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.

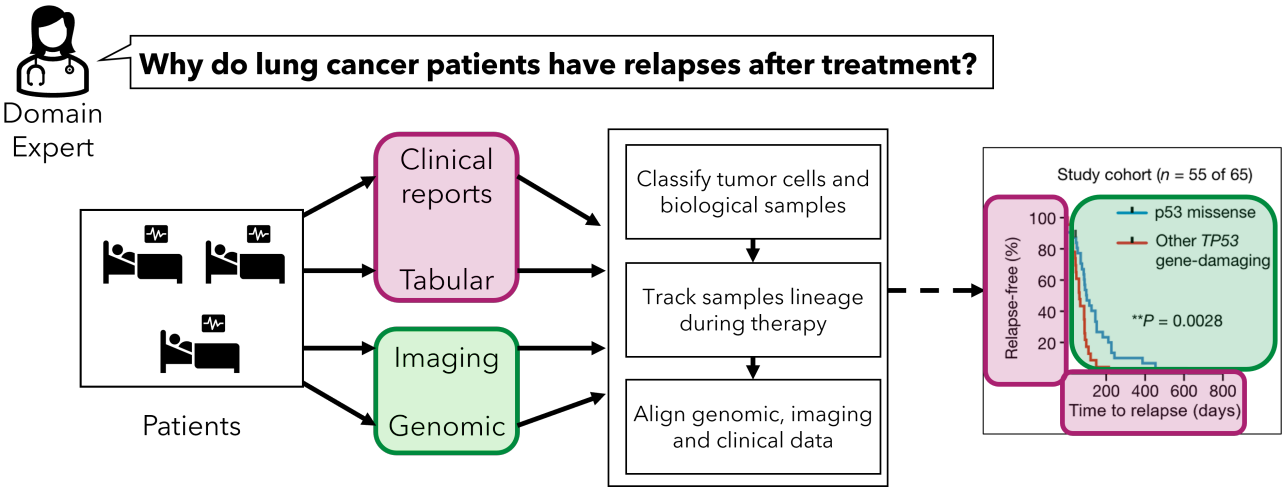

Figure 1: An example data-to-insight pipeline inspired from [15]. To answer and obtain insights about a clinical research question, biomedical researchers are required to manually integrate and process multimodal data.

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

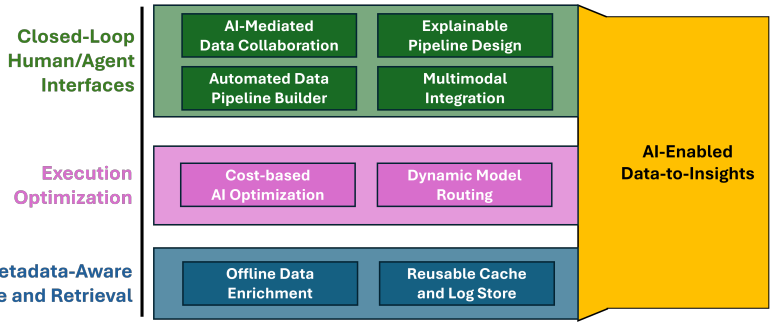

Figure 2: An overview of the components we are building towards an AI-enabled data-to-insight system.

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.

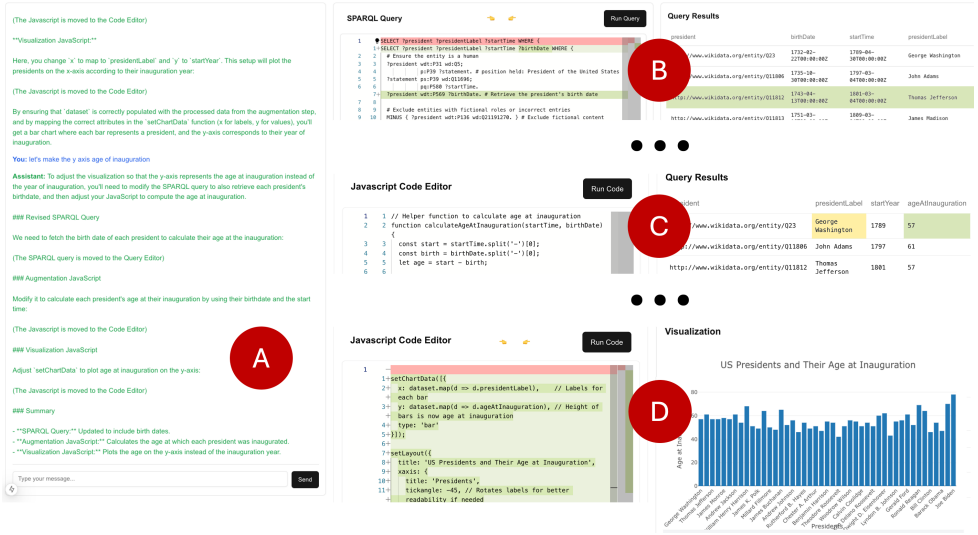

Figure 3: An early prototype of the Sunroom system, with a chat box (A) on the left hand side, and three code and output panels on the right hand side illustrating the pipeline: (B) runs a SPARQL query on a knowledge graph and returns a table; (C) cleans and prepares the data in Javascript, returning 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.

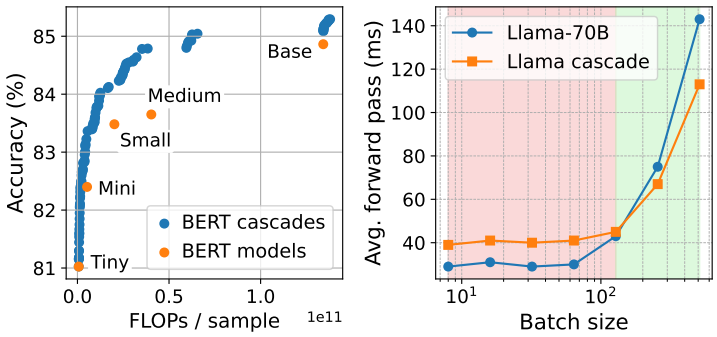

Figure 4: Left: Pareto frontier of FLOPs vs. accuracy for fine-tuned BERT classifiers on Sentiment-140 [16]. Right: Runtimes of Llama-70B and a cascade incurring 84% of its FLOPs.

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

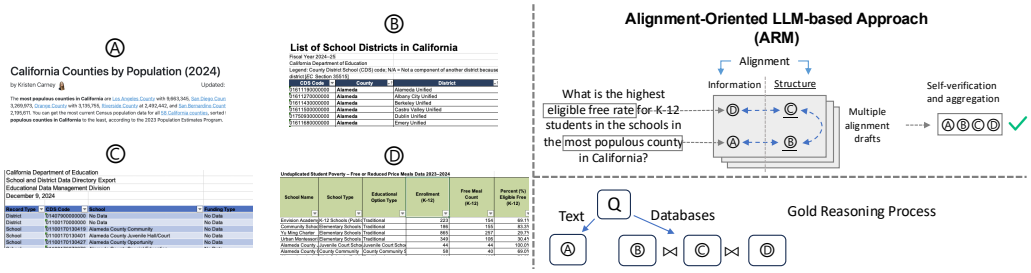

Figure 5: Overview of ARM, our alignment-based retrieval method for join-aware retrieval.

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.