to the closed-world assumption that enables clear and well-scoped data processing. However, once we acknowledge that relevant documents may span multiple databases and repositories—as is often the case in practice—the problem becomes more difficult to define and, consequently, more challenging to solve.

In this perspective article, we conceptualize the problem of data discovery, situating it in relation to data search and related tasks, and explain how it contributes to unlocking value from data (Section 2). We then present a landscape of solutions for data discovery over tabular data (Section 3), structured around a reference architecture that highlights the distinction between identification and retrieval. This architecture also serves as a framework to introduce key contributions from our group. Building on this foundation, we examine the emerging role of large language models (LLMs) and the evolving architectures for retrieval-augmented generation (RAG) and agentic systems in shaping the future of data discovery (Section 4). This discussion sets the stage for one of the paper's central claims: data discovery is fundamentally a socio-technical challenge. As such, no purely technical solution—not even those based on LLMs—can fully address it. Instead, we advocate for socio-technical approaches and introduce an emerging line of research, data ecology, that explores this broader framing (Section 5). We conclude with a discussion of the current state of the problem and outline what is needed to advance future solutions.

2 Conceptualizing Data Discovery

In this section, we introduce a set of definitions to support the discussions that follow (Section 2.1) and examine the role of data discovery in enabling value extraction from data (Section 2.2).

2.1 Basic Definitions

Data and Documents. We draw a clear distinction between data and documents [6]. Data refers to an abstract representation of some aspect or condition of reality, while a document is a representation of data. The same data may be represented in different documents. For instance, consider the concept of "room temperature": the data corresponds to the measured value itself, whereas a document might be a handwritten note recording the temperature, a value in a CSV file, or a row in a database table.

Documents exist in many forms. This heterogeneity is a core challenge of data integration and by extension data discovery.

Definition 2.1 (Information Need) *The data necessary to address a task.*

We refer to it as an information need rather than a data need because the ideal data is not always directly accessible; instead, we often rely on data that indirectly reveals information about the target. For example, if the goal is to predict whether a loan application should be approved, the ideal data would be whether the customer will default. However, that data may not be directly available. Instead,

we might have access to related data—such as credit history or financial status—that provides indirect evidence about the likelihood of default. In such cases, an analyst may articulate their information need as the set of attributes (data) that relate to a customer's probability of default.

A key aspect of information needs is that they are articulated by an agent—human or bot—as part of pursuing a task. For example, the scenario illustrated in Figure 1 depicts multiple agents within an organization, each facing a data discovery problem. The information need expressed by the decision maker (bubble 1) may be high-level—for instance, "we need to project sales for this new product for next quarter." This initial request triggers several subtasks, such as reporting on past sales, assembling features for a predictive model, and ultimately compiling these derived data products into a report to support decision-making. The key point is that, when expressed by humans, information needs are typically formulated so that the next person in the workflow can interpret and act on them. Regardless of how an information need is expressed, the central challenge of data discovery is to identify which documents satisfy it.

Definition 2.2 (Data Discovery) *The problem of identifying and retrieving documents that satisfy an information need.*

In data discovery, the solution is a document. In some cases, this document already exists, and the task is simply to locate and retrieve it. This is typical of web search, information retrieval, or dataset search, where the goal is to identify existing webpages, documents, or tables. In other cases, however, the document does not exist in a ready-made form and must be constructed—through integration or fusion—as part of the discovery process. For example, multiple documents may need to be combined and transformed to produce a new document that satisfies the user's information need. Similarly, large language models (LLMs) combined with search can synthesize a document from information scattered across multiple sources, as seen in systems like Google's Deep Dive Search, Perplexity.ai, and ChatGPT with retrieval.

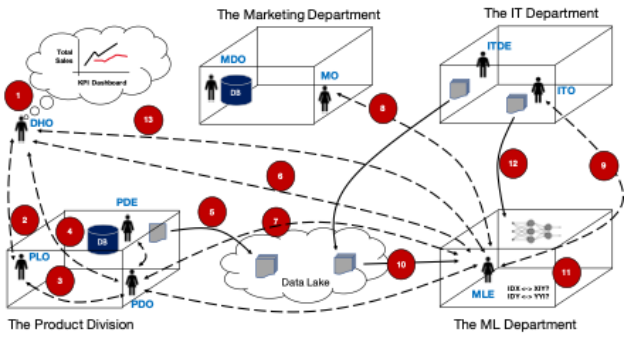


Figure 1: Simplified diagram of an organization where multiple stakeholders with information needs (indicated by the red numbered circles) interact with data systems and among themselves to solve a data task

Identify and Retrieve. Two key terms in the definition of data discovery are *identify* and *retrieve*. *Retrieving* refers to the mechanistic process of delivering a document to the user so they can proceed with their task. *Identifying*, by contrast, involves determining which document satisfies an information need—that is, which document represents data that matches the articulated need. *Identifying* and *retrieving* are related to the separation between cognitive-viewpoint and document-based view of the "anomalous states of knowledge" theory of information retrieval [3]. Successfully solving this process entails several challenges:

1. it requires a precise and unambiguous articulation of the information need, which in turn depends on the user's understanding of the task they are trying to accomplish;
2. the information need must be expressed in a form that the discovery system can use to match against available data—for example, by explaining a request to a librarian, entering keywords into a search engine, or writing a natural language query to an LLM;
3. the discovery system must be capable of interpreting the information need and searching across large volumes of data to find relevant matches.

We distinguish between *content* and *context* information needs [2]. Content information needs can be addressed by examining the contents of documents themselves. However, identifying a relevant document is not always possible through content alone. Context information needs require metadata about the document—such as who created it, how it has been processed, and other provenance—related attributes.

A critical piece of metadata in data discovery is a document's *provenance*. Provenance is essential for verifying the origin of a document and the chain of agents who created or modified it. In a world flooded with data—much of it of questionable quality—provenance becomes increasingly valuable, often serving as the primary means of distinguishing reliable information from noise. We emphasize provenance because, in manual discovery processes, it is typically maintained informally by the people involved, who can explain how a document was produced if asked. As discovery processes become more automated, however, preserving provenance requires deliberate effort to capture the **tribal knowledge** that resides in the minds of those humans. A central goal of data discovery is to encode this implicit knowledge into documents so they can be used more effectively to satisfy information needs—a topic we expand on later in the paper.

2.2 The Role of Data Discovery in the Data Value Chain

To understand the value of data discovery solutions, it is useful to situate them within the broader process of deriving value from data. Data has instrumental value—that is, it generates value when used to perform a specific task. However, data is not accessed directly; it is accessed through documents. The utility of a document for a given task depends both on the value of the data it represents and the cost of using that document [6]. We conceptualize the process of deriving value from data as a sequence of the following steps:

1. Formulate the information need associated with the task.
2. Discover a document that satisfies the information need. This document may already exist within the organization or may need to be assembled from multiple existing documents.
3. Analyze and process the document to produce a solution to the data task.

Data discovery plays a central role in the second step of the process, and—as we will argue later—it must also support the first step by helping agents articulate their information needs. The cost of data discovery reflects the time, effort, and resources required to identify documents that satisfy an information need. Because this cost is nonzero, it reduces the overall utility of the data. Therefore, the purpose of data discovery solutions is to minimize these costs, thereby increasing the utility that can be derived from data.

Most data management technologies function as instruments for reducing costs, and data discovery fits this role too. But data discovery is more than a cost reduction technique.

Beyond Cost Reduction. In the conceptual model outlined above, the first step is to formulate an information need arising from a data task. However, selecting which data task to pursue—among many possibilities—is itself a critical decision that shapes the value derived from data. Data discovery can play a generative role in this process by helping to surface new questions (as in hypothesis-driven discovery) that lead to the formulation of new tasks. At its best, data discovery does more than reduce costs: it can help define entirely new use cases, opening up fresh avenues for value creation.

3 Landscape of Data Discovery Solutions for Tabular Data

We present an overview of solutions for tabular data discovery. Our goal is not to provide a comprehensive survey of the literature[2], as such a review would exceed the scope of this paper. Instead, we focus on highlighting key ideas proposed in this area and clarifying how they relate to one another. This technical foundation will serve as the scaffolding for our central argument in the next section: that data discovery is a socio-technical problem.

The literature on data discovery can be categorized based on the emphasis and objectives of each contribution. Some work focuses on identification interfaces, while others emphasize retrieval mechanisms. A substantial portion of the research centers on foundational building blocks—techniques designed to support navigation through large repositories of tables. In Section 3.1, we introduce a reference architecture that we have recently used to illustrate how the challenges of tabular data discovery are interconnected. We then provide an overview of our group's work in this area and relate it to other relevant efforts to contextualize our main contributions (Section 3.2).

3.1 Applying Separation of Concerns to Data Discovery

A reference architecture embodies a *separation of concerns* strategy, which breaks down complex engineering problems into more narrowly defined, and thus more manageable, subproblems. Such architectures describe a set of components and their interactions, reflecting our current understanding of the problem space. In prior technical work, we have presented detailed reference architectures [16]. Here, we focus on highlighting the separation of concerns among three key components to help structure this brief overview.

First, an identification module that processes users' questions (Section 3.1.1). Second, a retrieval engine that accesses relevant documents (Section 3.1.2). Third, a discovery engine that orchestrates a strategy involving:

1. applying identification techniques to derive search criteria;
2. combining these criteria into a plan that leverages available indices; and
3. answering the question—often with user involvement.

We conclude the section with a set of examples that illustrate how these three components work in practice (Section 3.1.3).

3.1.1 Identification Interfaces

Data discovery solutions must offer a means for agents to express their information needs. For tabular data, identification interfaces include:

Keyword Search. Soon after keyword search became the standard interface for expressing information needs on the web, it was adapted to relational database management systems through early prototypes that had significant academic impact in the years that followed.

Query-by-Example (QBE). QBE interfaces were originally proposed as an alternative to SQL for querying relational databases [28]. The same idea—where agents provide examples of the desired output—has since been adapted for use in data discovery systems.

Task-Oriented. Instead of explicitly articulating the information need induced by a task, agents provide a direct specification of the task itself. When an evaluation function for the task is available, it can be supplied to the system, which then assumes responsibility for discovering data that satisfies the

[2]See, for example, [23] for surveys that summarize techniques used in discovery systems

task requirements. This approach was formalized for tabular data in Metam [13], with an earlier version for supervised machine learning tasks introduced in ARDA [4] and later applied in Kibana [20].

Personalization. When agent demand is high, systems can capture signals about what agents request and how satisfied they are with the results. These signals can then be used to build personalized models that better align with individual information needs. In web search, Google pioneered this approach, which has since become standard practice. In the context of tabular data discovery, certain data catalogs that capture *behavior*—the "B" in the ABCs of metadata [18]—lay the groundwork for more sophisticated personalization strategies.

3.1.2 Retrieval Strategies: Indices

It is useful to conceptualize the solution to the identification problem as producing a concrete search criterion. A retrieval strategy then takes this search criterion—such as a keyword, question, range, or spatial polygon—and uses an index to return matching documents. We define an index broadly as any data structure that facilitates the efficient location of documents that match a given search criterion.

Table Representation. Indices used in data discovery represent tables in ways that facilitate matching them against search criteria—and in doing so, they induce specific representations of the tables themselves. Several types of indices are commonly used. Full-text search indices represent each column of a table as a sparse vector within a vector space model. Vector indices use embedding models to encode entire tables or their components as dense vectors [1, 9, 24]. The design space for such vector representations is large and rapidly evolving, particularly with the emergence of retrieval-augmented generation (RAG) systems for tabular data, as we discuss later. Another approach is to represent data using graphs, which is common in knowledge bases that encode relationships between terms in the data.[3]

Data catalogs, or "databases of metadata," are widely used in industry. When discovery questions can be answered using metadata alone, these systems are effective in identifying relevant tables. Many data catalogs also incorporate behavioral signals derived from user interactions—corresponding to the "B" in the ABCs of metadata [18]—which further enhance their ability to support discovery.

The indices described above are compatible and often complementary. Many data discovery systems combine multiple indexing strategies to expand their ability to answer diverse queries. For example, modern data catalogs increasingly incorporate vector-based and graph-based techniques alongside traditional metadata indexing.

Building Blocks. When discovery questions require combining multiple documents, simple retrieval of individual tables is insufficient. To support more complex queries, a range of techniques—referred to here as building blocks—can be integrated with the indexing methods described above. Notable examples include indices that facilitate table joins, unions, and augmentations (conceptually, a combination of joining and unioning). Additionally, other building blocks enrich the metadata available to discovery systems by annotating columns and tables or by generating descriptive summaries.[4]

3.1.3 Examples

We present a representative, though not exhaustive, set of systems to illustrate the identification, retrieval, and discovery engine components discussed above.

Discover [19]. Discover accepts keywords as input—serving as its identification strategy—and returns a combination of tables from a given relational schema that contain those keywords (retrieval). The

[3] This is a vast design space, and a comprehensive treatment of knowledge bases for data discovery is beyond the scope of this paper.

[4] This task is referred in the literature as semantic annotation.

discovery engine constructs a compact answer by identifying a Steiner tree of join operations connecting the relevant tables.

Infogather [26]. Infogather takes an input table and returns an augmented version of it—adding rows or columns sourced from a collection of other tables. Its primary technical contribution is a retrieval index specifically designed to support such augmentation operations. The discovery engine implements algorithms to search this index and produce the final result.

Goods [17]. Goods takes keywords as input and returns relevant tables based on those keywords (identification). Its retrieval index encodes both the content of tables and rich metadata, including references from code (e.g., ETL and transformation pipelines) and user annotations. The discovery engine uses traditional information retrieval techniques to rank the results.

Aurum [7]. Aurum accepts programs written in a domain-specific language (DSL) that allows users to describe their information needs (identification). It uses a graph-based index to determine which subset of documents satisfies the DSL query. The discovery engine orchestrates the execution and combination of results as specified by the program.

LlamaIndex [22]. When equipped with the table reader plugin, LlamaIndex creates a vector representation for each row in the input tables (retrieval) and uses a vectorized form of a natural language question (identification) to find relevant tables. The discovery engine matches these vectors against the table representations and uses a large language model (LLM) to further refine the results.

3.2 My Group's Work

The diagram in Figure 2 presents our group's technical contributions to the area of data discovery and serves as a reference throughout this section to summarize key lessons learned. Before introducing the diagram, I offer a brief summary of the vision that has guided my work in this area from the outset.

Vision. There are countless compelling problems in data discovery. Working on any of them presents opportunities to contribute solutions that others can build upon, gradually advancing toward more complete systems. However, my philosophy has been that this piecemeal approach risks getting stuck in a kind of local optima—focusing on technically interesting subproblems that, while valuable, may not significantly advance the broader goal: helping people find the data they need.

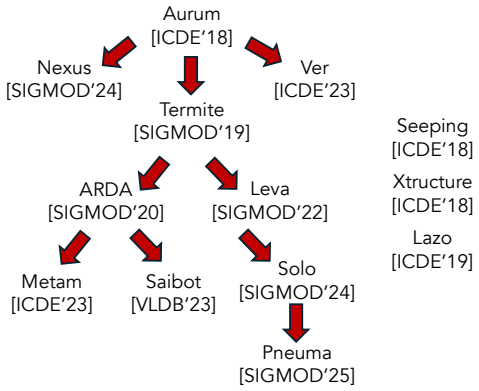


Figure 2: Portfolio of data discovery work

Over the past few years, I have prioritized an end-to-end approach to data discovery. This strategy comes with both drawbacks and advantages. One drawback is that building full prototypes takes considerable effort. Often, the resulting systems are not immediately deployable or impactful outside the lab—bridging that gap requires even more work. The longer development cycles also mean feedback loops can be slower.

The benefit, however, is substantial: pursuing end-to-end solutions reveals where the real bottlenecks lie. I use the term bottleneck informally to refer to two things i) technically interesting problems for which we lack good solutions, and ii) engineering challenges that differ in kind from those tackled by existing systems. Through this approach, I believe we have developed a strong sense of both.

3.2.1 Overview of the Group's Data Discovery Work

Our group's work is summarized in the diagram shown in Figure 2. Each node represents a technical paper, and edges indicate that one work builds upon another. While no summary can fully capture the richness and detail of these contributions, the diagram helps surface the main themes. We highlight those themes next, organized according to the reference architecture introduced earlier.

A Spectrum of Identification Techniques. When users have a clear idea of what they want, they can articulate their query using a domain-specific language (DSL) with systems like Aurum [7] or specify a utility function with Metam [13], the latter having introduced the concept of goal-driven data discovery. In situations where users cannot precisely define their information needs, alternative methods are available. For example, Aurum supports keyword search; Termite [9] offers vector search over tables and columns; and Ver [16] enables users to specify the desired table via a QBE-style interface. Finally, in systems such as Solo [25] and Pneuma [2], users express their information needs through natural language questions.

Table Representation. Supporting the identification strategies described above are a variety of table representation methods developed across our work. Seeping Semantics [10] was one of the first efforts to use distributed representations (embeddings) to capture the semantics of data in the context of data discovery. Termite extends this approach by embedding all values in a document, and Leva [27] advances the line further by exploring more effective techniques for computing embeddings at the table, column, row, and cell levels. Many of the insights from these efforts are incorporated into Solo and, more recently, Pneuma [2].

Complementing these vector-based approaches, systems such as Aurum [7], Nexus [14, 15], Xtructure [21], Metam [13], and Ver [5, 16] rely on sets of profiles to represent the underlying documents, offering a structured alternative to embedding-based representations.

Indices. Table representations are indexed using a variety of structures, depending on the system. Graph-based indices are used in Aurum, Ver, Metam, and ARDA [4]; vector databases are employed by Leva, Solo, and Pneuma; Lazo [11] uses hash-based indices; and Xtructure [21] relies on regular expression (regex) indices.

Broad Applications. Aurum, Seeping, and Ver support general-purpose discovery queries and offer the broadest interfaces among our systems, accommodating a wide range of information needs. In contrast, ARDA [4], Saibot [20], and Metam [13] focus on data augmentation. While ARDA and Saibot are tailored to supervised machine learning tasks, Metam generalizes this approach to any task for which a user can define a utility function.

Beginning with Leva, and continuing with Solo and Pneuma, we introduced RAG-like architectures that accept natural language questions and return relevant tables (i.e., table search). However, these systems do not support on-the-fly table assembly. Finally, Nexus targets hypothesis generation, where the hypothesis space is defined in terms of correlations among variables.

3.2.2 Tenets and Lessons Learned

Reflecting during and after the work described above has led to a number of insights that may be valuable to others working in this area.

Tenet 1: The North Star Is Satisfying Information Needs Directly. The overarching goal remains constant: help users find data that advances their work. This principle has guided many disciplines, though the techniques evolve. As systems improve, user expectations rise—pushing the target forward. This is a healthy dynamic, as long as we continue to learn how people want to access and use information, and we engineer systems that serve those real-world needs.

Tenet 2: Separate Identification from Retrieval. Clearly distinguishing between identification (understanding what data is needed) and retrieval (locating and delivering documents representing that data) is essential. This separation, rooted in software engineering principles like separation of concerns, helps both system design and research progress. Variants of this distinction have long existed in fields such as library science and information retrieval, albeit under different terminology. Adopting this lens helps bring conceptual clarity and makes it easier to incorporate insights from adjacent disciplines.

Tenet 3: Engineer for Evolving Signals. Data discovery does not have exact solutions; it resembles information retrieval more than deterministic SQL processing. As a result, systems must be engineered to flexibly incorporate new signals—reference datasets, human input, LLM prompts, knowledge graphs, and more. A guiding principle: avoid discarding documents early. Instead, cast a wide net, rank based on available signals, and update the ranking as new signals emerge. Always expose the basis for ranking to the user, and let them decide how to weigh or filter results.

Tenet 4: Favor End-to-End Solutions. As discussed earlier, pursuing end-to-end systems—rather than isolated point solutions—yields deeper insight into what actually limits progress. While point solutions are valuable (and abundant in the literature), end-to-end approaches reveal both technical and practical bottlenecks, and thus offer a clearer path toward usable systems. These efforts are more demanding, but ultimately more impactful.

Tenet 5: Be Cautious with Benchmarks. When benchmarks accurately reflect real-world problems, they are invaluable—they support reproducibility, enable progress tracking, and foster community alignment. But when a benchmark simplifies or misrepresents a problem, it can be misleading. In such cases, it encourages over-optimization on artificial targets, diverting attention from what actually matters in practice. Reproducibility is important, but not at the cost of chasing the wrong goals.

4 How do LLMs Change the Game?

Addressing data discovery requires solving both the identification and retrieval problems. Each involves a range of technical challenges—some of which remain especially difficult and unresolved.

- **Multiple Document Formats.** Understanding a table often requires access to contextual information—metadata that describes or qualifies the table's content—which is frequently not stored in tabular form. Much of this context resides in heterogeneous formats, including spreadsheets, PDFs, code, and other unstructured or semi-structured documents. If, as argued in Tenet 2 (Engineer for Evolving Signals), effective discovery depends on leveraging all available signals, then robust support for manipulating multiple document formats—or multimodal data—is essential.
- **Semantic Ambiguity.** Even with robust support for multiple document formats, resolving semantic ambiguity remains a core challenge. It requires determining which signals to prioritize in order to disambiguate meaning in specific contexts. Semantic ambiguity is pervasive in data discovery—from interpreting the unit of a value in a cell, to selecting an appropriate join key, to inferring a user's intent. These ambiguities often lack a single correct resolution, making adaptive, context-aware reasoning essential.
- **Scalability.** The value of data discovery increases as the volume of data grows—especially when manual exploration becomes impractical. At the same time, the imperative to incorporate new signals (as discussed in Tenet 2) continuously expands the set of documents under consideration. Processing large volumes of data—often requiring complex computations—poses significant scalability challenges, both in terms of system performance and engineering effort.

4.1 The Promise of LLMs

Large Language Models (LLMs) play a central role in many current research efforts. When conceptualized as vast repositories of documents equipped with natural language interfaces—and as systems capable of adapting their responses to specific information needs—they offer a compelling solution to the identification problem in data discovery. This makes LLMs a particularly interesting artifact to consider in the broader landscape of data discovery systems.

First, they readily provide approaches to tackle the first two problems above:

Querying Multimodal Data. Internally, LLMs represent all input, regardless of its original format, as vectors. These vector representations can be accessed and manipulated through natural language, effectively turning language into a Rosetta Stone-like interface for querying multimodal data. This makes LLMs a promising tool for addressing the challenge of working across heterogeneous document formats.

Addressing Semantic Ambiguity. LLMs encode knowledge from vast collections of documents and can be leveraged to resolve many forms of semantic ambiguity. While individual instances, such as interpreting a value's unit or understanding user intent, are often trivial for humans, the variety and unpredictability of such cases make it difficult to formalize a comprehensive set of rules. LLMs offer a compelling alternative: they can empirically address this long tail of ambiguities through contextual inference, without requiring explicit rule specification.

Second, LLMs can be leveraged both to help users articulate their information needs and to interpret those needs in ways that guide the system toward meaningful answers.

4.2 Challenges Ahead

Despite the promise, LLMs also introduce challenges: i) how to incorporate external data; ii) scalability.

Incorporating External Data. There are three broad approaches to incorporating external data, such as the document collections over which discovery queries are run, into LLM-based systems. First, external data can be incorporated during the training process, embedding the information directly into the model's parameters. Second, it can be provided as part of the input context at inference time. Third, external data can be accessed dynamically using a Retrieval-Augmented Generation (RAG) architecture, in which relevant documents are retrieved and passed to the model alongside the user's query.

Incorporating data during the training process, even through fine-tuning, is costly and raises concerns about data leakage, which may conflict with organizational priorities around privacy, compliance, and intellectual property. In contrast, in-context learning, RAG architectures, and self-play environments, where agents interact dynamically with data, offer more flexible and privacy-preserving alternatives. While these approaches have primarily been developed for unstructured data, our group and others are now extending them to tabular data with promising results, which we summarize next.

RAG and agentic architectures combine traditional systems for solving the retrieval problem with an LLM that receives both the question and partial retrieval results to generate an answer. This is where LLMs offer a distinct advantage: they help address the identification problem by interpreting the user's intent and connecting it to relevant data.

Despite promising early results, it remains too soon to determine the right RAG architecture for data discovery. However, we expect that the growing interest in this space will soon lead to practical and usable solutions—particularly if the scalability challenge, which we explain next, is addressed.

Scalability Challenge. While data discovery already faced scalability challenges before the advent of LLMs, those challenges are amplified by the use of LLMs and RAG architectures [25]. The primary source of this increased burden is the reliance on vector representations, which are both computationally expensive to generate—requiring resource-intensive model inference—and large in size, often spanning

hundreds or thousands of dimensions. This makes it difficult not only to index data efficiently, but also to keep vector representations incrementally updated as the underlying data evolves. Despite these challenges, the substantial research activity and industry investment in this area suggest that practical solutions are likely to emerge.

Overall, LLMs represent an exciting and promising direction for data discovery. Given the substantial research and industry resources dedicated to overcoming their limitations, it is reasonable to expect significant progress in the coming years. However, the precise architectures and scalability mechanisms that will ultimately enable robust, real-world data discovery systems remain uncertain, highlighting the richness and complexity of the research landscape. Even once these technical challenges are addressed, the question of how fully they will resolve the broader data discovery problem remains open. In the next section, we offer some perspectives on what may still be missing.

5 Data Discovery is a Socio-Technical Problem

We believe that LLM-augmented data discovery architectures will, in the near future, meaningfully assist humans in identifying and retrieving documents that satisfy their information needs [8]. Yet, these systems will not fully resolve the data discovery problem. The stronger claim we make in this section is that no technical solution alone—regardless of how advanced—can fully solve data discovery.

The reason is subtle. For a technical solution to effectively help a user, two conditions must be met: the necessary information needs to be stored as documents, and the discovery tool they use must be able to locate those documents.

Up to now, data discovery research has mainly dealt with the second problem: finding and retrieving documents that satisfy a user's information need, often taking for granted that the necessary data is already in document format. We anticipate that near-future systems, like RAG and agentic systems, will begin to tackle this assumption. But the main takeaway from this section is that solving the first condition—getting the data represented as documents—is a tougher nut to crack without fully recognizing that data discovery is a socio-technical issue.

5.1 Data Discovery as a Socio-Technical Problem

The ultimate success of technical solutions hinges on the extent to which necessary data is captured in accessible documents. Often, much of the vital context needed to interpret, transform, and leverage these documents resides solely with data employees. Although these employees occasionally document datasets and processes, thereby generating valuable resources for technical systems, much of this knowledge remains undocumented. This is frequently due to the high cost of documentation, a lack of incentives (as it's often not considered part of their core responsibilities), or simply because the information is too recent to have been recorded.

To achieve data discovery in such scenarios, it's crucial to fully acknowledge the social component. This involves extracting the necessary data from the minds of data employees so that it can be converted into documents and then utilized by technical solutions. However, a number of reasons make this challenging:

- While we've characterized the identification process as something a single user undertakes, it's often the case that the "user" actually represents a team of individuals who must collectively decide what information is most important.
- Although human cognition is crucial to the identification process, our understanding of how these cognitive functions operate is currently insufficient to influence them beneficially.

Simply put, we need to make an effort to pinpoint what information users (or teams) actually need, and a separate effort to get the necessary documents from data employees who often store that data in their minds. Data discovery can't be solved without successfully connecting this supply of information with the demand. This realization was central to an article we wrote a few years back, where we introduced data markets as a solution for data discovery within organizations [12]. Since then, we've been building on these concepts through our research agenda, Data Ecology.

5.2 Data Ecology

Data ecology examines how documents are created and shared within data ecosystems, which are made up of different agents working to complete data-related tasks. This field proposes that by identifying all existing dataflows (the transfers of documents between agents), we can determine several things: which positive dataflows are already present and should be maintained; which beneficial dataflows are missing and should be encouraged; which negative dataflows exist and should be stopped; and which harmful dataflows don't yet exist and should be prevented from ever forming.

To achieve this control over dataflows, data ecology introduces tools for developing and evaluating *interventions*. These interventions are changes to the ecosystem designed to produce the desired dataflow effects. They can be technical (like the data escrow system we developed to manage data release and prevent leakage), but also economic (suchives as incentives), legal, and social (like establishing norms).

Understanding data discovery as a socio-technical issue and an organization as a data ecosystem to be analyzed via the dataflows that occur changes the game. A designer's objective then is to implement interventions that establish productive dataflows. This means encouraging data employees to document their *tribal knowledge* and helping data users articulate their information needs. Only when these pieces are in place can future data discovery systems effectively bridge the gap between data supply and demand.

The core idea is to foster an internal data ecosystem where data experts contribute their knowledge as documents to assist those who need to consume that data. In our previous paper, we explored mechanisms to encourage these dataflows within organizations, including gamification strategies like bonuses and monetization. We believe such internal data markets hold significant promise for addressing information needs at various levels, as previously illustrated in Figure 1. However, the design possibilities for these ecosystems are vast, and this remains an open problem. Our research efforts in recent years have primarily focused on developing the foundational concepts of Data Ecology.

6 Conclusions

Solving data discovery will significantly enhance data utility by lowering the costs associated with identifying and retrieving relevant documents, thereby accelerating progress in both science and industry. Historically, data discovery has been hampered by persistent challenges like semantic ambiguity and scalability, which are inherently difficult to overcome. However, with the rapid commoditization of large language models (LLMs) and our growing understanding of how to effectively integrate them with architectures like RAG, we now have access to a technology that could dramatically speed up our progress in data discovery. We are optimistic that the technical hurdles of data discovery will be largely resolved in the near future.

At the same time, we recognize that data discovery is fundamentally a socio-technical problem. Even if we had a perfect technical solution, we'd still face hurdles: helping users clearly define their information needs and convincing data providers to document the valuable knowledge they hold in their heads. To bridge this gap between data supply and demand, our Data Ecology research agenda has been exploring various data ecosystems, including internal data markets. While the technical challenges are

still significant, we believe that highlighting the socio-technical dimension of this problem can greatly enhance the effectiveness of future solutions.

References

[1] Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Transformers for tabular data representation: A survey of models and applications. *Transactions of the Association for Computational Linguistics* 11 (2023), 227–249.

[2] Muhammad Imam Luthfi Balaka, David Alexander, Qiming Wang, Yue Gong, Adila Krisnadhi, and Raul Castro Fernandez. 2025. Pneuma: Leveraging LLMs for Tabular Data Representation and Retrieval in an End-to-End System. *Proceedings of the ACM on Managment of Data* (2025).

[3] Nicholas J Belkin. 1980. Anomalous states of knowledge as a basis for information retrieval. *Canadian journal of information science* 5, 1 (1980), 133–143.

[4] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. 2020. ARDA: automatic relational data augmentation for machine learning. *arXiv preprint arXiv:2003.09758* (2020).

[5] Kevin Dharmawan, Chirag A Kawediya, Yue Gong, Zaki Indra Yudhistira, Zhiru Zhu, Sainyam Galhotra, Adila Alfa Krisnadhi, and Raul Castro Fernandez. 2024. Demonstration of Ver: View Discovery in the Wild. In *Companion of the 2024 International Conference on Management of Data*. 428–431.

[6] Raul Castro Fernandez. 2025. What is the Value of Data?: A Theory and Systematization. *ACM/IMS Journal of Data Science* (2025).

[7] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1001–1012.

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

[9] Raul Castro Fernandez and Samuel Madden. 2019. Termite: a system for tunneling through heterogeneous data. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 1–8.

[10] Raul Castro Fernandez, Essam Mansour, Abdulhakim A Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping semantics: Linking datasets using word embeddings for data discovery. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 989–1000.

[11] Raul Castro Fernandez, Jisoo Min, Demitri Nava, and Samuel Madden. 2019. Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1190–1201.

[12] Raul Castro Fernandez, Pranav Subramaniam, and Michael J Franklin. 2020. Data market platforms: trading data assets to solve data problems. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1933–1947.

[13] Sainyam Galhotra, Yue Gong, and Raul Castro Fernandez. 2023. Metam: Goal-oriented data discovery. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2780–2793.

[14] Yue Gong and Raul Castro Fernandez. 2024. Demonstrating Nexus for Correlation Discovery over Collections of Spatio-Temporal Tabular Data. In *Companion of the 2024 International Conference on Management of Data*. 524–527.

[15] Yue Gong, Sainyam Galhotra, and Raul Castro Fernandez. 2024. Nexus: Correlation Discovery over Collections of Spatio-Temporal Tabular Data. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–28.

[16] Yue Gong, Zhiru Zhu, Sainyam Galhotra, and Raul Castro Fernandez. 2023. Ver: View discovery in the wild. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 503–516.

[17] Alon Halevy, Flip Korn, Natalya F Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*. 795–806.

[18] Joseph M Hellerstein, Vikram Sreekanti, Joseph E Gonzalez, James Dalton, Akon Dey, Sreyashi Nag, Krishna Ramachandran, Sudhanshu Arora, Arka Bhattacharyya, Shirshanka Das, et al. 2017. Ground: A Data Context Service.. In *CIDR*. Citeseer.

[19] Vagelis Hristidis and Yannis Papakonstantinou. 2002. Discover: Keyword search in relational databases. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 670–681.

[20] Zezhou Huang, Jiaxiang Liu, Daniel Gbenga Alabi, Raul Castro Fernandez, and Eugene Wu. 2023. Saibot: A Differentially Private Data Search Platform. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3057–3070.

[21] Andrew Ilyas, Joana MF da Trindade, Raul Castro Fernandez, and Samuel Madden. 2018. Extracting syntactical patterns from databases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 41–52.

[22] llamaindex. [n. d.]. Llamaindex. `https://www.llamaindex.ai`

[23] Norman W Paton, Jiaoyan Chen, and Zhenyu Wu. 2023. Dataset discovery and exploration: A survey. *Comput. Surveys* 56, 4 (2023), 1–37.

[24] Liane Vogel, Benjamin Hilprecht, and Carsten Binnig. 2023. Towards foundation models for relational databases [vision paper]. *arXiv preprint arXiv:2305.15321* (2023).

[25] Qiming Wang and Raul Castro Fernandez. 2023. Solo: Data Discovery Using Natural Language Questions Via A Self-Supervised Approach. *Proceedings of the ACM on Management of Data* 1, 4 (2023), 1–27.

[26] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 97–108.

[27] Zixuan Zhao and Raul Castro Fernandez. 2022. Leva: Boosting machine learning performance with relational embedding data augmentation. In *Proceedings of the 2022 International Conference on Management of Data*. 1504–1517.

[28] Moshe M. Zloof. 1977. Query-by-example: A data base language. *IBM systems Journal* 16, 4 (1977), 324–343.

Towards LLM-Augmented Database Systems OR The Relational Model is dead! Long live the Relational Model!

Carsten Binnig
Technical University of Darmstadt & DFKI

Abstract

Relational databases have powered critical systems for decades. Yet, despite their promises of a simple data model and an easy-to-use query language, relational systems impose high overheads of using them, which are inherently rooted in the relational model. In this paper, we coin these overheads *Relational Taxes* as users not only face high *query taxes* due to SQL's complexity and schema dependence, but also *data taxes* from the need to manually extract data into tables before being able to query them. In this paper, we present the efforts that we have spent over the last years to build what we call *LLM-augmented databases*—a new class of systems that integrates large language models with relational principles to cut the relational taxes to enable intuitive and efficient querying over diverse data modalities. Rather than replacing the relational model, we extend it to cut the relational taxes as long-standing overheads while preserving its foundational strengths in terms of performance.

1 Introduction

Relational Databases: A Success Story. Relational databases are the foundation of nearly every modern application. As Bruce Lindsay once famously stated, *"Relational databases are the foundation of Western civilization"*. This sentiment reflects the reality that critical infrastructure across banking, government, healthcare, supply chains, e-commerce, and scientific research depends on relational database systems. Moreover, the relational model has empowered the development of highly sophisticated techniques for data management, including powerful query optimization strategies, efficient query execution engines, and data access methods. These advances have enabled relational systems to scale to massive workloads, and they even continue to thrive in the era of cloud computing. As a community, we have made remarkable progress in advancing relational systems to handle increasing volumes of data and complex analytical workloads with high performance and reliability.

The Promise of the Relational Model. The invention of the relational model, along with the emergence of SQL as a standard query language, offered significant advantages over previous paradigms such as the hierarchical and network models. Two core promises of the relational model stood out, which promised to simplify data management in contrast to prior approaches:

1. *An Easy-to-Use Data Model.* The relational model organizes data into simple, tabular structures known as relations (or tables). An important aspect of the relational model was that it decoupled logical representation from physical storage and thus enabled users to interact with data without needing to understand the underlying implementation. In contrast to earlier models that required explicit navigation through linked records, the relational model thus offers a declarative way of expressing information needs.
2. *Simple and Intuitive Querying.* Along with the relational model came SQL. Originally, SQL was meant as a Structured English Query Language to enable querying of data by non-technical users.

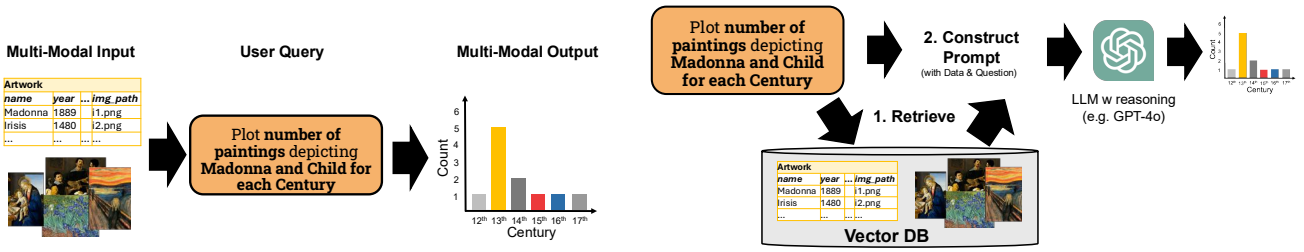
Instead of specifying how to retrieve data, users state what they want in a language that is close to natural language but with clear semantics. This was a profound shift away from imperative query interfaces required by hierarchical or network models.

Did the Relational Model fulfill its Promise? Despite its original promises, however, in this paper we argue that the relational model has not delivered on its original promises. Over the past 50 years, the accumulated experience of using relational databases has revealed significant overheads inherent in the relational model itself. These inefficiencies are deeply rooted in the design principles of the relational paradigm. We refer to these inefficiencies as *Relational Taxes*. In this paper, we identify two primary forms of relational taxes:

1. *Query Taxes of the Relational Model*. Despite its declarative nature, SQL has become a complex and verbose language. Writing non-trivial SQL queries often requires tens or even hundreds of lines of code, making queries hard to write, read, and maintain. Furthermore, users must have in-depth knowledge of the database schema, including table names, attribute names, and how literals are represented in the database, which are often driven by technical aspects and not meant to be understandable by users. For example, to query medical data, users often need to understand technical codes like ICD-10 rather than using intuitive terms like "fever". This disconnect significantly raises the barrier to entry and limits usability.
2. *Data Taxes of the Relational Model*. Real-world relational databases often feature complex database schemas with hundreds of interrelated tables. As such, identifying the correct tables and relationships can be a daunting task for users. Moreover, not all data originates in tabular form. Instead images, text documents, sensor logs, and other modalities are increasingly common as native data formats. Yet, relational systems require such non-tabular data to be manually transformed and integrated into tables before it can be queried, resulting in substantial manual overhead in data preparation and ingestion.

Cutting the Relational Taxes. As such, a natural question that arises is how to cut these taxes. In fact, one might ask if relational databases are still required in the age of AI and, in particular, LLMs. Recent advancements in LLMs, such as GPT-4, demonstrate promising capabilities in understanding and answering natural language queries over diverse modalities, including text and images, without the need to first extract structured information. In fact, the recent generation of LLMs can even support complex reasoning tasks, which allows these models to answer multi-hop queries that resemble relational joins across multi-modal data sources. However, we argue that a purely LLM-driven approach for query answering—where the LLM acts as a black-box engine for query answering—falls short due to several inherent drawbacks: hallucinations stemming from the generative nature of LLMs, a lack of transparency in reasoning processes, and most importantly inefficiencies in processing even medium-sized datasets.

Our Vision: LLM-Augmented Databases. To address these limitations, we present our vision of LLM-augmented databases—a new class of database systems that integrates large language models (LLMs) tightly with the relational database stack to cut the relational taxes. First, instead of using SQL as the query interface, LLM-augmented databases provide more intuitive query interfaces, including natural language querying and SQL-inspired abstractions that alleviate the need for users to know and understand complex schemas or precise data encodings. At the core, these query interfaces empower users to query data using high-level semantics concepts, while the underlying system handles the translation of the user query, as well as data discovery and binding of data to the query. Moreover, LLM-augmented databases remove the burden of manual data extraction by allowing direct querying of other data sources, such as images and text, that can be used as complex data types.



(a) Example Scenario (b) LLM-based Question Answering

Figure 1: Multi-modal Question Answering: (a) Example scenario on an artwork dataset composed of a table (artwork metadata), images of the artwork, and a text collection (artwork descriptions). (b) Using LLMs for multi-modal question answering using Reasoning and Retrieval-Augmented Generation.

Contributions and Outline. In this paper, we present our comprehensive vision for LLM-augmented databases, grounded in our group's recent work and research contributions. In Section 2, we outline the architectural principles of how we envision LLM-augmented databases, showing how they can be built on the foundational strengths of relational systems while avoiding the relational taxes. In this section, we also introduce key abstractions and design patterns that enable seamless multi-modal querying, many of which have already influenced emerging systems in this space. Sections 3–5 then present three concrete case studies demonstrating LLM-augmented databases in action, which span different layers of the database stack from query compilation and optimization, over query execution, to data storage in LLM-augmented databases. Finally, in Section 6, we discuss the future challenges and opportunities of LLM-augmented databases before we conclude in Section 6.

2 Our Vision: LLM-augmented Database Systems

As discussed before, an LLM-augmented database system enhances classical relational database systems with LLMs to enable the combination of the following two new functionalities: (1) multi-modal input data to cut data taxes and (2) new query interfaces (e.g., natural language or SQL-inspired abstractions) that alleviate the need for users to know and understand complex schemas or precise data encodings. In the sequel, we more precisely define which data and queries are in the scope of an LLM-augmented database system as we envision in this paper. To make more clear what type of scenarios we envision to support in our first version of LLM-augmented databases, we present a scenario that can be supported by such system. Figure fig:example shows a user querying a multi-modal dataset of a museum that stores both metadata (available as a table) and pictures of artworks (stored as images) exhibited in the museum.

2.1 Why LLMs are not enough!

One could now raise the question of why modern LLMs are not enough, as multi-modal question answering is already supported by recent LLMs such as GPT-4o. To understand the downsides of such question answering approaches, let us look at Figure 1b, which shows how a RAG-based question answering approach can be used to answer questions over multi-modal datasets. The main idea is that the user question is forwarded as part of the input to the LLM, together with the relevant data sources that are retrieved from a vector database based on the user question. Based on this input prompt, the LLM then reasons over the user's question and data and generates its output, which is forwarded to the user as a response. However, this LLM-based approach for query answering comes with significant downsides:

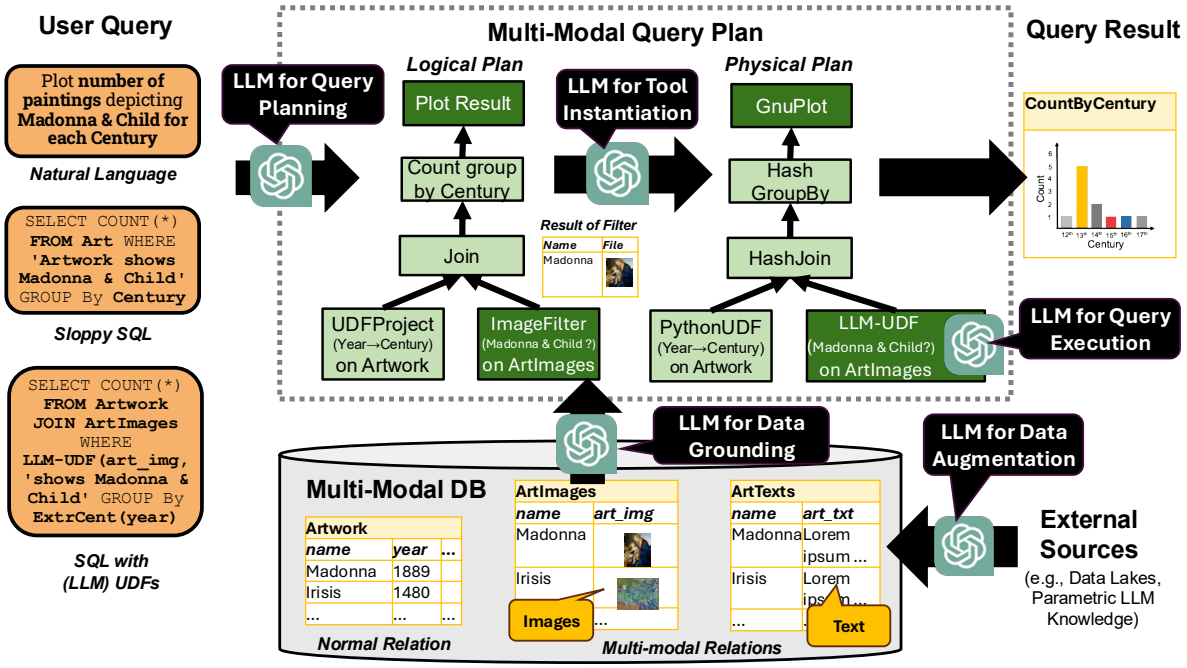


Figure 2: Sketch of an LLM-Augmented DBMS for Multi-modal Question Answering. The idea of an LLM-augmented DBMS is centered around a multi-modal relational data model where relations can include complex types such as images and text. Building on this model, the system supports several query interfaces that abstract away the exact schema, enabling users to query data using natural or imprecise language. For execution, queries are compiled into multi-modal query plans composed of classical and multi-modal operators, facilitating efficient query processing and optimization across diverse data modalities. LLMs are used in several places in LLM-Augmented DBMS, including query planning and execution, as well as data grounding and augmentation.

1. *Limited Query Complexity:* While question answering approaches with LLMs are able to handle more and more complex queries, which also stems from the fact of recent developments such as reasoning, they are still limited in various dimensions. For example, LLMs are known to be notoriously "bad in math. As such, queries that involve arithmetic or aggregations (e.g., counting as in our example) often fail.
2. *Limited Scalability of Query Answering:* In the LLM-based approach, query answering is mapped to inference passes through the LLM parameters (which could be trillions) to generate the query response. This notion of query answering, however, introduces high latencies and costs even for small data sets, and aggregating over large datasets at the scale of PetaBytes that databases can handle today easily is far from being tangible with LLMs.
3. *Hallucination & Black Box Answers:* Query answering using LLMs is purely *generative* using the LLM as a black-box; i.e., the output (query result) is generated in a probabilistic manner based on the input and it is not clear how the LLM came up with the answer. As such, LLMs can come up with query answers that are not grounded in the input data at all (also known as hallucination).

2.2 A Sketch of an LLM-augmented DBMS

In the following, we sketch our vision of LLM-augmented DBMS as shown in Figure 2. The sketch we present in this section builds on key concepts of our prior work (e.g., CAESURA, ELEET, OmniscientDB) that pioneered ideas in this field. Many of these ideas have also been proposed in parallel by other

systems such as ThalamusDB, Palimpzest, Lotus, and Galois that rely on similar notions. We refer to the parallel ideas and put them in context.

Data Model & Query Interfaces. At the core, as a *data model* an LLM-augmented DBMS uses multi-modal relations for storing and processing data, while multi-modal data is represented as complex data types (image, text) for columns of a table. For example, as shown in Figure 2 (bottom), the images are stored in the `art_img` column of the `ArtImages` table. Using the concept of relations as a core data model allows an LLM-augmented DBMS to build and extend on the efficient techniques for query execution (e.g., hash joins) and query plans for optimization over multi-modal relations, as we discuss below.

As *query interfaces*, we envision that LLM-augmented DBMSs support various alternatives. Similar to the LLM-based approach, we believe that natural language queries are one important option as they allow users to query data without knowing the exact representation of data in the database. However, understanding natural language queries and mapping them into an executable format comes with its own challenges, as natural language queries can be ambiguous or incomplete. As such, we envision that users can also use alternative query interfaces. One interesting direction that we are currently working on is a variant of SQL (which we call Sloppy SQL) that allows users to formulate "imprecise" SQL queries without knowing the exact data representation. For example, Figure 2 (left) shows a Sloppy SQL statement where tables and attribute names are used in the query that have no direct correspondence in the schema. Similarly, a natural-language predicate is used for filtering, which does not specify an attribute at all. For such queries, the task of the LLM-augmented DBMSs is to ground the query into the actual representation of the database, which is part of the query translation as we discuss next. Finally, users who know the schema can also use precise SQL extended with LLM UDFs. This direction is also what current database vendors integrate into their products. While that way, users can precisely formulate queries, it also demands that the user know the schema in detail (i.e., it comes with high query taxes).

Query Execution using Multi-Modal Plans. For execution, an LLM-augmented DBMS translates user queries into what we call a multi-modal query plan — an idea we have pioneered in our work around CAESURA. A multi-modal query plan resembles a query processing pipeline — containing processing steps (i.e., operators) that can deal with multi-modal relations as input (i.e., tables that can but must not contain image and text columns).

As shown in Figure 2 (center), such a multi-modal query plan contains classical operators (light green) that operate on columns without images and text, as well as multi-modal operators (dark green), which are applied to image and text columns. For instance, the multi-modal query plan in the example contains a classical operator (UDFProject) that scans over the `Artwork` table and uses a classical UDF to compute the century column from the year attribute according to the user query. Moreover, the plan also contains a multi-modal operator for selecting individual images from an image collection (see `art_img` column of `ArtImages` table). In particular, in the example, the multi-modal operator (ImageFilter) is used to filter out pictures that depict *Madonna & Child*.

An important property of a multi-modal plan is that the output of every operator is yet another multi-modal relation that can be processed by upstream operators. In our case, we apply joins and aggregates known from classical databases to compute the aggregate result (i.e., count by century). Moreover, the final result is then rendered again by a multi-modal operator that creates a visualization from the query result, which is, in fact, again a multi-modal relation with one row that contains the visualization of the aggregate result as a bar chart.

2.3 Benefits of using Multi-Modal Plans

Using the notion of query plans to answer multi-modal queries comes with many benefits that alleviate the limitations of a pure LLM-based approach discussed before.

1. *Execution outside LLM (if possible)*: First, different from a pure LLM-based approach, we do not map the full query execution into an inference pass of an LLM, but we use efficient and scalable algorithms such as hash-joins and hash-aggregates for query answering when possible and only rely on LLMs when needed (e.g., to realize an image filter using an LLM-UDF).
2. *Plan Optimizations*: Second, an even more important benefit is that using plans for query execution allows us to optimize such plans by extending well-known approaches known from relational databases to multi-modal relations. For example, one important technique in classical databases is join ordering to decide on which order to filter and join data sources, which might even have a higher impact on multi-modal relations if we can thus reduce the number of calls to an LLM (which are typically highly expensive).
3. *Separating Logical & Physical Operations*: An idea we have pioneered in CAESURA before others used this separation is the use of logical and physical plans for multi-modal query answering, which is a well-known technique in classical databases to select an algorithm for executing a logical operator (e.g., using a hash-join to execute a classical join). In multi-modal databases, this separation also allows us to reason about which algorithm to use. For example, an image or a text filter can be realized using an expensive LLM-UDF but also more lightweight (smaller) models (e.g., BLIB or our own ELEET model) that are faster but potentially less accurate.
4. *Plans are Explanations:* A major issue of a pure LLM-based approach for query answering, as shown in Figure 1b, is that it is a black-box, which is, in particular, problematic if the result is a hallucination; i.e., it is not clear how the LLM generated its output. When using plans for query answering as shown in Figure 2 (center) instead, this is different as the user can inspect the plan and reason about whether the steps of the plan operations in fact answer the user query. Moreover, users can inspect intermediate results and thus find out which operation (e.g., an image filter) produced a potentially wrong answer.

2.4 Where LLMs can be used in an LLM-augmented DBMS?

In the following, we outline how LLMs can enable the core functionalities of an LLM-augmented DBMS in various components of the system.

LLM for Query Planning. To answer queries over multi-modal relations, the system first translates user input into an executable multi-modal query plan. For natural language queries, this involves decomposing the query into a sequence of high-level reasoning steps that map to a fixed set of supported operators (i.e., tools). Our CAESURA system pioneered a robust approach that incrementally constructs and executes such plans, tolerating ambiguity and partial understanding. For Sloppy SQL, where users issue vague or underspecified SQL-like queries, the LLM is used to identify and resolve references to tables and attributes based on semantic similarity, context, and schema hints.

LLM for Data Grounding. A critical step in query translation is data grounding, where the LLM determines which data sources and columns are relevant to a given query. This is especially important for complex data types such as images and text, which may not have clear structural indicators. We provide the LLM with representative samples from the data to support this decision-making process during query planning. This capability enables the system to intelligently select appropriate tables and columns even in the absence of explicit schema references, an idea explored in our earlier work on CAESURA.

LLM for Tool Instantiation. Once a plan is generated, the individual operators within a multi-modal query plan need to be instantiated with specific parameters, which is done in the transition from logical to physical plan. This includes generating prompts for LLM UDFs (e.g., determining whether an image depicts "Madonna and Child"), constructing Python code for classical UDFs (e.g., converting a year to a century), or configuring visualization tools (e.g., specifying Gnuplot parameters for a bar chart). LLMs are used in this step to synthesize these components based on the query intent and available data, enabling dynamic and context-aware operator instantiation. These ideas are a key part of the CAESURA framework.

LLM for Query Execution. During query execution, LLMs are used to evaluate user-defined functions on multi-modal content. For instance, an LLM UDF may be used to filter image columns based on whether they depict a particular scene, extract semantic information from text, or define join conditions over similar images or between related images and texts. Such operations go beyond traditional relational processing and require careful optimization of LLM invocation to maintain performance. To this end, we developed ELEET, a technique for efficient execution using smaller, specialized LLMs tailored for multi-modal query workloads, as discussed in more detail in Section 4.

LLM for Data Augmentation. Traditional databases operate under a closed-world assumption—they can only answer questions using stored data. To overcome this limitation, LLMs enable open-world querying by augmenting existing data with external sources. For example, when a user query involves information not present in the database (e.g., background details about Pablo Picasso), the LLM can retrieve and integrate relevant facts from external corpora, such as data lakes, or use the parametric knowledge of LLMs as a data source as demonstrated in our OmniscientDB system [1].

2.5 The Landscape of LLM-augmented Data Systems

Integrating non-tabular data into relational workflows is a key challenge that different systems approach [2–5]. In this paper, we present a holistic vision of an LLM-augmented Data Systems that provides a holistic view of how we think such a system should look in the future, which combines several of our seminal results which we explain in Sections 3 to 5. First, with CAESURA [6], we have pioneered the idea of using multi-modal query plans that were created from a natural language query. By contrast, Palimpzest [3] and DocETL [5] handle unstructured data through similar ideas using also pipelines that are very close to what we called multi-modal plans in CAESURA, however, they suggest alternative query interfaces: Palimpzest lets users write declarative queries over document collections, and DocETL provides a YAML-based pipeline language for complex document-processing tasks. These pipeline systems complement the SQL-centric approaches above. Moreover, ELEET enables fast and efficient multi-modal operators (e.g., for implementing filter operations on text documents) using a targeted small language model to extract structured fields from text, achieving up to 575× speedups over LLM baselines [7]. In contrast, ThalamusDB similarly allows SQL queries to directly reference visual and audio content, using a pool of zero-shot classifiers and an approximate query optimizer to minimize label requests for predicates [2]. LOTUS also targets integration of unstructured content via its semantic operators over tables of text [4]. Finally, OmniscientDB implicitly uses an LLM to *augment* relational tables via simple SQL commands [1]. Again, other similar work has been suggested. The most relevant work for this is Galois, a system for querying pre-trained LLMs with SQL [8]. This approach exploits the automatic generation of a logical query plan from a SQL script that serves as a structured chain of thought, guiding the LLM in processing complex queries effectively.

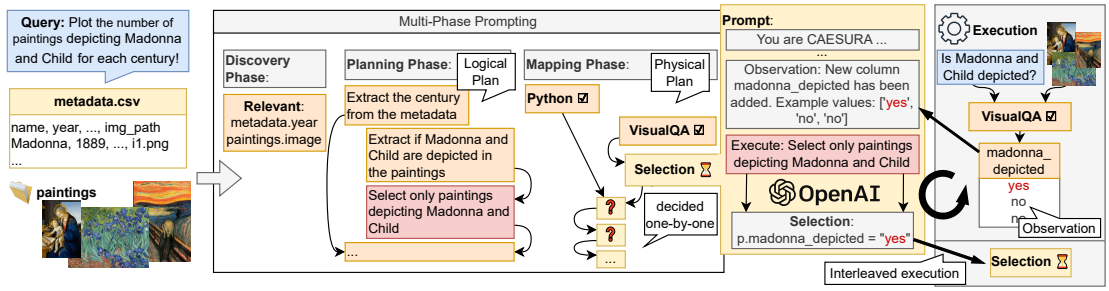


Figure 3: Query Planning in CAESURA. CAESURA transforms the query into a multi-modal query-plan using a series of prompts using an LLM. In the *Discovery Phase*, the LLM is prompted to identify data items relevant for the query, such as relevant columns and datasets. In the *Planning Phase*, the LLM is prompted to construct a sequence of steps to satisfy the user request (Logical Plan). The final *Mapping Phase* is interleaved with *Execution*: a physical operator is chosen for each of the logical steps and executed incrementally. That allows the LLM to take the output of previous executions into account when choosing the physical operator and operator arguments (e.g., filter conditions as depicted in the figure) to enhance plan correctness.

3 CAESURA - LLMs for Query Planning

In essence, as discussed before, in CAESURA we orient ourselves on the phases of traditional query planning and first generate a logical plan, which is afterwards translated to a physical plan. In the following, we summarize the query planning process and the use of LLMs for the planning as shown in .

3.1 Phases of Query Planning

Splitting the process of query planning into several phases allows us to tailor the prompts for query planning to the specific decisions of each phase. See Figure 3 for an overview of the three phases, which we elaborate in more detail in the following. In a nutshell, we first identify the relevant data sources, then in the planning phase we let the LLM generate the logical plan, and finally, in the mapping phase we let the LLM select the operators to obtain a physical plan.

Discovery Phase. In the first phase, we decide which data sources (e.g., in a data lake) provide relevant information for the current query. We only briefly describe this phase, because the focus of this paper is on query planning. In essence, CAESURA first narrows down the relevant tables, image collections, etc., using dense retrieval (similar to Symphony [9]). Afterwards, for tabular data sources, we prompt the LLM to decide which columns of the retrieved data are relevant to the user query.

Planning Phase. In the planning phase, which is at the core of CAESURA, the LLM is prompted to come up with a logical query plan that contains a natural language description of all steps necessary to satisfy the user's request. The prompt consists of several parts: (1) a description of the data, (2) the capabilities of CAESURA, (3) an output format description, and (4) finally, the user query and an instruction telling the model to come up with a plan. Notice how the multi-modal data is presented to the LLM: it is modeled as a special two-columned table where one column has the special datatype IMAGE. The capabilities of CAESURA describe the logical actions that CAESURA can take with the help of the available operators, as can be seen in the example. Using this prompt, the LLM generates a stepwise (textual) plan that describes the logical plan in the output format specified in the input prompt. The generated stepwise plan is then parsed by CAESURA into a logical plan.

Mapping (Tool Instantiation) and Interleaved Execution. In the last phase, each previously determined logical step is mapped to a physical operator (and its input arguments) using a prompt

similar to the one in Figure 3 (right). The prompt for this phase contains a short summary of the operators and what they can be used for. Moreover, we do not decide on all the physical operators for all logical steps at once. Instead, we incrementally decide for each step and then execute it directly.

3.2 Experimental Results: Planning Quality

In our experiments reported in [6], we were primarily interested in whether CAESURA is able to construct correct query plans. Below, we report the most interesting findings from these experiments.

Datasets and Workloads. Since there does not yet exist a benchmark for the scenarios we envision for CAESURA, we constructed two multi-modal datasets. (1) The *artwork* dataset (with tables and images) resembles the example from Figures 1. The dataset contains a table about painting metadata as well as an image collection containing images of the artworks. We use Wikidata to construct both the metadata table and the image corpus: for the metadata table, we extract title, inception, movement, etc., for all Wikidata entities that are instances of 'painting'. (2) The second dataset is the *rotowire* dataset (with tables and text) [10], which consists of textual game reports of basketball games, containing important statistics (e.g., the number of scored points) of players and teams that participated in each game.

Our Results. For this experiment, we are interested in the query planning abilities of LLMs. Hence, we skip the data discovery step and assume perfect retrieval (to not measure retrieval performance). The results that show the quality of the planning — how often CAESURA created a correct plan — are shown in Table 1. The queries used in this experiment are clustered along several aspects. Importantly, these queries were not used for tuning the prompts during the development. We see that CAESURA using GPT-4 is better than ChatGPT-3.5 and is even able to correctly translate 87.5% of queries despite never being fine-tuned on the queries. The approach works especially well on the artwork dataset, where CAESURA is able to translate all queries to correct query plans. However, we also see that there is still room for improvement. In particular, on the Rotowire dataset, which consists of more tables and contains texts instead of images, fewer queries are translated correctly.

Models	**ChatGPT-3.5**		**GPT-4**	
Plan type	logical	physical	logical	physical
Artwork overall	79.2%	70.8%	100%	100%
Rotowire overall	50.0%	41.7%	87.5%	75.0%
Single modality	79.2%	75.0%	100%	92.7%
Multiple modalities	50.0%	37.5%	87.5%	83.3%
All	64.6%	56.2%	93.8%	**87.5%**

Table 1: CAESURA Planning Quality. The table shows the fraction of correctly translated plans for the different datasets, modalities, and output formats. We show the percentage of correctly generated logical plans, as well as physical plans.

4 ELEET - LLMs for Query Execution

In this section, we introduce ELEET, a query execution engine designed to support so-called multi-modal operators (MMOps), which operate over structured tables and text collections. While general-purpose LLMs such as GPT-4 offer the potential to perform such operations, their high computational cost makes them impractical for query execution. Instead, ELEET centers around a compact and efficient language model — the ELEET model — specifically trained to execute operations like joins and filters over multi-modal input. ELEET achieves this by combining pointer-based extraction with targeted pre-training and fine-tuning strategies, enabling fast and accurate execution of relational-style queries directly over text.

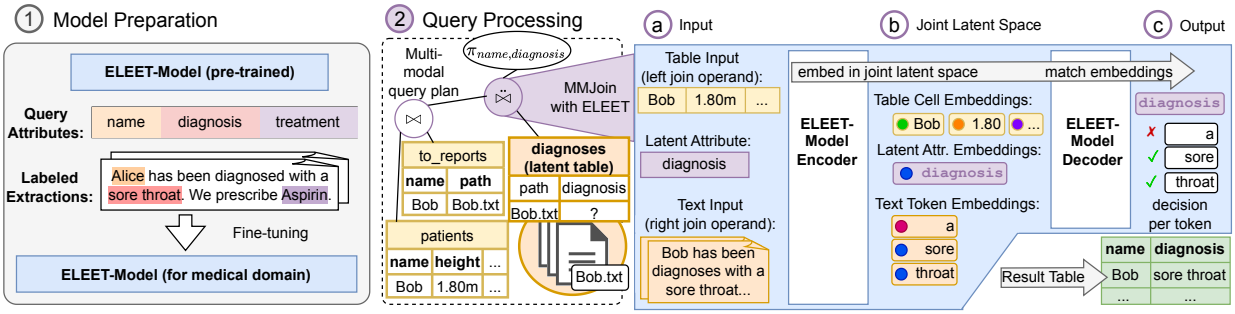


Figure 5: Overview of ELEET. In an offline phase, the ELEET-model can be fine-tuned for unseen domains ①. Fine-tuning the ELEET model for an unseen domain is a one-time effort and requires a small sample of a few labeled texts. ② For query execution, ELEET uses multi-modal query plans that contain traditional (white) and multi-modal database operators (purple). To compute the result of a multi-modal operation such as a join over texts, the ELEET-model is used (see ⓐ to ⓒ): During the execution of a multi-modal operation, the ELEET model first computes embeddings of the query attributes, texts, and table input ⓐ, using its encoder ⓑ. Afterwards, the ELEET-model matches text token embeddings to query attribute embeddings to extract the output table from the text using its extractive decoder ⓒ, which decides which tokens qualify for a given query attribute.

4.1 The ELEET Model

A Simple Example. Figure 4 illustrates the usage of ELEET in a practical scenario. Consider a medical database that stores structured patient attributes alongside textual diagnostic reports. If all data were tabular, querying correlations (e.g., between age and diagnosis) would be straightforward using standard SQL. However, when crucial information like diagnoses is embedded in unstructured text, extracting it for analysis becomes a labor-intensive task involving custom NLP pipelines. ELEET eliminates this barrier: it enables querying over the text by treating it as a latent table. Instead of manually coding extract-transform-load (ETL) routines, users can issue queries that include text attributes as if they were columns in a regular table, and ELEET retrieves the necessary information from the text during query execution.

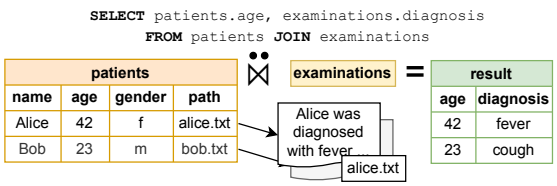


Figure 4: ELEET Example. A query that executes a multi-modal join between a patient table and examination reports. ELEET analyzes the texts and extracts values for the diagnosis from each examination report.

The Model. At the heart of ELEET is a compact model with only 140M parameters, trained specifically for extractive query operations over multi-modal data. Given a structured tuple, a textual document, and a latent attribute (e.g., *diagnosis*), the model identifies relevant spans in the text that match the semantics of the operator. Unlike autoregressive LLMs, ELEET avoids decoding token-by-token; instead, it uses pointer-based supervision to select answer spans in a single forward pass.

During pre-training, ELEET learns to extract latent attributes from weakly structured documents. This training procedure enables generalization to diverse linguistic styles and schemas. At inference time, ELEET can perform operations such as filtering, joining, and projection by extracting values conditioned on structured context. Importantly, the model is adaptable: only a handful of labeled examples are needed to fine-tune ELEET for high-accuracy performance on previously unseen domains.

A Sketch of a Multi-Modal Join Figure 2 depicts a multi-modal join using the ELEET-Model. The join in Figure 2 needs to extract the diagnosis for each patient tuple coming from the first join

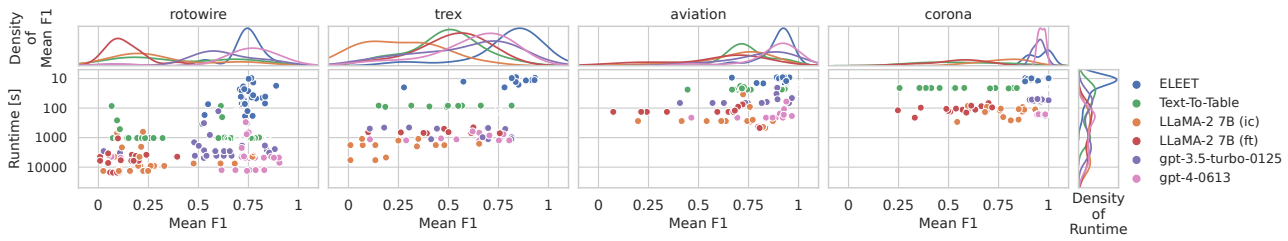


Figure 6: Performance and Accuracy comparison of ELEET with GPT-4 (175B), LLaMA-2 (7B), and T5 (11B) on multi-modal query execution tasks across different datasets. The plot shows F1 score (y-axis) versus query execution time (x-axis), where lower values of time and higher values of F1 score indicate better performance.

of the *patients* and *to_ reports* table (i.e., each patient can have multiple reports). For executing this query, we feed the attributes into the ELEET-model to extract from text (i.e., *diagnosis*; called *latent attribute*) together with the patient data from the first join and the text documents to be joined into the ELEET-model. For example, for joining the patient tuple of Bob with his patient report in Figure 2 ⓐ, ELEET feeds the patient tuple (containing name, height, ...), the latent attribute *diagnosis*, and the patient report of Bob into the ELEET-model. For extracting the diagnosis, the encoder of our ELEET-model maps all inputs into a joint latent space ⓑ. Afterwards, the decoder identifies spans of texts in the report that qualify as diagnosis, such as the text span *sore throat* in Figure 2 ⓒ. Finally, the result row $\{\text{name} \mapsto \text{Bob}, \text{diagnosis} \mapsto \text{sore throat}\}$ with the extracted values from the text is materialized.

In this section, we present the results of our experimental evaluation, which justifies the design of ELEET. We constructed a challenging benchmark containing 70 multi-modal query plans over four data sets that we used in the original evaluation in [7]. We show that ELEET is more efficient and accurate than existing baselines, including fine-tuned LLM for executing multi-modal queries.

4.2 Experimental Results: Query Performance and Accuracy.

To show how ELEET compares to using state-of-the-art LLMs as operators, we conducted a set of experiments in the original publication [7]. Below, we report the most interesting findings from these experiments.

Dataset and Benchmark. To validate ELEET's effectiveness, we conducted an extensive experimental study on a new benchmark comprising 70 queries across four diverse multi-modal databases: sports (Rotowire), Wikipedia-derived data (T-REx), aviation incident reports, and COVID-19 status updates. These databases include a mix of numeric, categorical, and textual attributes, and span domains that differ in style, granularity, and data cleanliness.

The benchmark queries test a range of operators—joins, selections, unions, scans, and aggregations—all requiring the integration of structured tables with free-text collections. Each query plan includes between one and three MMOps, simulating real-world analytic tasks.

Our Results. Our results demonstrate that ELEET consistently achieves high F1 scores across all datasets while maintaining sub-second latency per query. Notably, it outperforms much larger models such as LLaMA-2 (7B) and GPT-4 (175B) in both speed and accuracy on extractive tasks. This efficiency stems from ELEET's lightweight design, optimized decoder, and task-specific training. While LLaMA-2 and GPT-4 require few-shot prompting or costly autoregressive decoding, ELEET's architecture allows direct span extraction in a single pass.

Figure 6 presents a performance comparison of ELEET with GPT-4 (175B), LLaMA-2 (7B), and T5 (11B) on the multi-modal query execution tasks. The plot shows a trade-off between F1 score and execution time for each model across the benchmark datasets. ELEET consistently achieves higher F1 scores than the other models, while also maintaining significantly lower query execution time. This highlights ELEET's efficiency in handling complex multi-modal queries without sacrificing accuracy.

5 OmniscientDB - LLM for Database Augmentation

Traditionally, databases are required to explicitly capture all relevant facts in order to be queried by the user. However, this so-called closed-world assumption significantly limits the ways in which a database can be used. For example, think of a database that stores information about movies. While a breakdown of the revenue by actor might be a query a user wants to issue, the actor information might not be stored in the database. Today, the only way to make additional information available for querying is to explicitly integrate additional data sources into the database, which requires extensive manual efforts.

Idea and Simple Example. With OmniscientDB [1], we presented our vision of an open-world DBMS that can automatically augment existing databases with world knowledge for the execution of SQL queries. To do so, OmniscientDB can not only generate additional tables on-the-fly but also complete existing tables with user-requested rows or columns. To enable this, OmniscientDB makes use of the world knowledge that is implicitly stored in LLMs such as GPT-4 etc.

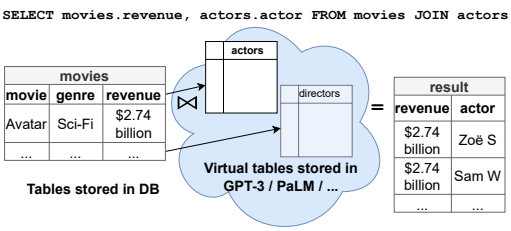


Figure 7: OmniscientDB Example. A SQL Query that joins a *movies* table stored in the database with an *actors* virtual table derived from an LLM (e.g., GPT-4). The join operation allows augmenting the *movies* table with actor information using a simple SQL query.

To illustrate the benefits of OmniscientDB by an example, imagine a data scientist trying to analyze the revenues of recent movies, as mentioned before. For the questions of the data scientist, important information like the starring actors or directors, however, is not contained in the dataset, despite their potentially large impact on the movies' revenues. While traditionally, such information would need to be explicitly integrated first, with OmniscientDB, the data scientist could simply use the knowledge stored in the LLM for augmentation. For instance, OmniscientDB automatically generates the missing information about the actors starring in the movies, allowing a more extensive analysis.

Virtual Tables. OmniscientDB leverages the knowledge implicitly stored in the parameters of LLMs for augmentation and makes it available for querying via SQL using so-called virtual tables. Virtual tables can be treated by users just like traditional tables. However, they are not explicitly stored in the database but instead act as a proxy for the knowledge stored in LLMs. For instance, in the example above, the user is able to join the movies table with the virtual *actors* table to generate additional information about actors, as shown in Figure 7. The information about actors is materialized on-the-fly during the execution of the query by prompting the LLM. Furthermore, in addition to virtual tables, we envision that OmniscientDB also allows adding virtual columns to existing tables. For example, the movie table might miss a column for the production year, which could also be extracted from a LLM.

Challenges and Vision. A key challenge of OmniscientDB is that the knowledge that OmniscientDB is able to make available is obviously bound by the parametric knowledge in LLMs. However, LLMs have been extended by retrieval mechanisms to retrieve external knowledge [11–13] and are even able to perform web searches [14] or interact with APIs [15]. Hence, our vision is to exploit these rapid

advancements and make the abilities and knowledge of modern LLMs available for users to easily augment their datasets with OmniscientDB.

Going forward, we envision OmniscientDB to become a true open-world database, allowing for arbitrarily complex database operations. This not only includes joins and unions with virtual tables, to generate missing columns or rows, but also other operations. For instance, existing incomplete tables could be marked as semi-virtual, meaning that some tuples or values are stored explicitly, but additional missing tuples can be generated on the fly if needed. Furthermore, we want virtual tables to be updatable, in case the knowledge in the LLM is outdated, or support that the database itself suggests information that could be used for augmentation.

6 Research Roadmap

We have seen that using today's state-of-the-art LLMs is an interesting direction to enable a new generation of database systems to cut query and data taxes by LLM-augmented database systems for multi-modal question answering as outlined in Section 2. However, there are still an abundance of interesting open challenges to be solved, as well as many opportunities that have not yet been widely researched. Below, we discuss some of the major challenges and opportunities we deem interesting for the community.

6.1 Challenges & Opportunities

Challenge: Query Optimization & Execution. A major future challenge is designing effective optimization strategies for query execution in multi-modal database systems, where queries span structured data, text, and images. Traditional cost- and rule-based optimization techniques fall short in this setting, especially when queries involve operators over collections of images or text documents — and require substantial rethinking: (1) A central challenge is cardinality estimation for multi-modal data. Unlike structured data, estimating result sizes for queries over images or text requires novel techniques (e.g., filter out all images which show Madonna & Child as shown in Figure 1a). One promising direction is the use of embedding-based histograms, which cluster similar content to enable approximate count estimates. Such estimates are essential for choosing efficient join orders in multi-modal query plans. (2) Another key challenge is physical operator selection when incorporating LLMs into query plans. Operators applied to unstructured data (e.g., filtering images or text) can vary widely in both cost and accuracy depending on the implementation strategy. Query optimizers must learn to choose between lightweight rule-based methods, traditional operators, and high-cost, high-accuracy LLM inference—balancing computational efficiency with the accuracy demands of the query. (3) Finally, adaptive query optimization must be revisited and extended to accommodate the unpredictable behavior of LLMs and the nature of multi-modal data. This involves building systems that can adjust execution strategies on the fly based on the observed performance and accuracy. Reinforcement learning presents a promising avenue here.
Although some initial work exists in these areas, many open questions remain. Addressing them is critical to building robust, efficient multi-modal database systems capable of handling real-world workloads.

Challenge: Towards Community-Standard Benchmarks. A key challenge for the field is the lack of standardized benchmarks to evaluate and compare multi-modal database systems. While individual areas where LLMs are used in databases, such as Text-to-SQL, benefit from well-established community benchmarks like SPIDER and BIRD, no equivalent benchmarks exist for LLM-augmented databases to realize multi-modal database systems as envisioned in this paper. Instead, current evaluations rely on small, system-specific benchmarks that typically reflect narrow use cases tailored to the strengths of

individual systems. A first pressing question is: what constitutes a meaningful and comprehensive set of use cases that can assess the capabilities of multi-modal databases and guide future development? Addressing this will require selecting diverse application domains and building on existing datasets to design challenging scenarios that thoroughly test multi-modal functionality. Equally important is the development of a set of accepted evaluation metrics. These should go beyond accuracy and query runtime and include aspects such as scalability under high-volume multi-modal workloads, and other aspects, such as missing data, which require systems to understand when a query can not be answered. Understanding such aspects will be critical for advancing the field. Finally, benchmark suites must also assess system robustness, including performance under out-of-distribution inputs and degraded or incomplete data. Such tests are essential for evaluating the reliability and real-world applicability of multi-modal database systems.

Opportunity: Beyond SQL with Semantic Operations. Various query interfaces have been proposed for multi-modal database systems, including natural language queries, SQL-like queries with semantic predicates (e.g., filters or joins based on natural language), and SQL extended with LLM-based UDFs. While diverse in form, these interfaces typically follow a common pattern: they translate into SQL-style query plans extended with multi-modal operators that operate on unstructured data such as images or text. However, by using LLMs for query planning—as we propose in this paper—query interfaces can support far more expressive, goal-driven interactions. Consider, for example, a crime-incident database used by investigators to identify a suspect based on multiple sources: textual witness reports, police image data, and structured records such as registered weapons. The user's high-level query might be as simple as: Find the suspect involved in the crime on Jan. 15, 2018 in SQL City. Solving such a query requires a sequence of interdependent steps: analyzing intermediate results (e.g., matching details from witness reports), reasoning across modalities, and potentially backtracking if a search path proves unproductive. This raises the question: can a database system assist users in navigating such multi-turn, exploratory queries instead of requiring them to manually issue and manage each step? To address this, we explore the potential for goal-driven query interfaces, where the system plans and executes a sequence of multi-modal queries to solve a user-defined problem, asking for clarification only when necessary (e.g., when the available data is insufficient). Interestingly, the sequences of generated queries can directly act as an explanation of how the problem was solved.

Opportunity: Open-World Databases A compelling opportunity for LLM-augmented databases lies in enabling open-world databases—systems capable of answering queries that go beyond the facts explicitly stored in the database. Our initial work on OmniscientDB takes a step in this direction by leveraging the parametric knowledge of LLMs to augment tables with additional rows and columns at query time. However, this is only a starting point. A more ambitious goal is to enrich databases not only with internal model knowledge but also with external data sources, such as those found in data lakes. For example, consider the query: "How many pictures show Madonna and Child and are by Leonardo da Vinci?"—where the artist information is not present in the existing database (as shown in Figure 1a). To answer such queries, relevant external data (e.g., a CSV containing artwork metadata) must be retrieved and seamlessly integrated with the current schema. LLMs can assist by interpreting retrieved datasets, reasoning about their relevance, and proposing integration strategies, such as joining external and internal tables. When mismatches arise (e.g., differing artwork titles or formats), semantic joins—based on natural language predicates and embedding similarity—can help align the data sources effectively. Finally, we envision a future in which these retrieval and integration operations are embedded directly into the query planning process. A database system could dynamically perform multiple retrievals to incrementally augment its data and even backtrack or revise earlier augmentations if they fail to contribute to answering the query.

6.2 Potential Limitations

One might assume that a key limitation of our vision is the requirement for users to preprocess and load data into structured, multi-modal relations before querying. This could be seen as a significant barrier to adoption, particularly in real-world scenarios where data often resides in raw, unstructured form across diverse sources in data lakes. However, as discussed in our vision, this is not a hard requirement. Users could also begin with minimal or even no structured data and incrementally augment their data by retrieving relevant content from data lakes at query time. In this mode, the system constructs multi-modal relations on-the-fly, guided by the user's query intent. In the most extreme version of this vision, no schema is defined upfront—instead, the schema is synthesized dynamically as part of the query answering process.

This schema-on-demand approach is a direction we have not yet explored in our work, but it represents a highly promising avenue for future research. It would enable querying data lakes directly without prior data curation or integration—essentially inverting the traditional data engineering pipeline. In such a setting, data retrieval must be tightly coupled with on-the-fly data cleaning and integration. For example, aligning join keys, resolving type mismatches, or eliminating duplicates may all need to be performed as part of query execution. This is particularly critical for multi-modal data, where inconsistencies and redundancies are common, and where traditional schema-based assumptions about cleanliness and integration no longer hold. As such, developing mechanisms for adaptive, query-driven data cleaning and integration becomes a central research challenge in realizing this more flexible and powerful model of multi-modal data interaction.

7 The Relational Model is Dead! Long Live the Relational Model!

In this paper, we have argued that relational databases impose substantial overhead on users, both in terms of expressing queries and preparing data—particularly in the context of complex, multi-modal information needs. Despite these limitations, we do not advocate for abandoning the relational model altogether. Instead, we propose to retain the relational model as an internal abstraction for future data systems, leveraging its proven strengths in query optimization, physical execution, and scalability. However, we argue that it should no longer serve as the primary interface exposed to users. To bridge this gap, we envision data systems that extend the relational model with support for complex data types and provide more intuitive query interfaces—ranging from natural language to schema-agnostic structured languages. Central to this vision is the integration of large language models (LLMs), which enable semantic query interpretation, data discovery, and access to heterogeneous and non-tabular data sources. We refer to such systems as *LLM-augmented databases*: systems that preserve the robustness and efficiency of relational engines internally while offering a more accessible and expressive user experience externally. In doing so, we reconcile the internal utility of the relational model with the evolving demands of modern data-centric applications.

Acknowledgements

First and foremost, I would like to thank all my PhD students and Postdocs who are working on the topics and directions mentioned in this paper, as well as my collaborators. Moreover, I would like to thank also sponsors who support this work, such as the LOEWE program in Hesse (Reference III 5 - 519/05.00.003-(0005)), the DFG project MAgiQ BI 2011/3-1, hessian.AI at TU Darmstadt, as well as DFKI Darmstadt.

References

[1] V. Kakar *et al.*, "Omniscientdb: Augmenting relational queries with world knowledge," in *AIDM@SIGMOD*, 2023.

[2] S. Cheng *et al.*, "Thalamusdb: Querying multimodal data with language," in *VLDB*, 2024.

[3] B. Ding *et al.*, "Palimpzest: Declarative llm query optimization," in *SIGMOD*, 2024.

[4] N. Yaghmazadeh *et al.*, "Lotus: Declarative semantic queries over text tables," in *VLDB*, 2024.

[5] M. Zhou *et al.*, "Docetl: Declarative document processing with llms," in *CIDR*, 2024.

[6] T. Rekatsinas, Z. Zhang *et al.*, "Caesura: Query planning with language models for multi-modal databases," in *CIDR*, 2024.

[7] S. Simeonidou, T. Kraska *et al.*, "Eleet: Efficient learned execution for entity-centric text tables," *VLDB*, 2025, to appear.

[8] M. Saeed *et al.*, "Querying large language models with SQL," in *EDBT*, 2024.

[9] N. Tang, C. Yang, Z. Zhang, Y. Luo, J. Fan, L. Cao, S. Madden, and A. Y. Halevy, "Symphony: Towards trustworthy question answering and verification using RAG over multimodal data lakes," *IEEE Data Eng. Bull.*, vol. 48, no. 4, pp. 135–146, 2024.

[10] S. Wiseman, S. M. Shieber, and A. M. Rush, "Challenges in Data-to-Document Generation," *arXiv:1707.08052 [cs]*, Jul. 2017.

[11] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, "REALM: retrieval-augmented language model pre-training," *CoRR*, vol. abs/2002.08909, 2020.

[12] K. Lee, M. Chang, and K. Toutanova, "Latent retrieval for weakly supervised open domain question answering," in *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pp. 6086–6096.

[13] P. S. H. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[14] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, X. Jiang, K. Cobbe, T. Eloundou, G. Krueger, K. Button, M. Knight, B. Chess, and J. Schulman, "Webgpt: Browser-assisted question-answering with human feedback," *CoRR*, vol. abs/2112.09332, 2021.

[15] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," *CoRR*, vol. abs/2302.04761, 2023.

Scalable Image AI via Self-Designing Storage

Utku Sirin, Victoria Kauffman, Aadit Saluja, Florian Klein, Jeremy Hsu,
Vlad Cainamisir, Qitong Wang, Konstantinos Kopsinis, Stratos Idreos
Harvard University

Abstract

Image AI has the potential to improve every aspect of human life. Image AI, however, is very expensive. We identify that the root cause of the problem is a long-overlooked and largely unexplored dimension: storage. Most images today are stored as JPEG files. JPEG is designed for digital photography. It maximally compresses images with minimal loss in visual quality. We observe that JPEG is a fixed design. AI problems, however, are diverse; every problem is unique in terms of how data should be stored and processed. Using a fixed design, such as JPEG, for all problems results in excessive data movements and costly image AI systems. This paper presents Image Calculator, a self-designing storage system that finds the optimal storage for a given image AI task. The Image Calculator achieves this by identifying design primitives for image storage and creating a design space comprising thousands of storage formats based on these design primitives, each capable of storing and representing data differently, with varying accuracy, inference and training time, as well as space consumption trade-offs. It efficiently searches within this design space by building performance models and using locality among its storage formats. It exploits the inherent frequency structure in image data to efficiently serve inference and training requests. We evaluate the Image Calculator across a diverse set of datasets, tasks, models, and hardware. We show that Image Calculator can generate storage formats that reduce end-to-end inference and training times by up to 14.2x and consumed space by up to 8.2x with little or no loss in accuracy, compared to state-of-the-art image storage formats. Its incremental computation and data-sharing schemes over frequency components allow scalable inference- and training-serving systems.

1 Efficient Image AI Storage

Image AI Has Potential to Improve Every Aspect of Human Life. Image AI has shown great success in numerous areas, providing more productive and safer services and tools. Medical doctors now use image AI to support their decision-making process in the early and late detection of diseases. Companies deploy image AI tools to enhance worker safety conditions. Manufacturing companies use image AI in their production pipelines to increase productivity. Farmers use image AI to efficiently monitor the conditions of their crops and take preventive actions against potential diseases, further improving their yields [1–3].

Image AI is Prohibitively Expensive. Every year, billions of digital cameras capture trillions of images. These images consume thousands of petabytes of storage in the cloud. Numerous AI applications process these massive datasets for various purposes, such as personalized content management, image reconstruction, and visual captioning. Applications access data over remote storage servers and need to move and process data in compute nodes using high-end processors. This incurs an immense dollar cost for cloud vendors and application developers. It costs millions of dollars to train and deploy a single model, which generates hundreds of thousands of pounds of carbon emissions, making image AI inaccessible [4, 5].

Inference and Training are Equally Costly. The lifetime of an AI model includes two main stages: training and inference. Training is where the model learns how to perform the task. Inference is where the model is deployed and performs a learned task. Both training and inference are important cost components. While some applications perform frequent re-trainings by newly arriving data, some applications are deployed across billions of devices concurrently and continuously performing inference, and some other applications perform both training and inference equally frequently [6–9].

Where Does Time Go? In Figure 1, we examine the end-to-end inference times of four state-of-the-art AI models: ResNet, EfficientNet, ShuffleNet, and MobileNet. For this experiment, the server is an A100 GPU machine. The data resides on disk as JPEG files. The model resides on the GPU. Each inference call requires reading the data from disk to main memory, decoding it from the bytes stream into a human-recognizable image, transferring it from main memory to GPU, and executing the AI model on the GPU. The graph on the left-hand side presents the normalized number of floating-point operations (FLOPs) required to execute each model over an image, and the graph on the right-hand side presents the end-to-end inference time. The figure shows that, although models become smaller and smaller, and the number of FLOPs decreases by as much as 98%, the inference time remains constant. The reason is that reducing the number of flops reduces the model execution time, i.e., the GPU time, but not the other time components. When the GPU time is reduced, the other time components surface up, and the inference time remains the same[1]. This suggests that inference time is a complex function of multiple time components, and reducing inference time requires reducing all time components. We performed the same experiment for training time and reached the same conclusion.

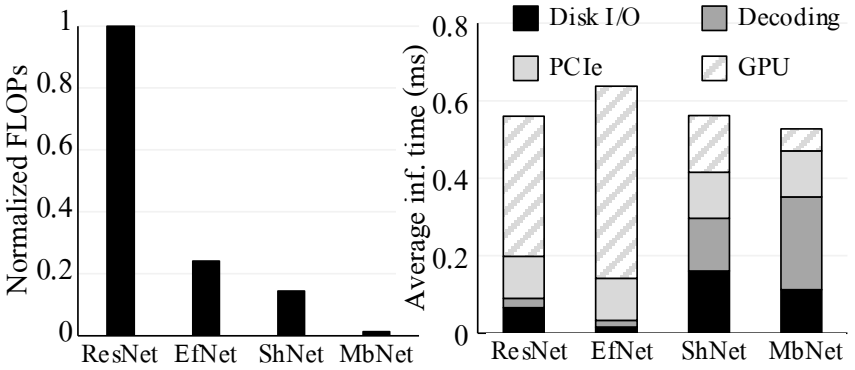


Figure 1: Reducing image AI cost requires improving all cost components.

Storage is Key. We observe that all the time components depend on one main factor: the amount of data moved/processed. This is determined by the storage format used for images. First, the storage format defines how we compress and store the data as well as how much data we read from the disk or network. Therefore, the storage format defines the I/O and decoding times. Second, PCIe time depends on the amount of data transferred over the PCIe link. PCIe stands for Peripheral Component Interconnect Express, and it serves as a bridge between the CPU and GPU. The amount of transferred data over PCIe depends on image size and, hence, storage format. Third, storage also determines the GPU time. The state-of-the-art AI models for images, i.e., CNNs and visual transformers, are primarily composed of transformers and convolutions, where inference and training times heavily depend on the height and width, i.e., the spatial dimensions of the image, which are part of the storage format.

JPEG is Designed for Digital Photography. Most images today are stored as JPEG files. JPEG is designed for digital photography. It lossily compresses images, where lost information minimally distorts the visual quality of the image. Users capture images for artistic, recreational, and/or communication purposes and store them using JPEG's algorithm so that stored images consume significantly less space

[1]EfficientNet has a higher GPU time than ResNet, although it requires only a quarter of FLOPs due to its more expensive pointwise operations (ReLU & BatchNorm).

than their raw versions. Thanks to JPEG's compression algorithm, storing and transferring images became much cheaper, which allowed the proliferation of images across the internet and social media.

AI is Diverse. We observe that JPEG is a fixed design. AI problems, however, are diverse. Every image AI problem has unique characteristics regarding how data should be stored and processed. We define an AI problem as a quadruple: dataset, AI model, machine, and performance budget. Any change in any dimension of an AI problem creates a completely new problem and requires a re-design of the storage. For example, a storage with five milliseconds of inference budget could be completely different than a storage with a hundred milliseconds inference budget. Therefore, using a fixed design for all problems is inefficient and can lead to severe performance issues due to excessive data movement.

Thesis. Our thesis is as follows.

> *Storage determines end-to-end image AI cost. Using a single storage for all problems results in inefficient image AI systems. Efficient systems need to tailor storage to the AI problem.*

Solution: Self-Designing Storage for Image AI. We introduce Image Calculator, a self-designing storage system for image AI. Figure 2 presents an overview of the Image Calculator. Unlike the current state of the art, which uses fixed storage designed for a single purpose, such as minimizing interference with the human eye, Image Calculator automatically generates and manages new storage for every AI task. Image Calculator achieves this by identifying design primitives for image storage and creating a rich design space of storage formats based on these primitives, each capable of storing and representing images at a different trade-off between space, inference/training time, and accuracy. The Image Calculator determines the optimal storage format for inference and training, given source data, hardware, and performance budget [12]. It efficiently searches within its design space based on performance models and locality among its storage formats. Formats with similar features also have similar performances. Starting from information-dense formats, Image Calculator quickly identifies high-quality storage candidates, allowing for fast and scalable inference and training. It exploits the inherent frequency structure in image data to efficiently scale model serving across a large number of applications. The Image Calculator breaks images into pieces, i.e., frequency components, and stores, transfers, and processes images frequency by frequency, rather than image by image, as conventionally done. This dramatically reduces data communication between clients & servers, providing a fast and scalable inference and training serving [13].

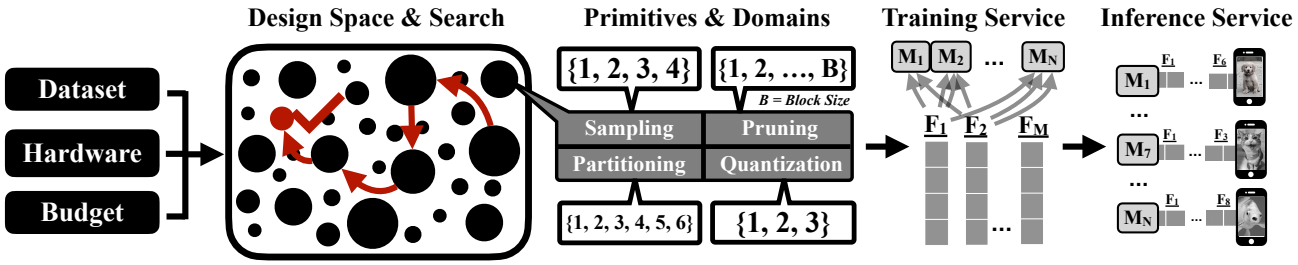


Figure 2: Image Calculator generates new image storage formats as combinations of fundamental design primitives. It exploits locality among storage formats for efficiently finding high-quality candidates. It stores, transfers, and processes images frequency by frequency instead of file by file for efficient serving of inference and training requests. Domains represent ID values rather than actual values. Sampling ID 4 means no sampling, for example. M_1 refers to first AI model, and F_1 refers to first frequency component.

2 Design Space

The Image Calculator adapts storage to a specific AI task. It achieves this by finding design primitives for image storage and creating a design space of storage formats based on design primitives.

Design Primitives for Image Storage. Creating a design space requires breaking down image storage formats and understanding their main design primitives. A rich design space encompasses a large number of candidates, each offering a distinct trade-off between space, inference/training time, and accuracy. We studied most used image/video storage formats such as JPEG [14], JPEG2000 [15], Learned JPEG [16], PNG [17], Bitmap [18], WebP [19], HEIF [20], AVC [21], and HEVC [22]. We show that four main design primitives cover all standard storage formats: (i) **Sampling**, (ii) **Partitioning**, (iii) **Pruning**, and (iv) **Quantization**. Sampling defines removing a set of columns and/or rows from the image. Partitioning refers to decomposing an image into blocks and storing or reading them block by block rather than as a whole image. Pruning refers to the removal of unnecessary data from the image. Quantization refers to reducing the magnitude of the values in image data so that they can be encoded with a smaller number of bits.

These four primitives govern all canonical storage formats. JPEG in its default settings, for example, samples images at every other row and column, partitions them into 8x8 blocks, and uses a specific 8x8 quantization matrix to quantize every data value within the block with a specific quantization factor. Learned JPEG, on the other hand, in addition to performing the same sampling, partitioning, and quantization as standard JPEG, also prunes unuseful data based on a learned model defining which data items are useful and which are not. Video storage formats follow image storage formats, except that they use sophisticated algorithms for exploiting the temporal relationship within the video data. They, too, use similar primitives, such as partitioning, sampling, and quantization when storing data.

Frequency-Domain Representation. There is one design primitive that all standard formats make the same decision: image representation. Standard formats use a human-recognizable color space representation of images, such as the red-green-blue (RGB) representation. Recently, several studies have shown that AI problems can be solved equally successfully by using frequency-domain representation [16, 23]. This representation enables fast image reconstruction, thereby significantly reducing inference time. Image Calculator follows this trend of frequency-domain representation, aiming for high efficiency in AI inference and training. When reading an image, it simply converts the byte stream into a set of frequency coefficients in a single step and uses it as is to train AI models and perform inference with them. This significantly reduces image reconstruction time and also enables scalable image representation, thanks to efficient data pruning, as described later in this section.

Domains for Design Primitives. Design primitives define the fundamentals when storing images. They are, however, useful only when we know how to perform each design primitive. A domain defines the set of possible ways to perform a specific design primitive. Every design primitive has its domain. Every combination of every domain value across all design primitives defines a different storage format, and all such combinations collectively define the total design space. There are, however, exponentially many ways to perform each design primitive. An RGB image, for example, can be sampled in an exponentially large number of ways. Similarly, we can partition the image into blocks of any size and prune them in an exponentially large number of ways. Consider a block size of 8x8, i.e., images are partitioned into 8x8 blocks when storing, and assume that we apply the same pruning strategy to all blocks of the image. How many possible pruning strategies are there for, say, six different block sizes: $8x8$, $16x16$, $32x32$, ..., $256x256$? For $8x8$ block size, there are $2^{8x8} = 2^{64}$ possible pruning strategies. For $16x16$ blocks, there are $2^{16x16} = 2^{256}$ strategies. In total, this makes $2^{8x8} + 2^{16x16} + ... + 2^{256x256} > 10^{150K}$, which is beyond any computational capacity to search within.

Dimensionality Reduction. We perform sensitivity analysis for each design primitive to determine a feasible domain. Our goal is to reach conclusions for each design primitive such that certain values

of their domains perform significantly better than others. This way, we aim to determine the most valuable set of design decisions for each primitive and create a feasible yet high-quality design space. When performing the sensitivity analysis, we fix values of all design primitives except the primitive under test. We then vary the values of the primitive under test from a very small value to a very large value and analyze how various AI problems, using different datasets and AI models, perform across different ranges of values. If we identify a group of domain values that are worse than others, we exclude them from the domain. If all domain values perform similarly, we uniformly sample a feasible set of domain values. This way, we identify the most valuable design decisions for each design primitive and create a high-quality design space.

#1 Sampling. Sampling primitive removes some rows and columns of pixels from the data. It has two aspects: which sampling strategy to use and which channels to sample. For the first aspect, we use four sampling strategies: sampling every other column, sampling every other row, sampling every other row and column, and no sampling. We observe that these strategies effectively cover the range. For the second aspect, we perform the following analysis. Once they are captured, images are typically stored in the red-green-blue (RGB) color space. Most standard formats transform them into Y-Cr-Cb color space before storing them. The reason is that Y-Cr-Cb color space separates brightness information from the color information. Y channel carries the brightness information, i.e., it is the black-and-white version of the original image. Cr and Cb channels carry the color information. Standard formats observe that the human eye is more sensitive to brightness information than to color information. Hence, they use a more aggressive sampling strategy for the color channels than for the brightness channel. We test this hypothesis against AI problems and observe that it does not hold for AI problems. Color information can be equally successful as brightness information for performing various AI tasks. Hence, once the sampling strategy is chosen, we sample all channels using the same sampling strategy.

#2 Partitioning. Partitioning primitive decides how to decompose images into blocks. We analyzed six block sizes of $8x8$, $16x16$, ..., and $256x256$. Each block size provides a different representation with varying accuracy offerings for different AI tasks. We observed that no block size is better than the others. Hence, we keep all six block sizes and use them as the domain of the partitioning primitive.

#3 Pruning. Pruning primitive removes some data values from the image. It is similar in definition to sampling but differs in its implementation. Most standard formats store data in the so-called frequency domain. Once an image is captured, it is usually in a color space, such as RGB. When storing images, most standard formats perform a frequency transformation, such as the Discrete Cosine Transformation (DCT) [10], which converts color-space representation into a frequency-space representation. This transformation is on top of the color-space transformation mentioned in the previous paragraph. Frequency-space representation allows reasoning about the data. Every value is a weight, i.e., a coefficient for a specific signal with a specific frequency. High-frequency signals represent fine-grained details on the image, whereas low-frequency signals represent coarse-grained information, such as the shape of a human body. Standard formats profile the human eye and observe that it is more sensitive to low-frequency signals than to high-frequency signals. Hence, standard formats compress high-frequency signals more aggressively than low-frequency signals, providing efficient storage.

Low-Frequency Coefficients Are Much More Useful Than High-Frequency Coefficients. We use the frequency structure in image data to design the pruning domain. We test four pruning strategies. As an image is a two-dimensional data, i.e., a two-dimensional signal, frequency transformation includes signals whose frequencies increase vertically and horizontally. In our first pruning strategy, we always retain the lowest horizontal and vertical frequency signals and prune everything else. As we prune less and less, we add higher and higher frequency signals, both in the horizontal and vertical dimensions. In our second pruning strategy, we always retain the highest frequency signals in both horizontal and vertical directions. As we prune less and less, we add lower and lower frequency signals both at horizontal and vertical dimensions. In our third and fourth pruning strategies, we retain the highest horizontal

and lowest vertical frequency signals, as well as the highest vertical and lowest horizontal frequency signals. Again, we add more and more signals in the opposite direction as we prune less and less. The third and fourth strategies test whether horizontal or vertical frequencies play a different role in image AI problems. We compare each strategy based on their performance using various AI problems. We observe that low-frequency signals have a clear superiority over all other types of signals. Hence, when pruning, we first prune the highest frequency coefficients at both horizontal and vertical dimensions and prune lower and lower frequency signals as we prune more. This constitutes the domain of the pruning primitive. For a given image, there is now a strict order and a small number of possible pruning strategies in terms of what signals should be pruned and what signals should be retained.

Removing Frequency Coefficients Allows Scalable Representation of Images. Eliminating the unuseful frequency signals has two advantages. First, it allows for significantly reducing the data size, as we physically remove some values from the dataset. Secondly, it provides a scalable representation of images in the main memory. We eliminate unuseful frequency signals outside the boundaries of the chosen triangle, as shown in Figure 3. As a result, the spatial dimensions of the image are reduced. For example, the image size in Figure 3 is reduced from 256x256 to 24x24 after we remove the frequency signals outside the chosen triangle. This reduction in terms of the spatial dimensions of the image allows for reducing disk I/O, decoding, PCIe times, and also GPU time.

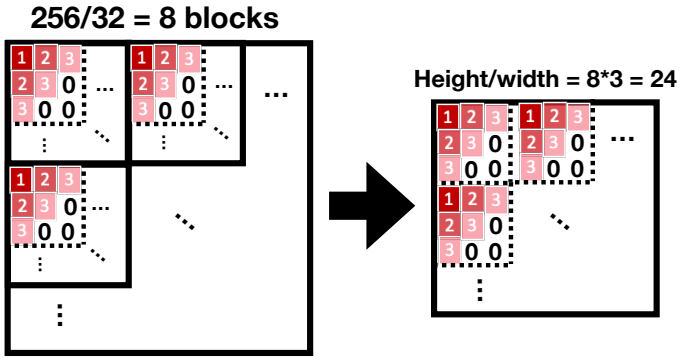


Figure 3: Image Calculator removes frequency coefficients outside the boundary of the chosen triangle.

#4 Quantization. Quantizing a value refers to dividing it with a quantization factor and rounding it to its nearest integer. This reduces the magnitude of the values, allowing them to be encoded with a smaller number of bits. Quantization is essential for reducing the consumed space, though it does not affect PCIe and GPU times and only minimally affects disk I/O and CPU decoding times. Even a small degree of quantization, e.g., quantizing with a small factor of 2, dramatically reduces the amount of data stored. This is because every value becomes smaller and hence needs a smaller number of bits to encode. Furthermore, as the values decrease, the number of repetitions across all values within the dataset increases. Modern encoding algorithms such as DEFLATE [11] exploit repetitions in the data and use a symbol table, where the most frequently occurring data items are encoded with the smallest number of bits. As values are quantized more and more, they get closer to each other and, hence, benefit more and more from repetitions. The trade-off is to quantize values as much as possible without losing much accuracy. In our experiments, we observed that quantizing values up to a factor of 20 typically does not result in any accuracy loss. Quantizing with factors larger than 20 results in an accuracy loss, and typically in proportion to the quantization factor. Hence, we choose 20 as the minimal quantization factor and select 50 and 100 for medium and high degrees of quantization, respectively.

Desing Space: ~4K Storage Formats. Having defined the domains, we can now count the total number of elements in our design space. There are three quantization factors, four sampling strategies, six block sizes, and as many pruning strategies as the dimension of each block size. This variation in total sums up to 4104 storage formats, which is large yet feasible enough to search within.

3 Inference Time

Having defined the design space, Image Calculator first reduces inference time by adapting storage to the AI task. It assumes the AI model is part of the AI task and, hence, user-specified. It aims to find the best-performing storage formats across various inference time budgets, allowing users to decide which storage format works best for their own use case. Image Calculator uses **performance models** to achieve this. A performance model is a predictive model that maps every element in the design space to a metric of interest, such as inference time, space consumption, or accuracy. Building a performance model for inference time and space consumption is straightforward, as hardware is easily accessible today through various cloud vendors, and it is inexpensive to profile hardware and obtain real-life measurements. Building a performance model for accuracy is a challenging task. The brute-force method would train an AI model for each storage format in the design space and offer an accuracy-cost trade-off across ∼4K candidates. This, however, means thousands of AI model training and is prohibitively expensive.

Bucket Sampling Reduces Accuracy Model Construction Time by 3x. Image Calculator uses sampling, interpolation, and transfer-learning-based methods to construct a performance model for the accuracy metric efficiently. We first observe a diminishing return relationship between inference time and accuracy. As storage formats become increasingly costly in terms of inference time, they also deliver higher accuracy. This is because high inference time implies expensive storage formats, which are characterized by high information density and, therefore, deliver high accuracy. Image Calculator exploits the diminishing returns curve and performs bucket sampling over the design space of storage formats. It first partitions the design space into a set of buckets, where each bucket contains storage formats with similar inference times. It then performs bucket sampling such that the sampled buckets define the overall shape of the trade-off curve reasonably well. It then trains an AI model specifically for the storage formats included in the sampled buckets. We show that this reduces the number of storage formats for which Image Calculator requires training an AI model from scratch from ∼4K to ∼1.3K, i.e., provides about 1/3 of savings.

Transfer Learning Further Reduces Accuracy Model Construction Time by 10x. We observe that storage formats in Image Calculator's design space are correlated. Consider the storage format that uses 16x16 blocks, prunes all frequency signals except the lowest four, and employs no sampling, with a quantization factor of 20. This storage format stores the data only slightly differently than the storage format that uses the same block size, sampling strategy, and quantization factor but prunes all frequency signals except the lowest five. Therefore, an AI model trained with a specific storage format could share its learned model parameters, i.e., weights, with other storage formats to accelerate their learning. In deep learning literature, this is referred to as transfer learning. Image Calculator trains only one AI model from scratch using the storage format that includes the highest amount of information, i.e., the format that is least pruned, sampled, and quantized. It then shares its weights with all other sampled storage formats. We analyzed four transfer learning strategies: (i) successively going backward from the most information-dense format to the least information-dense format in a sequential manner, (ii) going forward from the least information-dense format to the highest information-dense format in a sequential manner, (iii) parallel transfer learning where all formats learn from the highest information-dense format, and (iv) parallel transfer learning where all formats learn from the lowest information-dense format. We observe that the first transfer learning strategy, which involves successively transitioning from the most information-dense format to the least information-dense format in a sequential manner, yields the highest performance. Hence, we chose this as the Image Calculator's transfer learning algorithm. We show that transfer learning further reduces the accuracy model construction cost by 10x, as every storage format now needs ∼10x less number of epochs to train its AI model for achieving a similar level of accuracy.

Interactive Exploration of Trade-Off. Overall, bucket-sampling, interpolation, and transfer learning bring about ∼30x reduction in computing a performance model for accuracy, compared to the brute-force method of training ∼4K AI models. Once performance models are built, the user can interactively explore the design space and its various trade-offs, making an informed decision on which storage format to use and what accuracy to expect at runtime. Furthermore, all applications using the same dataset but different resource budgets can use the same accuracy-time/space models to make their decisions. As AI applications are typically deployed at a scale of millions across various types of hardware and software platforms and capacities, performance models offer efficient and scalable means for determining the right storage format for each application. We demonstrate that Image Calculator's storage formats can reduce inference time by up to 14.2x and consume space by up to 8.2x with little to no loss in accuracy compared to JPEG and its recent variants across diverse datasets, AI models, and AI tasks [12].

4 Training Time

Next, the Image Calculator aims to reduce training time. Training time encompasses storage format search time, AI model search time, specifically neural architectural search time, and hyperparameter optimization time. There are two challenges in reducing the training time. The first one is reducing storage format, architectural, and hyper-parameter search times. The second one is performing data augmentation in the frequency domain.

Reducing Storage Format Search Time via Unsupervised Learning. Image Calculator reduces storage format search time by using unsupervised learning. It exploits locality among storage formats in its design space. Closely related storage formats also exhibit similar performance. Image Calculator trains an AI model that creates one embedding for each storage format. It then starts with the densest storage format and traces lower-budget storage formats that are closest to it by keeping at least one storage format per inference time budget. It stops when it hits the most sparse storage format. Training an AI model that computes embeddings requires a single AI model training. Finding the densest storage format is a constant-time operation, as we know which storage format is densest in Image Calculator's design space, which is usually among the top-performing storage formats. Tracking back to lower-budget formats that are close to the densest format is also inexpensive, as it involves simply computing cosine similarity across a range of storage formats. As a result, finding high-quality storage formats using unsupervised learning is highly efficient, costing roughly the same as training a single AI model and allowing for scalable storage format search times. We demonstrate that unsupervised learning can identify storage formats with up to 10% higher accuracy than randomly chosen storage formats. Furthermore, they have only 3-5% lower accuracy than the optimal storage formats, which would take hundreds of AI model training using transfer learning, as described in the previous section.

Scaling Neural Architecture Search and Hyper-Parameter Optimization Time via Cheap Storage Formats. Having found a family of high-quality storage formats, Image Calculator performs neural architecture search and hyper-parameter optimization with one of these formats. The user provides two performance budgets: training time budget and inference time budget. The Image Calculator performs architectural and hyper-parameter searches using a storage format that fits within the inference budget, as much as the training budget allows. To illustrate, if the user has an inference budget of 100 microseconds, the Image Calculator determines which storage format, among those produced by its unsupervised learning algorithm, fits within the 100-microsecond budget. It then performs the architectural and hyper-parameter search with that format. This way, it finds the optimal architecture and hyper-parameter for that specific storage format, which allows searching for small training budgets that would otherwise not be possible and/or yield very poor qualities. Image Calculator can work with any neural architecture search algorithm, such as weight-sharing [7], neural predictor [24], and

zero-cost [25] algorithms, and any hyper-parameter optimization algorithm, such as sequential [26], and synchronous [27] and asynchronous [28] parallel methods.

Data Augmentation in Frequency Domain. Most neural architecture search studies perform repeated model evaluations, i.e., they train different candidate neural architectures at least partially to evaluate how high/low quality they are. Training an AI model requires data augmentation, such as randomly cropping and resizing different parts of the image and horizontally or vertically flipping the images. Data augmentation today is performed in the visual RGB domain. The Image Calculator uses data augmentation algorithms in the frequency domain. This enables the elimination of the costly image reconstruction phase during training, significantly improving individual training times. It employs existing digital signal processing algorithms for basic image manipulations in the frequency domain, such as block composition/decomposition and sub-band approximation, and implements nine popular data augmentations based on various combinations of these algorithms. This way, it can accelerate individual training time by up to three times.

5 Model Serving

Model-serving systems aim to provide fast and efficient inference [29–33] and training [34] at scale, where scale is defined by the number of applications concurrently submitting requests with varying latency/time and accuracy requirements. In its next step, Image Calculator scales inference and training times across a large number of applications. Model-serving systems use sophisticated algorithms to optimize resource efficiency, prediction latency, and accuracy jointly. The trade-off typically arises from using various types of AI models with differing cost-accuracy trade-offs. Existing studies all rely on a fixed storage, e.g., JPEG. Image Calculator replaces JPEG with its own design space of storage formats, thereby dramatically reducing all data movement costs. Additionally, the Image Calculator leverages the inherent frequency structure in image data to incrementally perform inference and efficiently share data during training. It breaks images into pieces, i.e., frequency components, and transfers, stores, or processes images frequency component by frequency component rather than file by file, as conventionally done.

Inference Serving: Adaptive AI Model and Storage Selection with Incremental Computation. When serving for inference, Image Calculator communicates with registered applications and asks only for the first set of frequency components for a batch of images. It then performs inference by using only these first frequency components for all images. For those images in which the Image Calculator reaches a threshold of confidence score, it finalizes the image AI task and returns the results. For those who do not, the Image Calculator requests additional frequency components and continues performing inference until it reaches a threshold of confidence score for all images in the batch. This way, it transfers only as much data as necessary for each image, which dramatically reduces data movement and processing costs when serving for inference. The Image Calculator performs inference using an ensemble of AI models. It uses an adaptive model and storage selection algorithm. Given a latency budget, it starts with a large ensemble of AI models over sparse storage formats. As more and more frequency components are transferred over the network, it reduces its ensemble size while increasing the density of the images with newly arriving frequency components. Its goal is to support sparse storage formats with large ensembles, allowing it to reach high confidence early and minimize data movement over the network. Image Calculator proactively prefetches the next frequency components while running the ensemble of AI models, overlapping AI model execution time with data transfer time to reduce end-to-end latency further.

Training Serving: Sharing Data Across Frequency Columns. When serving for training, Image Calculator uses a collaborative training algorithm for all AI models it maintains in its pool and their

variants to perform inference with varying numbers of frequency components. Like its inference-serving component, Image Calculator uses the inherent frequency structure within image data for efficient training. It stores images frequency-component by frequency-component instead of file by file. Every image has a fixed number of frequency components, and each frequency component is represented as an array of values. Storing images using the frequency structure is similar to storing a relational database table column by column rather than row by row. In this analogy, a relational table corresponds to a batch of images, a row corresponds to an individual image, and a column corresponds to a specific frequency component across the entire batch of images. In Image Calculator's pool, there are AI models that recognize an increasing number of frequency components. Storing images based on frequencies allows for efficient data sharing across AI models that use a similar set of frequency components. The Image Calculator co-locates such AI models and shares the I/O cost among them during training. This dramatically reduces the I/O cost, as otherwise, Image Calculator would need to create a separate file for every version of the image, even though they are part of the same storage, and would need to perform a separate I/O operation for each. We demonstrate that Image Calculator reduces I/O costs by up to 9x, thanks to its data sharing scheme over frequency columns [13].

6 Summary & Future Work

Image Calculator is a self-designing storage system that creates and manages storage tailored to a specific image AI task. It builds a design space of storage format, where every format offers a different accuracy-cost trade-off in terms of accuracy, inference/training time, and space consumption. It uses performance models and locality among storage formats to efficiently search for high-quality candidates. It breaks images into frequency components and stores, transfers, or processes them frequency by frequency, rather than file by file, for efficient serving of inference and training requests.

The AI stack has been heavily optimized, from the infrastructure layer, which uses massively parallel architectures with high memory bandwidth [35], to the framework layer, which employs libraries with efficient performance tuning methods, such as operator fusion and just-in-time compilation [36]. Storage has been the only component that is "general purpose", meaning it is not specialized for AI. Image Calculator challenges this convention and makes the case for an adaptive storage for image AI tasks. It opens the black box of image storage and sets an ambitious goal: to have AI storage for images that can replace JPEG.

In its next steps, we aim to expand Image Calculator's design space for better and cheaper formats, as well as for new domains and problems such as health [37], autonomous vehicles [38], 3D image reconstruction [39], and video games [40]. Given that neural architectures have design spaces that are orders of magnitudes larger than Image Calculator's design space (10^{21} vs. $4K$), the potential is huge. The Image Calculator's idea is simple and can be applied to other data types, such as video and text. For example, information retrieval systems use similar primitives to Image Calculator for efficient text processing, such as sampling [41], pruning [42], and quantization [43]. The Image Calculator enables the reimagining of the entire image AI pipeline in terms of how data is moved and processed, from bits to the system level. We plan to enhance Image Calculator to an end-to-end data system that holistically manages all data objects produced and consumed by an image AI pipeline, such as activation maps and gradients [44]. Image Calculator makes inference a viable option on CPUs, offering a new cost-performance trade-off. CPUs are equipped with a variety of micro-architectural features that database workloads have long been using [45–51]. We plan to study novel data systems that provide fast and scalable performance on CPUs by efficiently using micro-architectural resources.

References

[1] Chooch. How to use Computer Vision AI for Detecting Workplace Hazards? *https://www.chooch.com/blog/computer -vision-ai-safety-technology-to-detect-workplace-hazards*, 2023. Accessed on Nov 14, 2024.

[2] Alice Gomstyn and Alexandra Jonker. How to use Computer Vision AI for Detecting Workplace Hazards? *https://www.ibm.com/topics/smart-farming*, 2023. Accessed on Nov 14, 2024.

[3] Gang Yu et al. Accurate Recognition of Colorectal Cancer with Semi-supervised Deep Learning on Pathological Images. *Nature Communications*, 12 (6311), 2021.

[4] Matic Broz. How Many Pictures are There (2024): Statistics, Trends, and Forecasts. *https://photutorial.com/photos-statistics*, 2024. Accessed on July 31, 2024.

[5] Edge Delta Team. Breaking Down The Numbers: How Much Data Does The World Create Daily in 2024? *https:// edgedelta.com/company/blog/how-much-data-is-created-per-day*, 2024. Accessed on May 5, 2025.

[6] Emma Strubell et al. Energy and Policy Considerations for Deep Learning in NLP. *ACL*, 2019.

[7] Han Cai et al. Once-for-All: Train One Network and Specialize it for Efficient Deployment. *ICLR*, 2020.

[8] Forbes. Google Cloud Doubles Down On NVIDIA GPUs For Inference. *https://www.forbes.com/sites/moorinsights /2019/05/09/google-cloud-doubles-down-on-nvidia-gpus-for-inference*, 2019. Accessed on May 16, 2023.

[9] HPCwire. AWS to Offer Nvidia's T4 GPUs for AI Inferencing. *https://www.hpcwire.com/2019/03/19/aws-upgrades-its-gpu-backed-ai-inference-platform*, 2019. Accessed on May 16, 2023.

[10] Vasudev Bhaskaran and Konstantinos Konstantinides. Image and Video Compression Standards. *Springer*, 1995.

[11] Peter Deutsch. DEFLATE Compressed Data Format Specification version 1.3. *https://datatracker.ietf.org/doc/html/ rfc1951*, 1996. Accessed on December 19, 2024.

[12] Utku Sirin and Stratos Idreos The Image Calculator: 10x Faster Image-AI Inference by Replacing JPEG with Self-designing Storage Format. *SIGMOD*, 2024.

[13] Utku Sirin et al. Frequency-Store: Scaling Image AI by A Column-Store for Images. *CIDR*, 2025.

[14] Gregory K. Wallace. The JPEG Still Picture Compression Standard. *IEEE Transactions on Consumer Electronics*, 38 (1), 1992.

[15] JPEG. JPEG 2000. *https://jpeg.org/jpeg2000*, 2023. Accessed on Jan. 02, 2023.

[16] Kai Xu et al. Learning in the Frequency Domain. *CVPR*, 2020.

[17] Mark Adler et al. PNG Specification. *https://www.w3.org/TR/2003/REC-PNG-20031110*, 2003. Accessed on December 19, 2024.

[18] DigicamSoft. *https://www.digicamsoft.com/bmp/bmp.html*. Accessed on May 5, 2025.

[19] Google for Developers. An Image Format for the Web. *https://developers.google.com/speed/webp*. Accessed on May 5, 2025.

[20] HEIF Technical Information. *https://nokiatech.github.io/heif/technical.html*. Accessed on May 5, 2025.

[21] Gregory K. Wallace. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13 (7), 2003.

[22] Gary J. Sullivan et al. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22 (12), 2012.

[23] Lionel Gueguen et al. Faster Neural Networks Straight from JPEG. *NeurIPS*, 2018.

[24] Colin White et al. How Powerful are Performance Predictors in Neural Architecture Search? *NeurIPS*, 2021.
[25] Mohamed S. Abdelfattah et al. Zero-cost Proxies for Lightweight NAS. *ICLR*, 2021.
[26] Stefan Falkner et al. Robust and Efficient Hyperparameter Optimization at Scale. *ICLR*, 2018.
[27] Stefan Falkner et al. A Generalized Framework for Population based Training. *KDD*, 2019.
[28] Liam Li et al. A System for Massively Parallel Hyperparameter Tuning. *MLSys*, 2020.
[29] Daniel Crankshaw et al. Clipper: A Low-Latency Online Prediction Serving System. *NSDI*, 2017.
[30] Francisco Romero et al. INFaaS: Automated Model-less Inference Serving. *USENIX ATC*, 2021.
[31] Jashwant Raj et al. Cocktail: A Multidimensional Optimization for Model Serving in Cloud. *NSDI*, 2022.
[32] Ferdi Kossmann et al. CascadeServe: Unlocking Model Cascades for Inference Serving. *arXiv/2406.14424*, 2024.
[33] Alind Khare et al. SuperServe: Fine-Grained Inference Serving for Unpredictable Workloads. *NSDI*, 2025.
[34] Wei Wang et al. Rafiki: Machine Learning as an Analytics Service System. *VLDB*, 2018.
[35] NVIDIA. H100 Tensor Core GPU. *https://www.nvidia.com/en-us/data-center/h100*. Accessed on May 5, 2025.
[36] William Wen. Introduction to Torch.compile. *https://pytorch.org/tutorials/intermediate/torch_compile_tutorial.htm* 2023. Accessed on May 5, 2025.
[37] Google AI. Health AI. *https://ai.google/applied-ai/health*. Accessed on May 5, 2025.
[38] Autonomous Vehicles Factsheet. *Center for Sustainable Systems, University of Michigan*, Pub. No. CSS16-18, 2024.
[39] Rao Fu et al. GigaHands: A Massive Annotated Dataset of Bimanual Hand Activities. *CVPR*, 2025.
[40] Hongchi Xia et al. Video2Game: Real-time, Interactive, Realistic and Browser-Compatible Environment from a Single Video. *CVPR*, 2024.
[41] Zhen Yang et al. TriSampler: A Better Negative Sampling Principle for Dense Retrieval. *AAAI*, 2024.
[42] Yueqian Lin et al. SpeechPrune: Context-aware Token Pruning for Speech Information Retrieval. *ICME*, 2025.
[43] James O' Neill and Sourav Dutta. Improved Vector Quantization For Dense Retrieval with Contrastive Distillation. *SIGIR*, 2023.
[44] Utku Sirin and Stratos Idreos. Data Storage and Management for Image AI Pipelines. *SIGMOD*, 2025.
[45] Utku Sirin et al. Micro-architectural Analysis of In-memory OLTP. *SIGMOD*, 2016.
[46] Utku Sirin et al. OLTP On A Server-grade ARM: Power, Throughput and Latency Comparison. *Damon*, 2016.
[47] Utku Sirin et al. A Methodology for OLTP Micro-architectural Analysis. *Damon*, 2017.
[48] Utku Sirin and Anastasia Ailamaki. Micro-architectural Analysis of OLAP: Limitations and Opportunities. *VLDB*, 2020.
[49] Utku Sirin et al. Performance Characterization of HTAP Workloads. *ICDE*, 2021.
[50] Utku Sirin et al. Micro-architectural Analysis of In-memory OLTP: Revisited. *The VLDB Journal*, 2021.
[51] Utku Sirin. Micro-architectural Analysis of Database Workloads. *PhD Thesis. EPFL*. 2021.

Towards AI-Enabled Data-to-Insights Systems

Gerardo Vitagliano[1], Jun Chen[2], Peter Baile Chen[1], Ferdi Kossmann[1],
Eugenie Lai[1], Chunwei Liu[1], Matthew Russo[1], Sivaprasad Sudhir[1], Anna Zeng[1],
Ziyu Zhang[1], Michael J. Cafarella[1], Tim Kraska[1], Sam Madden[1]
[1]MIT, USA, [2] Independent
{gerarvit, peterbc, ferdi.kossmann, eylai, chunwei, mdrusso,
siva, annazeng, sylziyuz, michjc, kraska, madden}@csail.mit.edu,
chjuncn@gmail.com

Abstract

Modern data systems are more and more capable and efficient at processing large volumes of data, both unstructured and structured. However, an open challenge is to build a comprehensive automated system for end-to-end data science, from data discovery and exploration to visualization and statistical modeling. We introduce the notion of an "AI-enabled data-to-insights" system, and describe an architectural framework comprised of user-agent interactions, AI-powered automation, execution optimization, and specialized storage and infrastructure. For each of these layers in the framework, we outline the current challenges and we outline our vision of different system components and tools towards building an integrated data-to-insights system. Finally, we present an overview on our benchmarking efforts and preliminary results that motivate further research to guide the design of novel architectures to process data intensive and multimodal workloads.

1 Introduction

Data-powered AI systems have revolutionized the interaction between humans and information and drastically accelerated scientific research, business intelligence, and data-powered analysis. Systems based on Large Language Models (LLMs) and Foundation Models (FMs) have automated much of the engineering of applications and data analysis pipelines. These tools can perform sophisticated processing of large collections of structured and unstructured data, including tabular data, text, images, and video. Until recently, data science applications relied on extensive and brittle hand-engineering and human expertise to design, implement, deploy, and debug. As a motivating example, consider the following data science pipeline detailed in Figure 1, inspired from real experiments performed by the scientists in [15].

In this example, a biomedical researcher is investigating "small cell lung cancer", a particular type of lung cancer which is characterized by a high percentage of patients relapsing after receiving treatment. The researchers set up a clinical study to analyze the genome evolution of tumor cells in patients before, during, and after treatment. During the course of the study, biological samples are obtained by hospitalized patients, together with medical reports handwritten by hospital doctors; structured data from laboratory examinations, e.g., blood sampling; imaging data from MRI scans and cell histology; and genomics data from the biological samples.

Some of the key data-driven steps involved in the scientific research are:

1. Classify biological samples corresponding to small cell lung cancer, based on imaging and genomic data.

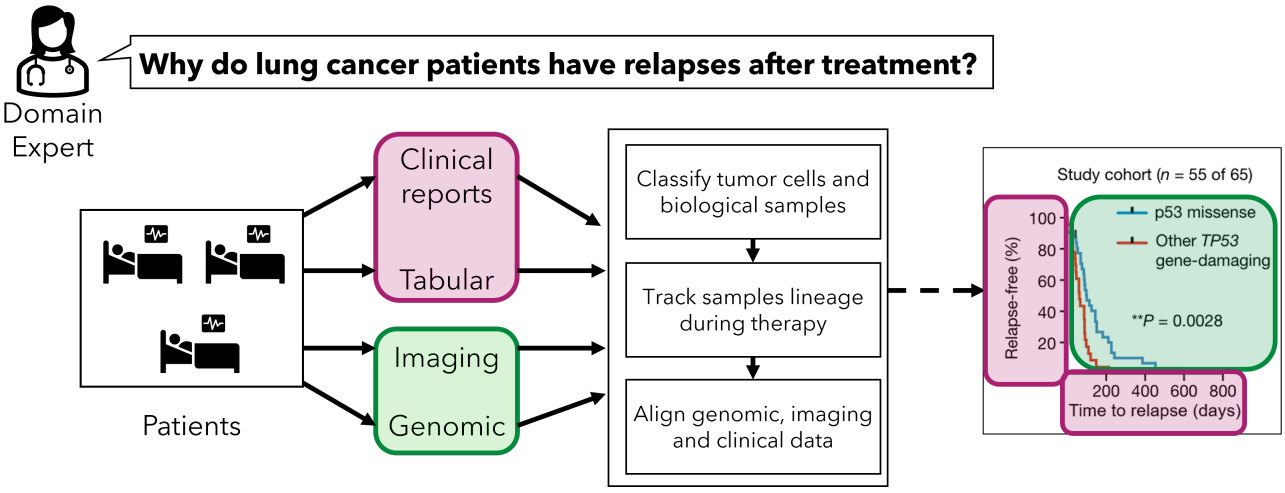


Figure 1: An example data-to-insight pipeline inspired from [15]. To answer and obtain insights about a clinical research question, biomedical researchers are required to manually integrate and process multimodal data.

2. Track samples collected during therapy; identify their lineage and evolution in individual tumor locations.

3. Align data obtained through different sources, including MRIs, blood samples, and clinical data to identify interesting patterns associated with relapse probability.

Through a complex data-driven pipeline, involving both significant computation and human effort to label and clean data, the authors of the study hypothesize that relapse probability and occurrence are highly correlated with a specific gene mutation (Figure 1, right plot extracted from [15]). Although the authors of [15] carried most of the steps in the pipeline through manual labor and domain expertise, we envision that AI technology will soon enable fully automated and/or low-code workflows.

Fulfilling this vision with current technology requires a full stack of data/AI processing systems and tools. In Figure 2, we detail three abstraction levels of such a mature data-oriented AI stack and some of the individual components we are building towards this vision:

- **Closed-Loop Human/Agent Interfaces**: current AI systems are heavily focused on textual input/outputs and single users. We envision data-driven applications that can leverage LLM capabilities beyond purely chat-based interfaces, including tools to formally design and execute agentic pipelines, and support a collaborative end-to-end data science process. Furthermore, a mature data-to-insight system must feature automated AI components as first-class citizens. For data science applications, we expect intelligent agents to be in charge of fully automating end-to-end pipeline building as well as integrating multimodal data without the need of extensive human labeling or transformations.

- **Execution Optimization**: complex systems that include LLM processing capabilities presents novel architectural challenges. Often, their performances are hindered by processing costs and latency, as each inference operation requires expensive memory and computation. We envision an integrated system that includes several optimizations to execute AI workloads including dynamic model routing and cost-based optimization of semantic operators.

- **Metadata-Aware Storage and Retrieval**: data-to-insight pipelines often process and generate large amounts of data. Systems must include specialized solutions to handle infrastructural and

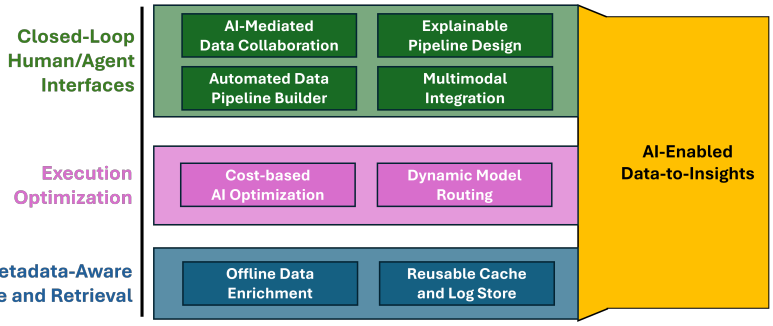


Figure 2: An overview of the components we are building towards an AI-enabled data-to-insight system.

storage challenges optimized vector storage and algorithms for retrieve-augmented generation (RAG), components to manage cache and user interaction logs to improve pipeline efficiency and quality over time, and interfaces to exchange data with external systems.

In the remainder of this paper, we outline some of the research opportunities and concrete steps we are undertaking to build such an AI-enabled data-to-insight stack and address the challenges of current data-oriented AI systems. In Section 2, we discuss our vision for the architecture of data-to-insight systems and detail the features that we deem fundamental to implement in such systems. In Section 3, we describe the human/agent interface layer for pipeline design, including a specialized user-agent interface, an automated system for explainable pipeline building, and the challenges multimodal integration. In Section 4, we highlight the challenges of optimizing the execution of complex LLM-powered pipelines, and outline a proposal for a cost-based optimizer and a model serving system. In Section 5, we introduce metadata-aware storage and retrieval to empower reasoning in data-to-insights systems, including a join-aware retrieval framework, an offline data enrichment framework, and a reusable cache and log store component. In Section 6, we discuss the challenges of benchmarking data-to-insight systems and present a novel benchmark with initial experiments that motivate the need for further development. Finally, we conclude the paper with an outlook on the open challenges in Section 7.

2 Data-to-Insights Systems

The production and consumption of data-driven insights is often a collaborative process involving multiple human roles and a diverse technology stack. *Contributors*, e.g., data engineers, have the technical expertise to build data-to-insight pipelines; *authors*, e.g., data analysts, use these pipelines to produce data artifacts; and *consumers*, e.g., business stakeholders, have the domain expertise to make decisions based on the insights derived from the artifacts. All parties possess specialized knowledge but require collaboration to leverage the unique expertise of all actors involved. Without seamless collaboration, changes to software pipelines, data artifacts, or insights can lead to costly coordination and time-consuming feedback loops. Our vision for mature AI-assisted data-to-insight systems includes specialized components to address these socio-technical challenges, including assisted development systems, user-agent interfaces, and specialized storage and retrieval components.

Contributors conventionally assemble data pipelines using diverse tools: integrated and interactive programming environments, i.e., Jupyter notebooks or IDEs, GUI-assisted tools, i.e., spreadsheets or dashboard builders, and ad-hoc scripts in a variety of languages, i.e., bash or SQL. While each step in the pipeline may be individually inspected and debugged, the overall end-to-end data flow cannot be easily revised or optimized holistically.

Existing code generation agents [2, 4] that leverage LLMs or automated systems can save contributors time and enable quicker design and development. However, when the generated code does not work as intended out-of-the-box, users who don't understand the code struggle to debug its subtle issues or provide high-level feedback to coding agents [23, 33, 38]. This highlights the need for intermediate computational results and human-in-the-loop feedback in a tight development loop to debug artifacts produced by automatically generated pipelines. Consequently, the productivity wins offered by traditional coding assistant agents do not necessarily entail quicker insights, particularly for complex data pipelines where contributors, authors, and consumers have to collaborate to discover new data insights. To address these inefficiencies, we envision a system that supports AI-mediated collaboration across the life cycle of data artifacts. We aim to make the production of data artifacts substantially cheaper and faster by dramatically reducing the coordination overhead between human and AI agent actors involved, thus improving the productivity of each participant. Rather than eliminating human collaboration, we aim to augment it through intelligent tooling and tighter feedback loops.

A full-fledged system that realizes this vision should include a few crucial features:

Closed-loop human/agent interfaces: The system should cover the end-to-end life cycle of data artifacts. The system should include an AI agent that constantly reads and revises the data artifact to detect missing, inconsistent, or obsolete information. When possible, it will take action to fix the artifact, such as updating a pipeline, searching a data lake for a piece of missing information, or answering a collaborator's question directly without bothering the responsible author. In other cases, it should warn end users about potential problems. Such a "closed loop" will allow feedback from the consumer of a data artifact to be communicated upstream to the author, and thereby exploited by both humans and AI programming tools. As a result, domain expert consumers can give direct artifact feedback and in many cases see a refined artifact before the author does any work at all. Further, this closed loop can facilitate the connection between the author and contributors of a data artifact, as the contributor might add query results or crucial facts that become part of the overall data artifact.

Execution Optimization: Considering the costs and latency associated to the execution of AI operators, a mature system needs a dedicated and optimized execution layer. Within this layer, calls to LLMs and FMs are considered physical operators, and optimization proceeds through the exploration of many equivalent plans to meet a user-defined mix of cost, latency, and quality. Example optimizations include (i) leveraging a spectrum of models with different cost-accuracy tradeoffs, (ii) partitioning, caching, or summarizing inputs to reduce the effect of context-length costs, or (iii) batching requests on the fly to ride out bursty demands. Because the quality of LLM outputs is stochastic and hard to predict, a key challenge of execution optimization is estimating the cost/quality curves online and identifying a Pareto frontier of physical operators.

Metadata-aware storage and retrieval: The social process of creating data artifacts produces a large amount of data that is currently not leveraged in conventional data systems. Unlike most of today's data pipelines, lineage of data artifacts must be explicitly stored and retrieved, allowing versioning, iterative refinement, and reuse. The system should leverage these precious metadata as part of the artifact production process. Examples include information about customer state, the responsibilities of individual contributors, the connection string for particular databases, and verbal claims made by collaborating institutions.

In the remainder of the paper, we detail some of the prototype components of this architecture that we are actively developing towards a full-fledged AI-enabled data-to-insights system.

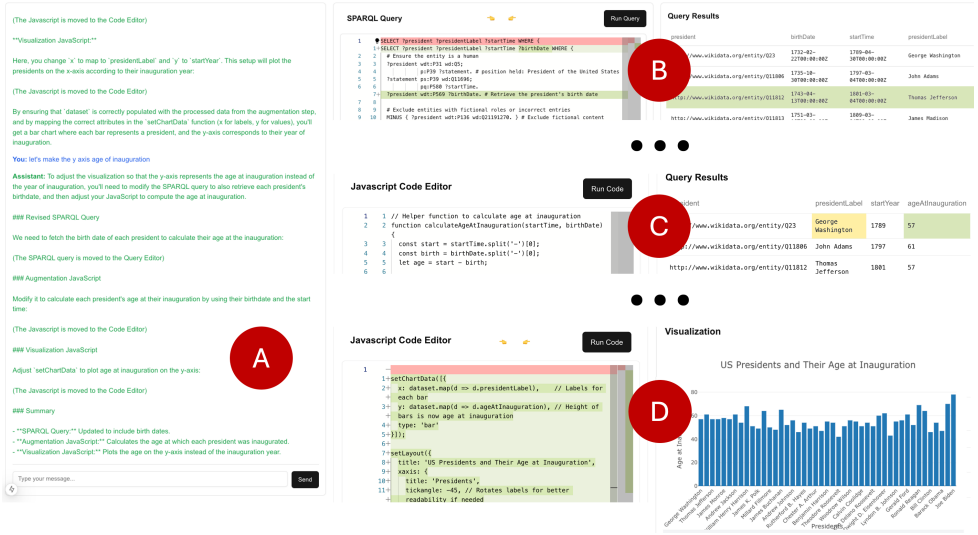


Figure 3: An early prototype of the Sunroom system, with a chat box (A) on the left hand side, and three code and output panels on the right hand side illustrating the pipeline: (B) runs a SPARQL query on a knowledge graph and returns a table; (C) cleans and prepares the data in Javascript, returning a modified table; and (D) uses a Javascript plotting library to visualize a bar chart. The highlighted content in the code and output panels represent recent code changes by the agent and output annotations left by the user, illustrating how the system enables human-agent feedback loops both on code and data artifacts.

3 Human/Agent Interfaces for Pipeline Design

In this section, we describe the main components we envision in a closed-loop framework for automated pipeline design and deployment. We introduce Sunroom, a prototype user interface for integrated pipeline design and debug; we describe the design space for automated, agentic pipeline design; and we outline the challenges and opportunities of multimodal data within data-to-insight systems.

Sunroom is a data pipeline builder that combines a comprehensive chat experience with a shared, live dataflow execution environment. Irrespective of the user being a data engineer, analyst, or consumer, our system can streamline iterative data artifact development with the assistance of an AI expediter. Sunroom incorporates several essential components:

- A user interface that enables the user to engage with the agent, pipeline source code, and data artifacts
- An agent interface to facilitate interactions with a coding oracle
- A workflow repository with source code and configuration files
- An execution result data store
- A task orchestrator with associated execution workers

In traditional data orchestration systems, the user has full, direct control to coordinate pipeline execution, but there is little room for an agent to assist the user in determining their data orchestration tasks; the source code and data repositories are available only for the benefit of the orchestrator, workers, and broader execution environment. Conversely, in AI-IDE systems, an automated coding agent takes the place of the user in having full, direct control; the user is only able to instruct the agent to make code changes, but the lack of intermediate result visibility hampers their ability to evaluate pipeline

correctness (as in these domains, correctness is determined more by the transformations applied on the data and consequential data output than the specific program behavior). Sunroom addresses these challenges by providing both the user and agent with full visibility into the source code, execution results, and orchestration of workflow tasks, allowing both humans and AI agents to provide feedback and input to any stage of the processing and output workflow. This is a notable departure from the code-generation features of existing workflow engines that employ LLM-driven features [1, 3, 41], as LLM-driven code changes are scoped to one pipeline step at a time and are not data-aware unless the user manually adds the context into the conversation (at time of publication).

While the five aforementioned components are essential for a Sunroom-like system, the primary challenge lies in representing the environment such that both humans and agents are able to digest critical information and take action in near real-time. For example, when working at scale or in production, debugging ETL pipeline issues would be served well by an application like Sunroom, but tables too large to fit in memory are also too large to fit in context for RAG queries; however, giving value-level feedback for agentic revision can be a challenge when tables are conventionally compressed. In larger-scale scenarios, approaches to execution optimization as in Section 4 stand to play a major role in realizing this type of application.

We envision the evolution of human-agent interfaces to support more sophisticated and adaptive execution behavior: as these systems accumulate historical execution data through continuous interaction, an automated orchestrator could dynamically prompt users for appropriate feedback or run longer sequences of the pipeline in full autonomy.

3.1 Explainable Pipeline Design

LLMs have demonstrated impressive capabilities to automate code generation, structured data querying, and semantic understanding [14, 32, 45]. While they can assist with coding and some semantic reasoning, they often lack deep contextual awareness of a given project's nuances. Moreover, data workflows are often iterative and non-deterministic, requiring constant adaptation based on intermediate results and human intuition. To address this gap between LLM capabilities and the needs of real-world data science we explore an agentic approach—one where LLM agents do not simply assist in isolated tasks but also help with orchestrating and optimizing entire workflows, intelligently coordinating different agents to construct, execute, and refine data pipelines dynamically.

In AI-assisted pipeline design, the fundamental "black box" nature of LLMs makes errors unpredictable and inconsistent, even across similar inputs. Hence, trust, explainability, and transparency of automatically produced pipelines remains a significant barrier, as users struggle to determine when model outputs can be accepted without verification. These challenges necessitate thoughtfully designed human-agent interfaces that:

- Support to build customized data processing pipelines through an intuitive, visual, drag-drop interface.
- Enable human feedback both at an individual operator level as well as at the broader pipeline level. The feedback should be stored and leveraged to improve the quality of subsequent executions of the pipelines.
- Annotate the output for individual steps of pipelines with confidence metrics to address accuracy concerns and provide greater insights into automatically generated results.
- Label operators with complexity and security requirements, for which users can provide test-cases which are automatically run during the execution of the pipeline.

We envision an agentic framework that fulfills the above features and enables data scientists to author pipelines at a higher-level of abstraction than before. An orchestrator agent will be in charge of understanding natural language directives, designing a pipeline of steps to reach the desired output, and finally invoking specialized agents to implement the pipeline steps. At the core of this agentic approach, we identify the following fundamental features:

Agent Primitives: A successful automated data-to-insight pipeline is predicated on a set of verified and trustworthy specialized agents. To design such agents, it is necessary to define the set of primitive operations that must be implemented by each individual operator. One of the challenges is that different pipelines on diverse datasets and domain may have unique features, e.g., geographical data may require spatial reasoning while biomedical genomics data may require sequence alignment.

Pipeline Orchestration: We define orchestration as the challenge of determining which agents to execute next at a given moment in time. Orchestration can be implemented statically (e.g. with pre-defined trigger rules for each agent) or in a more dynamic fashion (e.g. with a coordinator determining which agents to execute next). Effective orchestration must have three attributes: context management, to ensure that the correct agents are invoked when necessary; responsiveness and adaption to changes in the workload input(s) as well as its intermediate state; resilient execution handling, to be able to recover and correct run-time failure.

Self-Adaptation to Downstream Feedback: A real world data-to-insight workflow typically has an exploratory nature, with subsequent iterations of pipeline design and implementation depending intermediate outputs and insights obtained from available data. Therefore, a truly automated system must include a closed-loop framework, automating the feedback process. One of the challenges of building pipelines that can self-adjust to downstream outputs, is the diversity of formats for the data artifacts produced. For example, outputs of pipelines may be qualitative in nature (e.g., textual reports, or visual artifacts); quantitative data (e.g., a set of measures); or even statistical models (e.g., a regression model). Hence, to automatically parse useful downstream information, it is necessary to define a comprehensive and well-designed space of feedback signals.

3.2 Multimodal Integration

Foundational models can embed text, images, and other modalities in a shared vector space [34, 46]. As natural-language queries may also be encoded as vectors, multimodal question answering can be achieved using the Retrieval Augmented Generation (RAG) paradigm [27], by fetching the most similar documents to the query and having LLMs generate an answer grounded on evidence. However, this paradigm has several shortcomings for large and multimodal data inputs:

Offline preprocessing: As embedding models have usually limited contexts for input data, documents must be chunked (in sentences, lower resolution images, etc.) and/or possibly transpiled (i.e., non-textual inputs are described as text) at indexing time, before the question-answering stage. Query-agnostic chunking often degrades retrieval for complex reasoning, and query-aware chunking for multimodal data is still little studied.

Query-agnostic similarity: Vector search may returns data semantically similar to the query, but not necessarily the most fitting for the output of the query itself. Consider a multimodal query seeking for the brand of a dress in a photo, for instance, may yield images with similar poses instead of ones highlighting the garment. To overcome this gap, methods such as query rewriting [7], decomposition [39], or enriching structured data with graphs [18] are necessary.

Inter-modality gap: Verbalizing data in non-text modalities into a textual description is a popular approach for multimodal data processing. However, some aspects of multimodal data are difficult to capture with only transformer models and natural languages. For example, if two portraits have similar lighting designs, image models are better suited to reason about the connection than textual descriptions.

Towards this end, we propose multimodal embedding ensembles, a method of representing multimodal data entries with sets of embeddings from all applicable models. Multi-vector search primitives can subsequently help with the retrieval.

We have seen the unique features of human/agent interfaces for pipeline design. Implementing and using a full-fledged AI-enabled system is associated with extensive computation using large models. The next section details our vision on optimizing the execution of LLM-based operators to provide the scalability and efficiency required for complex data-to-insights pipelines.

4 Execution Optimization

The use of LLM-powered systems to automate the design of data pipelines and provide semantic capabilities to data operators [29, 31, 43, 48] often comes with scalability and cost concerns. In this section, we outline the challenges of executing large workloads of LLM-powered operators at the scale required for data-intensive processing pipelines such as data-to-insight pipelines. First, we will describe Abacus, our proposed framework to optimize the quality, cost, and latency of data pipelines using LLM operators. Then, we introduce Ken, a component to provide data-to-insight systems with fine-grained control over model serving, to expose more cost-accuracy-latency trade-offs while boosting inference throughput.

The execution cost of LLM-powered operators is dominated by three primary bottlenecks:

- **Cost/accuracy granularity:** A single "best" model exposes only one latency–quality operating point. A data system instead requires a continuum of trade-offs to provide cheaper or more effective executions.
- **Context length:** LLM computation scales quadratically with the number of input tokens. Large data inputs often exceed the context window current hardware can process efficiently [30].
- **Burstiness:** The volume of data processed in AI workloads is non-stationary but characterized by interactive spikes that may constrain model performances.

A mature data-to-insight system that leverages extensive AI-based operators must include an optimization layer to guarantee scalable and affordable execution for data pipelines. Inspired by relational operators [19], recent systems [22, 29, 40, 48] proposed declarative frameworks for semantic, LLM-based operators. The core design of these frameworks is to identify of logical and physical operators that implement a specific AI-based program (a sequence of semantic operations), and then to explore the space of equivalent implementations of such program as to optimize a given user requirement. The challenges of semantic optimization lie within the nature of LLM systems: due to their non-deterministic behavior and high processing cost, the execution of a plan may lead to varying degrees of output quality, economic cost, and latency. The Pareto-frontier of possible implementations is hard to explore because, unlike relational operators—semantic operator quality is uncertain, and we lack principled methods to quickly and cheaply estimate physical operators' performance.

4.1 Cost-based AI optimization

We propose Abacus [36], a general-purpose, extensible optimizer that can optimize the execution of LLM programs with respect to output quality, dollar cost, or latency. Abacus can optimize either along individual dimensions, e.g., having the best quality irrespective of the execution costs, or perform constrained optimization with respect to constraints on different dimensions of system performance, e.g. having the best possible output quality subject to an upper bound on cost.

The core intuition of Abacus is inspired by the Cascades query optimizer [17]: we implement rule-based transformations to define a space of physical plans. Each operator defined in a program can be executed with a set of equivalent physical implementations: for example, using LLM tools to perform a semantic filter out data records can be implemented using different models, or different prompting strategies. To address the aforementioned challenge of predicting the performances and cost of different physical implementations, Abacus models the problem of finding useful operators as an infinite-armed bandit problem [5, 6]. Given a sample execution budget (e.g., a fixed amount of LLM calls), Abacus samples the performance of equivalent physical implementations on a subset of the data inputs and builds a Pareto frontier of implementations. To accelerate the search process and increase the quality of the estimates, Abacus can leverage prior beliefs about operator performance, e.g., if larger models are known to be more expensive, it will avoid repeated samples to estimate their costs. Finally, in order to support constrained optimization objectives, we extend the traditional Cascades algorithm to keep track of the Pareto frontier of physical plans. For more details about the optimization and estimation processes of Abacus, we refer readers to [36], where we also provide experimental evidence that these algorithmic contributions help Abacus obtain consistent Pareto-optimal pipeline executions.

4.2 Dynamic Model Routing

A dedicated execution optimization layer for data-to-insight pipelines should not only entail declarative optimization: an orthogonal problem lies in the availability and provisioning of a broad and diverse set of model configurations to choose from. Without diverse model alternatives for serving LLM requests, the space of available physical optimizations is restricted.

Consider the example of having two models available for a given task, a tiny and a large model variant. The former may be too inaccurate, while the latter may incur unnecessary cost, limiting the optimizer ability to make effective trade-offs. Hence, the efficiency of these trade-offs also depends on how effectively the underlying system can serve these models. We argue that alongside declarative optimization, a dynamic and highly configurable model serving environment is essential to fully unlock AI-enabled data-to-insight systems.

For a given hardware provisioning, we can add points to the latency-accuracy trade-off curve by adopting *model cascades* [49] as an effective mechanism to trade off lower computational cost (e.g., FLOPs) for lower accuracy. In a cascade, inputs are first processed by a lightweight, inexpensive model that produces both a prediction and a confidence score. If the confidence exceeds a predefined threshold, the prediction is accepted; otherwise, the input is escalated to a more powerful model for reprocessing. Intuitively, this allows model cascades to process "easy" samples with cheap models and "hard" samples with expensive models. Through their tunable certainty thresholds, cascades expose a high-resolution trade-off between output quality and computational cost (e.g., FLOPs). Furthermore, cascades can significantly save computation at little quality degradation (Figure 4). However, using cascades to serve online requests on GPUs introduces two key challenges that can limit their effectiveness, or even make them degrade the system's performance:

Additional data transfer: Computing a model's output requires the GPU to repeatedly transfer weights and inputs to its arithmetic units. This often dominates the time taken to perform the arithmetic itself. Model serving systems mitigate this by *batching* multiple samples together and predicting their output in the same forward pass, amortizing the weight-transfer. Higher batch sizes incur more arithmetic and for large batch sizes, the arithmetic eventually dominates the time of the data transfer. For example, Figure 4 shows Llama-70B's runtime remains flat up to a batch size of 128, indicating memory transfer is the bottleneck; it then increases as the arithmetic becomes the bottleneck. Therefore, cascades only boost end-to-end throughput with large batch sizes, when their computational savings exceed the extra queuing delay.

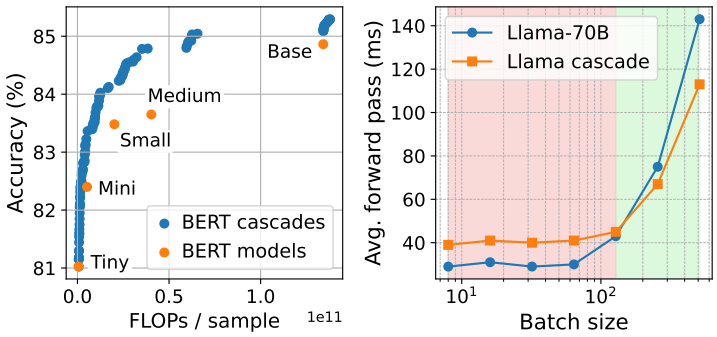


Figure 4: Left: Pareto frontier of FLOPs vs. accuracy for fine-tuned BERT classifiers on Sentiment-140 [16]. Right: Runtimes of Llama-70B and a cascade incurring 84% of its FLOPs.

Higher memory footprint: Model cascades require multiple models to reside in GPU memory. Often, their combined weights cannot fit on a single GPU. In such cases, individual models cannot be replicated as frequently as when served alone. For generative LLMs, throughput further depends on the amount of free GPU memory, which limits the number of concurrent requests, and therefore the batch size of a forward pass. The GPU must be able to run enough requests concurrently to reach batch sizes where cascades could help in the first place.

To address these shortcomings, we propose Ken, a dedicated model serving system that (i) exposes a high-resolution trade-off space between cost, accuracy, and latency, and (ii) enables more efficient ML inference. In our experiments, we characterize its behavior and demonstrate that Ken achieves a 1.7x - 3.3.x reduction in p95 latency over strong baselines [24].

We have described the challenges associated to the cost of data-intensive pipelines using LLM-powered operators, and some of the optimizations for their execution that can be leveraged by AI-enabled data-to-insight systems through specialized components. In the next section, we will introduce the opportunities of metadata-aware storage and retrieval to further enhance the output quality of AI operators.

5 Metadata-Aware Storage and Retrieval

When processing data with LLM and AI-powered tools, research showed the importance of domain-specific, external data and metadata to achieve high quality outputs [21, 30, 37]. Hence, a mature AI-enabled system requires strong retrieval capabilities to handle a broad spectrum of tasks with sufficient accuracy. In this section, we propose three metadata-aware strategies for data storage and retrieval that can serve as specialized components within a full-fledged data-to-insight system: (1) input data alignment for join-aware retrieval, (2) offline data enrichment for annotating data collections, and (3) log-augmented generation for reusing previous computations and reasoning traces.

5.1 Join-aware Retrieval

The current paradigms for retrieval, RAG and CAG, typically operates in an iterative, online, and isolated fashion assuming that the relevant data for a given LLM operation is readily available and are primarily concerned with its retrieval. However, information is often found scattered across data sources, therefore naive document retrieval can lead to incomplete answers due to missing connections between data sources. For instance, as shown in Figure 5, to answer the question "What is the highest eligible free rate for K-12 students in the schools in the most populous county in California?", the model might retrieve data on schools and eligible free rate, and separately a list of California counties and

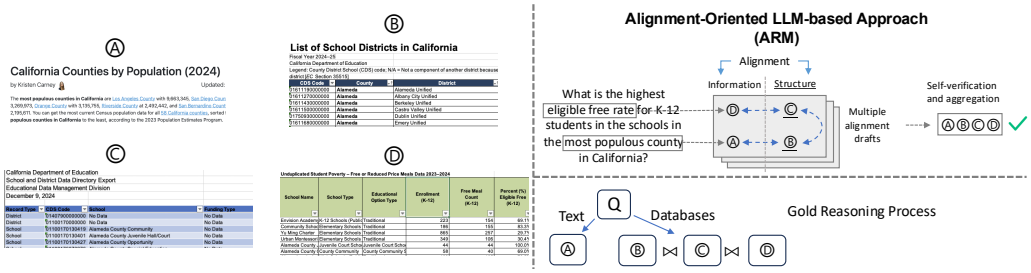


Figure 5: Overview of ARM, our alignment-based retrieval method for join-aware retrieval.

their population. However, in the most general case there is no guarantee that the retrieved data has explicit join keys (e.g., if school locations are cities, not counties) which would result in poor or empty results for the main question.

We propose ARM, standing for Alignment-Oriented Method [10, 11], to enables an efficient and comprehensive retrieval for queries for which information is found in separate documents. First, we decompose complex queries in easier sub-questions, and we aligning both the main question and its decomposed sub-questions with existing data objects (e.g., the tables or documents in a data collection). Specifically, ARM retrieves not only based on the semantic similarity of data objects to the query, but also based on the compatibility between objects themselves. Using the similarity and compatibility scores, we generate multiple alignment drafts that account for both informational and structural alignment, and ultimately select only the objects from these drafts that the model is most confident in. We evaluated ARM on three complex retrieval tasks involving both passages and tables (OTT-QA [13], Bird [28], and 2WikiMultiHop [20]), and found that it substantially outperforms traditional RAG systems—with or without query decomposition—as well as ReAct-style agent-based frameworks [51].

5.2 Offline Data Enrichment

Existing sparse (e.g., BM25) or dense (e.g., vector embeddings) retrieval methods often rely on complex, costly *online* computations to boost retrieval performance. Specifically, these methods typically begin by retrieving a broad set of data objects (such as documents or tables) from a collection that are potentially relevant to the query. They then utilize more powerful LLMs to assess the relevance between the query and the retrieved objects *online*, re-ranking the results to return only a small subset of the most relevant items. However, this retrieve-and-rerank process happens online for each query anew, which incurs high cost and latency. Moreover, they are inherently limited by the performance of the sparse or dense retrievers.

To overcome these shortcomings, we propose ENRICHINDEX [9], a retrieval approach which uses LLM *offline* to build semantically-enriched retrieval indices, by performing a single pass over all data objects in the retrieval corpus once during ingestion time. The semantically-enriched indices can enhance the effectiveness of current sparse and dense retrieval methods and, in turn, complement retrieve-then-rerank retrieval pipelines. We evaluated ENRICHINDEX on five complex retrieval tasks, involving passages and tables (Spider2 [26], Beaver [8], Fiben [42], Bright [44], and NQ [25]), and found that it outperforms strong online LLM-based retrieval systems with higher recall and significantly fewer tokens, greatly reducing the online latency and cost.

5.3 Reusable Cache and Log Store

Current LLMs and LLM-based agentic frameworks [51] handle user tasks in isolation, without remembering prior interactions. This lack of memory hinders their ability to reuse past reasoning, resulting in

repetitive thought processes and an inability to reflect on previous tasks. For instance, when solving a task T composed of three sub-tasks $T_1 \rightarrow T_2 \rightarrow T_3$, an LLM decomposes it and addresses each step sequentially. Later, when presented with task T', which consists of $T'_1 \rightarrow T_2 \rightarrow T_3$, the LLM repeats the process from scratch, unaware that the reasoning for sub-tasks $T_2 \rightarrow T_3$ has already been performed and could be reused. We propose log-augmented generation [12], LAG, a framework that directly reuses prior computation and reasoning from past logs at inference time. As part of the framework, we represent logs using KV values corresponding to a subset of tokens in past reasoning traces to represent the full reasoning context—reducing size while enabling context-dependent interpretation. We evaluated LAG on four knowledge- and reasoning-intensive datasets (Musique [47], 2WikiMultiHop [20], GPQA [35], and MMLU-Pro [50]), and found that our method significantly outperforms standard agentic systems without log usage and existing reflection and KV caching techniques, achieving superior effectiveness and efficiency.

Together with human/agent interfaces and execution optimization, we envision that metadata-aware storage and retrieval, as implemented by the aforementioned components, is a fundamental feature for AI-enabled data-to-insight systems. Before concluding our paper, the next section introduces our efforts to benchmark existing AI systems on their capabilities of solving complex data-to-insight tasks.

6 Benchmarking Data-to-Insight Systems

As argued throughout the paper, a mature AI-enabled system to automate data-to-insights should support users across whole data pipelines, from data discovery, wrangling and cleaning, to data visualization and statistical modeling. Current research has evaluated LLMs mostly on unit tasks, e.g., text-to-SQL, code generation, or question answering. Therefore, an integrated system is hard to assess holistically. We argue for the need of a robust benchmark to guide further progress in automated data-to-insight pipelines.

An ideal benchmark should have the following features:

Scale and heterogeneity: Real data science pipelines may involve tens to thousands of files with varying formats, e.g., CSVs, PDFs, JSON logs, or web tables. Synthetic or toy datasets are not fit to reproduce the messiness of real data files; hence they must include real-world data sources.

Task complexity: Benchmark questions must correspond to complex and realistic goals. To solve a realistic data science benchmark, a system under test should demonstrate pipeline design, implementation, and debugging capabilities. Moreover, a benchmark should be comprised of real-world datasets, large, potentially domain-specific and in a raw, unclean format.

Comprehensive evaluation: Considering the complexity of end-to-end pipelines, evaluation must be sufficiently fine-grained to provide insightful analysis into the failure modes of systems. Ideally, a benchmark should assess the end-to-end correctness of results, a correct design of the pipelines, and a correct implementation of the individual pipeline steps - to ensure that the produced pipelines can be consistently applied over potentially different data inputs (e.g., newer versions of the data).

Reproducibility and openness: Tasks should draw from public data sources, ship with reference solutions, and avoid dependencies on proprietary APIs so that any researcher can evaluate systems using the benchmark.

Following these principles, we introduce KRAMABENCH, the first suite that tests complete, automated data-science pipelines over real data lakes. The benchmark includes 104 tasks covering 1 700 raw files drawn from 24 sources in six domains; each task is a natural-language objective that demands several sub-tasks to be completed, e.g., data discovery, data wrangling, and statistical analysis. The benchmark scores (i) fully automated runs, (ii) pipeline design quality, and (iii) correctness of individual sub-tasks, exposing where systems break down.

Table 1: Evaluation results for the 106 data science pipelines of KramaBench on 6 baseline LLM systems.

	Models					
Evaluation setting	GPT-o3	GPT-4o	Claude-3.5	Llama3-3Instruct	DeepSeek-R1	Qwen2-5Coder
End-to-end automation	9.64%	1.62%	7.45%	1.19%	3.14%	3.72%
Pipeline Design	40.60%	30.83%	31.06%	26.74%	18.94%	27.35%
Pipeline Implementation	12.95%	9.27%	10.65%	8.28%	12.08%	7.52%

We manually curated all individual tasks and sub-tasks starting from real-world data science pipelines, and for each task provides a manually verified reference solution. The benchmark artifacts could be found at `http:\\www.github.com\\mitdbg\KramaBench`.

Table 1 presents some of the results of six different LLM systems on the 106 data science pipelines in the KramaBench benchmark. As it can be noted, none of the six baseline LLMs can solve even 10% of the benchmark fully automatically. The best performing baselines are GPT-o3 with 9.64%, and Claude-3.5 with 7.45%. These results underscore how brittle current agents are once they must discover data, write code, run it, and return a correct answer without human help. When it comes to pipeline design, systems have a higher score, showing that the models can sketch reasonable pipeline blueprints far more often than they can implement and execute them. Our experiments show that, even when the system is given reference sub-tasks, the success rate of pipeline implementation hovers around 10% across different models. In our analysis, we identified that models struggle to implement sub-tasks with correct parameters, that are data-input specific (e.g., using the right wrangling logic).

7 Conclusions

In this paper, we outlined our vision for AI-enabled data-to-insight systems. Automating data science end-to-end is a challenging yet rewarding research direction to pursue. Realizing this vision, however, demands integrated systems that span a full stack of components. We outlined some of the active areas of research across this stack: interactive and collaborative human-agent systems, automated AI data processing, optimized execution of AI operators, and specialized storage for agentic and reasoning workflows. We described a new benchmark to provide the first holistic lens on this challenge, and the results are sobering: even the most capable foundation models today solve fewer than one-tenth of end-to-end pipelines, stumble on large data volumes, and reveal sharp drop-offs between planning, implementation, and execution. These findings show that significant gaps remain before trustworthy, scalable "data-to-insight" automation is achieved.

We believe that the database community has both the data system expertise as well as all technical building blocks necessary to make rapid progress towards AI-enabled data-to-insights systems, and we are eager to collaborate with researchers and practitioners who share this goal.

Acknowledgements

We are grateful for the support from the DARPA ASKEM Award HR00112220042, the ARPA-H Biomedical Data Fabric project, NSF DBI 2327954, a grant from Liberty Mutual, and the Amazon Research Award. Additionally, our work has been supported by contributions from Amazon, Google, and Intel as part of the MIT Data Systems and AI Lab (DSAIL) at MIT, along with NSF IIS 1900933. This research was sponsored by the United States Air Force Research Laboratory and the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the

authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] Ascend.io, 2015.

[2] Github copilot, 2021.

[3] Activepieces, 2022.

[4] Cursor, 2023.

[5] R. Agrawal. The continuum-armed bandit problem. *SIAM Journal on Control and Optimization*, 33(6):1926–1951, 1995.

[6] J.-Y. Audibert and R. Munos. Algorithms for infinitely many-armed bandits. pages 1729–1736, 01 2008.

[7] C.-M. Chan, C. Xu, R. Yuan, H. Luo, W. Xue, Y. Guo, and J. Fu. Rq-rag: Learning to refine queries for retrieval augmented generation. *arXiv preprint arXiv:2404.00610*, 2024.

[8] P. B. Chen, F. Wenz, Y. Zhang, D. Yang, J. Choi, N. Tatbul, M. Cafarella, Ç. Demiralp, and M. Stonebraker. Beaver: an enterprise benchmark for text-to-sql. *arXiv preprint arXiv:2409.02038*, 2024.

[9] P. B. Chen, T. Wolfson, M. Cafarella, and D. Roth. Enrichindex: Using llms to enrich retrieval indices offline. *arXiv preprint arXiv:2504.03598*, 2025.

[10] P. B. Chen, Y. Zhang, M. Cafarella, and D. Roth. Can we retrieve everything all at once? arm: An alignment-oriented llm-based retrieval method. *arXiv preprint arXiv:2501.18539*, 2025.

[11] P. B. Chen, Y. Zhang, and D. Roth. Is table retrieval a solved problem? exploring join-aware multi-table retrieval. *arXiv preprint arXiv:2404.09889*, 2024.

[12] P. B. Chen, Y. Zhang, D. Roth, S. Madden, J. Andreas, and M. Cafarella. Log-augmented generation: Scaling test-time reasoning with reusable computation. *arXiv preprint arXiv:2505.14398*, 2025.

[13] W. Chen, M.-W. Chang, E. Schlinger, W. Wang, and W. W. Cohen. Open question answering over tables and text. *arXiv preprint arXiv:2010.10439*, 2020.

[14] M. Fan, J. Fan, N. Tang, L. Cao, G. Li, and X. Du. Autoprep: Natural language question-aware data preparation with a multi-agent framework, 2025.

[15] J. George, L. Maas, N. Abedpour, M. Cartolano, L. Kaiser, R. N. Fischer, A. H. Scheel, J.-P. Weber, M. Hellmich, G. Bosco, et al. Evolutionary trajectories of small cell lung cancer under therapy. *Nature*, 627(8005):880–889, 2024.

[16] A. Go, R. Bhayani, and L. Huang. Twitter sentiment classification using distant supervision. CS224N Project Report 1(2009), Stanford University, 2009.

[17] G. Graefe. The cascades framework for query optimization. *IEEE Data(base) Engineering Bulletin*, 18:19–29, 1995.

[18] H. Han, Y. Wang, H. Shomer, K. Guo, J. Ding, Y. Lei, M. Halappanavar, R. A. Rossi, S. Mukherjee, X. Tang, et al. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309*, 2024.

[19] J. M. Hellerstein, M. Stonebraker, and J. Hamilton. *Architecture of a Database System*. Now Publishers Inc., Hanover, MA, USA, 2007.

[20] X. Ho, A.-K. D. Nguyen, S. Sugawara, and A. Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.

[21] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025.

[22] S. Jo and I. Trummer. ThalamusDB: Approximate Query Processing on Multi-Modal Data. *Proceedings of the ACM on Management of Data*, 2(3):1–26, May 2024.

[23] A. Karpathy, February 2025.

[24] F. Kossmann, Z. Wu, A. Turk, N. Tatbul, L. Cao, and S. Madden. Cascadeserve: Unlocking model cascades for inference serving. *arXiv preprint arXiv:2406.14424*, 2024.

[25] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[26] F. Lei, J. Chen, Y. Ye, R. Cao, D. Shin, H. Su, Z. Suo, H. Gao, W. Hu, P. Yin, et al. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*, 2024.

[27] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems (NeurIPS)*, 33:9459–9474, 2020.

[28] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, et al. 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*, 36:42330–42357, 2023.

[29] C. Liu, M. Russo, M. Cafarella, L. Cao, P. B. Chen, Z. Chen, M. Franklin, T. Kraska, S. Madden, R. Shahout, et al. Palimpzest: Optimizing ai-powered analytics with declarative query processing. CIDR, 2025.

[30] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics (TACL)*, 12, 2024.

[31] L. Patel, S. Jha, M. Pan, H. Gupta, P. Asawa, C. Guestrin, and M. Zaharia. Semantic operators: A declarative model for rich, ai-based data processing, 2025.

[32] R. Peeters, A. Steiner, and C. Bizer. Entity matching using large language models, 2024.

[33] M. Potthast, M. Hagen, and B. Stein. The dilemma of the direct answer. *SIGIR Forum*, 54(1), Feb. 2021.

[34] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning (ICML)*, pages 8748–8763, 2021.

[35] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

[36] M. Russo, S. Sudhir, G. Vitagliano, C. Liu, T. Kraska, S. Madden, and M. Cafarella. Abacus: A cost-based optimizer for semantic operator systems, 2025.

[37] J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia. ARES: An automated evaluation framework for retrieval-augmented generation systems. In K. Duh, H. Gomez, and S. Bethard, editors, *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 338–354, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

[38] A. Sarkar, A. D. Gordon, C. Negreanu, C. Poelitz, S. S. Ragavan, and B. Zorn. What is it like to program with artificial intelligence?, 2022.

[39] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations (ICLR)*, 2024.

[40] D. Satriani, E. Veltri, D. Santoro, S. Rosato, S. Varriale, and P. Papotti. Logical and physical optimizations for sql query execution over large language models. SIGMOD '25, New York, NY, USA, 2025. Association for Computing Machinery.

[41] S. Schelter and S. Grafberger. Messy code makes managing ml pipelines difficult? just let llms rewrite the code!, 2024.

[42] J. Sen, C. Lei, A. Quamar, F. Özcan, V. Efthymiou, A. Dalmia, G. Stager, A. Mittal, D. Saha, and K. Sankaranarayanan. Athena++ natural language querying for complex nested sql queries. *Proceedings of the VLDB Endowment*, 13(12):2747–2759, 2020.

[43] S. Shankar, T. Chambers, T. Shah, A. G. Parameswaran, and E. Wu. Docetl: Agentic query rewriting and evaluation for complex document processing, 2024.

[44] H. Su, H. Yen, M. Xia, W. Shi, N. Muennighoff, H.-y. Wang, H. Liu, Q. Shi, Z. S. Siegel, M. Tang, et al. Bright: A realistic and challenging benchmark for reasoning-intensive retrieval. *arXiv preprint arXiv:2407.12883*, 2024.

[45] Y. Sui, M. Zhou, M. Zhou, S. Han, and D. Zhang. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, WSDM '24, page 645–654, New York, NY, USA, 2024. Association for Computing Machinery.

[46] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[47] H. Trivedi, N. Balasubramanian, T. Khot, and A. Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

[48] M. Urban and C. Binnig. Demonstrating caesura: Language models as multi-modal query planners. In *Companion of the 2024 International Conference on Management of Data*, pages 472–475, 2024.

[49] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.

[50] Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[51] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

TAIJI: MCP-based Multi-Modal Data Analytics on Data Lakes

Chao Zhang, Shaolei Zhang, Quehuan Liu, Sibei Chen, Tong Li, Ju Fan*
Renmin University of China
{fanj@ruc.edu.cn}

Abstract

The variety of data in data lakes presents significant challenges for data analytics, as data scientists must simultaneously analyze multi-modal data, including structured, semi-structured, and unstructured data. While Large Language Models (LLMs) have demonstrated promising capabilities, they still remain inadequate for multi-modal data analytics in terms of accuracy, efficiency, and freshness. First, current natural language (NL) or SQL-like query languages may struggle to precisely and comprehensively capture users' analytical intent. Second, relying on a single unified LLM to process diverse data modalities often leads to substantial inference overhead. Third, data stored in data lakes may be incomplete or outdated, making it essential to integrate external open-domain knowledge to generate timely and relevant analytics results. In this paper, we envision a new multi-modal data analytics system to address the aforementioned limitations. Specifically, we propose a novel architecture built upon the Model Context Protocol (MCP), an emerging paradigm that enables LLMs to collaborate with knowledgeable agents. First, we define a semantic operator hierarchy tailored for querying multi-modal data in data lakes and develop an AI-agent–powered NL2Operator translator to bridge user intent and analytical execution. Next, we introduce an MCP-based execution framework, in which each MCP server hosts specialized foundation models optimized for specific data modalities. This design enhances both accuracy and efficiency, while supporting high scalability through modular deployment. Finally, we propose an updating mechanism by harnessing the deep research and machine unlearning techniques to refresh the data lakes and LLM knowledge, with the goal of balancing the data freshness and inference efficiency.

1 Introduction

Modern data-intensive applications continuously generate diverse types of data stored in data lakes [16], encompassing structured data (e.g., tables, graphs), semi-structured data (e.g., JSON, HTML), and unstructured data (e.g., text, images, video). This diversity introduces a critical challenge of *data variety* [15], underscoring the need for integrated analysis across *multi-modal datasets*, which often contain complementary information essential for extracting timely and valuable insights. For example, in healthcare scenarios, physicians simultaneously analyze heterogeneous patient data, such as X-ray images and textual diagnostic reports, to support accurate and timely diagnoses. In e-commerce, users seek products by jointly exploring visual content, textual descriptions, and structured metadata.

Recently, the rapid advancement of Large Language Models (LLMs) has opened new opportunities for analyzing multi-modal data at the semantic level through natural language interactions. For example, GPT-4 [2] demonstrates the ability to reason over text, images, and JSON data within a unified conversational interface. However, despite these advancements, current LLM-based approaches have yet to fully unlock the potential of multi-modal data analytics, primarily due to the following limitations:
Limitation 1: Limited Query Expressiveness. Existing approaches typically support querying only a subset of data modalities and suffer from limited expressiveness in capturing complex user intents. For example, ELEET [24] supports analytics over text and tabular data; Palimpzest [11] is restricted to

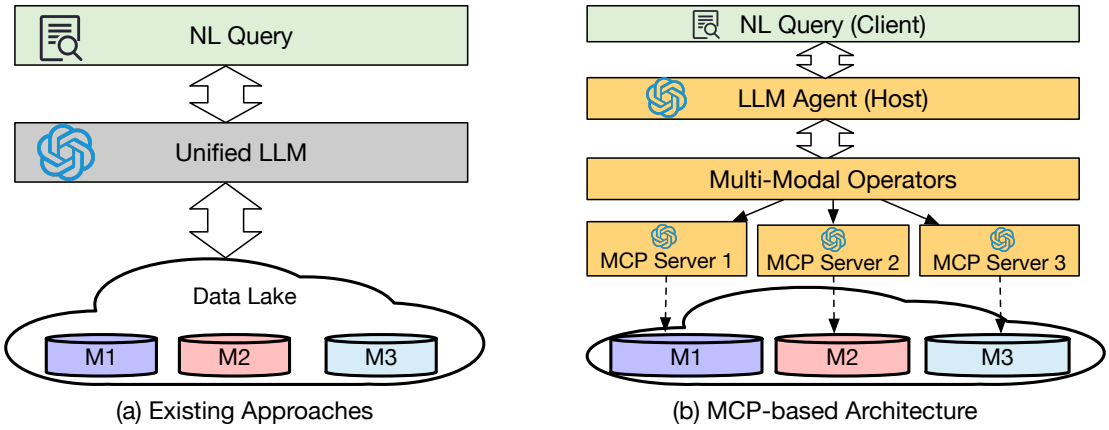


Figure 1: An Illustrate Example on MCP-based Multi-Modal Data Analytics.

text and image processing; and AOP [25] primarily targets document and text data. Underlying these systems are three main strategies for query translation: (1) mapping natural language (NL) directly to modality-specific operations [25], (2) translating NL to SQL using NL2SQL techniques [6, 13], or (3) requiring users to write declarative SQL-like query languages [11]. However, these methods remain inadequate for data lakes, where user queries are often ambiguous and span a wide range of data modalities.

Limitation 2: High Inference Overhead. Most existing approaches rely on a single, unified LLM to process heterogeneous data modalities, which leads to significant inference overhead. This issue is attributed to the computational cost of state-of-the-art LLMs, such as GPT-4, which contains over a trillion parameters. Moreover, queries that span multiple modalities introduce complexity in query planning and execution, further impacting efficiency. From a query processing perspective, users typically expect low-latency responses, making it imperative to reduce inference overhead.

Limitation 3: Knowledge and Data Staleness. Data lake is often incomplete or becomes outdated over time, which undermines the freshness and reliability of analytical results. Moreover, the knowledge embedded in LLMs can also become stale as new data and emerging patterns (e.g., novel diseases or evolving user behavior) are not reflected in the original training data. For example, an LLM may fail to recognize symptoms of a newly discovered disease or interpret related medical images correctly. Existing approaches largely ignore this issue and operate under static assumptions. Therefore, it is essential to develop mechanisms for augmenting stale data and refreshing LLM knowledge to ensure up-to-date and contextually relevant analytics.

Model Context Protocol (MCP) [8] is a novel framework that standardizes the interaction among AI agents (e.g., knowledge-grounded LLMs), external tools (e.g., database engines, function calls), and data resources (e.g., structured and unstructured sources). By defining a unified interface, MCP enables seamless integration of real-time inputs, environmental variables, and domain-specific constraints, thereby supporting robust and scalable decision-making across diverse applications. In the context of multi-modal data analytics, MCP provides an effective abstraction layer over heterogeneous data lakes. Users can interact with AI agents through natural language, while the analytics workload is offloaded to dedicated MCP servers, each tailored to handle a specific data modality. Moreover, with the growing ecosystem of MCP servers, thousands of off-the-shelf components will be readily available for processing multi-modal data efficiently.

Building upon the capabilities of MCP, we propose a novel multi-modal data analytics system, named Taiji, to address the aforementioned limitations. Taiji introduces a new architecture that

leverages MCP to offload modality-specific analytical tasks to dedicated MCP servers. As illustrated in Figure 1, TAIJI differs fundamentally from conventional approaches that rely on a single, unified LLM to process all modalities (e.g., M1, M2, M3 represent different data modalities). Instead, TAIJI adopts a client–host architecture: the client receives an NL query, and a host-side LLM agent interprets the user intent. This agent decomposes the query into a set of modality-specific operators and formulates a structured query plan. Each sub-plan is dispatched to an appropriate MCP server, where a tailored LLM, which is optimized for the corresponding modality, executes the analysis. This modular and distributed design yields several key advantages: (1) higher inference accuracy by leveraging specialized models, (2) improved scalability through parallelism across servers, and (3) significantly reduced inference overhead compared to monolithic LLM-based solutions.

In summary, this paper has the following key contributions:

(1) **MCP-Based Multi-Modal Analytical System.** We propose a novel multi-modal data analytics framework built upon the Model Context Protocol (MCP), which addresses the limitations of existing LLM-based systems. The core idea is to assign each MCP server a tailored LLM optimized for a specific data modality, embracing the principle that "one size does not fit all".

(2) **Semantic Operator Hierarchy.** We introduce a hierarchical set of semantic operators that spans structured, semi-structured, and unstructured data. This design enables TAIJI to support advanced tasks such as cross-modal joins, while maintaining compatibility with existing operators for relational data, documents, graphs, images, audio, and video, thus avoiding "reinventing the wheels".

(3) **Query-Driven Model Optimization and Data Discovery.** We develop a query-driven fine-tuning strategy to optimize the reasoning ability of LLMs on each MCP server. Moreover, we propose a unified embedding-based semantic representation combined with a hybrid indexing mechanism for multi-modal data discovery.

(4) **Dynamic Knowledge and Data Refreshing.** We propose a twofold updating mechanism to ensure freshness of both the data and the LLMs. First, we leverage deep research techniques to enrich the data lake with the latest documents and web content. Second, we introduce a lightweight editing mechanism (supporting insert, update, and delete operations) augmented by machine unlearning techniques to refresh LLM knowledge.

2 Preliminaries

2.1 Problem Definition

Multi-Modal Query Processing (MMQP). Given a natural language (NL) query Q over a collection of datasets $\mathbb{D}$, where each dataset $D_i \in \mathbb{D}$ is associated with a modality $M_i \in \mathbb{M}$, the goal is to compute a result set R that satisfies the query predicates, where each result tuple $r \in R$ may contain data items spanning multiple modalities in $\mathbb{M}$. To evaluate the performance of MMQP, we adopt three standard metrics: (1) recall, which measures the completeness of the retrieved results, (2) precision, which quantifies the ratio of the correct ones among the returned results. and (3) latency, which reflects the query execution efficiency.

In general, the MMQP problem has the following challenges:

Challenge 1: Cross-Modality Query Planning. Unlike traditional query planning in relational databases that focuses solely on relational operators, orchestrating operator pipelines across multiple data modalities is more complex. This complexity arises from the heterogeneous nature of operators and data formats. Therefore, MMQP requires an effective yet lightweight mechanism to generate reasonable execution plans.

Challenge 2: Efficient Inference on Large-Scale Data. Compared with using a general-purpose LLM with a massive number of parameters, employing a tailored, relatively lightweight LLM for

modality-specific data can significantly reduce inference overhead. However, this approach still encounters scalability challenges when dealing with large volumes of data, particularly unstructured data such as text, images, or videos.

Challenge 3: Data and Knowlege Freshness. High-quality data in the data lake is often insufficient to precisely and comprehensively answer user queries. Therefore, it is crucial to continuously update both the data lake and the LLM knowledge within each MCP server. However, open-domain data sources are vast and challenging to explore, making integration into the data lake cumbersome and resource-intensive.

2.2 Related Work

In recent years, LLMs have made remarkable advances in natural language understanding, generation, and reasoning, opening up new opportunities for multi-modal data management. By expressing their query requirements in natural language, users can interact with systems that leverage LLMs to interpret query intent and dynamically orchestrate external databases, knowledge bases, or APIs to execute complex analytical tasks.

CAESURA [23] (also known as ELEET [24]) utilizes LLMs as multimodal query planners that translate NL inputs into executable multimodal query plans. Its key idea is leveraging LLMs to understand user intent and dynamically construct efficient query workflows tailored to diverse data modalities. Building upon this, Gorilla [18] enhances LLMs' reasoning and tool-use capabilities by incorporating retrieval-augmented generation (RAG), enabling real-time access to external knowledge bases and APIs during multimodal query processing. Toolformer [21] further advances LLM tool integration by fine-tuning models with limited examples, empowering them to autonomously invoke external APIs, such as database engines and computational tools, during generation. This overcomes LLMs' traditional limitation to static text generation, enabling dynamic interaction with external data sources for complex analytical tasks. ThalamusDB [9] introduces a hybrid architecture combining a dedicated query planner with deep learning inference. It allows SQL queries to include NL predicates, offloading parts of the execution to neural models for image and text processing, while leveraging approximate query processing (AQP) to balance efficiency and accuracy. However, it still relies on manual annotations, which may hinder scalability. PALIMPZEST [11, 12] extends declarative query languages to express AI workloads, generating optimized plans that jointly consider the cost of both AI inference and relational operations. AOP [25, 26] proposes defining semantic operators over documents, text, and structured tables, and employs LLMs to iteratively generate executable query plans for multi-modal content.

However, existing works have three main limitations: First, they support only a few data modalities and are hard to extend, i.e., adding new modalities or operators often requires heavy manual work. In contrast, Taiji supports a wide range of multimodal operators with a scalable MCP-based architecture. Second, most systems use a single LLM to handle all data types, which leads to low efficiency and accuracy. Taiji instead uses tailored LLMs for different modalities, improving performance. Third, they focus on static data and ignore updates. Taiji addresses this by automatically enriching the data and refreshing model knowledge over time.

3 Taiji's Overview

Figure 2 presents an overview of Taiji, which consists of three main components: **MCP Client Manager**, **MCP Server Manager** and **MCP Augmentor**. In the following, we introduce them in details.

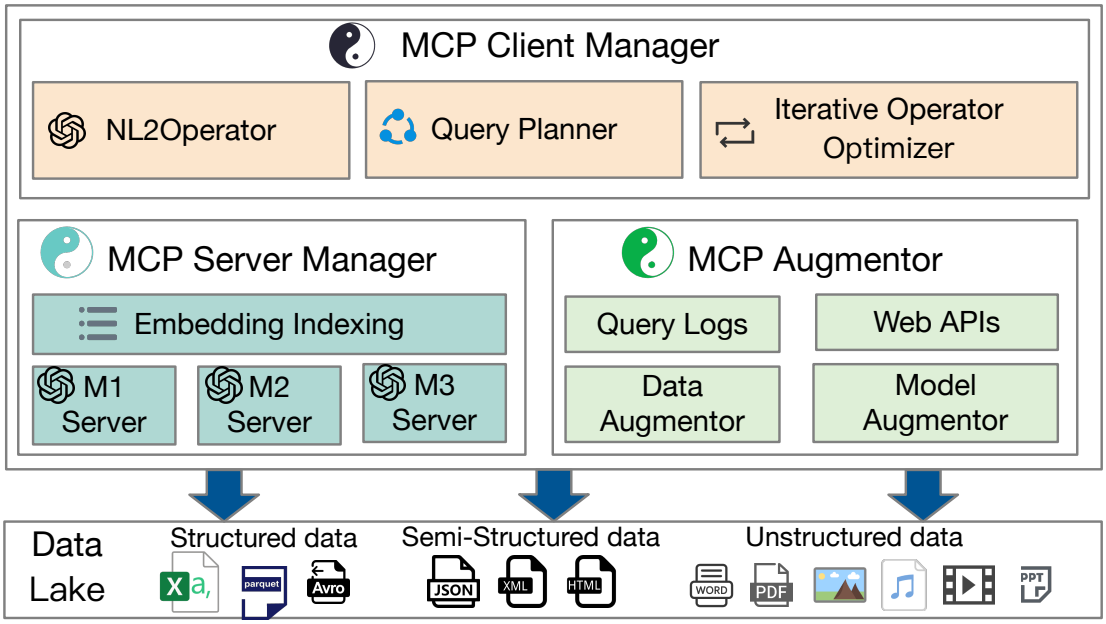


Figure 2: An Overview of Taiji.

3.1 MCP Client Manager

3.1.1 NL2Operator

Understanding user intent is critical for accurate query answering. However, neither SQL-like declarative languages nor UDF-based procedural languages are well-suited for this task. This is due to two main challenges: First, NL queries are inherently ambiguous and difficult to interpret accurately. Second, translating NL queries into complex declarative or procedural languages increases the likelihood of inaccuracies or errors.

To address this, we propose NL2Operator, a method that defines a hierarchical set of semantic operators, each linked to specific MCP servers. An LLM-based agent maps user queries to these operators based on query intent. For example, semi-structured data processing is handled by an intermediate MCP server, which can either process the query directly or delegate it to specialized sub-servers (e.g., for JSON, HTML, XML). This hierarchical design offers three advantages: (1) Scalability and flexibility, by modularizing the processing across different servers; (2) High concurrency support, through distributed workload balancing; (3) Simplified translation, by mapping NL to high-level modality-specific operators, reducing the burden on the LLM.

3.1.2 Query Planner

The goal of query planner is to find the execution plan with the lowest cost, thereby improving overall efficiency. Based on a hierarchy of semantic operators, we construct a directed acyclic graph (DAG) to represent the query workflow. We then implement a sampling-based cost estimation optimizer, which includes selectivity estimation, cost modeling, plan evaluation, and final plan selection.

The optimization process works as follows: First, an operator cost sampler collects runtime statistics, such as per-tuple processing time and selectivity, for each operator. Using this data, the optimizer builds a latency-based cost model to evaluate candidate plans. The plan with the lowest estimated execution time is selected for execution. This sampling-based approach supports accurate, data-driven

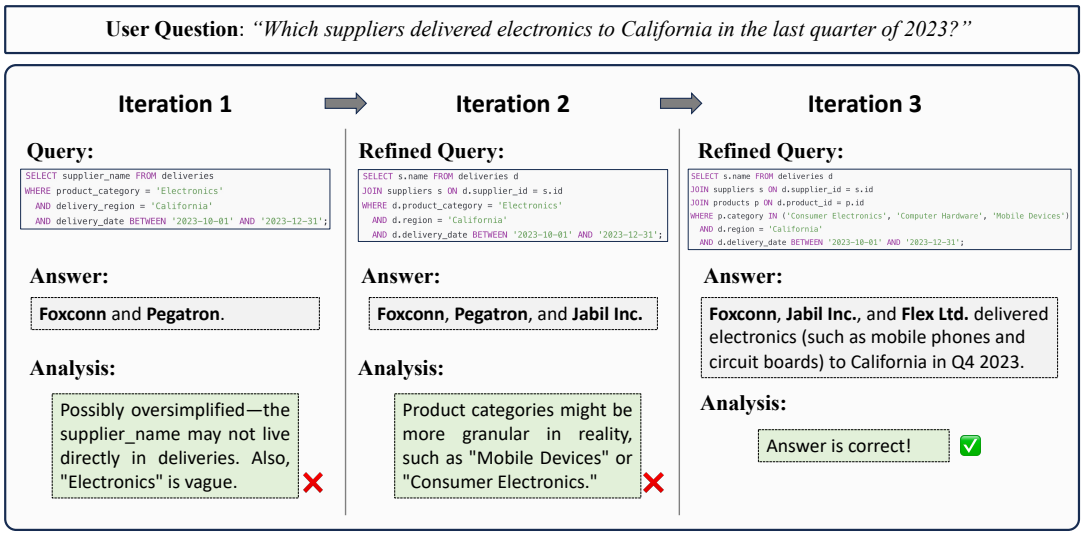


Figure 3: An illustrative example of iterative operator optimization in Taiji.

cost prediction and adaptive plan selection. It offers two key advantages: (1) it maintains reliable cost estimates under different data distributions and runtime conditions, and (2) it produces more deterministic plans compared to LLM-based query planning.

3.1.3 Iterative Operator Optimizer

Each MCP server is responsible for processing data from a specific modality, autonomously interacting with its underlying data source, whether structured (e.g., SQL-based) or unstructured (e.g., document corpus), via an Iterative operator optimization [10, 22]. Unlike traditional pipelines that rely on a single retrieval pass, Taiji introduces a bidirectional communication channel between the MCP server and the MCP client (or host). This feedback loop enables the server to request prompt refinements or clarifications from the upstream controller when retrieval results are insufficient, supporting a dynamic, feedback-driven, and iterative optimization process.

An example of the iterative architecture in Taiji is illustrated in Figure 4. The process starts when the MCP server receives a sub-task from the central LLM agent. Guided by the task's semantics and structure, the server formulates an initial query and retrieves candidate results from its local database. These results are evaluated along multiple dimensions, including coverage, redundancy, ambiguity, and informativeness. If deficiencies are detected (e.g., sparse or misaligned results), the server engages a reasoning-based refinement loop to revise the query and improve alignment with the task's underlying intent. This iterative cycle continues until the result set meets a predefined confidence threshold. Moreover, we employ a curriculum learning strategy that gradually increases task complexity, from simple key-value lookups to multi-hop reasoning over relational and multimodal data, thus guiding the agent through progressively more abstract levels of query planning and evaluation.

3.2 MCP Server Manager

3.2.1 Embedding Indexing

For high-dimensional vector data, traditional vector indexing methods [3] typically construct a graph structure by linking neighboring data points based on nearest-neighbor distance. However, when metadata-based filters are applied, many of these connections become invalid, resulting in disconnected

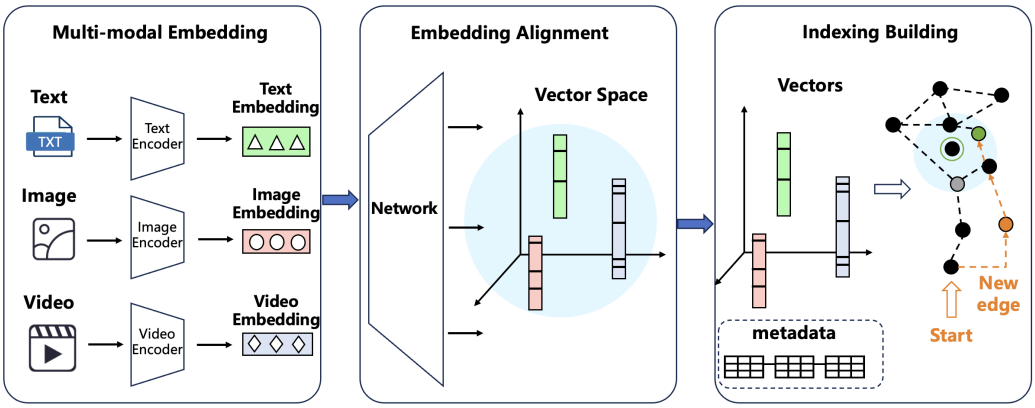


Figure 4: Embedding of Multi-Modal Data.

subgraphs. This leads to search failures, as the index may no longer be able to reach the true nearest neighbors. To address this, TAIJI introduces a filter-aware vector indexing approach designed to maintain search effectiveness even under filtering constraints. The key idea is to selectively augment the vector graph by adding edges that connect only those data points that comply with the filtering conditions. This ensures that the search process remains efficient while achieving high recall. Figure 4 illustrates the core concept. The key components of our method include:

(1) Condition-aware graph augmentation: Dynamically reinforce valid connections among filter-compliant nodes to ensure index traversability even after metadata-based filtering is applied.

(2) Hybrid search robustness: Maintain a balance between filtering precision and vector search accuracy by preserving essential traversal paths within the index.

(3) Recall preservation: Ensure that the augmented index structure delivers competitive nearest-neighbor retrieval performance, comparable to that of unfiltered searches.

This approach extends traditional vector search to support constrained queries where both semantic similarity (via embeddings) and structured criteria (via metadata) must be jointly satisfied.

3.2.2 Multi-Modal MCP Servers

Multi-Modal MCP Servers deal with heterogeneous data ecosystems by unifying structured (e.g., relational databases), semi-structured (e.g., JSON logs, XML), and unstructured data (e.g., text, images, sensor streams) under a single adaptive framework. Leveraging LLMs and multi-modal AI architectures, these servers dynamically interpret, classify, and contextualize diverse data formats through techniques such as natural language processing (NLP) for unstructured text, computer vision for visual data, and graph-based reasoning for structured relationships. By embedding semantic understanding into the MCP layer, the system autonomously generates metadata, enforces cross-modal data governance policies, and enables federated queries that bridge tabular sales records, semi-structured IoT telemetry, and unstructured social media feeds. For instance, an LLM-powered MCP Server could correlate customer sentiment (extracted from unstructured reviews) with structured transactional data to optimize supply chain decisions, while simultaneously parsing semi-structured maintenance logs to predict equipment failures.

3.3 MCP Augmentor

3.3.1 Data Augmentor

After executing queries on modality-specific databases, each MCP server autonomously initiates an update procedure involving both data augmentation and model capability enhancement, as shown in Figure 5. In the data augmentation phase, Taiji adopts a query-driven retrieval-then-synthesis strategy to keep the data lake semantically enriched and aligned with the latest developments. Upon completion of a user query, the MCP server triggers targeted information harvesting routines. These leverage both web crawlers and structured API-based agents configured to access dynamic and authoritative sources such as arXiv papers, GitHub repositories, Stack Overflow, and enterprise technical documentation.

The collected documents undergo a multi-stage processing pipeline designed to transform unstructured content into validated, semantically indexed knowledge.

(1) Structural Reconstruction: The pipeline begins with document structure parsing, utilizing tools such as ScienceParse [7] and GROBID [14] to extract key structured elements, including titles, abstracts, section headers, equations, tables, figures, and code snippets. These tools employ a combination of rule-based heuristics and machine learning techniques to reconstruct the hierarchical organization of scientific and technical documents from raw formats such as PDFs and HTML.

(2) Redundancy Elimination: In the redundancy elimination stage, a hybrid strategy combines MinHash-based fingerprinting for fast approximate set similarity detection with dense embedding-based retrieval. Specifically, documents are first fingerprinted using MinHash signatures generated over token-level n-grams. In parallel, dense vector embeddings are computed using models like Sentence-BERT [20], and clustered using HNSW-based approximate nearest neighbor search (via libraries like FAISS) to detect semantically similar passages. Duplicate or near-duplicate entries are then filtered out based on a configurable similarity threshold.

(3) Entity Extraction: Next, domain-adapted named entity recognition (NER) is applied using fine-tuned transformer-based models trained on labeled datasets from scientific and software domains. To resolve synonyms and disambiguate entities across sources, we apply context-aware entity linking using string similarity, co-occurrence statistics, and external knowledge bases (e.g., DBpedia, PapersWithCode APIs). To identify novel concepts and facts, we compute concept-level hashes using SimHash, which captures the semantic footprint of passages. These hashes are used to cluster conceptually similar content and measure deviation from existing entries in the data lake. We also apply graph-based semantic clustering over the extracted entities and their relationships to reveal emergent topics not previously indexed.

(4) Data Augmentation: In the final augmentation and indexing stage, each candidate entry is validated across multiple independent sources to ensure factual consistency and reliability. Only entries that are corroborated—i.e., referenced in at least two unrelated sources—are retained. The resulting validated knowledge units are indexed along multiple axes: modality (e.g., text, code, image), source credibility, and temporal metadata (e.g., crawl time, publication date). This indexing supports temporal knowledge reasoning, such as prioritizing more recent findings or applying decay-weighted relevance during downstream model retrieval.

3.3.2 Model Refreshing

In the Model Refreshing phase, to align model knowledge with the evolving data lake, we introduce a modality-specific parameter update mechanism tailored to three data operations: querying, insertion, and deletion. Thanks to the built-in subscription function of MCP server, the MCP client can monitor the changes of data resources via subcription, then we can conduct the model refreshing.

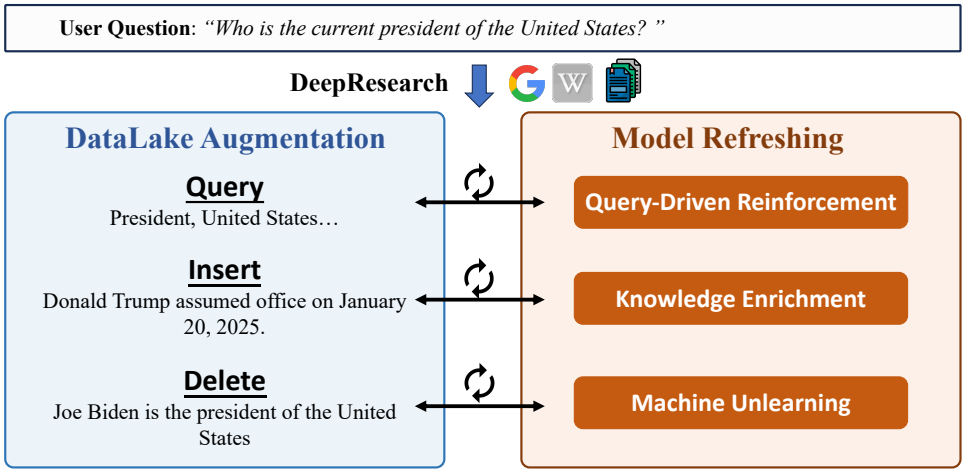


Figure 5: Data Augmentation and Model Refreshing in TAIJI.

(1) Query-Driven Reinforcement: To enhance model performance based on frequently asked or emergent user intents identified from query logs, we first perform an in-depth analysis to extract the most common and critical intent patterns. These patterns are then leveraged to generate task-specific training samples. Furthermore, to ensure that these synthesized samples effectively contribute to the model's learning process, we apply importance sampling during the fine-tuning phase. Importance sampling assigns a higher weight to these specific training examples, based on their frequency and relevance in real-world use cases. This mechanism helps the model prioritize the learning of high-value user intents while preventing overfitting to less common or irrelevant patterns. By combining these methods, we significantly improve the MCP server's ability to understand and respond to emergent user needs in real-time applications.

(2) Insert via Knowledge Enrichment: When new knowledge is ingested into the data lake, it undergoes a structured transformation where relevant text spans or structured tuples are converted into instruction-style training samples. These samples are then systematically appended to the fine-tuning dataset. The conversion process involves identifying key knowledge components and framing them as prompts and expected responses, ensuring that the newly introduced information is aligned with the model's reasoning mechanisms. By incorporating these structured samples, the model is exposed to explicit tasks that require reasoning over the updated knowledge, enabling it to more effectively process and apply this new information during inference. This method directly enhances the model's capacity for knowledge integration, thereby improving its performance on tasks that necessitate reasoning over both the pre-existing and newly added knowledge. Furthermore, this fine-tuning approach mitigates the risk of knowledge forgetting, as the model learns to consistently reference the most recent data alongside the foundational knowledge it was initially trained on.

(3) Deletion via Machine Unlearning: To address the removal of obsolete or sensitive knowledge, we employ advanced techniques such as gradient ascent unlearning or influence function-based data deletion to effectively erase the target information without compromising the integrity of the remaining model knowledge [27]. Specifically, we first compute data influence scores, which quantify the contribution of individual training samples to the model's predictions. These scores are utilized to identify the most influential data points that need to be removed. Subsequently, we employ a fine-tuning strategy where negatively-weighted gradients are applied to the model, directing it to unlearn the targeted knowledge. This process is executed in a manner that is mindful of the need to preserve the model's retained knowledge, ensuring that the unlearning of specific information does not lead to catastrophic forgetting

of other critical data or degrade the overall performance [5]. The integration of these techniques enables a controlled and efficient forgetting mechanism that minimizes interference with the model's pre-existing knowledge, making it an ideal approach for handling sensitive or outdated information.

This unified update mechanism ensures that each MCP server maintains alignment between its internal LLM and its evolving modality-specific data lake, enabling accurate, efficient, and up-to-date multi-modal response. In particular, it facilitates synchronized parameter tuning and schema-aware knowledge injection across modalities such as vision, audio, and text, thereby preventing semantic drift and enhancing the consistency of cross-modal representations. This design also supports incremental updates without requiring full model retraining, significantly reducing computational overhead while preserving reasoning fidelity.

4 Preliminary Experiments

In this section, we evaluate the performance of a prototype of Taiji, demonstrating the efficiency and effectiveness of the proposed MCP-based architecture on handling multi-modal data.

4.1 Experimental Setup

Dataset. We use the Craigslist furniture dataset [9], which contains both relational and image data, to evaluate execution accuracy and latency of Taiji. The dataset comprises 3000 listings from the "furniture" category in website craigslist.org, detailing items such as sofas, tables, and chairs. It consists of two tables: *furniture* and *image*. The *furniture* table includes information of each entity; with each furniture has one or more images related to it, the *image* table contains the path of images corresponding to each furniture.

We design three queries to verify that Taiji can demonstrate advantages concerning varied selectivity and task complexity. For simplicity, the query plan is fixed by performing a filter on the furniture table and then conducting matches on images with specified predicates. The intermediate result size refers to the size of set filtered by relational predicates, and image predicate refers to the image classification task. Table 1 gives the summary of the workload.

MCP Client Setting and Server Setting. As introduced before, we implement a disaggregated MCP-based data analytical system. Specifically, we leverage the ChatGPT API [1] in the MCP client, which is built upon the GPT-4.1 architecture [17]. Additionally, We use the Qwen2.5-VL-7B [4] as an MCP server to analyze complex images. We also deploy another MCP server [19] to manage the structured data and semi-structured data based on a PostgreSQL database.

Baseline. We compare our method to a pure GPT-4.1 model, which handles the query translation and data analysis altogether. On the contrary, our approach splits the task to three sub-tasks with MCP: query translation, table filtering, and image analysis. Note that as GPT-4.1 is not good at processing tabular data, we utilize PostgreSQL to help it handle the table filtering for a fair comparison.

Evaluation Metric. For the evaluation of Taiji, we employ three metrics: recall, accuracy and latency, which are respectively employed to evaluate correctness and efficiency. We compare the result set retrieved by client with the ground truth, to compute recall and precision. We also evaluate latency of performing the queries in an end-to-end fasion.

Table 1: Summary of the Workload

Query ID	Intermediate Size	Image Predicate	NL Query
1	33	images with two chairs	Find a set of two chairs
2	126	images with a black chair	Find a black leather chair
3	347	images with table and chair	Find a set of wood table and chair

4.2 Experimental Results

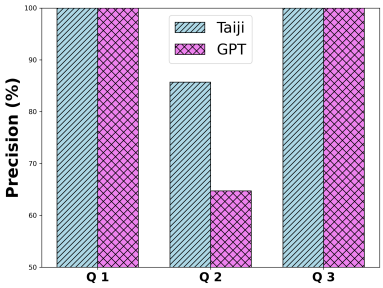


Figure 6: Precision Evaluation.

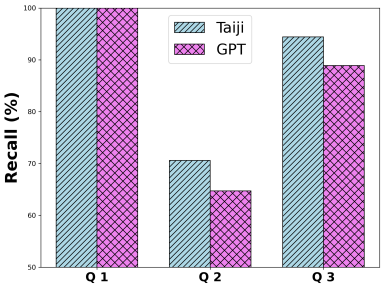


Figure 7: Recall Evaluation.

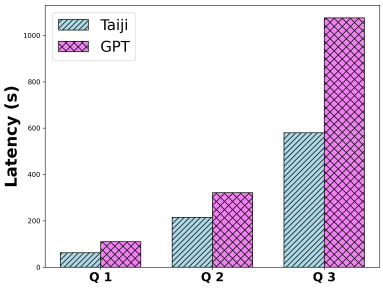


Figure 8: Latency Evaluation.

As shown in Figure 6 and Figure 7, Taiji also outperforms GPT in accuracy (for Q2) and recall (on Q2 and Q3). In specific, Taiji achieves an accuracy of 85% on Q2 while GPT has an accuracy of 65%.; Taiji has a recall of 71% and 94% on Q2 and Q3, respectively while GPT has a s a recall of 65% and 89%. Take Q2 as an example where Taiji is better than GPT both in precision and recall, we found that it is hard to find the black chair from the candidate images, the reason is that GPT can hardly figure out black chair and black sofa.

Figure 8 depicts the latency between Taiji and GPT-4.1. It is clearly visible that our approach is largely faster than GPT. As the intermediate results increase (i.e., Q1-Q3), the performance gap increases. On average, Taiji improves the latency by 43%. Take Q3 as an example where the intermediate size has reached up to 347, GPT is too heavy to offer high efficiency with its trillion parameters.

Implications of MCP-based Architecture over pure LLM. Through the preliminary experiments, we have the following insights. First, compared with pure LLM, MCP-based architecture can judiciously select and harness most proper LLMs to process multi-modal data. For instance, GPT may be excel at text processing and query translation, but it turns out that Qwen is better than GPT on image processing. Second, leveraging a tailored LLM (i.e., Qwen) with a much smaller number of parameters can not only improve the accuracy, but also it can result in a much lower inference overhead. This implies that, MCP-based architecture is very promising to combine many small LLMs for a wide range of data modality. Third, since MCP-based architecture can offload a sub-task to a local MCP server, it further reduces the burden of the centric LLM. Such an observation motivates us to explore more optimizations in the future, such as asynchronous execution, task scheduling, etc.

5 Challenges and Opportunity

MCP-based Data Security. Despite the salient features discussed in the previous sections, a major challenge of MCP-based data analytic is how to preserve the data security while performing the LLM-empowered data analytics. This is because MCP servers are developed by different vendors and contributors all over the world, and analyzing the plain text/documents in the data lake poses high risks of data leakage. For instance, the MCP SDK or build-in LLM agents may upload the data to the

web server. Therefore, it calls for new mechanisms to enable privacy-preserved data transmission and analytics. On the one hand, the data transmission tunnel should be secured among MCP host/clients, MCP servers, and resources. On the other hand, it shoud avoid accessing the plain text, other alternatives should be explored, such as query processing over ciphertext directly. However, it is challenging to balance the trade-off between query efficiency and data privacy.

Hybrid Deployment Optimization. In addition to the local deployment of MCP servers, it is also possible to deploy remote MCP servers, resulting in a hybrid deployment. Hence, it calls for new methods that can harness the capability of hyrid deployment. However, it is challenging to effectively determine the strategy of data placement and to migrate the data among the multiple MCP servers. Moreover, it is also hard to efficiently discover data of high quality due to the hugh search space and high latency.

Cost-Aware Model Mangement. As more and more MCP servers are involved, it has become a challenge to manage the multiple LLMs and their evolvement. The reason is two-fold. First, existing approaches [9, 24, 25] only care about the query performance, while neglecting the dollar cost of LLMs' invocation as they charge for tokens. Second, model knowledge can also be obsolete as discussed before, and performing the fine-tuning also incur significant cost, thus it is challenging to balance the knowledge freshness and training cost. As a result, it is required to judiciously select and fine-tune the LLM models with cost-aware optimization.

Multi-Modal Database Benchmark. Since there lacks a unified and standard multi-modal database benchmark, existing approaches basically evaluate the performance on different datasets and workloads. Hence, there is a pressing need to have a new multi-modal database benchmark that covers various data modality and workloads. However, it is challenging to collect or generate multi-modal data and workloads with both fidelity, utility, and generality, even for one modality, there could be numerous variants.

6 Conclusion

In this paper, we present Taiji, a novel multi-modal data analytical system. Particularly, we design a new architecture based on Model Context Protocol (MCP) that offloads the multi-modal operators to specific MCP server. We propose a series of components including NL2Operator query translation, multi-modal query planning, iterative RAG, embedding indexing, data augmentation, model refreshing. To the best of our knowledge, this is the first MCP-based architecture for multi-modal analytical system. We believe MCP architecture opens the door for the researches over multi-modal data management. In the future, we will finish implementing the whole system and conduct more experiments.

References

[1] Introduction of openai text generation apis. `https://platform.openai.com/docs/guides/text-generation`.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] I. Azizi, K. Echihabi, and T. Palpanas. Graph-based vector search: An experimental evaluation of the state-of-the-art. *Proc. ACM Manag. Data*, 3(1):43:1–43:31, 2025.

[4] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang, H. Zhong, Y. Zhu, M. Yang, Z. Li, J. Wan, P. Wang, W. Ding, Z. Fu, Y. Xu, J. Ye, X. Zhang, T. Xie,

Z. Cheng, H. Zhang, Z. Yang, H. Xu, and J. Lin. Qwen2.5-VL Technical Report. *arXiv preprint arXiv:2502.13923*, 2025.

[5] R. Chourasia and N. Shah. Forget unlearning: Towards true data-deletion in machine learning. volume 202, pages 6028–6073. PMLR, 2023.

[6] J. Fan, Z. Gu, S. Zhang, Y. Zhang, Z. Chen, L. Cao, G. Li, S. Madden, X. Du, and N. Tang. Combining small language models and large language models for zero-shot nl2sql. *Proceedings of the VLDB Endowment*, 17(11):2750–2763, 2024.

[7] A. I. for AI. Science parse: An open-source library for extracting structured data from scientific pdfs, 2018. `https://github.com/allenai/science-parse`.

[8] X. Hou, Y. Zhao, S. Wang, and H. Wang. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*, 2025.

[9] S. Jo and I. Trummer. Thalamusdb: Approximate query processing on multi-modal data. *Proceedings of the ACM on Management of Data*, 2(3):1–26, 2024.

[10] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

[11] C. Liu, M. Russo, M. Cafarella, L. Cao, P. B. Chen, Z. Chen, M. Franklin, T. Kraska, S. Madden, R. Shahout, et al. Palimpzest: Optimizing ai-powered analytics with declarative query processing. In *Proceedings of the Conference on Innovative Database Research (CIDR)*, 2025.

[12] C. Liu, M. Russo, M. Cafarella, L. Cao, P. B. Chen, Z. Chen, M. Franklin, T. Kraska, S. Madden, and G. Vitagliano. A declarative system for optimizing ai workloads. *arXiv preprint arXiv:2405.14696*, 2024.

[13] X. Liu, S. Shen, B. Li, P. Ma, R. Jiang, Y. Zhang, J. Fan, G. Li, N. Tang, and Y. Luo. A survey of nl2sql with large language models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109*, 2024.

[14] P. Lopez. Grobid: Generation of bibliographic data, 2020. `https://github.com/kermitt2/grobid`.

[15] J. Lu and I. Holubová. Multi-model databases: a new journey to handle the variety of data. *ACM Computing Surveys (CSUR)*, 52(3):1–38, 2019.

[16] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment*, 12(12):1986–1989, 2019.

[17] OpenAI and e. a. Achiam. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

[18] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

[19] M. C. Protocol. Model context protocol servers, 2025. `https://github.com/modelcontextprotocol/servers`.

[20] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 3982–3992, 2019.

[21] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

[22] K. Shuster, M. Komeili, L. Adolphs, S. Roller, A. Szlam, and J. Weston. Language models that seek for knowledge: Modular search and generation for dialogue and prompt completion. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 373–393, 2022.

[23] M. Urban and C. Binnig. Caesura: Language models as multi-modal query planners. *arXiv preprint arXiv:2308.03424*, 2023.

[24] M. Urban and C. Binnig. Eleet: Efficient learned query execution over text and tables. *Proc. VLDB Endow*, 17:13, 2024.

[25] J. Wang and G. Li. Aop: Automated and interactive llm pipeline orchestration for answering complex queries. CIDR, 2025.

[26] J. Wang, G. Li, and J. Feng. idatalake: An llm-powered analytics system on data lakes. *Data Engineering*, page 57.

[27] Y. Zhang, Z. Hu, Y. Bai, J. Wu, Q. Wang, and F. Feng. Recommendation unlearning via influence function. *arXiv preprint arXiv:2307.02147*, 2023.

Data Agent: A Holistic Architecture for Orchestrating Data+AI Ecosystems

Zhaoyan Sun, Jiayi Wang, Xinyang Zhao, Jiachi Wang, Guoliang Li
Department of Computer Science, Tsinghua University, Beijing, China
{liguoliang@tsinghua.edu.cn}

Abstract

Traditional Data+AI systems utilize data-driven techniques to optimize performance, but they rely heavily on human experts to orchestrate system pipelines, enabling them to adapt to changes in data, queries, tasks, and environments. For instance, while there are numerous data science tools available, developing a pipeline planning system to coordinate these tools remains challenging. This difficulty arises because existing Data+AI systems have limited capabilities in semantic understanding, reasoning, and planning. Fortunately, we have witnessed the success of large language models (LLMs) in enhancing semantic understanding, reasoning, and planning abilities. It is crucial to incorporate LLM techniques to revolutionize data systems for orchestrating Data+AI applications effectively.

To achieve this, we propose the concept of a 'Data Agent' – a comprehensive architecture designed to orchestrate Data+AI ecosystems, which focuses on tackling data-related tasks by integrating knowledge comprehension, reasoning, and planning capabilities. We delve into the challenges involved in designing data agents, such as understanding data/queries/environments/tools, orchestrating pipelines/workflows, optimizing and executing pipelines, and fostering pipeline self-reflection. Furthermore, we present examples of data agent systems, including a data science agent, data analytics agents (such as unstructured data analytics agent, semantic structured data analytics agent, data lake analytics agent, and multi-modal data analytics agent), and a database administrator (DBA) agent. We also outline several open challenges associated with designing data agent systems.

1 Introduction

In the past decade, the database community has made significant contributions to the Data+AI field [1]. On one hand, for AI4Data, our community leverages AI techniques to tackle offline NP-hard problems (e.g., index advisor [2, 3], view advisor [4], partition advisor [5], knob advisor [6], hint advisor [7]), online NP-hard problems (e.g., query rewrite [8], plan enumeration [9]), regression problems (e.g., cardinality estimation [10, 11], cost estimation [12], latency estimation [13]), prediction challenges (e.g., workload prediction [14]), and data structure design issues (e.g., learned indexes [15]). These efforts primarily focus on machine learning model design and selection, as well as the design and selection of data/query/environment features. However, adapting these techniques to changes in data, queries, tasks, and environments poses a significant challenge, as they rely heavily on experts tuning to accommodate different scenarios. On the other hand, for Data4AI, our community extends database optimization techniques to ease the deployment of AI, including in-database machine learning (ML) training and inference [16], data preparation [17], data cleaning [18], data integration [19], feature management [20], and model management [21]. The main obstacle with these methods is achieving autonomous orchestration of system pipelines without labor-intensive involvement.

The core obstacle preventing existing Data+AI techniques from adapting to varying scenarios is their limited ability in semantic understanding, reasoning, and planning, as shown in Figure 1. Fortunately,

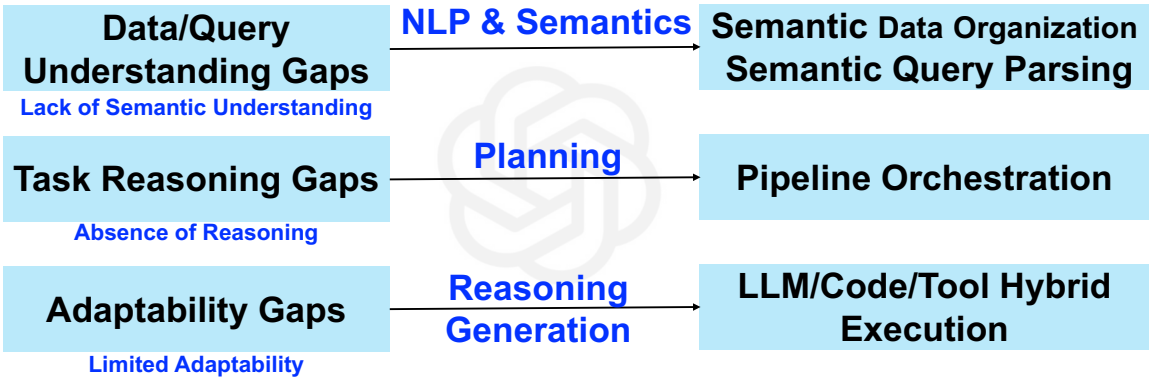


Figure 1: Challenges of Data+AI Systems.

large language models (LLMs) possess these capabilities [22, 23], and we aim to leverage them to revolutionize Data+AI systems. To accomplish this, we propose the 'Data Agent,' a comprehensive architecture designed to orchestrate Data+AI ecosystems by focusing on data-related tasks through knowledge comprehension, reasoning, and planning abilities. We outline several challenges in designing data agents:

Challenge 1: How can we understand queries, data, agents, and tools?

Challenge 2: How can we orchestrate effective and efficient pipelines to bridge the gaps between user requirements and the underlying heterogeneous data (e.g., data lakes)?

Challenge 3: How can we schedule and coordinate agents and tools to improve the effectiveness?

Challenge 4: How can we optimize and execute pipelines to improve the efficiency?

Challenge 5: How can we continuously improve pipeline quality with self-reflection?

We begin by proposing a holistic architecture to address these challenges. We present a robust architecture for developing data agents, which includes components for data comprehension and exploration, understanding and scheduling within the data engine, and orchestrating processes through pipeline management. Subsequently, we demonstrate several data agent systems, such as a data science agent, data analytics agents (including unstructured data analytics agents [19], semantic structured data agents, and data lake agents [41]), and a database administrator (DBA) agent [42, 43]. Finally, we identify some open challenges associated with designing data agent systems.

Our contributions can be summarized as follows:

(1) We introduce the data agent, which autonomously handles data-related tasks with capabilities for knowledge comprehension, automatic planning, and self-reflection.

(2) We present a holistic architecture for orchestrating a data agent system.

(3) We showcase three types of data agent systems: the data science agent, the data analytics agents, and the DBA agent.

(4) We provide several open research challenges related to data agents.

2 Data Agent

The Data Agent is designed to autonomously carry out data-related tasks with capabilities for knowledge comprehension, automatic planning, and self-reflection.

Data Agents require to consider six key factors as shown in Figure 2.

Perception: This involves understanding the environment, data, tasks, agents, and tools. It requires aligning the Data Agent through offline fine-tuning or by preparing offline prompt templates.

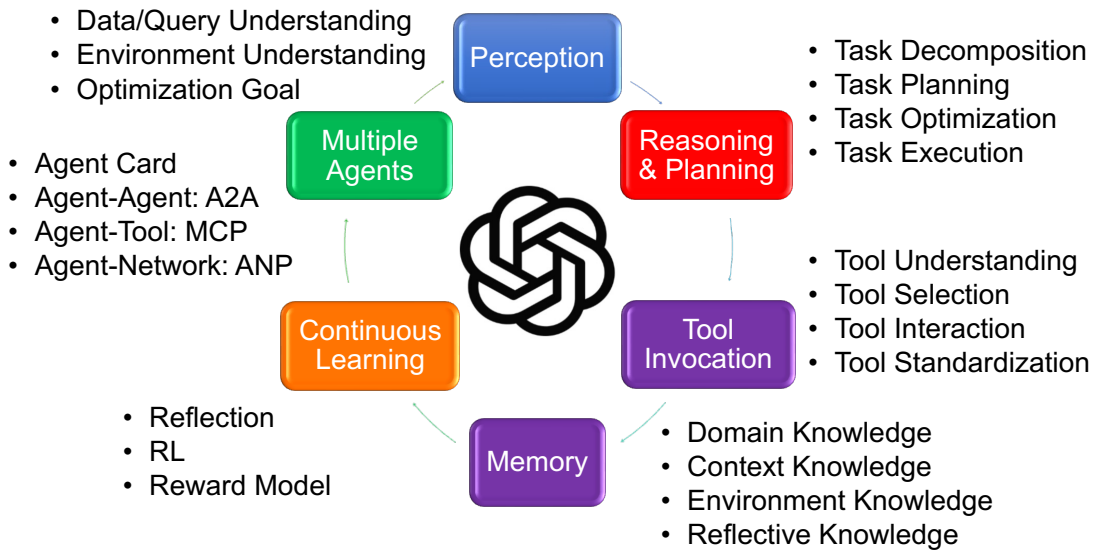


Figure 2: Key Factors of Data Agents.

Reasoning and Planning: Planning focuses on creating multi-step pipeline orchestration, while reasoning involves making single-step decisions or actions. Each action may require further exploration of reasoning/planning or invoking a tool (to acquire domain data or knowledge).

Tool Invocation: The agent can call upon tools to perform calculations, access domain-specific data, or provide instructions to environments. The Model Context Protocol (MCP) facilitates alignment between agents and tools, ensuring that information and states are exchanged in a standard format to prevent information drift [24]. Intermediate inference results from different models can be understood and reused across the system.

Memory: This includes long-term memory, such as domain-specific and environmental knowledge, and short-term memory, like user context. Typically, a vector database is used to store and query these memory data. Other types of memory, such as reflective memory, will also be used to enhance planning abilities and performance.

Continuous Learning: Continuously improving the agent to make it smarter is vital. This relies on self-reflection, reinforcement learning, and reward model techniques for self-improvement.

Multiple Agents: Individual agents may struggle to handle diverse tasks effectively, as each agent has its own strengths but also limitations. Thus, integrating multiple agents to collaborate and coordinate is necessary for complex tasks. This approach enhances system robustness and improves parallelism and efficiency.

We propose a comprehensive architecture for building data agents, encompassing data understanding and exploration, data engine understanding and scheduling, and pipeline orchestration, as shown in Figure 3. Figure 4 shows the detailed architecture.

Data Understanding and Exploration Agents aim to organize and understand data to facilitate discovery and access by the agent. A unified semantic catalog offers a well-structured metadata system (e.g., schema and metadata index), enhancing data access performance. The data fabric provides a unified view of heterogeneous data by linking and integrating diverse data, allowing easy data retrieval by the agent. Semantic data organization and semantic indexes are also very improve to improve the data agent efficiency. Importantly, there are numerous tools for data preparation, cleaning, and integration.

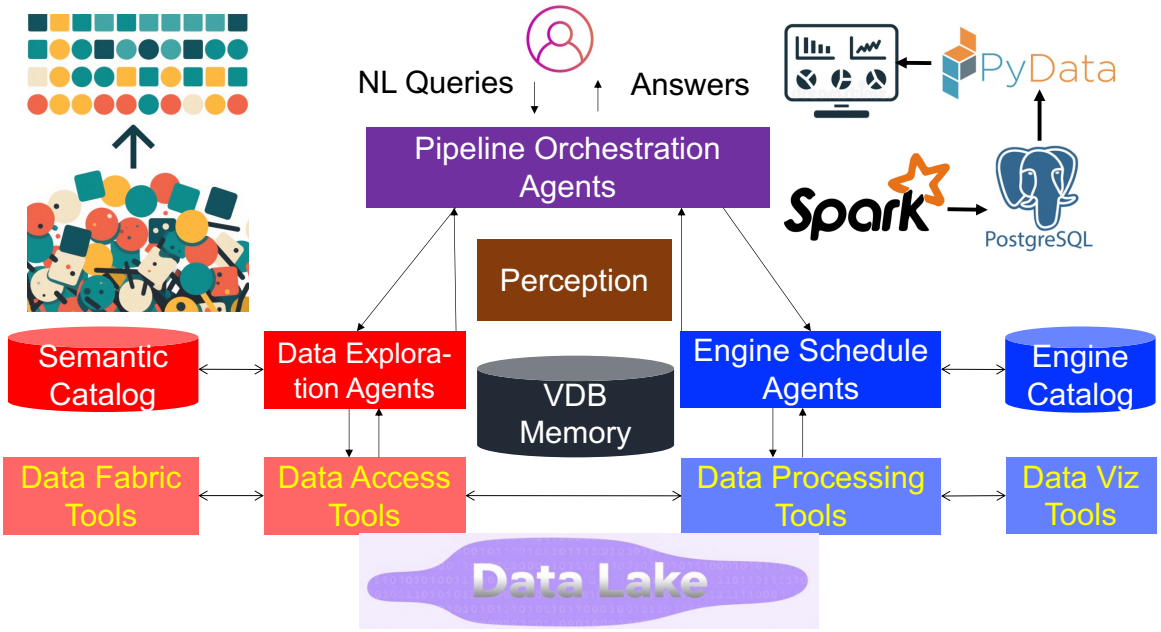


Figure 3: Data Agents Framework.

This component will also devise effective strategies to utilize these tools efficiently.

Data Engine Understanding and Scheduling Agents focus on comprehending and scheduling data processing engines, such as Spark, DBMSs, Pandas, and PyData. Given that different agents and tools have varying skill sets, it is essential to profile the specific capabilities of each engine and coordinate them to execute complex tasks effectively.

Pipeline Orchestration Agents are responsible for generating pipelines based on user-input natural language (NL) queries and the data catalog. They break down complex tasks into smaller, manageable sub-tasks that can be executed sequentially or in parallel to achieve the overall goal. Given that both NL queries and the underlying data exist in an open-world context, these agents must leverage the understanding, reasoning, and self-reflection capabilities of large language models (LLMs) to create high-quality plans. Subsequently, the pipelines need to be optimized to improve latency, cost, or accuracy, and engine agents are invoked to efficiently execute the pipelines.

Memory encompasses long-term memory, such as domain and environmental knowledge, as well as short-term memory, like user context and reflective context. Vector databases are typically used to manage this memory for enhancing the performance.

Perception is tasked with comprehensively understanding the surrounding environments and the specific tasks at hand.

Agent-Agent Interaction is designed to coordinate and collaborate multiple agents to tackle decomposed sub-tasks. It comprises three key components: agent profiling and selection, agent interaction and coordination, and agent execution. Agent Profiling and Selection involves building profiles for agents, enabling the system to choose the most suitable agents for specific tasks. Agent Interaction and Coordination focuses on the coordination and interaction of multiple agents to effectively address sub-tasks. Thanks to agent-to-agent (A2A) protocols, we can facilitate communication between agents and synchronize their statuses via A2A [25]. Agent Execution aims to execute multiple agents either in a pipeline or in parallel, enhancing the system's fault tolerance and fast recovery.

Agent-Tool Invoking is utilized for calling upon appropriate tools. Given the multitude of data

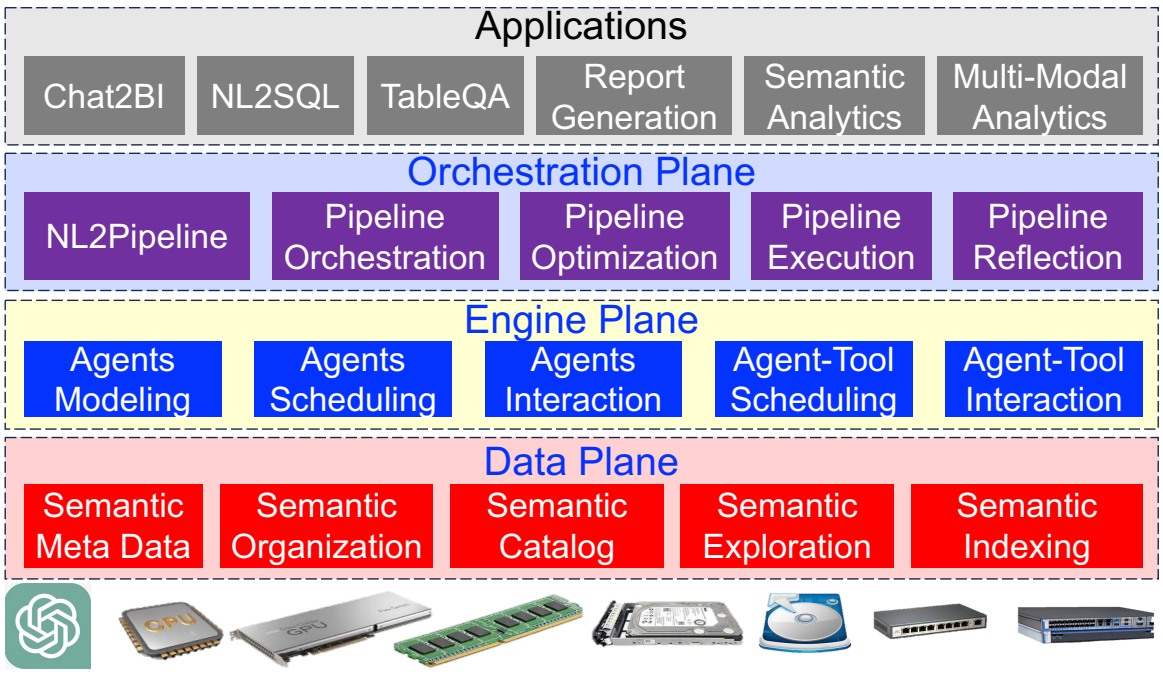


Figure 4: Data Agents Architecture.

processing tools available, such as Pandas and PyData, it is necessary to select the right tool for each task. The challenge lies in profiling and scheduling these tools effectively. Fortunately, the Model Context Protocol (MCP) allows us to easily integrate new tools into the data agent system.

3 *iDataScience*: A Multi-Agent System on Data Science

We first introduce the data agent architecture of *iDataScience* (see Figure 5), and then present the components of *iDataScience*.

3.1 Overview of *iDataScience*

iDataScience is designed to adaptively handle data science tasks by flexibly composing the complementary capabilities of diverse data agent, which is a challenging open research problem. As shown in Figure 5, *iDataScience* comprises an offline stage and an online stage.

Offline Data Agent Benchmarking. This stage aims to construct a comprehensive data agent benchmark that can cover diverse data science scenarios by composing basic data skills. First, given a large corpus of data science examples, we employ LLM for quality filtering and data skill discovery. Next, to organize the skills, we build a hierarchical structure via recursive clustering. Each skill is also assigned an importance score based on its overall frequency or user-defined priorities. Then, to reflect the capability requirements of specific data science scenario, we sample important skills probabilistically according to their scores and use LLM to generate corresponding test cases.

Besides, to ensure unbiased agent evaluation for an online task, benchmark test cases should be adaptively aggregated based on their similarity to the task. We thus construct an efficient index to enhance the performance of similarity search over the test cases. We will discuss the detailed techniques in Section 3.2.

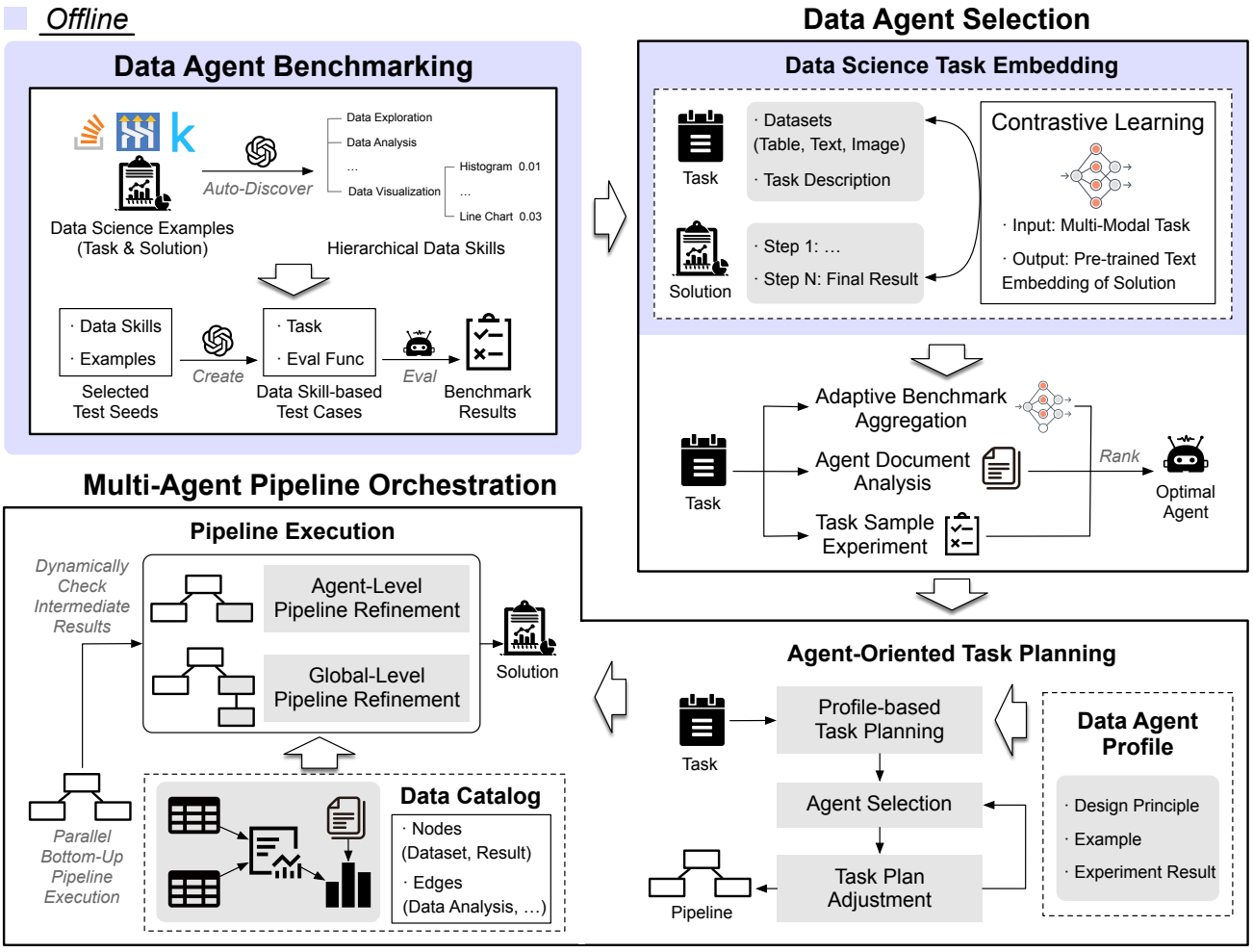


Figure 5: Overview of *iDataScience*.

Online Multi-Agent Pipeline Orchestration. Given an online data science task, this stage autonomously decomposes the task into a pipeline of sub-tasks aligned with data agent capabilities, selects an appropriate agent for each sub-task, and dynamically refines the pipeline to ensure both efficiency and robustness.

(1) Data Agent Selection. To effectively utilize benchmark results, we design a task embedding method to measure the similarity between benchmark test cases and the online task. To this end, we fine-tune a task embedding model to align task embeddings with the pre-trained text embeddings of corresponding task solutions, which capture the capability requirements and reasoning complexity inherent in the tasks. We then build an embedding index over the benchmark test cases. For an online task, we utilize the index to efficiently retrieve top-k relevant test cases, and evaluation scores of test cases are adaptively aggregated based on their similarity to guide agent selection. The data agent with the highest aggregated score is selected as optimal.

Besides, for special cases where benchmark is unsuitable, we can also evaluate the agents through structured document analysis or via sample task experiments. We will discuss the technical details in Section 3.3.

(2) Multi-Agent Pipeline Orchestration. Given an online task, we first use LLMs to decompose the task into a pipeline of interdependent sub-tasks using specialized agent profiles. Each sub-task is

assigned to an appropriate data agent selected as previously described. We also iteratively refine the plan with additional adjustments such as sub-task merging or decomposition.

Then, we execute the pipeline in a parallel bottom-up manner based on its topological order. To ensure robustness, we dynamically refine the pipeline based on intermediate results, including (*i*) sub-task modification at agent level, and (*ii*) global-level re-planning, where complete intermediate results are stored as datasets in our data catalog to prevent redundant computation. Once all sub-tasks are executed, *iDataScience* outputs the final task result to the user. We will discuss the technical details in Section 3.4.

Integration of New Data Agent. *iDataScience* is designed to be extensible, allowing integration of new data agents through agent profile construction based on agent document analysis. Additionally, when sufficient time and resources are available, *iDataScience* can further enhance the agent profile by executing the benchmark, thereby improving the accuracy of agent selection and pipeline orchestration. Once integrated, the new agent can seamlessly collaborate with existing agents within our multi-agent pipeline orchestration framework.

3.2 Data Agent Benchmarking

Given the wide range of data science tasks across diverse application domains, existing benchmarks are typically constrained to a limited set of pre-defined task types [26, 27], and thus poorly evaluate data agents in more flexible use cases. To address this limitation, we introduce a data skill-based benchmark, which can adaptively create test cases to cover different data science scenarios (see Figure 5). Specifically, we first use LLMs to extract data skills from a large corpus, whose compositions can represent the capability requirements of diverse tasks. Next, we construct a hierarchical structure that captures the semantic relationships among these skills, with each skill assigned an importance score based on its overall frequency or user-defined priorities. Then, during data agent evaluation, we can compose random sets of relevant skills tailored to the target scenario, and prompt LLMs to generate corresponding test cases for adaptive and comprehensive assessment.

3.2.1 Hierarchical Data Skill Discovery

Automatic Data Skill Discovery. To better understand the coverage of data science tasks, we first collect data science examples from diverse sources, including online websites (e.g., StackOverflow), professional competitions (e.g., Kaggle), and existing data science benchmarks. Each data science example typically contains three components: (*i*) "task": a natural language task description, as well as multi-modal datasets involved (e.g., tables, texts, images); (*ii*) "solution": a detailed sequence of steps outlining the procedure of completing the task; (*iii*) "associated data skills": the fundamental data science capabilities utilized in the solution, whose extraction procedure is described in the following paragraph.

We use LLMs to automatically filter out high-quality examples and discover associated data skills in three steps. First, since the data science example can contain noisy content and intricate code snippets, we provide the task description and solution to LLMs, and instruct it to summarize the procedure as a sequence of steps in concise natural language. Second, we utilize LLMs to evaluate the quality of example summaries, and exclude low-quality examples that either lack implementation details or provide only partial task solution. Third, given the summarized steps of data science examples, we further use LLMs to extract associated data skills [28]. For instance, given the example 'Do lower-income students perform worse on a math test than higher-income students?', its solution reveals the data skills: 'new column derivation', 'filter by column', and 'linear regression'.

Data Skill Hierarchy Construction. The mess of extracted data skills can be systematically organized into a hierarchical structure through recursively clustering semantically similar skills within the same

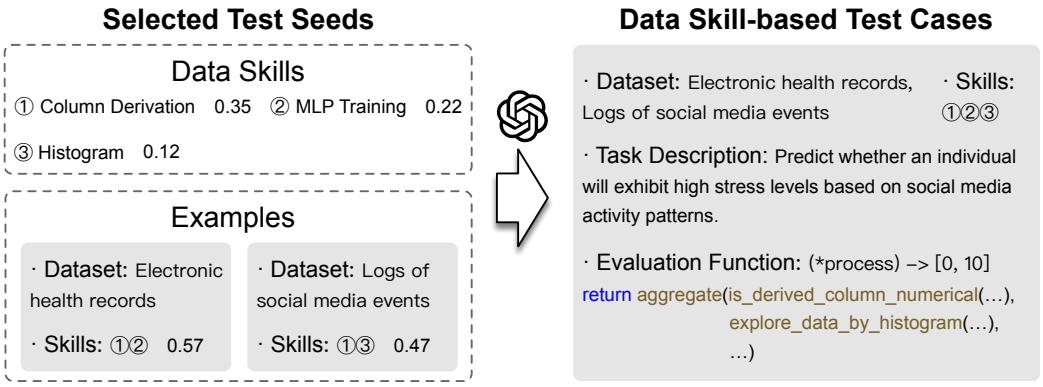


Figure 6: Example of Data Skill-based Benchmark Construction.

group. Specifically, we first use pre-trained text embeddings to represent the semantics of the data skills. Then, these embeddings are clustered using Gaussian Mixture Models [29], where Uniform Manifold Approximation and Projection method is used to reduce dimensionality by approximating the local data manifold [30]. Next, for each skill cluster, we instruct LLM to generate a summary that captures the common skill theme of these grouped skills. The cluster summaries serve as higher-level nodes in the hierarchy. If the skill number within some cluster falls below a pre-defined threshold, such clusters are designated as leaf nodes. Conversely, if some cluster still contains skills exceeding the threshold, the same clustering procedure is recursively applied to construct a more fine-grained hierarchical structure under the same parent cluster node.

Data Skill Weighting. To distinguish the relative importance of data skills, we initially use their occurrence frequencies across data science examples as a general proxy score. Specifically, for each leaf node in the skill hierarchy, we compile the skills within the corresponding cluster, and count the number of examples in which any of these skills are associated. The score of the leaf node is calculated as the ratio of its associated example count to the total number of examples. Then, the scores of higher-level nodes can be recursively computed by aggregating the scores of their corresponding children nodes.

Besides, we also support users to adaptively adjust the scores of critical data skills for practical needs. For instance, if users identify 'missing value handling' as a more critical data cleaning skill, the score of corresponding node can be manually increased. To maintain a constant total score among sibling nodes, the scores of the remaining sibling nodes are proportionally decreased. The scores of descendant nodes are also adjusted accordingly, preserving their relative proportions within their respective sub-trees.

3.2.2 Data Skill-based Benchmark Construction

The test case in the benchmark consists of two components: (*i*) "task": a task description along with associated datasets; (*ii*) "evaluation function": an executable function that outputs a scalar value within the range $[0, 10]$, quantifying the correctness and performance of data agents. However, directly using LLMs to generate test cases leads to low-quality benchmarks for two reasons. First, the generated test cases cannot fully capture the data science capabilities required for specific scenario. Second, the test case may not adequately reflect conditions in real-world applications.

To address these challenges, we propose a data skill-based benchmark construction method (see Figure 6). First, we sample a random set of k data skills from the leaf nodes of the hierarchy, with probabilities proportional to their importance scores. Second, to enhance the realism of the generated test case, we retrieve data examples from the corpus that are pertinent to the selected skills, and incorporate them into LLM's input context for in-context learning. Specifically, we assign a relevance

score to each data science example based on its associated data skills. The score of an example is computed as the sum of the importance scores of the k selected skills present in its associated skill set. We select the examples with top scores as representative.

Then, we provide LLM with the selected data skills and data science examples, instructing it to synthesize test cases by leveraging both its pre-trained knowledge and in-context examples. The synthesis process adheres to the following criteria: (*i*) the datasets must be drawn from those provided in the examples; (*ii*) the synthesized task and evaluation function should both involve the application of all the specified data skills; and (*iii*) the evaluation function should be composed of a set of sub-functions, each of which returns a boolean value indicating whether a specific evaluation criterion is satisfied by the task results. The final scalar score is then computed based on the aggregate results of these sub-functions, a design choice that promotes greater consistency across different test cases [31].

Two special cases also require consideration. First, if the dataset size of some selected example exceeds the context window that LLMs can effectively process [32], a subset of the dataset can be randomly sampled, supplemented with its metadata (e.g., column descriptions of the table) to preserve essential information. Second, if certain evaluation standards are too ambiguous to be directly implemented as executable code, LLMs can be invoked within the evaluation function to assess the task results guided by a set of generated rules.

Note that the complexity of test cases grows rapidly with the number of data skills k [33], which users can specify to tailor the benchmark to practical applications.

3.3 Data Agent Selection

Since many data agents have been proposed to address similar data science tasks [26, 27], a straightforward approach to agent selection is to leverage their benchmark results. Specifically, we can assess agent performance by aggregating the evaluation scores of test cases and selecting the agent with the highest overall score. However, this method overlooks the degree of relevance between benchmark test cases and the target task, and thus we should re-weight the test cases to ensure an unbiased assessment.

To address this issue, we first train an embedding model for data science tasks, where tasks with similar embeddings require similar data science competencies and reasoning process. The unbiased evaluation score of an agent is then computed using these embeddings, as a weighted aggregation of the test case evaluation scores, where the weights reflect the similarity of each test case to the target task. The agent achieving the highest score is regarded as the optimal choice.

Besides, we also propose alternative evaluation methods for the special cases where the benchmark is unsuitable. For time-constrained settings, the agent is evaluated through structured document analysis using LLM, without actually executing the benchmark. For tasks with extremely high execution time or resource demands (e.g., model training on large datasets), we generate samples from the target task and utilize them to further refine agent selection.

3.3.1 Data Science Task Embedding.

Since the data science task typically contains multi-modal datasets and textual descriptions, we can use multi-modal large language models (MLLMs) to obtain its semantic embedding [34], where both text and images are natively supported and tables can be represented as serialized text in CSV format. Large datasets can be substituted with data samples and metadata to fit the context window of MLLM, as discussed in Section 3.2.2. However, such embedding method is sensitive to irrelevant information like (*i*) domain of the task and (*ii*) representation format of datasets, despite these factors minimally affect actual complexity of solving the task.

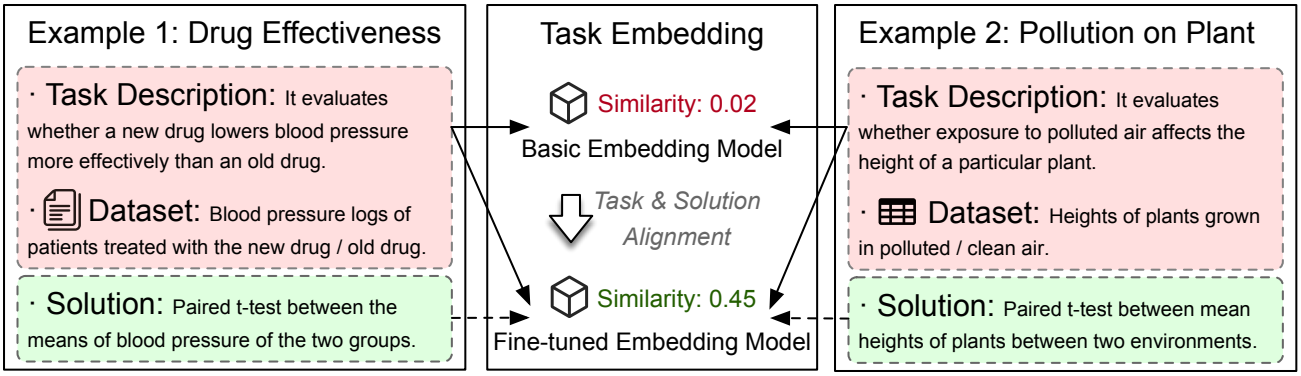


Figure 7: Example of Fine-tuned Data Science Task Embedding.

Table 1: Comparison of Data Agent Selection Methods.

Method	Accuracy	Online Overhead	Offline Overhead
Adaptive Benchmark Aggregation	Mid	Low	High
Agent Document Analysis	Low	Mid	Low
Task Sample Experiment	High	High	Low

To overcome this limitation, we introduce a fine-tuning process that aligns data science task embedding with the pre-trained text embedding of its correct solution (see Figure 7). We again use the corpus of high-quality data science examples annotated with associated data skills, as discussed in Section 3.2.1. We employ a contrastive learning framework: correct solutions to the task are treated as a positive example, while solutions to tasks involving mutually exclusive data skills serve as negative examples. Using these training pairs, we compute the Multiple Negatives Ranking Loss [35], which minimizes embedding distances for positive pairs and maximizes embedding distances for negative pairs. Once training converges, the embeddings exhibit meaningful alignment, allowing similarity metrics to reliably identify data science tasks with analogous solution procedures, which in turn reflect comparable performance of a data agent.

3.3.2 Heterogeneous Data Agent Selection

We propose three methods for selecting the most suitable data agent for a given task, i.e., adaptive benchmark aggregation, agent document analysis, and task sample experiment, each offering distinct advantages and disadvantages (see Table 1).

Adaptive Benchmark Aggregation. In the offline stage, we run the benchmark using the candidate data agents, resulting in each agent associated with evaluation scores corresponding to each test case. We also build an embedding index over the test cases using fine-tuned task embedding model to support efficient similarity search. Then, for an online task, we similarly generate its embedding and identify the most relevant test cases with top-k embedding similarities. The weights assigned to these k selected test cases are their normalized similarities such that their sum is equal to one. We thus obtain an unbiased evaluation score for each agent, which is computed by the weighted sum of its evaluation scores across these selected test cases. The agents are then ranked by the aggregated scores, and the top one with the highest score is identified as the most suitable choice.

Agent Document Analysis. Sometimes, it is impractical to evaluate the data agent by executing time-intensive benchmark. For instance, when integrating a new agent into *iDataScience*, a temporary

evaluation method is required to enable its inclusion in the agent selection framework prior to completing its full benchmark evaluation. Thus, given an online task, we can assess the data agent by using LLM to directly analyze agent documents (e.g., research paper, repository Readme files).

Specifically, we instruct LLMs to focus on three structured artifacts of the document, which are critical for agent evaluation: (*i*) Design Principle. It describes the intended usage scenarios and technical contributions of the agent, which are often extracted from the introductory sections of the documents. We can use LLMs to compare the given task with these design specifications, determining whether the agent is appropriately targeted for the task. (*ii*) Representative Example. It can be treated analogously to test cases in the benchmark, where higher embedding similarity to the given task indicates greater suitability of the agent. (*iii*) Experiment Result. It describes the experimental results of the agent on various datasets in comparison with baselines. First, if details regarding the datasets or baselines are missing, we can prompt LLMs to use search engine as tools to retrieve and complete the missing information. Then, we use LLMs to compare the target task with the experimental datasets, and predict the agent's performance based on reported experimental results.

Finally, guided by the analysis of aforementioned three key artifacts, we employ LLMs to generate an evaluation score in the range $[0, 10]$, which is comparable to benchmark scores.

Task Sample Experiment. If the given task involves large datasets or demands large resources, agent selection can be further refined through sample experiments. Specifically, we randomly sample the task dataset to construct a task sample. Next, we select the agents with the top scores based on the evaluation and execute the task sample using these candidates. Finally, we compare their execution results and use LLMs to determine which agent most effectively solves the task sample.

3.4 Multi-Agent Pipeline Orchestration

For complex data science tasks that exceed the capabilities of any single data agent [36–38], it is essential to compensate for their limitations through a unified collaborative framework. However, most existing work define a deterministic pipeline with a fixed set of data agents based on human expertise, limiting their ability to leverage the broader range of available agents through extensible integration [39]. To this end, we propose a flexible and adaptive pipeline orchestration algorithm that accommodates diverse tasks and agents, and effectively manages the intricate reasoning process through agent collaboration. Specifically, we first introduce an agent-oriented task planning approach that decomposes the given task into a pipeline of sub-tasks by leveraging specialized agent profiles, heterogeneous agent selection for each sub-task, and LLM-based plan adjustment to ensure both effectiveness and efficiency. Next, we execute the pipeline interactively, with dynamic pipeline refinement to adapt to intermediate results through either agent-level modifications or holistic re-planning.

3.4.1 Agent-Oriented Task Planning

Following existing studies [40], we feed the given task and a set of agent profiles to LLMs, and instruct LLMs to generate a task plan comprising sub-tasks, each associated with a relevant subset of datasets and aligned with capabilities of some agents. We further prompt LLMs to evaluate the task plan for completeness and non-redundancy, ensuring that the sub-tasks collectively address the given task in its entirety while avoiding unnecessary or duplicate sub-tasks. Our approach is distinguished by three key aspects: (*i*) design of specialized data agent profiles, (*ii*) heterogeneous agent selection for each sub-task, and (*iii*) adaptive task plan adjustment.

Data Agent Profile Construction. To fully describe the capabilities of the data agent, we construct the agent profile comprising three components: (*i*) "design principle", (*ii*) "representative example", and (*iii*) "experiment result", paralleling the key artifacts extracted from agent document as discussed

in Section 3.3.2. We can enhance the agent profile by including its benchmark results in addition to document information. Specifically, for the "representative example" component of an agent, we incorporate benchmark test cases by quantifying its performance deviation on each test case t_i as $\frac{s_i - mean_i}{mean_i}$, where s_i denotes the evaluation score of the given agent on t_i, and $mean_i$ represents the average score of t_i across all data agents. Test cases exhibiting the highest and lowest deviation scores are selected as positive and negative examples respectively. For the "experiment result" component, we further incorporate overall benchmark metrics (e.g., accuracy, recall), using the seed data skills associated with the benchmark as descriptive metadata. Note that our agent profile design is also compatible with the "Agent Card" interface defined by the Agent-to-Agent (A2A) protocol [25].

Heterogeneous Data Agent Selection. To construct the context for each sub-task during agent selection, we first employ LLMs to analyze the dependency relationship among sub-tasks in the task plan, representing them as a directed graph with sub-tasks as nodes and dependencies as edges. Next, for a given sub-task, we treat the intermediate results of its dependent sub-tasks as auxiliary datasets, and prompt LLMs to synthesize mock dataset descriptions to serve as the sub-task input. Then, we apply the agent selection method described in Section 3.3 to identify the most suitable agent for the sub-task.

Task Plan Adjustment. After sub-task decomposition and agent assignment, we further refine the task plan by applying adjustments such as additional decomposition and merging. First, if no suitable data agent can be selected for a given sub-task (e.g., all agents with evaluation scores below a pre-defined threshold), we instruct LLM to revise the task plan by further decomposing the sub-task into simpler sub-tasks.

Besides, to enhance efficiency of task plan, we attempt to merge correlated sub-tasks, thereby reducing the complexity of execution pipeline. Specifically, we first provide the dependency graph of sub-tasks to LLMs and instruct LLMs to identify sub-graphs where the nodes (*i*) involve interrelated datasets, and (*ii*) can be coherently aggregated into a larger, logically consistent sub-task. Next, for each identified sub-graph, we prompt LLMs to generate a consolidated sub-task that summarizes the constituent sub-tasks within the sub-graph. Then, we apply the heterogeneous data agent selection method to identify a suitable agent for the new sub-task. If a suitable agent is successfully selected, the original sub-graph is replaced with the new sub-task and its corresponding agent assignment.

3.4.2 Pipeline Execution

Unlike traditional approaches that depend on fixed data agents and rigid workflows, *iDataScience* is designed to dynamically adapt to the diversity and unpredictability of both data agents and data science tasks. As illustrated in Figure 5, *iDataScience* can execute its pipeline interactively, dynamically refining its pipeline at both the agent level and the global level based on intermediate results and execution states.

Parallel Pipeline Execution. To enhance execution efficiency, following the dependency graph of sub-tasks, we execute the sub-tasks in parallel bottom-up along their topological order in the graph. Once their dependent sub-tasks are completed, each sub-task is executed using the corresponding intermediate results, proceeding iteratively until all sub-tasks are completed and the final result is obtained. During execution, each sub-task is carried out by its assigned agent, after which we use LLMs to verify whether the generated intermediate results adequately fulfill the sub-task. If discrepancies are identified, they trigger dynamic refinement of the pipeline to ensure robustness, as described next.

Agent-Level Pipeline Refinement. When a sub-task fails to be resolved by the assigned data agent, we first use LLMs to reflect potential failure causes based on the sub-task input, including: (*i*) sub-task description, (*ii*) dataset, and (*iii*) intermediate results. For example, if a failure stems from an ambiguous

sub-task description that the agent misinterprets, we use LLMs to rephrase it, drawing on insights from the failure logs. Next, if required datasets or intermediate results are missing, we use LLMs to further examine the available datasets and sub-tasks to identify supplementary information. Then, if intermediate results are improperly formatted, LLMs are employed to refine it by further analyzing the solution process of the corresponding sub-task.

Besides, if a failure cannot be attributed to unexpected sub-task inputs, it may result from an incorrect agent selection. In such cases, we review the ranking of data agents based on their evaluation scores for the given sub-task, and select the next-best agent with a slightly lower score than the previously chosen one.

Lastly, we re-execute the sub-task using refined input information or a newly assigned data agent.

Global-Level Pipeline Refinement. If a sub-task failure cannot be resolved via agent-level refinement, a holistic re-planning of the entire pipeline is required. First, during pipeline execution, we systematically store the intermediate results of sub-tasks in a data catalog (see Figure 5), which ensures preservation of previously computed results and avoids redundant computations. Specifically, the output generated by each sub-task is treated as a new dataset. For this dataset, metadata is created by LLMs based on the sub-task's input and the process used to solve it. These newly formed datasets are then incorporated into the data catalog. Then, during task re-planning, we apply the agent-oriented task planning algorithm in Section 3.4.1, treating all elements in the data catalog as available datasets.

4 Data Analytics Agents

We first summarize the overview of data analytics agents [19, 41]. We then introduce four data analytics agents, including unstructured data analytics agents, semantic structured data analytics agents, data lake analytics agents, and multi-modal data analytics agents.

Overview of Data Analytics Agent. In the offline phase, the data analytics agent generates a semantic catalog and builds semantic indexes for a variety of data types. It also defines semantic operators, such as semantic filter, semantic group-by, semantic sorting, semantic projection, semantic join, and others. Each semantic operator is represented logically (e.g., as an entity that satisfies certain conditions) to facilitate matching natural language segments with these semantic operators. Each semantic operator is also associate with multiple physical operators, e.g., execution by LLMs, pre-programmed functions, LLM coding, etc. When processing an NL query, the data analytics agent decomposes the query into sub-tasks and orchestrates them into a pipeline. The agent then optimizes this pipeline by selecting the optimal sequence of semantic operators and executes the pipeline efficiently by calling the semantic operators.

Unstructured Data Analytics Agent. It supports semantic analytics on unstructured data using natural language queries [19]. The challenges include orchestrating a natural language query into a pipeline, self-reflecting on the pipeline, optimizing the pipeline for low cost and high accuracy, and executing the pipeline efficiently. We propose a logical plan generation algorithm that constructs logical plans capable of solving complex queries through correct logical reasoning. Additionally, we introduce physical plan optimization techniques that transform logical plans into efficient physical plans, based on a novel cost model and semantic cardinality estimation. Finally, we design an adaptive execution algorithm that dynamically adjusts the plan during execution to ensure robustness and efficiency. In addition, we can extract a semantic catalog for unstructured data and use it to guide pipeline orchestration.

Semantic Structured Data Analytics Agent. Existing database systems operate under a closed-world model, which limits their ability to support open-world queries. To overcome these limitations, we integrate databases with LLMs to enhance the capabilities of database systems. By using LLMs as

semantic operators, we can support open-world data processing functions such as semantic acquisition, extraction, filtering, and projection. This approach allows us to extend SQL to incorporate these LLM-powered semantic operators, creating what we call semantic SQL. Additionally, NL queries can be transformed into semantic SQL using specialized NL2SQL agents. To execute semantic SQL effectively, we propose three techniques. First, we replace some semantic operators with traditional operators. For instance, instead of using "semantic acquire the capital of China," we can use 'city = Beijing.' Second, we design a multi-step filtering process to accelerate the processing of semantic operators, including embedding-based filtering and small LLM-based filtering. Third, we estimate the cost of semantic operators to determine the most efficient order for executing multiple semantic operations.

Data Lake Analytics Agent. Its aim is to perform data analytics on semi-structured and unstructured datasets [41]. However, integrating LLMs into data analytics workflows for data lakes remains an open research problem due to the following challenges: heterogeneous data modeling and linking, semantic data processing, automatic pipeline orchestration, and efficient pipeline execution. To address these challenges, we propose a data lake agent designed to handle data analytics queries over data lakes. We introduce a unified embedding approach to efficiently link heterogeneous data types within a data lake. Additionally, we present a set of semantic operators tailored for data analytics over data lakes. Our iterative two-stage algorithm facilitates automatic pipeline orchestration, incorporating dynamic pipeline adjustment during query execution to adapt to intermediate results. Overall, the data lake agent represents a significant advancement in enabling high-accuracy, practical data analytics on data lakes. Unlike previous approaches that rely on lossy data extraction or are constrained by SQL's rigid schema, the data lake agent effectively employs the semantic understanding capabilities of LLMs to provide a more comprehensive and efficient solution.

Multi-modal Data Analytics Agent. We also design data agents to support multi-modal data, e.g., audio, video. First, the integration and management of heterogeneous data types, such as text, images, audio, and video, are critical yet challenging tasks that require robust frameworks for seamless merging into cohesive datasets. Representing the multi-modal data in a unified format demands advanced embedding techniques to maintain unique characteristics while enabling analysis. Semantic understanding across different modalities is essential for extracting meaningful insights, necessitating sophisticated NLP, computer vision, and audio processing algorithms. Additionally, designing a flexible query system capable of interpreting and executing complex multi-modal queries is crucial. Aligning and fusing the data from various modalities ensures a coherent data view, while scalable methods are necessary for handling increasing data volumes efficiently.

5 DBA Agent

Database administrators (DBAs) often face challenges managing multiple databases while providing prompt responses, as delays of even a few hours can be unacceptable in many online scenarios. Current empirical methods offer limited support for database diagnosing issues, further complicating this task. To address these challenges, we propose a DBA agent, which is a database diagnosis system powered by LLMs [42, 43]. This system autonomously acquires knowledge from diagnostic documents and generates well-founded reports to identify root causes of database anomalies accurately. The DBA agent includes several key components. The first is to extract knowledge from documentation automatically. The second is to generate prompts based on knowledge matching and tool retrieval. The third conducts root cause analysis using a tree search algorithm. The fourth optimizes execution pipelines for high efficiency. Our results demonstrate that the DBA agent significantly outperforms traditional methods and standard models like GPT-4 [22] in analyzing previously unseen database anomalies.

6 Opportunities and Challenges

Theoretical Guarantee. A Data Agent may not always deliver 100% accurate results due to potential hallucinations from LLMs and semantic operators [44]. Therefore, it is crucial to provide a theoretical guarantee for the reliability of data agent systems.

Self-Reflection and Reward Model. A Data Agent needs to continuously enhance its accuracy and efficiency. Providing feedback to the Data Agent is essential for its self-improvement, and designing effective self-reflection techniques and reward models are viable strategies to tackle these challenges.

Data Agent Benchmark. It is crucial to develop benchmarks for Data Agents to evaluate these systems effectively, particularly in areas like data science, data analytics, and database administration.

Security and Privacy. Ensuring security and privacy in data agents involves protecting sensitive information from unauthorized access while maintaining compliance with privacy regulations.

Scalability and Performance. Scalability and performance challenges for data agents involve efficiently managing large and complex datasets while maintaining high processing speed and accuracy. As data volume and complexity grow, agents must be designed to scale seamlessly without degradation in performance.

7 Conclusion

In this paper, we propose the concept of a Data Agent for autonomously supporting Data+AI applications. We design a comprehensive architecture for developing a Data Agent, which includes components for data understanding and exploration, data engine comprehension and scheduling, and pipeline orchestration. We also present examples such as the data science agent, data analytics agents, and a DBA agent. The introduction of the Data Agent will present numerous challenges that require attention from the data community to address effectively. Moreover, there are numerous opportunities to develop data agents in various areas, including database development, database design, data transformation, data flywheel, and data fabric, etc.

Acknowledgments

This paper was supported by National Key R&D Program of China (2023YFB4503600), NSF of China (62525202, 62232009), Shenzhen Project (CJGJZD20230724093403007), Zhongguancun Lab, Huawei, and Beijing National Research Center for Information Science and Technology (BNRist). Guoliang Li is the corresponding author.

References

[1] G. Li, X. Zhou, L. Cao. AI meets database: AI4DB and DB4AI. *SIGMOD*, 2021.

[2] Y. Wu, X. Zhou, Y. Zhang, G. Li. Automatic Database Index Tuning: A Survey. *TKDE*, 2024.

[3] W. Zhou, C. Lin, X. Zhou, G. Li. Breaking It Down: An In-Depth Study of Index Advisors. *VLDB*, 2024.

[4] Y. Han, C. Chai, J. Liu, G. Li, C. Wei, et al. Dynamic materialized view management using graph neural network. *ICDE*, 2023.

[5] X. Zhou, G. Li, J. Feng, L. Liu, W. Guo. Grep: A graph learning based database partitioning system. *SIGMOD*, 2023.

[6] X. Zhao, X. Zhou, G. Li. Automatic database knob tuning: A survey. *TKDE*, 2023.
[7] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, et al. Bao: Making learned query optimization practical. *SIGMOD*, 2021.
[8] X. Zhou, G. Li, C. Chai, J. Feng. A learned query rewrite system using monte carlo tree search. *VLDB*, 2021.
[9] X. Yu, C. Chai, G. Li, J. Liu. Cost-based or learning-based? A hybrid query optimizer for query plan selection. *VLDB*, 2022.
[10] J. Wang, C. Chai, J. Liu, G. Li. FACE: A normalizing flow based cardinality estimator. *VLDB*, 2021.
[11] J. Sun, J. Zhang, Z. Sun, G. Li, N. Tang. Learned cardinality estimation: A design space exploration and a comparative evaluation. *VLDB*, 2021.
[12] B. Hilprecht and C. Binnig. Zero-Shot Cost Models for Out-of-the-box Learned Cost Prediction. *VLDB*, 2022.
[13] J. Wehrstein, C. Binnig, F. Özcan, S. Vasudevan, Y. Gan, et al. Towards Foundation Database Models. *CIDR*, 2025.
[14] X. Zhou, J. Sun, G. Li, J. Feng. Query performance prediction for concurrent queries using graph embedding. *VLDB*, 2020.
[15] Z. Sun, X. Zhou, G. Li. Learned index: A comprehensive experimental evaluation. *VLDB*, 2023.
[16] Y. Guo, G. Li, R. Hu, Y. Wang. In-database query optimization on SQL with ML predicates. *The VLDB Journal*, 34 (1), 12, 2025.
[17] C. Chai, J. Liu, N. Tang, J. Fan, D. Miao, et al. In-database query optimization on SQL with ML predicates. *SIGMOD*, 2023.
[18] S. Siddiqi, R. Kern, M. Boehm. SAGA: A Scalable Framework for Optimizing Data Cleaning Pipelines for Machine Learning Applications. *SIGMOD*, 2023.
[19] J. Wang, G. Li. Aop: Automated and interactive llm pipeline orchestration for answering complex queries. *CIDR*, 2025.
[20] X. Zhou, C. Chen, K. Li, B. He, M. Lu, et al. Febench: A benchmark for real-time relational data feature extraction. *VLDB*, 2023.
[21] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, et al. Ray: A Distributed Framework for Emerging AI Applications. *OSDI*, 2018.
[22] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, et al. GPT-4o System Card. *arXiv:2410.21276*, 2024.
[23] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv:2501.12948*, 2025.
[24] Anthropic. Introducing the Model Context Protocol. *Newsroom Anthropic*, 2024.
[25] R. Surapaneni, M. Jha, M. Vakoc, T. Segal. Announcing the Agent2Agent Protocol (A2A). *Google for Developers*, 2025.
[26] X. Hu, Z. Zhao, S. Wei, Z. Chai, Q. Ma, et al. InfiAgent-DABench: evaluating agents on data analysis tasks. *ICML*, 2024.
[27] L. Jing, Z. Huang, X. Wang, W. Yao, W. Yu, et al. DSBench: How Far Are Data Science Agents to Becoming Data Science Experts?. *ICLR*, 2025.
[28] A. Didolkar, A. Goyal, N. R. Ke, S. Guo, M. Valko, T. Lillicrap, et al. Metacognitive capabilities of llms: An exploration in mathematical problem solving. *NIPS*, 2024.
[29] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, et al. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. *ICLR*, 2024.
[30] L. McInnes, J. Healy, J. Melville. UMAP: Uniform Manifold Approximation and Projection for

Dimension Reduction. *arXiv:1802.03426*, 2020.

[31] Z. Liu, P. Wang, R. Xu, S. Ma, C. Ruan, et al. Inference-time scaling for generalist reward modeling. *arXiv:2504.02495*, 2025.

[32] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, et al. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*, 12, 2024.

[33] D. Yu, S. Kaur, A. Gupta, J. Brown-Cohen, A. Goyal, et al. Skill-Mix: a Flexible and Expandable Family of Evaluations for AI models. *ICLR*, 2024.

[34] Z. Jiang, R. Meng, X. Yang, S. Yavuz, Y. Zhou, et al. Vlm2vec: Training vision-language models for massive multimodal embedding tasks. *ICLR*, 2025.

[35] N. Reimers, I. Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *EMNLP-IJCNLP*, 2019.

[36] J. Fine, M. Kolla, I. Soloducho. Data Science Agent in Colab: The future of data analysis with Gemini. *Google for Developers*, 2025.

[37] S. Hong, Y. Lin, B. Liu., B. Liu, B. Wu, et al. Data Interpreter: An LLM Agent For Data Science. *arXiv:2402.18679*, 2024.

[38] Z. You, Y. Zhang, D. Xu, Y. Lou, Y. Yan, et al. DatawiseAgent: A Notebook-Centric LLM Agent Framework for Automated Data Science. *arXiv:2503.07044*, 2025.

[39] P. Trirat, W. Jeong, Wonyong, S. J. Hwang. AutoML-Agent: A Multi-Agent LLM Framework for Full-Pipeline AutoML. *ICML*, 2025.

[40] A. Li,Y. Xie, S. Li,F. Tsung, B. Ding, et al. Agent-Oriented Planning in Multi-Agent Systems. *ICLR*, 2025.

[41] J. Wang, G. Li, J. Feng. iDataLake: An LLM-Powered Analytics System on Data Lakes. *IEEE Data Eng. Bull.*, 49(1), 57-69, 2025.

[42] X. Zhou, G. Li, Z. Sun, Z. Liu, W. Chen, et al. D-bot: Database diagnosis system using large language models. *VLDB*, 2024.

[43] Z. Sun, X. Zhou, J. Wu, W. Zhou, G. Li. D-Bot: An LLM-Powered DBA Copilot. *SIGMOD-Companion*, 2025.

[44] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2), 1-55, 2025.

Data Engineering

It's FREE to join!

TCDE

tab.computer.org/tcde/

The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society is concerned with the role of data in the design, development, management and utilization of information systems.

- Data Management Systems and Modern Hardware/Software Platforms
- Data Models, Data Integration, Semantics and Data Quality
- Spatial, Temporal, Graph, Scientific, Statistical and Multimedia Databases
- Data Mining, Data Warehousing, and OLAP
- Big Data, Streams and Clouds
- Information Management, Distribution, Mobility, and the WWW
- Data Security, Privacy and Trust
- Performance, Experiments, and Analysis of Data Systems

The TCDE sponsors the International Conference on Data Engineering (ICDE). It publishes a quarterly newsletter, the Data Engineering Bulletin. If you are a member of the IEEE Computer Society, you may join the TCDE and receive copies of the Data Engineering Bulletin without cost. There are approximately 1000 members of the TCDE.

Join TCDE via Online or Fax

ONLINE: Follow the instructions on this page:

www.computer.org/portal/web/tandc/joinatc

FAX: Complete your details and fax this form to **+61-7-3365 3248**

Name _____

IEEE Member # _____

Mailing Address _____

_____

Country _____

Email _____

Phone _____

TCDE Mailing List

TCDE will occasionally email announcements, and other opportunities available for members. This mailing list will be used only for this purpose.

Membership Questions?

Xiaoyong Du
Key Laboratory of Data Engineering and Knowledge Engineering
Renmin University of China
Beijing 100872, China
duyong@ruc.edu.cn

TCDE Chair

Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, QLD 4072, Australia
zxf@uq.edu.au

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314

Non-profit Org.
U.S. Postage
PAID
Los Alamitos, CA
Permit 1398