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.

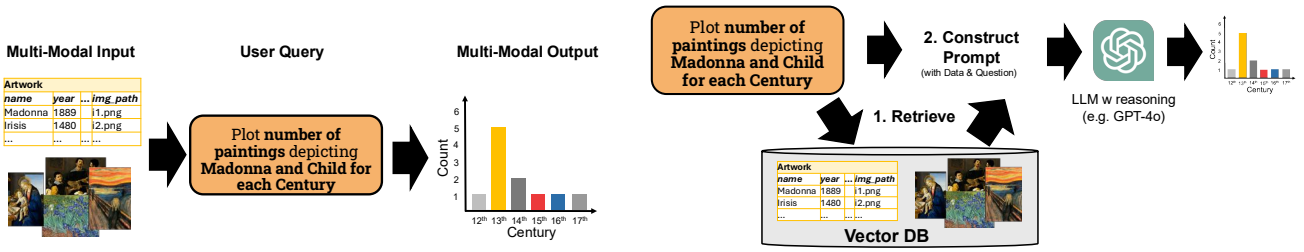
Instead of specifying how to retrieve data, users state what they want in a language that is close to natural language but with clear semantics. This was a profound shift away from imperative query interfaces required by hierarchical or network models.

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:

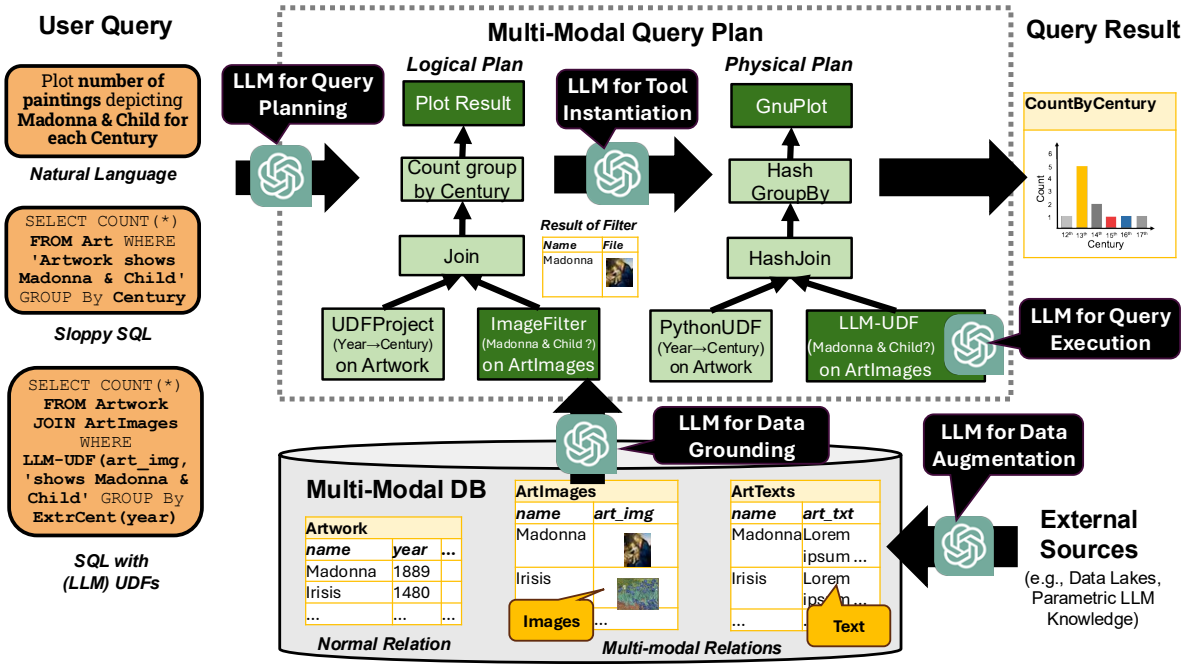


Figure 2: Sketch of an LLM-Augmented DBMS for Multi-modal Question Answering. The idea of an LLM-augmented DBMS is centered around a multi-modal relational data model where relations can include complex types such as images and text. Building on this model, the system supports several query interfaces 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.

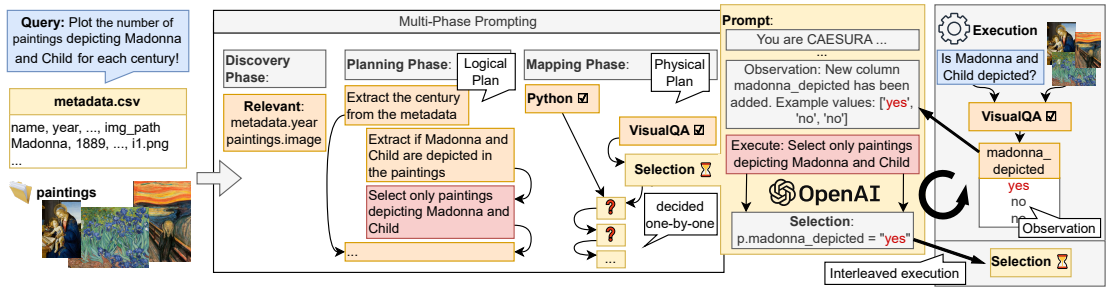


Figure 3: Query Planning in CAESURA. CAESURA transforms the query into a multi-modal query-plan using a series of prompts using an LLM. In the *Discovery Phase*, the LLM is prompted to identify data items relevant for the query, such as relevant columns and datasets. In the *Planning Phase*, the LLM 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.

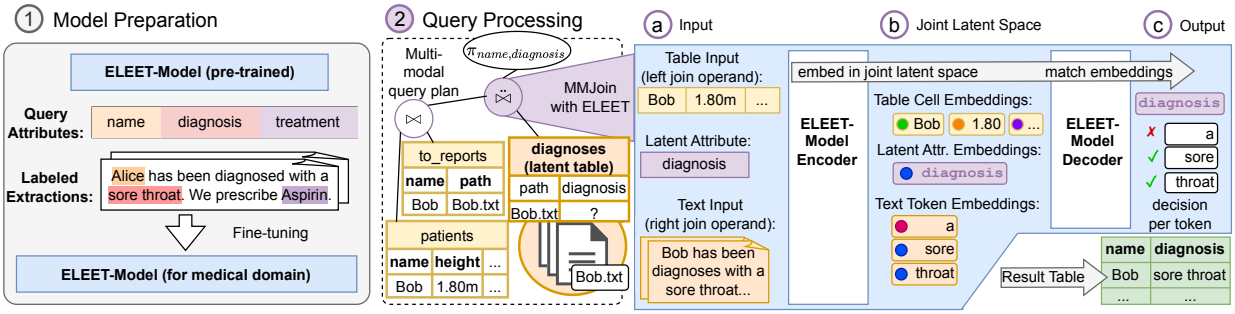


Figure 5: Overview of ELEET. In an offline phase, the ELEET-model can be fine-tuned for unseen domains ①. Fine-tuning the ELEET model for an unseen domain is a one-time effort and requires a small sample of a few labeled texts. ② For query execution, ELEET uses multi-modal query plans that contain traditional (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.

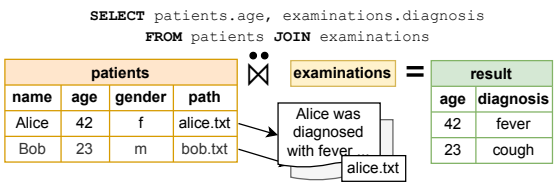


Figure 4: ELEET Example. A query that executes a multi-modal join between a patient table and examination reports. ELEET analyzes the texts and extracts values for the diagnosis from each examination report.

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

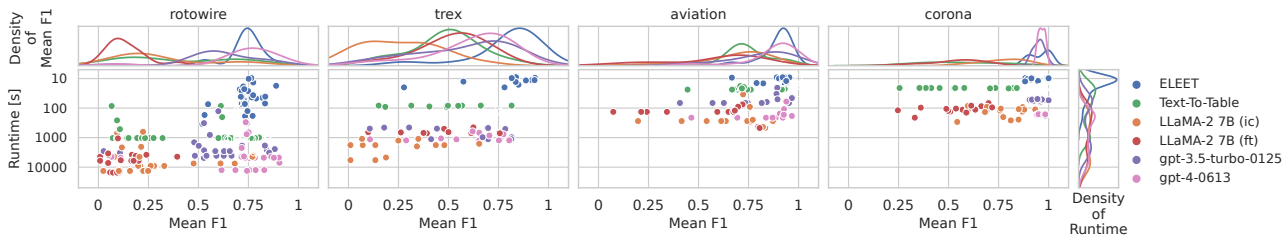


Figure 6: Performance and Accuracy comparison of ELEET with GPT-4 (175B), LLaMA-2 (7B), and T5 (11B) on multi-modal query execution tasks across different datasets. The plot shows F1 score (y-axis) versus query execution time (x-axis), where lower values of time and higher values of F1 score indicate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

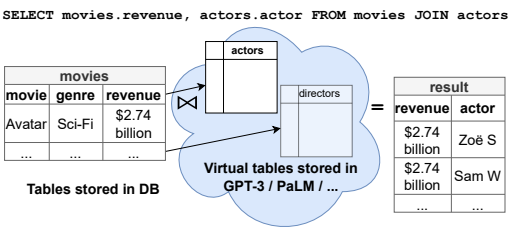


Figure 7: OmniscientDB Example. A SQL Query that joins a *movies* table stored in the database with an *actors* virtual table derived from an LLM (e.g., GPT-4). The join operation allows augmenting the *movies* table with actor information using a simple SQL query.

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.