Bulletin of the Technical Community on

Data Engineering

Mar 2025 Vol. 49 No. 1 IEEE Computer Society

Letters

Special Issue on Large Language Models and Data Systems

Conference and Journal Notices

The TC on Data Engineering

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

The Data Engineering Bulletin

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

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

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

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

Letter from the Editor-in-Chief

Traditional data management systems built on relational models have, for decades, provided the bedrock of reliable data storage and efficient queries. Yet their inherently rigid structure can constrain advanced analytics and sophisticated automation—limitations that have grown more pronounced as applications demand ever richer insight from increasingly varied data. Large Language Models (LLMs) promise a disruptive new chapter, offering a deep, context-aware grasp of both data and user intent. By moving beyond the static assumptions of relational schemas, LLMs can potentially transform how we query, integrate, and analyze data, easing the manual labor of transformations and freeing us to explore more intuitive, semantic ways of managing information.

We are, however, still in the early days of tapping these capabilities for real-world data systems. It remains unclear which designs or methodologies will prove most effective, how well these models will scale, or the degree to which they can seamlessly merge with long-standing DBMS technologies. Some paths will lead to breakthroughs in usability and analytics power; others may reveal unforeseen complexities that challenge our notions of correctness, performance, and trust. It is precisely this uncertainty—coupled with the tremendous opportunities—that makes LLMs such an exciting frontier for data management research and practice.

This issue explores how LLMs open transformative possibilities in data management. For example, instead of relying on tightly defined schemas or specialized data formats, LLMs enable a more flexible approach that blends both structured and unstructured information. Their capacity for richer semantic understanding can alleviate long-standing burdens in tasks like data integration and data quality, where mismatched schemas and sparse documentation often lead to labor-intensive fixes.

Equally important, LLMs encourage us to rethink analytics from a more holistic perspective. They provide a single interpretive layer for text, time-series, images, relational tables, and other data types, allowing diverse inputs to be processed with greater coherence and context-awareness. This unifying capability can make data exploration more intuitive, especially in large-scale environments where data is varied and often lacks a strict organizational scheme. In doing so, LLMs help unlock insights that might otherwise remain obscured, fostering a new level of accessibility and analytic power for modern data systems.

The contributors to this issue offer important glimpses into what the age of LLMs and generative AI might look like for data management. Trummer's work on operator-level customizations reveals how systems can adapt to shifting query workloads in near-real time, while Freire et al. demonstrate how complex data integration tasks can be simplified by deeper semantic modeling. Lin et al. push further by showing how LLMs essentially behave like experienced data engineers, orchestrating cleaning and reconciliation efforts that were once painfully manual. Wang et al. illustrate how large and unstructured data repositories can be tamed through language-driven interfaces, making analytics more flexible and intuitive. Throughout these pages, the central lesson is that we are beginning to glimpse a future where LLMs help us rethink the very nature of data systems: how queries are processed, how data is unified, and how organizations can derive insights from an ever-expanding universe of information.

It is my pleasure to extend special thanks to Dr. Steven Euijong Whang, the Associate Editor who brought together these forward-thinking works. As you delve into the research presented here, I invite you to envision how data management might evolve in an era where generative AI not only understands our questions and the data itself, but also orchestrates every necessary process through a flexible suite of tools. The opportunities are immense, and I look forward to the breakthroughs that our field is poised to achieve.

Haixun Wang
EvenUp

Letter from the Special Issue Editor

Large Language Models (LLMs) are increasingly being integrated into databases, data analytics, and emerging data systems. These models are capable of infusing human-level expertise into every stage of data processing, from data discovery to predicting user intent. This issue explores cutting-edge research on the current and future applications of LLMs across various data systems.

LLMs are becoming integral to modern databases. Beginning with the critical tasks of data discovery and integration, the paper *Large Language Models for Data Discovery and Integration: Challenges and Opportunities* by Juliana Freire et al. offers a comprehensive analysis of how LLMs can aid in these tasks while highlighting ongoing challenges. LLMs can help decode the semantics of datasets, making them more accessible and usable. Next, LLMs can enhance query processing. In *LLMs and Databases: A Synergistic Approach to Data Utilization*, Fatma Özcan et al. describe how LLMs leverage their vast, pre-trained knowledge to effectively translate natural language into SQL queries. The authors also develop pre-trained cardinality estimation models and foundation database models inspired by LLMs to solve database performance problems. Finally, SQL programming itself also benefits from LLMs. The paper *Customizing Operator Implementations for SQL Processing via Large Language Models* by Immanuel Trummer introduces the GenesisDB system, which uses LLMs to generate relational operator code for SQL queries. It is even possible to generate code required for running benchmarks like TPC-H.

Beyond databases, LLMs are being utilized in data analytics and next-generation data systems. The paper *iDataLake: An LLM-Powered Analytics System on Data Lakes* by Jiayi Wang et al. presents the iDataLake system, which automates analytics on multi-modal data lakes using LLMs, leveraging their semantic understanding to provide a comprehensive and efficient solution. Data engineering workflows also benefit from LLM integration. The paper *Large Language Models as Pretrained Data Engineers: Techniques and Opportunities* by Yin Lin et al. introduces the UniDM system, which embeds LLMs into key stages of the data engineering process, including data wrangling, analytical querying, and table augmentation for machine learning. LLMs can even enhance the proactivity of data systems. The paper *LLM-Powered Proactive Data Systems* by Sepanta Zeighami et al. demonstrates how LLMs can anticipate user intent, perform transformation operations, and adapt both structured and unstructured data to align with user needs. Lastly, the paper *Top Ten Challenges Towards Agentic Neural Graph Databases* by Jiaxin Bai et al. explores Agentic Neural Graph Databases (NGDBs), which extend neuralization processes to automate data management. The paper discusses the challenges of refining the interfaces, learning, inference, and system components of traditional NGDBs.

Together, these studies represent the frontier of LLM integration into both established and emerging data systems. LLMs are clearly driving impactful advancements in data systems and will continue to shape their evolution in the future. We would like to thank all the authors for their valuable contributions. We also thank Haixun Wang for the opportunity to put together this special issue, and Jieming Shi for his help in its publication.

Steven Euijong Whang
Korea Advanced Institute of Science and Technology

Large Language Models for Data Discovery and Integration: Challenges and Opportunities

Juliana Freire {juliana.freire@nyu.edu}, Grace Fan, Benjamin Feuer,
Christos Koutras, Yurong Liu, Eduardo Pena, Aécio Santos, Cláudio Silva, Eden Wu
New York University, New York, USA

Abstract

The exponential growth of data across diverse sources—from the web and scientific repositories to enterprise systems—has amplified the need for effective data discovery and integration methods. While data integration has been extensively studied for decades, traditional approaches face limitations in generalization and scalability, often requiring a time-consuming and error-prone process that demands manual effort. Dataset discovery, though a more recent research direction, encounters similar challenges. A fundamental issue across both domains is semantic heterogeneity and ambiguity, as existing methods frequently struggle to understand the underlying meaning of data, limiting their effectiveness across diverse applications.

Large Language Models (LLMs) have emerged as a promising solution to enhance both data integration and discovery tasks. These models encode vast amounts of knowledge from extensive training data, enabling them to contextualize datasets—both their schema and contents—within real-world semantics. Notably, LLMs have demonstrated remarkable generalization capabilities, successfully performing various data tasks even without task-specific fine-tuning. This paper examines how LLMs are being applied across various integration and discovery tasks, synthesizing recent developments in this rapidly evolving field. We identify key limitations in current methods and suggest potential directions for future research, offering insights into both the potential and challenges of LLM-driven solutions.

1 Introduction

The increasing availability of datasets presents unprecedented opportunities for innovation across government, industry, and scientific research. By integrating data from multiple sources, it is possible to address complex and important questions that would otherwise be difficult or impossible to answer. For example, integrating genomic, transcriptomic, proteomic, and clinical data from multiple studies allows scientists to carry out pan-cancer analyses and identify common patterns, shared molecular mechanisms, and unique characteristics across cancer types [74]. Similarly, in climate science, combining satellite imagery with ground sensor data improves the accuracy of climate models and predictions [140]. Data integration unlocks new insights, driving advancements across many domains.

Despite efforts to make data FAIR–findable, accessible, interoperable and reusable [133]–discovering and integrating datasets remains a significant challenge [12, 49, 62, 96]. While datasets published on the Web are easily *accessible*, finding specific datasets can be difficult. They are often spread across different repositories that typically rely on simple keyword-based search interfaces, which are insufficient for users to express important information needs [10, 62, 105]. This challenge is further compounded by data heterogeneity: different datasets may use varying terminology for the same attribute or, conversely, the same term for distinct concepts. Such heterogeneity hinders the accuracy and recall of discovery queries,

creating barriers to *interoperability* and *re-usability* and making it difficult to combine data from diverse sources [27].

Dataset Discovery and Integration. Dataset discovery involves systematically identifying and retrieving relevant datasets from data repositories or data lakes [8, 10, 36, 38, 42, 109]. Datasets can be discovered through text-based search, where information needs are expressed in keyword-based queries that are matched against metadata and contents [124, 126, 151], as well as dataset-oriented queries, where users provide a query table and search for relevant, related datasets [3, 25, 29, 30, 43, 51, 60, 90, 105, 160]. These search paradigms largely rely on schema similarity or content-based relevance, but incomplete metadata and semantic heterogeneity remain key challenges.

Data integration refers to the process of combining data from disparate sources into a unified, coherent dataset [27]. It encompasses several key tasks such as schema matching [102], which identifies semantically similar schema elements (e.g., attributes) across datasets, and entity resolution [19], which determines when different records refer to the same real-world entity. Data integration has been a long-standing research area, with a rich body of literature addressing its challenges [69]. Related concepts have emerged in specific domains. For instance, *data harmonization* is a common concept in digital healthcare [88] and social sciences [33] that refers to the process of combining datasets with the explicit goal of maximizing dataset compatibility and comparability [16]. Ultimately, data integration aims to create datasets with standardized formats and structures that enable meaningful comparisons and analysis while addressing challenges such as semantic heterogeneity, data quality, and structural differences across sources. Despite many advances in the area, data integration remains a time-consuming and error-prone task.

Dataset discovery and data integration share several fundamental challenges related to understanding tabular datasets, their schemas and attributes. For instance, to integrate two datasets, one must identify correspondences between attributes across the datasets. Similarly, in dataset-oriented discovery, these correspondences must be determined to enable effective retrieval of related datasets, e.g., that can be joined or concatenated with a query dataset [38, 42, 105].

LLMs for Data Discovery and Integration. State-of-the-art methods for both dataset discovery and integration struggle with semantic heterogeneity and ambiguity, as they often fail to *understand* the semantics of the underlying data, limiting their effectiveness. Recent advances in language models, particularly *Large Language Models* (LLMs), offer promising opportunities to address these challenges [44]. LLMs encode vast amounts of knowledge from extensive training data, allowing them to contextualize datasets—both their schema and contents—within real-world semantics. Moreover, these models have demonstrated the ability to generalize beyond their training data and perform data integration tasks [45, 59, 89] without task-specific fine-tuning. Unsurprisingly, the data management community has shown growing interest in leveraging LLMs for these tasks. In this paper, we survey recent advances, challenges, and opportunities in applying LLMs to dataset discovery and integration.

Contributions. While previous surveys and studies have examined the role of language models in data management [150, 155], they have largely focused on smaller models like BERT or overlooked the specific challenges of using LLMs for dataset discovery. In contrast, this paper explores the emerging role of LLMs, such as GPT and LLaMA, which primarily operate through prompt-based mechanisms, and how these models are reshaping dataset discovery and integration practices. We begin with an overview of language models in Section 2. We then review established techniques for dataset discovery and integration and examine how LLMs have been used to address key challenges in Sections 3 and 4. Finally, in Section 5, we discuss the limitations of existing LLM-based approaches, particularly challenges related to scalability and consistency, and outline potential research directions in both dataset discovery and integration, including opportunities to leverage their synergy to improve each of these tasks.

2 Large Language Models (LLMs): An Overview

A *Language Model* (LM) is trained to model the likelihood of future (autoregressive) or missing (masked or denoising) tokens in a sequence, conditional on all previous tokens in that sequence (the input, often called the context) [111]. These tokens are presumed to be drawn from some fixed vocabulary, typically consisting of common subwords and characters in particular languages. In this work, for simplicity's sake, we refer to all discriminative language models whose final output is an embedding as *Pretrained Language Models* (PLMs), and all generative language models whose final output is a string as *Large Language Models* (LLMs). In fact, LLMs lack a formal definition, as the threshold for what qualifies as large is subjective, and has evolved over time. The phrase was not, for instance, employed by [26] when they introduced their 400 Mn parameter language model, although the phrase "large pre-trained model" is. The authors in [5] use the phrase several times in reference to GPT-3, a 175 Bn parameter language model, and are likely responsible for the term's popularization.

One notable and often discussed property of LLMs that distinguishes them from PLMs is their ability to learn to solve novel problems "in-context" – either through few-shot learning, where they are provided examples with or without instructions, or zero-shot learning, where only instructions are provided [5, 101]. It has been observed that such abilities, which extend well beyond traditional language modeling to encompass, in principle, any task which can be expressed in language, tend to emerge as data and parameter count scale up [22, 104].

The current standard practice with LLMs is to train them in two stages. In the pretraining stage, raw strings are curated and filtered from a range of sources, tokenized, and served for training. This stage, which generally accounts for over 90% of training compute, is where the model learns world knowledge and internal algorithms it will rely on, such as copying [92]. In the post-training stage, the LLM learns to follow instructions and adopt human preferences [22, 94, 130]. It is fine-tuned on a variety of tasks presented as instructions [120], and typically aligned with human preferences using some form of human preference feedback [94]. Post-training has been observed to work better with larger models [22, 130], which in turn has increased the emphasis on scale. Another, more recent area of intensive research has been the scaling of test-time compute, exemplified by OpenAI's O1 and O3 models [157]. Prompting strategies such as chain-of-thought (CoT), which encourage the model to generate more tokens before answering, give the autoregressive model more "time to think" and can improve performance on complex arithmetic, commonsense, and symbolic reasoning tasks [131].

As LLMs can be applied to any task which can be expressed in natural language, attempting to use them to solve long-standing problems in processing tabular data is both logical and appealing. Recent approaches have used LLMs for data cleaning and integration [71, 89, 118, 138, 146], data profiling [45, 52], transforming tables [53, 91], reasoning and question answering over tables [142, 156], and more. However, using LLMs for table related tasks entails several challenges [28, 57, 82, 117]. Naive serialization of large tables as context rapidly explodes the number of tokens per query, increasing the cost and decreasing the efficacy of autoregressive models [73, 86]. Test-time scaling methods exacerbate this challenge, as they rely on the availability of long context, which they consume with their internal chains of thought [157]. Although many methods for subsampling long contexts for tabular data have been proposed, some of which we will discuss later, they tend to vary in efficacy depending on the particulars of the task and the data. The inherent heterogeneity of tabular data poses another kind of challenge: a table can contain anything, ranging from dense numerical attributes to sparse or high-cardinality categorical features, to natural language strings and embeddings [41]. Last but not least, metadata for tables in the wild (e.g., from open-data repositories) is often incomplete; the most reliable source are column headers, but even this can be absent in web-scraped tables [7]. This fact makes it more challenging to deploy solutions which rely on table metadata.

In the following sections, we discuss recent work on using LLMs on tabular data, specifically data

discovery and integration tasks, and highlight limitations and open research challenges.

3 Large Language Models for Dataset Discovery

As more tabular datasets become available from academic institutions, private companies, and governments, there is greater opportunity for innovation and advancements across technology, society, and the economy [37]. However, dataset collections, made available in open repositories or closed data lakes, often contain a large number of datasets with varying sizes and complexity, making manual exploration and retrieval practically infeasible. As a result, there has been growing interest in the data management community to develop discovery systems that enable users to efficiently explore and retrieve datasets from large collections.

Query-driven dataset (table) discovery systems address this challenge by allowing users to query for relevant tables from a data lake. These systems support different types of queries, including (1) keyword-based queries (e.g., users specify keywords such as "*new york*"); (2) natural language queries (e.g., users can ask, "*What is the expected wait time of taxi cabs in NYC?*"); and (3) query tables (e.g., users have a table about NYC Taxis and would like to find other *related* datasets). Methods that support query tables consider different notions of table relatedness. For instance, some systems find tables that can join with a query table on shared attributes [43, 159, 160], while others find tables that can union with a query table to extend it with additional tuples [25, 60, 90]. Additionally, some methods support task-specific table discovery, such as finding tables that are joinable and correlated with the query table to augment it with additional features to improve the performance of machine learning models [105].

In this section, we provide an overview of approaches to table discovery, and survey recent approaches that leverage LLMs for different discovery tasks. Table 1 summarizes LLM-based approaches for dataset discovery.

3.1 Table Search

Table search (or table retrieval) is a data discovery method similar to traditional web search– it aims to find tables that satisfy the information needs described in a textual query. Text-based table retrieval systems initially focused on matching keywords in user queries against the dataset metadata [4, 23, 115] or the content of tables [151], similar to conventional search engines. However, as language models evolved and became more sophisticated, the task advanced to address more complex challenges, such as identifying tables that can answer questions posed in natural language [124, 126]. Regardless of the content of the queries, these methods usually aim to generate a ranked list of tables, denoted as $(T_1, ..., T_k)$, selected from a collection of tables C, in response to a textual query q. This task is often referred to as *ad-hoc table retrieval* since the relevance of each table T_i is determined independently of the other tables T_j (where $i \neq j$). Consequently, the ranking assigns scores to each table, which are then arranged in descending order based on these scores.

Overview of Related Work. Zhang and Balog [151] were one of the first to formalize the table search problem in recent literature and propose deep-learning methods to match keyword queries to table content. However, the problem had previously appeared in earlier work in the context of web tables [7, 8] (we refer the reader to [152] for a longer list of related work). Since then, more recent work has followed and proposed improvements, including algorithms based on PLMs. For example, Chen et al. [14] proposed to leverage a pre-trained BERT model to encode the table content. To workaround BERT's input size limit, they proposed and evaluated different ways to select content from the table that improves the overall ranking quality. Inspired by TaBERT [143], a pre-trained LM that jointly learns representations for natural language sentences and semi-structured tables, Trabelsi et al. [121]

introduced StruBERT. This new model, which was designed for table search and matching, combines textual and structural information of a table to produce context-aware representations for both textual and tabular content. They expand on the concept of vertical self-attention from TaBERT and introduce horizontal self-attention, allowing for equal treatment of both dimensions of a table. Graph-based models have also been proposed that capture table layout, including tables with nested structure [124].

Table Search using LLMs. A common approach to leverage LLMs in table search is to use them as generators of training data to build smaller and more efficient models. For instance, Fujita et al. [46] explored different strategies for generating labels (relevant/irrelevant) for a given table and query. Silva and Barbosa [113] introduced a method for generating synthetic queries based on dataset descriptions. The resulting pairs of queries and descriptions are regarded as soft matches when training fine-tuned dense retrieval models for re-ranking. Wang and Fernandez [126] used an LLM to generate synthetic training data to train a lightweight encoder model that generates embeddings that are used to efficiently retrieve tables that answer natural language questions. Specifically, their pipeline includes a fine-tuned T5 model to translate SQL queries into natural language questions. Another possibility is to use LLMs for query understanding. For instance, Chen et al. [13] used GPT-3.5 Turbo to decompose a natural language query into multiple sub-queries that can potentially be mapped to different tables and columns. They aimed to solve the problem of answering questions that require retrieving multiple tables and joining them through a join plan that cannot be easily discerned from the user query.

Dataset search systems and infrastructure that power data portals [4, 23, 115] treat datasets as documents and rely on metadata (i.e., dataset names and descriptions) to build an inverted index for keyword-based queries. Findability is thus dependent on the quality of dataset descriptions. For data in the wild, descriptions are often incomplete and sometimes inconsistent with the data contents. Zhang et al. [147] proposed a data-driven approach that uses LLMs to automatically generate dataset descriptions and showed that the derived descriptions lead to improved accuracy and recall for table retrieval.

3.2 Query-by-Tables

Semantic Joinable Table Search. Joinable table search aims to find tables that can be joined with a query table to augment it with additional attributes. This type of search is useful for data scientists who want to find new features to improve machine learning models, enrich data for analysis and support decision-making. There are different types of joinable table search, including equi-join, which finds exact matches between joinable columns; fuzzy join, which finds approximate column matches; and semantic join, which matches tables based on semantic relationships between columns. Traditional techniques for joinable table search have often relied on syntactic similarity measures, including Jaccard similarity and set overlap, to find potential joinable tables [43, 68, 159, 160]. More recently, there has been a shift towards methods that capture semantic relationships between columns by using embeddings and pre-trained language models [24, 29, 30, 61] to improve precision and recall.

To find semantic joinable tables, PEXESO [29] encodes columns into high-dimensional vectors using word embeddings such as fastText and GloVE. Joinable tables are then retrieved by comparing vector representations using similarity predicates. Similarly, DeepJoin [30] encodes columns as vectors, and uses column vector similarity to find joinable tables. Unlike PEXESO, DeepJoin uses a pre-trained language model (DistilBERT or MPNet) as the column encoder, which is trained in a self-supervised manner. This way, DeepJoin is able to consider table semantics. To support both equi-joins and semantic joins, the model is fine-tuned on labeled data specific to each joinability task. WarpGate [24] also performs semantic join discovery by leveraging pre-trained language models. WarpGate uses pre-trained web table embeddings [50] to capture the semantic relationships between tables. TabSketchFM [61] was

introduced as a sketch-based tabular pre-training model that can be fine-tuned for different search tasks. TabSketchFM leverages data sketches to represent tabular data and combines embeddings of these sketches with column and token embeddings to create an input embedding for a BERT encoder model. While these approaches for semantic joinable search focus on table retrieval, DTT [91] addresses the challenge of joining values in semantically joinable columns. DTT leverages ByT5 and fine-tunes it to learn transformation rules to align and transform values for joins.

Table Union Search. In table union search, the objective is to discover tables that can be unioned (or concatenated) with a query table to extend it with additional tuples. This type of search is particularly valuable for data scientists who want to compile training or test data for machine learning models or expand the scope of their query tables to cover different geographical regions or time periods, among other use cases. Early work defined unionable tables as entity-complements that share the same subject column and similar schema [109]. More recently, Nargesian et al. [90] relaxed this assumption that unionable tables share the same schema as the query table. They formally defined table unionability based on attribute unionability, such that tables are considered unionable if they have attributes that originate from the same domain as the query table. Bogatu et al. [3] adopted this definition and used five similarity metrics to find unionable and joinable columns. This definition was further refined by Khatiwada et al. [60], who considered relationships between columns in addition to individual column unionability when finding unionable tables that share similar semantics as the query table.

Recent approaches for table union search [25, 38, 51] leverage pre-trained language models to capture column semantics more effectively. Starmie [38] finds unionable tables via self-supervised learning, namely contrastive learning, leveraging a pre-trained language model (RoBERTa) to capture table context when encoding column embeddings. To determine unionability, Starmie computes cosine similarity between column vectors and explores various column aggregation techniques to produce table unionability scores. Similarly, Pylon [25] employs self-supervised contrastive learning for table union search. Pylon explores different encoder models, including fastText, web table embeddings [50], and BERT, to generate column representations. In contrast, AutoTUS [51] shifts the focus to encoding the relationships between column pairs, rather than the columns themselves. By leveraging BERT, AutoTUS produces column relational representations that capture table contexts.

Query-by-Table Approaches using LLMs. Recent methods for joinable and unionable table search have largely been embedding based. While the use of large language models (LLMs) for these tasks has yet to be explored, it comes with its own set of challenges, which we discuss in Section 3.4. There are also many opportunities to take advantage of the power of LLMs to help perform these tasks, we discuss these in Section 5.

3.3 Other Goal-Oriented Dataset Discovery Tasks

Beyond the tasks described above, some discovery methods aim to satisfy more specific information needs. For instance, find tables that (1) satisfy specific distributional characteristics, such as percentile predicates [1], (2) are joinable and contain attributes correlated to columns in the input query table [35, 105, 106, 108, 122], (3) improve the performance of machine learning models [17, 55, 56, 78, 79], (4) uncover causal links among attributes [80, 144], (5) provide explanations for salient features in data [11, 18].

Overview of Related Work: PLMs and LLMs. Since the majority of work on goal-oriented tasks involves numerical data, the approaches used are typically based on traditional algorithmic techniques such as sketches and indexes. However, recent studies have shown that PLMs and LLMs can potentially be used in these tasks. Trummer [122] empirically demonstrates that PLMs can effectively predict correlations between table attributes using only their schemas in many cases. This shows that schemas

are important for data profiling and allow language models to extract insights about the data such as correlations between columns. For the problem of causal dataset discovery, where the goal is to discover datasets containing columns with causal relationships to those in a query table, Liu et al. [80] leverage LLMs to infer causal relationships. They use LLMs to determine whether there are causal links between the correlated tables and to infer the direction of these links. They experiment with GPT-3.5 and GPT-4 and different prompting techniques such as Chain-of-Thought [131] as well as fine-tuning, and show that GPT-4 performs particularly well in identifying causal relationships and their directions.

3.4 Limitations and Research Gaps

Table Discovery methods face similar challenges when leveraging PLMs and LLMs. First, these models rely primarily on textual values in tables and often struggle to capture the semantics of numerical values. This poses a significant limitation for tasks like joinable table search, when join columns are numerical, or unionable table search, that often requires identifying matching numerical columns. Additionally, LLMs may struggle with the complexity of joins, especially transitive joins that involve many tables. Scalability is another major concern, particularly when searching over a large number of tables. Last but not least, LLMs' susceptibility to hallucination raises concerns about their reliability [137].

The effectiveness of techniques that rely on pre-trained models are heavily dependent on the size of the training data. At the same time, the limited context windows of PLMs and LLMs make it challenging to process large amounts of tabular data. Moreover, PLM techniques often lack generalizability. While methods like TabSketchFM [61] demonstrate task generalizability–for example, a model fine-tuned for joinable table search can also be applied to table union search–they often struggle to generalize to unseen data or entirely new domains. Finally, embedding-based approaches lack interpretability, making it difficult to explain retrieval results. Despite these challenges, embedding-based methods highlight the importance of encoding table semantics into column representations when discovering related tables in a large data lake. Building on this idea, we discuss potential opportunities to utilize LLMs for data discovery in Section 5.

4 Large Language Models for Data Integration

Data integration pipelines depend on identifying and connecting relevant elements across disparate datasets. Schema matching and entity resolution represent foundational techniques that identify semantically related elements and enable the creation of unified views from heterogeneous data sources.

This section first provides definitions of these core data integration problems, followed by an overview of traditional state-of-the-art methods. We then examine recent innovations using Large Language Models (LLMs) for data integration tasks, analyzing their potential advantages and limitations. Our discussion primarily focuses on tabular datasets, which are ubiquitous across enterprises, the scientific community, and the web. Table 2 summarizes the LLM-based approaches for data integration discussed in this paper, highlighting their distinctive characteristics.

4.1 Schema Matching

Schema matching refers to the process of identifying semantic relationships between elements in different schemas. For tabular datasets, this process involves finding column pairs from different tables that are semantically similar. The typical input for a schema matching method consists of two or more tables, while the output comprises of either pairwise column correspondences or clusters of semantically similar columns. The effectiveness of schema matching approaches depends on two key factors: the input information considered and the similarity metrics employed to identify related columns. In what

Table 1: Characteristics of LLM-based methods in the literature for Table Discovery.

Papers	Query Type		Task				Models Type			Inference	
	Textual	Table	Table Search	Joinable	Unionable	Others	Open	Closed	Fine-tuned	Zero-shot	Few-shot
[46]	✓		✓					✓		✓	
[113]	✓		✓				✓		✓		
[126]	✓		✓				✓		✓		
[13]	✓		✓			✓		✓			
[80]		✓				✓		✓	✓	✓	
[91]		✓		✓					✓		✓
[147]	✓		✓				✓	✓		✓	

Table 2: Characteristics of LLM-based methods in the literature for Schema Matching (SM) and Entity Resolution (ER).

Papers	Task		Input			Models Type			Inference	
	SM	ER	Metadata	Values	External Knowledge	Open	Closed	Fine-tuned	Zero-shot	Few-shot
[59]	(equi-joins)		✓	✓			✓		✓	
[53]	(PK-FK)		✓				✓		✓	
[97]	✓		✓				✓		✓	
[112]	✓		✓				✓		✓	
[110]	✓		✓		✓		✓		✓	
[83]	✓		✓		✓	✓	✓			✓
[81]	✓		✓	✓			✓			✓
[89]	✓	✓	✓	(ER only)			✓		✓	✓
[136]	✓	✓	✓	(ER only)	✓	✓	✓			✓
[71]	✓	✓	✓	✓			✓	✓	✓	✓
[146]	✓	✓	✓	(ER only)		✓	✓	✓	✓	✓
[138]	✓	✓	✓	(ER only)		✓		✓		✓
[99]		✓		✓		✓	✓	✓	✓	✓
[128]		✓		✓			✓		✓	
[54]		✓	✓	✓		✓				✓
[39]		✓	✓	✓		✓	✓			✓
[116]		✓		✓		✓	✓	✓	✓	
[31]		✓	✓	✓		✓		✓	✓	
[15]		✓	✓	✓			✓		✓	✓

follows, we review approaches proposed in the literature that demonstrate considerable diversity in both the similarity metrics utilized and the types of information leveraged.

Overview of Related Work. Early schema matching methods relied primarily on syntactic similarity measures between columns [102] and their value distribution [149], drawing mainly on information captured in column names and their corresponding column values. Some approaches expanded beyond syntax by incorporating external knowledge sources such as dictionaries and domain-specific thesauri [84] to capture semantic relationships.

The emergence of word and character embedding models, such as GloVe [100] and fastText [85], enabled the creation of semantically rich column representations that could be compared for similarity assessment [9, 110]. However, these embedding-based methods have shown inconsistent behavior and struggled with noisy data [65]. Specifically, traditional word embedding models do not properly handle formatting discrepancies, such as typos, and they attach the same meaning to each syntactically unique word, without accounting for context and polysemy (words with multiple meanings).

More recently, PLM-based methods, which produce contextualized representations of text, have been proposed for schema matching. Specifically, several methods have been proposed to fine-tune PLMs and perform schema matching [32, 123, 154]. The effectiveness of these methods is comparable to or even better than schema matching techniques that rely on syntactic measures or word embeddings. However, these PLM-based methods require large amounts of labeled column pairs. This labeled set is not always available, like in the case of tables in the wild, and thus requires methods to generate training examples with similar data distributions as the test data, such as contrastive learning. In addition, other methods have been proposed that leverage other (self-)supervised deep learning models (e.g., GNNs) to produce column representations for schema matching [66, 67, 148].

Schema Matching with LLMs. Several methods use LLMs with varied prompting approaches and information. Narayan et al. [89] employed zero-shot and few-shot LLM prompts to address data cleaning and integration tasks. Their prompts consist of serialized attribute/data values and, optionally, task demonstrations (selected randomly or manually) from a pool of labeled data. When using only column names in few-shot LLM prompts, the authors showed improvement over the deep-learning state-of-the-art schema matching method in [148]. Kayali et al. [59] designed prompts specifically for identifying join columns between two distinct *pandas DataFrames*, providing instructions with data samples and answer templates to guide the LLM in completing a *pandas.merge* operation. Parciak et al. [97] explored various prompting strategies for matching source attributes to a target schema. Huang et al. [53] developed a method that utilizes LLM prompts containing table descriptions (which are generated using LLMs) and schema information to capture PK-FK relationships (a special type of column match) between column pairs. In an effort to improve prompts for schema matching, Xu et al. [136] incorporated manual rules and additional insights extracted from either the input datasets or external knowledge bases.

Beyond using off-the-shelf LLMs, several researchers have explored fine-tuning LLMs specifically for data wrangling and integration tasks. Li et al. [71] proposed Table-GPT, which fine-tunes LLMs to address various table-related tasks, including schema matching. It incorporates fine-tuning for both complex reasoning tasks and simpler table manipulation operations to enhance the model's overall table processing capabilities. They evaluated various prompt and table serialization templates, demonstrating the importance of diverse augmentation techniques during fine-tuning. However, their experimental results for schema matching are inconclusive, as both out-of-the-box and fine-tuned LLMs demonstrate "perfect" effectiveness. This study leaves several questions unanswered, including how the number of tuples in table serialization affects performance and how the approach scales to large tables (in terms of both columns and rows).

Similarly, Zhang et al. [146] developed Jellyfish, which also employs LLM fine-tuning for data wrangling and integration tasks. Unlike Table-GPT, Jellyfish places special emphasis on prompt content

and intended output, sometimes incorporating injected knowledge about input data and requiring answers that include reasoning information. Their evaluation shows that Jellyfish's fine-tuned models slightly outperform a previous schema matching technique [148] on a benchmark using only attribute names and descriptions (similar to the evaluation in [89]). Yan et al. [138] proposed a Mixture of Experts (MoE) approach based on LLMs for various data preprocessing tasks, including schema matching. Their results demonstrate improved effectiveness compared to both open-source and proprietary off-the-shelf models, while also showing better efficiency than Jellyfish's fine-tuned model [146], suggesting that MoE represents a promising approach.

It is worth noting that several studies [89, 97, 136, 138, 146] evaluate schema matching using datasets that only include table/attribute names and descriptions. However, there has not been a comprehensive evaluation to compare all these approaches.

Instead of using LLMs on input data to directly capture matches, a number of methods incorporate them as part of more complex schema matching pipelines. ReMatch [112] introduces a method that refines candidates based on embeddings before prompting the LLM. Specifically, the authors focus on the problem of matching a set of source tables to a set of target tables when table and attribute descriptions and names are given. Their method first transforms each table to a document that includes table/column names and descriptions; using GPT-4, each such document is transformed into an embedding. Then, for each source attribute they leverage embeddings to find the most relevant documents, i.e., tables, from which they retrieve the final matching target attributes based on the LLM response. Matchmaker [110] targets the same variant of schema matching problem and proposes a multi-stage method with several LLM-calls to suggest and refine candidate matches between source and target schemata, while also assigning confidence scores. In contrast to ReMatch, Matchmaker uses both LLM-calls and PLM embeddings to filter out candidate target attributes, while it also employs LLMs in a different way to refine and finally match them to the query source attribute: matching is formulated as a multiple choice question, where the task is to find the most relevant target column with respect to a given source column. Both methods show improvements over a state-of-the-art deep-learning schema matching method [148], while Matchmaker shows effectiveness gains over ReMatch and Jellyfish [146]. Instead of relying only on the information in the input datasets, Ma et al. [83] proposed to leverage knowledge graphs. Their method retrieves relevant knowledge graph triplets and uses them to augment LLM prompts for answering whether a source attribute corresponds to a target one, on top of providing their names, descriptions and demonstration examples; retrieval can be either LLM-based or employ vector search over PLM-based embeddings.

Magneto [81] introduces a new approach for schema matching that combines small-PLMs (SLMs) and LLMs in a novel way. Like ReMatch, it works in two steps. First, it leverages a pre-trained or fine-tuned SLM to produce embeddings of columns and given a source column, outputs, a ranked list of similar target attributes. Note that instead of relying on manually-labeled data, Magneto uses LLMs to generate training data for fine-tuning the SLM. For the second step, the ranked list of matches is given to an LLM for re-ranking. To deal with the context-window limitations of both PLMs and LLMs, they explored different methods for sampling and serializing tables (including values). Experimental results show that using LLMs for re-ranking can be effective, regardless of the SLM used to derive the initial ranking, and that Magneto outperforms or performs comparably to state-of-the-art methods, including [123] and [32].

Limitations and Research Gaps. Despite the progress in applying LLMs to schema matching, several critical limitations remain unaddressed. First, existing work largely ignore the challenges of processing large input sequences when tables contain a large number columns and rows. The tendency to disregard data instances in favor of relying solely on column names and descriptions restricts these methods' applicability in many real-world scenarios.

Interestingly, current LLM-based approaches predominantly focus on matching source data to standardized target schemas, with matching datasets to OMOP CDM (Observational Medical Outcomes Partnership Common Data Model) [93] emerging as the most common evaluation scenario. Since these benchmarks contain table/attribute names and descriptions, research has gravitated toward metadata-based schema matching, resulting in a scarcity of LLM-based methods that effectively utilize actual data instances.

Few studies include comparisons against other LM-based methods, impeding meaningful progress in the field. Instead, most methods report improvements against pre-PLM schema matching techniques or out-of-the-box models. Notably absent are detailed analyses of specific column match cases where LLM-based methods significantly outperform previous state-of-the-art approaches. The absence of granular performance analysis makes it difficult to identify genuine advancements and understand the specific strengths of LLM-based approaches. Furthermore, evaluation results from multi-tasking models like TableGPT [71] and Jellyfish [146] successfully demonstrate general applicability but fail to provide task-specific insights crucial for understanding schema matching performance.

While few-shot inference with LLMs can achieve state-of-the-art effectiveness, this approach requires carefully selected demonstration examples from labeled data pools. This presents a significant challenge, as ground truth data for schema matching tasks is notoriously limited. Even when labeled data exists, it may not accurately reflect the intrinsic characteristics of test data, potentially leading to suboptimal performance in real-world applications.

4.2 Entity Resolution

Entity Resolution involves identifying the same entities across different datasets. In tabular data, the objective is to determine which tuples refer to the same entity. Entity resolution pipelines typically consist of several distinct tasks [19]. Initially, input tables may undergo an optional pre-processing step that includes integration and curation tasks such as schema matching (as discussed in Section 4.1), data cleaning [21], and other preparatory operations. Following this, the blocking phase examines all tuples across datasets and produces a set of candidate pairs that might be semantically similar; essentially, blocking serves as the filtering step within an entity resolution pipeline. Finally, the matching phase evaluates each candidate pair and determines whether it constitutes a valid match. In the following sections, we discuss proposed methods that focus on either the blocking phase exclusively [95], the matching phase exclusively [76], or address both phases [20]. We then examine approaches that leverage LLMs for entity resolution tasks.

Overview of Related Work. The majority of entity resolution methods target the matching step. Specifically, several approaches rely on human experts to devise rules [40, 114], guide the entity matching process through crowd-sourcing [48, 125], or provide labeled data for training machine learning models [63]. Moreover, automated methods using supervised learning with deep learning models [34, 87] exhibit considerable effectiveness gains.

More recently, various methods have leveraged the representational capabilities of PLMs to build fine-tuned models that capture entity matches in a supervised manner [6, 75, 77, 98, 123]. While these approaches show performance improvements, they continue to require a substantial amount of labeled data. To address this limitation, alternative research directions have explored self-supervised [47, 127] and unsupervised [134, 145] frameworks. These approaches have shown comparable or superior effectiveness relative to supervised models, helping overcome the labeled data bottleneck that often constrains learning-based entity resolution methods.

To address the blocking phase of entity resolution, methods have been proposed to employ various filtering techniques to identify potential matches across datasets, including rule-based filtering, hash-based comparisons, sorted key comparisons, similarity functions, and ensemble methods combining

multiple approaches [95]. Deep learning solutions have also emerged in the blocking space [34, 119], which project tuples into embedding spaces and leverage these representations to construct clusters of potentially matching entities.

Moving beyond the traditional separation of blocking and matching, some researchers have proposed integrated approaches [70, 135]. Such methods demonstrate the benefits of addressing both tasks simultaneously, allowing signals from each phase to inform and improve the other's performance. This integrated perspective represents a promising direction for enhancing overall entity resolution effectiveness.

Entity Resolution with LLMs. Recent research has explored using LLMs out-of-the-box with dedicated prompts for direct entity matching decisions. Narayan et al. [89] demonstrated that an LLM prompt that serializes column names and values for tuple pairs alongside demonstration examples can sometimes outperform state-of-the-art solutions like Ditto [75, 77]. Peeters et al. [99] conducted a comprehensive experimental study on LLM-based tuple-pair matching. Their investigation encompassed multiple LLMs using both zero-shot and few-shot prompting approaches, while also exploring fine-tuning with labeled data from existing annotated dataset pairs. Their findings revealed that no single model or prompting technique consistently outperforms others, with traditional non-LLM supervised methods sometimes achieving superior results. Wang et al. [128] introduced a novel approach that considers matching of a single tuple against a set of tuples, rather than focusing on tuple-pair matching. This problem formulation enabled their method, ComEM, to explore diverse input-output prompting strategies and effectively combine them for enhanced performance. When compared against state-of-the-art approaches—including supervised methods (e.g., Ditto [77]), self-supervised techniques (e.g., Sudowoodo [127]), and out-of-the-box LLM prompts [99], ComEM demonstrated comparable or occasionally superior effectiveness. Dou et al. [31] explored the advantages of jointly training blocking and matching components through supervised learning. Specifically, they introduced an innovative architecture supports fine-tuning either traditional PLMs or instruction-tuned open-source LLMs for the matching phase. Their experimental results reveal that while instruction-tuned LLMs demonstrate superior effectiveness compared to PLMs, they still fall short of the performance achieved by proprietary models like GPT-4 when used out-of-the-box. Xu et al. [136] presented a method that claims applicability to entity matching tasks through the use of manual instructions and external knowledge integration. However, their evaluation focuses exclusively on schema matching, leaving the method's effectiveness for entity resolution tasks unverified.

Recent work has explored multiple strategies to optimize LLM-based entity matching, ranging from model architecture modifications to prompt engineering and computational efficiency improvements. Table-GPT [71] demonstrates that LLMs specifically tuned on tabular data outperform their untuned counterparts in both zero-shot and few-shot settings for tuple pair matching. Building on this foundation, Jellyfish [146] achieves superior entity matching results compared to both Table-GPT and traditional non-LLM methods. However, Jellyfish's performance advantage is most pronounced when the test data distribution is represented in the fine-tuning dataset, and comparisons against state-of-the-art LLMs like GPT-4 have yielded inconclusive results. A systematic study by Steiner et al. [116] further explores fine-tuning approaches across both open-source and proprietary LLMs, analyzing various data representation formats and example selection strategies. They also evaluated model robustness against domain shifts between training and test data. While their fine-tuned models demonstrate enhanced performance for in-domain generalization, an interesting finding reveals that zero-shot approaches actually achieve superior results when handling test data that comes from a domain different from the training data. This highlights an important trade-off between fine-tuning benefits and domain adaptability.

Alternative architectural approaches have also shown promise. The Mixture-of-Experts (MoE) LLM-based model proposed in [138] brings improvements in entity matching over standard open/closed-

source and fine-tuned models. It demonstrates superior performance compared to both conventional and fine-tuned LLM models, highlighting the advantages of specialized task experts within a unified framework. Taking a different approach, Seed [15] introduces a flexible system that leverages LLMs in two ways: either for code and training data generation, or for direct entity matching tasks. A key innovation of Seed is its optimization framework, which generates and selects execution plans based on both computational efficiency and effectiveness metrics. While Seed does not match the accuracy levels of state-of-the-art supervised methods like Ditto [77] or pure LLM approaches [99], it achieves a balance between effectiveness and LLM-cost efficiency.

Huh et al. [54] investigated demonstration example selection by leveraging pre-trained language models (PLMs) to embed tuple pairs and identify similar examples in the representation space. Interestingly, their findings reveal that this sophisticated approach does not consistently outperform simpler random or manual example selection methods. Addressing computational efficiency, BatchER [39] demonstrates that batch processing of tuple pairs within single prompts can achieve both cost savings and effectiveness gains. BatchER achieves comparable results to state-of-the-art PLM-based entity matching methods that require extensive labeled datasets, while maintaining lower computational overhead.

Limitations and Research Gaps. While LLMs show considerable promise for entity matching tasks, there are also important challenges. First, prompt engineering significantly impacts matching accuracy, with different model architectures exhibiting varying levels of prompt sensitivity. Peeters et al. [99] demonstrated that while GPT-4 maintains consistent performance across different prompt formulations, other models like GPT-mini, Llama2, Llama3.1, and Mixtral show higher sensitivity to prompt variations. This finding highlights the critical importance of careful prompt design, particularly when using more cost-effective models.

The evaluation landscape presents a mixed picture. Unlike schema matching, entity matching research benefits from common benchmark datasets, enabling direct comparisons between LLM-based approaches and traditional methods. However, inconsistent result presentation and reporting practices often lead to inconclusive comparisons. This challenge is particularly evident when trying to determine the precise advantages of LLM-based methods over existing approaches. A significant gap in current research is the lack of detailed analysis of specific use cases or scenarios where LLM-based methods demonstrate clear advantages over traditional approaches. While studies often report aggregate performance metrics, they frequently fail to identify and analyze the particular types of entity matching problems where LLMs excel or struggle compared to conventional methods. This limitation makes it difficult for practitioners to make informed decisions about when to adopt LLM-based solutions.

A significant challenge in deploying LLMs for entity matching is managing computational costs, particularly with hosted commercial models. The cost structure, based on token count for both input prompts and model outputs, becomes especially problematic when handling tuples with extensive textual attributes that require longer prompts. While advanced models like GPT-4 provide superior performance, their higher per-token costs create a substantial economic barrier compared to smaller models like GPT-mini. The cost-quality trade-off becomes more complex when considering performance optimization strategies. In-context learning can significantly improve matching accuracy through demonstration examples, but each additional example or rule increases the token count and consequently the operational costs [39]. Similarly, while fine-tuning offers a promising path to improve performance of smaller open-source models, its benefits are often limited to scenarios where test data closely aligns with the training domain [99, 116]. Furthermore, the fine-tuning process itself presents two significant barriers: the high computational overhead and the requirement for substantial labeled training data—a limitation shared with traditional PLM-based methods [77].

5 Research Challenges and Future Directions

Despite the rapid advancement in applying LLMs to data discovery and integration tasks, significant research challenges and opportunities remain unexplored. Building on our analysis of existing methods (Sections 3 and 4), we identify and discuss both task-specific challenges and broader research directions that warrant further investigation. Our discussion encompasses not only technical limitations identified in current approaches but also emerging opportunities to enhance the effectiveness and practical applicability of LLM-based solutions in this domain.

5.1 Dataset Discovery

Although the direct application of LLMs to large-scale tasks like semantic joinable table search and table union search faces *scalability* challenges, LLMs' sophisticated semantic understanding capabilities present promising opportunities for advancing dataset discovery tasks.

Improved Semantic Understanding and Metadata Enrichment. While existing PLM-based methods [24, 25, 30, 38, 51, 61] have demonstrated the value of semantic approaches in finding related tables, LLMs could potentially enable more precise semantic matching and relationship identification. LLMs have shown success in fundamental table understanding tasks, including column-type annotation [45, 59, 64, 71, 132, 146, 153], table-class detection [52, 59, 64], and column-relation extraction [59, 153]. These capabilities can be leveraged to enhance tables with rich semantic information, thereby improving joinable and unionable table search accuracy.

Cross-Task Integration. Advances in LLM-based entity matching (Section 4.2) could be extended to multi-table entity matching and attribute discovery. Similarly, schema matching techniques (Section 4.1) could be adapted to identify semantically similar columns for unionable table search. This cross-pollination of techniques offers promising paths for improvement. Research is needed on how to effectively combine different LLM-based tasks (e.g., type annotation, relation extraction, and matching) in a unified discovery pipeline.

Hybrid Approaches. Another direction of research is to further explore hybrid approaches that combine LLMs with traditional techniques. Such approaches could leverage LLMs' semantic understanding while maintaining the efficiency of established methods. Particularly promising is the integration of LLMs with existing join discovery techniques like sketches and indexes [30, 43, 106, 160], which could enhance emerging join-aware textual query methods that retrieve multiple tables [2, 13].

Explainable Discovery. Lastly, LLMs' ability to generate natural language explanations for semantic matches and relationships between columns and tables represents an underexplored opportunity. Such explanations could significantly improve user trust and understanding of discovery results, facilitating more effective use in downstream tasks. Research is needed on how to generate explanations that are both technically accurate and accessible to users with varying levels of expertise.

5.2 Schema Matching

Handling Instance Data. Current LLM-based schema matching methods predominantly focus on scenarios where only metadata is available for matching source datasets to target schemas (Section 4.1). However, schema matching must also handle more challenging scenarios where column names may be opaque or missing, making instance data the primary source of matching information. This disconnect presents several key challenges to the development of LLM-based schema matching approaches. A fundamental challenge is managing large input sequences when tables contain numerous columns and rows. While recent LLM architectures offer expanded context windows exceeding 100,000 tokens, two critical

issues emerge: First, studies indicate that LLMs face accuracy degradation with longer contexts [72] and efficiency challenges [139]. Second, even with larger context windows, the computational costs and performance trade-offs must be carefully balanced. This points to promising research directions. Rather than using LLMs for direct schema matching, approaches like [81] demonstrate the potential of leveraging LLMs as sophisticated reasoning engines in post-processing steps. Future research should explore approaches to efficiently sample and summarize instance data to create informative yet compact LLM inputs, develop adaptive strategies that selectively invoke LLM processing based on data characteristics, design cost-effective architectures that maintain accuracy while managing computational overhead.

Profiling for Schema Matching. LLMs present significant opportunities for enhancing schema matching through automated data profiling and metadata enrichment, particularly in scenarios where metadata is missing or unreliable. Recent works demonstrate promising applications: generating detailed table and column descriptions [52, 147], and inferring semantic types for columns [45]. The information obtained through profiling can be used to identify matching columns. In addition to improving the effectiveness of schema matching methods, the generated metadata an help prune the search space of potential column matches, contributing to lower computation cost and execution time.

5.3 Entity Resolution

Handling Long Entities. While LLM-based entity matching solutions benefit from significantly longer context windows compared to PLM-based methods, incorporating all attribute values into entity representations may not always improve matching accuracy. The impact of long attribute values on LLM reasoning can vary significantly - for instance, extensive user comments might either obscure or enhance product matching depending on their content. This complexity necessitates sophisticated pre-processing strategies for optimizing entity representations in LLM prompts. Several promising research directions emerge for effective pre-processing, including: the development of profiling techniques to identify and prioritize informative attributes; integration of schema matching techniques to focus on comparable attributes, following successful approaches from traditional entity matching methods [20]; and techniques for condensing long attribute values while preserving matching-relevant information [77]

Blocking. While LLMs have shown promise in entity matching (Section 4.2), their application to blocking—a crucial step in real-world entity resolution [95]—remains largely unexplored. This gap stems from the inherent complexity of blocking tasks: unlike pair-wise entity matching, blocking must efficiently process the entire search space of possible tuple pairs across datasets to identify candidate matches. The computational demands and complexity of this task make direct application of LLMs impractical, even with their expanding context windows and enhanced reasoning capabilities. However, LLMs can contribute to blocking effectiveness through indirect approaches. For example, through metadata enhancement – enriching tuples with additional semantic information, generating standardized representations of attribute values. Recent work [129] has demonstrated the potential of LLM-based tuple enrichment for improving blocking effectiveness

5.4 General Directions

Prompt Engineering. The effectiveness of LLMs in data integration and discovery tasks heavily depends on prompt engineering, but there are significant challenges in developing robust prompting strategies. Experimental evaluations of methods discussed in Sections 3 and 4 point out that there are no universally effective prompting templates and strategies. Even minor changes in table serialization formats or task descriptions can significantly impact performance [71, 99]. Moreover, the rapid evolution of prompting strategies [103] makes it difficult to establish best practices.

This complexity points to several critical research directions in automated prompt optimization and the integration of context and knowledge. Inspired by recent advances in automatic prompt engineering efforts [158], automated prompt optimization methods could be tailored to each given input and table-related tasks. Moreover, choosing suitable in-context examples [54] and effective retrievers for incorporating external knowledge to prompts [58] is equally important. Finally, recent advanced prompting techniques that enable tool calling abilities [141] are an exciting direction that allows the development of agentic systems that integrate existing efficient algorithms with LLM-based reasoning and interactive user interfaces [107].

Cost Considerations. Evaluating LLMs' feasibility in practical scenarios demands rigorous analysis of cost and runtime factors, which present significant deployment barriers. The current landscape presents a complex trade-off between hosted and open-source solutions, each with distinct challenges. Hosted models pose several significant challenges for practical deployment. Leading providers' high API costs per token [1][2][3] can accumulate rapidly, becoming prohibitive for large-scale deployments – involving large datasets or a large number of datasets. Response latency issues become particularly acute when processing large datasets, potentially impacting real-time applications.

Open-source models present a different set of challenges. Models like LLaMA2 require a significant upfront investment in specialized hardware infrastructure, particularly high-performance GPUs. Beyond the initial investment, organizations must consider ongoing operational and maintenance costs, as well as the technical expertise required for effective deployment and optimization. These factors can make the total cost of ownership substantial, even without per-token API fees.

This dichotomy highlights a critical research gap: the absence of standardized frameworks for comparing hosted and open-source models. Such frameworks would need to address multiple dimensions including effectiveness-cost ratios, scalability characteristics, total cost of ownership, and operational complexity. The development of comprehensive comparison methodologies would enable organizations to make more informed deployment decisions based on their specific needs and constraints.

Comprehensive Task-Specific Evaluations. Recent LLM-based approaches that target multiple tabular data curation and integration tasks [71, 138, 146] have established their effectiveness through extensive experimental evaluations that typically assess each task using dedicated benchmarks and compare against state-of-the-art methods using standard metrics like F1-score. Additionally, they often include task-agnostic ablation studies examining the impact of different prompting strategies, demonstration examples, and LLM architectures.

While these evaluations demonstrate the general applicability and advantages of LLM-based methods, they often lack granular, task-specific insights that could better illuminate their unique strengths. Current evaluation approaches rarely analyze specific challenging instances where LLMs demonstrate particular advantages. For example, cases involving columns or entities with significant syntactic differences but semantic similarities could provide valuable insights into LLMs' semantic understanding capabilities. Such fine-grained analysis could reveal where LLMs excel compared to traditional approaches. Future evaluations should therefore incorporate detailed analysis of specific challenging cases that highlight LLMs' distinctive capabilities. This could include examining performance on instances requiring complex semantic reasoning, handling of ambiguous or context-dependent matches, and cases where traditional methods typically struggle. By providing concrete examples of where LLMs bring meaningful improvements, such task-specific evaluations would offer stronger justification for their adoption and better guide their application in practice.

Acknowledgments. This work was supported by NSF awards IIS-2106888 and OAC-2411221, and the

[1] https://openai.com/api/pricing/
[2] https://www.anthropic.com/pricing#anthropic-api
[3] https://ai.google.dev/pricing#1_5flash

DARPA Automating Scientific Knowledge Extraction and Modeling (ASKEM) program, Agreement No. HR0011262087. The views, opinions, and findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the DARPA, the U.S. Government, or NSF.

References

[1] Lennart Behme, Sainyam Galhotra, Kaustubh Beedkar, and Volker Markl. Fainder: A fast and accurate index for distribution-aware dataset search. *Proceedings of the VLDB Endowment*, 17 (11):3269–3282, 2024.

[2] Jan-Micha Bodensohn and Carsten Binnig. Rethinking table retrieval from data lakes. In *Proceedings of the Seventh International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pages 1–5, 2024.

[3] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. Dataset discovery in data lakes. In *ICDE*, pages 709–720, 2020.

[4] Dan Brickley, Matthew Burgess, and Natasha Noy. Google dataset search: Building a search engine for datasets in an open web ecosystem. In *The World Wide Web Conference*, pages 1365–1375, 2019.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`.

[6] Ursin Brunner and Kurt Stockinger. Entity matching with transformer architectures-a step forward in data integration. In *23rd International Conference on Extending Database Technology, Copenhagen, 30 March-2 April 2020*, pages 463–473. OpenProceedings, 2020.

[7] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[8] Michael J Cafarella, Alon Halevy, and Nodira Khoussainova. Data integration for the relational web. *Proceedings of the VLDB Endowment*, 2(1):1090–1101, 2009.

[9] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *SIGMOD*, pages 1335–1349, 2020.

[10] Sonia Castelo, Rémi Rampin, Aécio Santos, Aline Bessa, Fernando Chirigati, and Juliana Freire. Auctus: A dataset search engine for data discovery and augmentation. *Proceedings of the VLDB Endowment*, 14(12):2791–2794, 2021.

[11] Yeuk-Yin Chan, Fernando Chirigati, Harish Doraiswamy, Cláudio T. Silva, and Juliana Freire. Querying and exploring polygamous relationships in urban spatio-temporal data sets. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 1643–1646. ACM, 2017. doi: 10.1145/3035918.3058741. URL `https://doi.org/10.1145/3035918.3058741`.

[12] Adriane Chapman, Elena Simperl, Laura Koesten, George Konstantinidis, Luis-Daniel Ibáñez, Emilia Kacprzak, and Paul Groth. Dataset search: a survey. *The VLDB Journal*, 29(1):251–272, 2020.

[13] Peter Baile Chen, Yi Zhang, and Dan Roth. Is table retrieval a solved problem? exploring join-aware multi-table retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2687–2699, 2024.

[14] Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Yinan Xu, and Brian D Davison. Table search using a deep contextualized language model. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 589–598, 2020.

[15] Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. Seed: Domain-specific data curation with large language models. *arXiv e-prints*, pages arXiv–2310, 2023.

[16] Cindy Cheng, Luca Messerschmidt, Isaac Bravo, Marco Waldbauer, Rohan Bhavikatti, Caress Schenk, Vanja Grujic, Tim Model, Robert Kubinec, and Joan Barceló. A general primer for data harmonization. *Scientific data*, 11(1):152, 2024.

[17] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David R. Karger. ARDA: automatic relational data augmentation for machine learning. *Proc. VLDB Endow.*, 13(9):1373–1387, 2020.

[18] Fernando Chirigati, Harish Doraiswamy, Theodoros Damoulas, and Juliana Freire. Data polygamy: the many-many relationships among urban spatio-temporal data sets. In *ACM SIGMOD*, pages 1011–1025, 2016.

[19] Peter Christen. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-Centric Systems and Applications. Springer, 2012. ISBN 978-3-642-31163-5. doi: 10.1007/978-3-642-31164-2. URL `https://doi.org/10.1007/978-3-642-31164-2`.

[20] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.*, 53 (6):127:1–127:42, December 2020. ISSN 0360-0300. doi: 10.1145/3418896.

[21] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*, pages 2201–2206, 2016.

[22] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022.

[23] CKAN. `https://ckan.org`, 2024.

[24] Tianji Cong, James Gale, Jason Frantz, H. V. Jagadish, and Çagatay Demiralp. Warpgate: A semantic join discovery system for cloud data warehouses. In *CIDR*, 2023.

[25] Tianji Cong, Fatemeh Nargesian, and HV Jagadish. Pylon: Semantic table union search in data lakes. *arXiv preprint arXiv:2301.04901*, 2023.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/V1/N19-1423. URL https://doi.org/10.18653/v1/n19-1423.

[27] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012. ISBN 0124160441.

[28] Haoyu Dong and Zhiruo Wang. Large language models for tabular data: Progresses and future directions. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2997–3000, 2024.

[29] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 456–467. IEEE, 2021.

[30] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. Deepjoin: Joinable table discovery with pre-trained language models. *Proceedings of the VLDB Endowment*, 16(10):2458–2470, 2023.

[31] Wenzhou Dou, Derong Shen, Xiangmin Zhou, Hui Bai, Yue Kou, Tiezheng Nie, Hang Cui, and Ge Yu. Enhancing deep entity resolution with integrated blocker-matcher training: Balancing consensus and discrepancy. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 508–518, 2024.

[32] Xingyu Du, Gongsheng Yuan, Sai Wu, Gang Chen, and Peng Lu. In situ neural relational schema matcher. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 138–150. IEEE, 2024.

[33] Joshua Kjerulf Dubrow and Irina Tomescu-Dubrow. The rise of cross-national survey data harmonization in the social sciences: emergence of an interdisciplinary methodological field. *Quality & Quantity*, 50:1449–1467, 2016.

[34] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. Distributed representations of tuples for entity resolution. *Proc. VLDB Endow.*, 11 (11):1454–1467, 2018. doi: 10.14778/3236187.3236198. URL http://www.vldb.org/pvldb/vol11/p1454-ebraheem.pdf.

[35] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. *Cocoa: Correlation coefficient-aware data augmentation*. Konstanz, Germany: OpenProceedings. org, University of Konstanz, University . . . , 2021.

[36] Mahdi Esmailoghli, Christoph Schnell, Renée J Miller, and Ziawasch Abedjan. Blend: A unified data discovery system. *arXiv preprint arXiv:2310.02656*, 2023.

[37] Grace Fan, Jin Wang, Yuliang Li, and Renée J. Miller. Table discovery in data lakes: State-of-the-art and future directions. In *SIGMOD Conference Companion*, pages 69–75. ACM, 2023.

[38] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. *Proc. VLDB Endow.*, 16(7):1726–1739, 2023.

[39] Meihao Fan, Xiaoyue Han, Ju Fan, Chengliang Chai, Nan Tang, Guoliang Li, and Xiaoyong Du. Cost-effective in-context learning for entity resolution: A design space exploration. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 3696–3709. IEEE, 2024.

[40] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. Reasoning about record matching rules. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1):407–418, 2009.

[41] Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, and Christos Faloutsos. Large language models(llms) on tabular data: Prediction, generation, and understanding – a survey, 2024. URL `https://arxiv.org/abs/2402.17944`.

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

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

[44] Raul Castro Fernandez, Aaron J. Elmore, Michael J. Franklin, Sanjay Krishnan, and Chenhao Tan. How large language models will disrupt data management. *Proc. VLDB Endow.*, 16(11): 3302–3309, 2023. doi: 10.14778/3611479.3611527. URL `https://www.vldb.org/pvldb/vol16/p3302-fernandez.pdf`.

[45] Benjamin Feuer, Yurong Liu, Chinmay Hegde, and Juliana Freire. Archetype: A novel framework for open-source column type annotation using large language models. *Proceedings of the VLDB Endowment*, 17(9):2279–2292, May 2024. ISSN 2150-8097. doi: 10.14778/3665844.3665857. URL `http://dx.doi.org/10.14778/3665844.3665857`.

[46] Yukihisa Fujita, Teruaki Hayashi, and Masahiro Kuwahara. Inferring relationships between tabular data and topics using llm for a dataset search task. In *2024 IEEE International Conference on Big Data (BigData)*, pages 6564–6573. IEEE, 2024.

[47] Congcong Ge, Pengfei Wang, Lu Chen, Xiaoze Liu, Baihua Zheng, and Yunjun Gao. Collaborem: A self-supervised entity matching framework using multi-features collaboration. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12139–12152, 2021.

[48] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. Corleone: Hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 601–612, 2014.

[49] Kathleen Gregory and Laura Koesten. *Human-centered data discovery.* Springer, 2022.

[50] Michael Günther, Maik Thiele, Julius Gonsior, and Wolfgang Lehner. Pre-trained web table embeddings for table discovery. In *Proceedings of the Fourth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pages 24–31, 2021.

[51] Xuming Hu, Shen Wang, Xiao Qin, Chuan Lei, Zhengyuan Shen, Christos Faloutsos, Asterios Katsifodimos, George Karypis, Lijie Wen, and Philip S. Yu. Automatic table union search with tabular representation learning. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 3786–3800. Association for Computational Linguistics, 2023.

[52] Zezhou Huang and Eugene Wu. Cocoon: Semantic table profiling using large language models. In *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics*, pages 1–7, 2024.

[53] Zezhou Huang, Jia Guo, and Eugene Wu. Transform table to database using large language models. *Proceedings of the VLDB Endowment. ISSN*, 2024.

[54] Joon Suk Huh, Changho Shin, and Elina Choi. Pool-search-demonstrate: Improving data-wrangling llms via better in-context examples. In *NeurIPS 2023 Second Table Representation Learning Workshop*, 2023.

[55] Andra Ionescu, Rihan Hai, Marios Fragkoulis, and Asterios Katsifodimos. Join path-based data augmentation for decision trees. In *2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW)*, pages 84–88. IEEE, 2022.

[56] Andra Ionescu, Kiril Vasilev, Florena Buse, Rihan Hai, and Asterios Katsifodimos. Autofeat: Transitive feature discovery over join paths. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 1861–1873. IEEE, 2024.

[57] Deyi Ji, Lanyun Zhu, Siqi Gao, Peng Xu, Hongtao Lu, Jieping Ye, and Feng Zhao. Tree-of-table: Unleashing the power of llms for enhanced large-scale table understanding. *arXiv preprint arXiv:2411.08516*, 2024.

[58] Xingyu Ji, Aditya Parameswaran, and Madelon Hulsebos. Target: Benchmarking table retrieval for generative tasks. In *NeurIPS 2024 Third Table Representation Learning Workshop*, 2024.

[59] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. Chorus: Foundation models for unified data discovery and exploration. *Proceedings of the VLDB Endowment*, 17(8):2104–2114, 2024.

[60] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. Santos: Relationship-based semantic table union search. In *SIGMOD*, 2023.

[61] Aamod Khatiwada, Harsha Kokel, Ibrahim Abdelaziz, Subhajit Chaudhury, Julian Dolby, Oktie Hassanzadeh, Zhenhan Huang, Tejaswini Pedapati, Horst Samulowitz, and Kavitha Srinivas. Tabsketchfm: Sketch-based tabular representation learning for data discovery over data lakes. In *NeurIPS 2024 Third Table Representation Learning Workshop*, 2024.

[62] Laura M Koesten, Emilia Kacprzak, Jenifer FA Tennison, and Elena Simperl. The trials and tribulations of working with structured data: -a study on information seeking behaviour. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 1277–1289, 2017.

[63] Pradap Venkatramanan Konda. *Magellan: Toward building entity matching management systems*. The University of Wisconsin-Madison, 2018.

[64] Keti Korini and Christian Bizer. Column type annotation using chatgpt. In *CEUR Workshop Proceedings*, volume 3462, pages 1–12, 2023.

[65] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. Valentine: Evaluating matching techniques for dataset discovery. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 468–479. IEEE, 2021.

[66] Christos Koutras, Rihan Hai, Kyriakos Psarakis, Marios Fragkoulis, and Asterios Katsifodimos. Sima: Effective and efficient data silo federation using graph neural networks. *arXiv preprint arXiv:2206.12733*, 2022.

[67] Christos Koutras, Jiani Zhang, Xiao Qin, Chuan Lei, Vasileios Ioannidis, Christos Faloutsos, George Karypis, and Asterios Katsifodimos. Omnimatch: Effective self-supervised any-join discovery in tabular data repositories. *arXiv preprint arXiv:2403.07653*, 2024.

[68] Oliver Lehmberg, Dominique Ritze, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. The mannheim search join engine. *J. Web Semant.*, 35:159–166, 2015.

[69] Maurizio Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, page 233–246, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581135076. doi: 10.1145/543613.543644. URL https://doi-org.proxy.library.nyu.edu/10.1145/543613.543644.

[70] Bing Li, Yukai Miao, Yaoshu Wang, Yifang Sun, and Wei Wang. Improving the efficiency and effectiveness for bert-based entity resolution. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(15):13226–13233, 2021.

[71] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. Table-gpt: Table fine-tuned gpt for diverse table tasks. *Proceedings of the ACM on Management of Data*, 2(3):1–28, 2024.

[72] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060*, 2024.

[73] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. Long-context llms struggle with long in-context learning, 2024. URL https://arxiv.org/abs/2404.02060.

[74] Yize Li, Yongchao Dou, Felipe Da Veiga Leprevost, Yifat Geffen, Anna P Calinawan, François Aguet, Yo Akiyama, Shankara Anand, Chet Birger, Song Cao, et al. Proteogenomic data and resources for pan-cancer analysis. *Cancer cell*, 41(8):1397–1406, 2023. PMCID: PMC10506762.

[75] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep entity matching with pre-trained language models. *Proceedings of the VLDB Endowment*, 14(1):50–60, 2020.

[76] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, Jin Wang, Wataru Hirota, and Wang-Chiew Tan. Deep Entity Matching: Challenges and Opportunities. *J. Data and Information Quality*, 13(1):1:1–1:17, January 2021. ISSN 1936-1955. doi: 10.1145/3431816.

[77] Yuliang Li, Jinfeng Li, Yoshi Suhara, AnHai Doan, and Wang-Chiew Tan. Effective entity matching with transformers. *The VLDB Journal*, 32(6):1215–1235, 2023.

[78] Jiaming Liang, Chuan Lei, Xiao Qin, Jiani Zhang, Asterios Katsifodimos, Christos Faloutsos, and Huzefa Rangwala. Featnavigator: Automatic feature augmentation on tabular data. *arXiv preprint arXiv:2406.09534*, 2024.

[79] Jiabin Liu, Chengliang Chai, Yuyu Luo, Yin Lou, Jianhua Feng, and Nan Tang. Feature augmentation with reinforcement learning. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 3360–3372. IEEE, 2022.

[80] Junfei Liu, Shaotong Sun, and Fatemeh Nargesian. Causal dataset discovery with large language models. In *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics*, pages 1–8, 2024.

[81] Yurong Liu, Eduardo Pena, Aecio Santos, Eden Wu, and Juliana Freire. Magneto: Combining small and large language models for schema matching. *arXiv preprint arXiv:2412.08194*, 2024.

[82] Weizheng Lu, Jing Zhang, Ju Fan, Zihao Fu, Yueguo Chen, and Xiaoyong Du. Large language model for table processing: A survey. *Frontiers of Computer Science*, 19(2):192350, 2025.

[83] Chuangtao Ma, Sriom Chakrabarti, Arijit Khan, and Bálint Molnár. Knowledge graph-based retrieval-augmented generation for schema matching. *arXiv preprint arXiv:2501.08686*, 2025.

[84] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *vldb*, volume 1, pages 49–58, 2001.

[85] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[86] Ali Modarressi, Hanieh Deilamsalehy, Franck Dernoncourt, Trung Bui, Ryan A. Rossi, Seunghyun Yoon, and Hinrich Schütze. Nolima: Long-context evaluation beyond literal matching, 2025. URL `https://arxiv.org/abs/2502.05167`.

[87] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 international conference on management of data*, pages 19–34, 2018.

[88] Yang Nan, Javier Del Ser, Simon Walsh, Carola Schönlieb, Michael Roberts, Ian Selby, Kit Howard, John Owen, Jon Neville, Julien Guiot, Benoit Ernst, Ana Pastor, Angel Alberich-Bayarri, Marion I. Menzel, Sean Walsh, Wim Vos, Nina Flerin, Jean-Paul Charbonnier, Eva M. van Rikxoort, Avishek Chatterjee, Henry C. Woodruff, Philippe Lambin, Leonor Cerdá Alberich, Luis Martí-Bonmatí, Francisco Herrera, and Guang Yang. Data harmonisation for information fusion in digital healthcare: A state-of-the-art systematic review, meta-analysis and future research directions. *CoRR*, abs/2201.06505, 2022. URL `https://arxiv.org/abs/2201.06505`.

[89] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. Can foundation models wrangle your data? *Proceedings of the VLDB Endowment*, 16(4):738–746, 2022.

[90] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. Table union search on open data. *Proc. VLDB Endow.*, 11(7):813–825, 2018.

[91] Arash Dargahi Nobari and Davood Rafiei. DTT: An example-driven tabular transformer for joinability by leveraging large language models. *Proceedings of the ACM on Management of Data*, 2(1):1–24, 2024.

[92] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 olmo 2 furious, 2024. URL https://arxiv.org/abs/2501.00656.

[93] omop. Observational medical outcomes partnership (omop) common data model (cdm). https://www.ohdsi.org/data-standardization, 2024.

[94] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

[95] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and Filtering Techniques for Entity Resolution: A Survey. *ACM Comput. Surv.*, 53(2):31:1–31:42, March 2020. ISSN 0360-0300. doi: 10.1145/3377455.

[96] Andrea Papenmeier, Thomas Krämer, Tanja Friedrich, Daniel Hienert, and Dagmar Kern. Genuine information needs of social scientists looking for data. *Proceedings of the Association for Information Science and Technology*, 58(1):292–302, 2021.

[97] Marcel Parciak, Brecht Vandevoort, Frank Neven, Liesbet M Peeters, and Stijn Vansummeren. Schema matching with large language models: an experimental study. *Proceedings of the VLDB Endowment. ISSN*, 2150:8097, 2024.

[98] Ralph Peeters and Christian Bizer. Dual-objective fine-tuning of bert for entity matching. *Proceedings of the VLDB Endowment*, 14:1913–1921, 2021.

[99] Ralph Peeters, Aaron Steiner, and Christian Bizer. Entity matching using large language models. *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, pages 529–541, 2025. doi: 10.48786/EDBT.2025.42. URL https://doi.org/10.48786/edbt.2025.42.

[100] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[101] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[102] Erhard Rahm and Philip A Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10:334–350, 2001.

[103] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*, 2024.

[104] Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. Multitask prompted training enables zero-shot task generalization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=9Vrb9D0WI4`.

[105] Aécio Santos, Aline Bessa, Fernando Chirigati, Christopher Musco, and Juliana Freire. Correlation sketches for approximate join-correlation queries. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1531–1544, 2021.

[106] Aécio Santos, Aline Bessa, Christopher Musco, and Juliana Freire. A sketch-based index for correlated dataset search. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2928–2941. IEEE, 2022.

[107] Aécio Santos, Eduardo HM Pena, Roque Lopez, and Juliana Freire. Interactive data harmonization with llm agents. *arXiv preprint arXiv:2502.07132*, 2025.

[108] Aécio Santos, Flip Korn, and Juliana Freire. Efficiently estimating mutual information between attributes across tables. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 193–206, 2024. doi: 10.1109/ICDE60146.2024.00022.

[109] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *SIGMOD Conference*, volume 10, pages 2213836–2213962, 2012.

[110] Nabeel Seedat and Mihaela van der Schaar. Matchmaker: Self-improving large language model programs for schema matching. In *GenAI for Health: Potential, Trust and Policy Compliance*, 2024.

[111] Sofia Serrano, Zander Brumbaugh, and Noah A. Smith. Language models: A guide for the perplexed, 2023. URL `https://arxiv.org/abs/2311.17301`.

[112] Eitam Sheetrit, Menachem Brief, Moshik Mishaeli, and Oren Elisha. Rematch: Retrieval enhanced schema matching with llms. *arXiv preprint arXiv:2403.01567*, 2024.

[113] Levy Silva and Luciano Barbosa. Improving dense retrieval models with llm augmented data for dataset search. *Knowledge-Based Systems*, 294:111740, 2024.

[114] Rohit Singh, Vamsi Meduri, Ahmed Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. Generating concise entity matching rules. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1635–1638, 2017.

[115] Socrata. `https://open-source.socrata.com`, 2024.

[116] Aaron Steiner, Ralph Peeters, and Christian Bizer. Fine-tuning large language models for entity matching. *arXiv preprint arXiv:2409.08185*, 2024.

[117] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei 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*, pages 645–654, 2024.

[118] Nan Tang, Chenyu Yang, Ju Fan, Lei Cao, Yuyu Luo, and Alon Halevy. Verifai: verified generative ai. *Conference on Innovative Data Systems Research (CIDR)*, 2024.

[119] Saravanan Thirumuruganathan, Han Li, Nan Tang, Mourad Ouzzani, Yash Govind, Derek Paulsen, Glenn Fung, and AnHai Doan. Deep learning for blocking in entity matching: a design space exploration. *Proceedings of the VLDB Endowment*, 14(11):2459–2472, 2021.

[120] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Vincent Zhao, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Pranesh Srinivasan, Laichee Man, Kathleen Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. Lamda: Language models for dialog applications, 2022.

[121] Mohamed Trabelsi, Zhiyu Chen, Shuo Zhang, Brian D Davison, and Jeff Heflin. Strubert: Structure-aware bert for table search and matching. In *Proceedings of the ACM Web Conference 2022*, pages 442–451, 2022.

[122] Immanuel Trummer. Can large language models predict data correlations from column names? *Proc. VLDB Endow.*, 16(13):4310–4323, September 2023. ISSN 2150-8097. doi: 10.14778/3625054.3625066. URL `https://doi.org/10.14778/3625054.3625066`.

[123] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Guoliang Li, Xiaoyong Du, Xiaofeng Jia, and Song Gao. Unicorn: A unified multi-tasking model for supporting matching tasks in data integration. *Proceedings of the ACM on Management of Data*, 1(1):1–26, 2023.

[124] Fei Wang, Kexuan Sun, Muhao Chen, Jay Pujara, and Pedro Szekely. Retrieving complex tables with multi-granular graph representation learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1472–1482, 2021.

[125] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11), 2012.

[126] Qiming Wang and Raul Castro Fernandez. Solo: Data discovery using natural language questions via a self-supervised approach. *Proceedings of the ACM on Management of Data*, 1(4):1–27, 2023.

[127] Runhui Wang, Yuliang Li, and Jin Wang. Sudowoodo: Contrastive self-supervised learning for multi-purpose data integration and preparation. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 1502–1515. IEEE, 2023.

[128] Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xuanang Chen, Xianpei Han, Hao Wang, Zhenyu Zeng, and Le Sun. Match, compare, or select? an investigation of large language models for entity matching. *arXiv preprint arXiv:2405.16884*, 2024.

[129] Yaoshu Wang and Mengyi Yan. Unsupervised domain adaptation for entity blocking leveraging large language models. In *2024 IEEE International Conference on Big Data (BigData)*, pages 159–164. IEEE, 2024.

[130] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2022.

[131] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL `http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html`.

[132] Lindsey Linxi Wei, Guorui Xiao, and Magdalena Balazinska. Racoon: An llm-based framework for retrieval-augmented column type annotation with a knowledge graph. *NeurIPS Third Table Representation Learning Workshop*, 2024.

[133] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9, 2016.

[134] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. Zeroer: Entity resolution using zero labeled examples. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1149–1164, 2020.

[135] Shiwen Wu, Qiyu Wu, Honghua Dong, Wen Hua, and Xiaofang Zhou. Blocker and matcher can mutually benefit: A co-learning framework for low-resource entity resolution. *Proceedings of the VLDB Endowment*, 17(3):292–304, 2023.

[136] Yongqin Xu, Huan Li, Ke Chen, and Lidan Shou. Kcmf: A knowledge-compliant framework for schema and entity matching with fine-tuning-free llms. *arXiv preprint arXiv:2410.12480*, 2024.

[137] Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. Hallucination is inevitable: An innate limitation of large language models. *arXiv preprint arXiv:2401.11817*, 2024.

[138] Mengyi Yan, Yaoshu Wang, Kehan Pang, Min Xie, and Jianxin Li. Efficient Mixture of Experts based on Large Language Models for Low-Resource Data Preprocessing. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pages

3690–3701, New York, NY, USA, August 2024. Association for Computing Machinery. doi: 10.1145/3637528.3671873.

[139] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Transactions on Knowledge Discovery from Data*, 18(6):1–32, 2024.

[140] Jun Yang, Peng Gong, Rong Fu, Minghua Zhang, Jingming Chen, Shunlin Liang, Bing Xu, Jiancheng Shi, and Robert Dickinson. The role of satellite remote sensing in climate change studies. *Nature climate change*, 3(10):875–883, 2013.

[141] Shunyu Yao, Jeffrey Zhao, Dian Yu, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022. URL `https://openreview.net/forum?id=tvI4u1ylcqs`.

[142] Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 174–184, 2023.

[143] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. TaBERT: Pretraining for joint understanding of textual and tabular data. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.745. URL `https://aclanthology.org/2020.acl-main.745`.

[144] Brit Youngmann, Michael Cafarella, Babak Salimi, and Anna Zeng. Causal data integration. *arXiv preprint arXiv:2305.08741*, 2023.

[145] Xiaocan Zeng, Pengfei Wang, Yuren Mao, Lu Chen, Xiaoze Liu, and Yunjun Gao. Multiem: Efficient and effective unsupervised multi-table entity matching. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 3421–3434. IEEE, 2024.

[146] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. Jellyfish: Instruction-tuning local large language models for data preprocessing. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8754–8782, 2024.

[147] Haoxiang Zhang, Yurong Liu, Aécio Santos, Juliana Freire, et al. Autoddg: Automated dataset description generation using large language models. *arXiv preprint arXiv:2502.01050*, 2025.

[148] Jing Zhang, Bonggun Shin, Jinho D Choi, and Joyce C Ho. Smat: An attention-based deep learning solution to the automation of schema matching. In *Advances in Databases and Information Systems: 25th European Conference, ADBIS 2021, Tartu, Estonia, August 24–26, 2021, Proceedings 25*, pages 260–274. Springer, 2021.

[149] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M Procopiuc, and Divesh Srivastava. Automatic discovery of attributes in relational databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 109–120, 2011.

[150] Meihui Zhang, Zhaoxuan Ji, Zhaojing Luo, Yuncheng Wu, and Chengliang Chai. Applications and challenges for large language models: From data management perspective. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 5530–5541. IEEE, 2024.

[151] Shuo Zhang and Krisztian Balog. Ad hoc table retrieval using semantic similarity. In *Proceedings of the 2018 world wide web conference*, pages 1553–1562, 2018.

[152] Shuo Zhang and Krisztian Balog. Web table extraction, retrieval, and augmentation: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(2):1–35, 2020.

[153] Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. Tablellama: Towards open large generalist models for tables, 2023.

[154] Yunjia Zhang, Avrilia Floratou, Joyce Cahoon, Subru Krishnan, Andreas C Müller, Dalitso Banda, Fotis Psallidas, and Jignesh M Patel. Schema matching using pre-trained language models. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 1558–1571. IEEE, 2023.

[155] Yunyi Zhang, Ming Zhong, Siru Ouyang, Yizhu Jiao, Sizhe Zhou, Linyi Ding, and Jiawei Han. Automated mining of structured knowledge from text in the era of large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6644–6654, 2024.

[156] Yilun Zhao, Lyuhao Chen, Arman Cohan, and Chen Zhao. Tapera: enhancing faithfulness and interpretability in long-form table qa by content planning and execution-based reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12824–12840, 2024.

[157] Tianyang Zhong, Zhengliang Liu, Yi Pan, Yutong Zhang, Yifan Zhou, Shizhe Liang, Zihao Wu, Yanjun Lyu, Peng Shu, Xiaowei Yu, et al. Evaluation of openai o1: Opportunities and challenges of agi. *arXiv preprint arXiv:2409.18486*, 2024.

[158] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[159] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. LSH ensemble: Internet-scale domain search. *Proc. VLDB Endow.*, 9(12):1185–1196, 2016.

[160] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. JOSIE: overlap set similarity search for finding joinable tables in data lakes. In *SIGMOD*, pages 847–864, 2019.

LLMs and Databases: A Synergistic Approach to Data Utilization

Fatma Özcan, Yeounoh Chung, Yannis Chronis, Yu Gan, Yawen Wang
Carsten Binnig, Johannes Wehrstein, Gaurav Kakkar, Sami Abu-el-haija
Google Inc

Abstract

Large language models (LLMs) are not merely changing computing; they are igniting a revolution across industries, from healthcare to finance. Their prowess in tackling complex problems, demonstrated by breakthroughs in chatbots, translation, and code generation, is undeniable. A notable breakthrough in data management, driven by LLMs, is the advancement of natural language to SQL translation. This technology has fueled substantial progress, making database interactions more accessible and enabling the deployment of numerous real-world applications. Yet, even these powerful models falter with latency-sensitive regression problems, a critical need in database performance optimization. Inspired by the foundational principles of LLMs, we are developing pre-trained cardinality estimation and foundation database models, bridging this gap and unlocking the next generation of database optimization.

1 Introduction

Large Language Models (LLMs), a key part of GenAI, have captured the attention of many people, technical and non-technical alike. LLMs are bringing significant changes to the field of computing and this is not just an incremental step; it's a fundamental shift on how to approach computing problems and data. Notably, LLMs have demonstrated a powerful capability to tackle challenging problems across diverse domains, including data management. For instance, LLMs have greatly improved NL2SQL (converting natural language to SQL queries), leading to high accuracy solutions and making commercial applications more feasible. Beyond NL2SQL, LLMs have the potential to help with other difficult data management tasks like data integration, data discovery, and semantic understanding of tables and schemas.

LLMs, trained with all data available on the Internet, are very adept at question answering over unstructured context. Combining LLMs with databases allows us to analyze structured and unstructured data together in innovative ways [1–4]. These emerging systems explore how to augment RDBMS with pre-trained knowledge of LLMs to drive insights from a broader knowledge base.

For years, the seamless integration of structured and unstructured data has remained an elusive goal for enterprises. Traditional approaches, relying on ETL and predefined entity extraction, suffer from inherent limitations: rigid schemas and batch-oriented processing lead to significant delays. However, the advent of Large Language Models (LLMs) is ushering in a paradigm shift. LLMs, with their inherent ability to process unstructured data, combined with database capabilities, are unlocking unprecedented analytical power. Innovative systems, such as Lotus [1] and Palimzest [2], are leveraging LLM-powered operators to facilitate on-demand information extraction from unstructured sources. This eliminates the need for predefined entities and minimizes processing latency, unlocking new possibilities for AI-driven data analysis.

The synergy between databases and LLMs extends far beyond their respective handling of structured and unstructured data. Databases operate within a closed-world paradigm, relying solely on their

stored information, while LLMs possess vast, pre-trained knowledge derived from the entire internet. Furthermore, their primary access mechanisms, SQL and natural language, respectively, offer distinct advantages. To realize the full potential of data analytics, we must explore more innovative ways to integrate these contrasting strengths.

LLMs have shown promise in addressing certain database challenges, including data wrangling[5] and semantic query equivalence checking[6]. These initial studies highlight the potential of LLMs for data management. However, LLMs currently struggle with regression tasks[7] and latency-sensitive problems, which are prevalent in database performance optimization. For example, cardinality estimation is one such latency sensitive regression problem, which can be solved more accurately using traditional ML models.

Early work on ML-based cardinality estimation focused on instance-based models, where a new model is trained for each data set and workload. However, one significant learning from LLMs's success is the importance of pre-trained models that can generalize to new tasks and data sets, with or without fine-tuning. Building on this observation, we build pre-trained cardinality estimation models. Inspired further by foundation models, we also explore the building blocks for database foundation models, and build data and query experts, which can be combined to solve many database performance problems.

In this paper, we first describe our findings and insights using LLMs to solve natural language to SQL. Then, we present our research on pre-trained cardinality estimation models, followed by foundation database models, which are inspired by foundation language models. Finally, we conclude with future research directions.

2 Natural Language to SQL with LLM

One particular domain that stands out to benefit from recent advancements in LLMs is Natural Language to SQL (NL2SQL). NL2SQL is a challenging task that requires translating ambiguous natural language questions into structured SQL queries over complex data schema [8]. Early approaches in NL2SQL relied on rule-based semantic parsing or seq-to-seq language models treating NL2SQL as a machine translation problem [9–11] with limited success. Recently, the latest LLMs with their superior language understanding and code generation capabilities have revolutionized the NL2SQL landscape. Since mid-2023, LLM-based approaches have dominated popular NL2SQL benchmarks, significantly outperforming earlier methods based on semantic parsing and machine translation [12–17].

LLMs lack the pre-trained knowledge of database schemas. Hence, each new database schema, with its different data model and semantics, pose a new challenge. This missing information needs to be provided as context to the LLM model. As such, we focus on schema linking and creating the right context for LLMs in our NL2SQL research.

2.1 Schema linking for NL2SQL

While LLMs excel at queries with less semantic ambiguity and simple schema, they can still struggle with more nuanced and ambiguous questions asked over complex data schema. Rather than directly translating natural language questions into SQL queries, LLMs can link relevant schema elements (e.g., tables, columns) to the natural language questions and generate SQL queries that align with users' intent. This so called schema linking is crucial for generating accurate and executable SQL queries. Many LLM based NL2SQL systems have focused on improving schema linking accuracy via careful relevant table and column retrieval [18, 19]. Recent studies [12, 20] showed that LLMs can generate accurate SQL queries given the entire database schema. The enhanced reasoning and "near-perfect" retrieval capabilities [21] of the latest LLMs, such as *gemini-1.5*, enable accurate schema linking during inference.

Table 1: Schema Selection Performance (TBR: Table Retrieval, CR: Column Retrieval) and Execution Accuracy on BIRD dev. Using the entire DB table schema without any relevant column selection yields comparable performance to using one of the state-of-the-art schema linking method; the ground truth TBR and CR results in the prompt boosts the SQL generation accuracy significantly.

	Filtered Schema [19]	All DB Schema [20]	Ground Truth Schema
Ex Acc (%)	64.08	64.80	72.43
TBR Recall (%)	97.69	100	100
TBR Precision(%)	89.72	34.29	100
CR Recall(%)	97.12	-	100
CR Precision(%)	69.43	-	100

Table 1 illustrates that the perfect relevant table and column retrievals (Ground Truth Schema) lead to accurate schema linking and give a significant boost to the generated SQL accuracy. It is important to note that such perfect table retrieval and column retrieval are infeasible in practice due to the question ambiguity as well as the semantic complexity of the underlying schema. High quality, but expensive, table and column selection can be done via multiple LLM calls (Filtered Schema [19]), and alternatively, one can pass the entire DB schema to the prompt, skipping relevant table and column retrievals entirely. In [20] we further study this alternative approach leveraging the extended context window of the latest LLMs, and evaluate the trade-offs of passing more contextual information for in-context learning.

2.2 In-context learning for NL2SQL

While providing table and column schema aids schema linking, it does not ensure accurate SQL generation. The model must comprehend the semantic correlation between the user's question and actual SQL query. This is further complicated by the absence of domain knowledge. For instance, consider a query such as "find patients with normal blood pressure". "Normal" is a domain-specific term that requires context and interpretation. Moreover, text search semantics frequently deviate from SQL semantics. A user might ask "find me all account holders in south bohemia," but it remains ambiguous how to translate "south bohemia" into a SQL predicate. Depending on the actual literal stored in the table, "South Bohemia", "south Bohemia", or "SOUTH BOHEMIA" are all possible options. Entity recognition, especially with proper nouns, introduces additional complexity.

One promising approach to address these challenges is in-context learning (ICL). ICL is a popular technique to guide the LLM to perform new tasks by providing a few-shot examples in the prompt. It has been demonstrated to improve the performance of LLMs in many challenging NLP tasks, such as math and reasoning [22]. We have found that in-context example selection is critical to NL2SQL accuracy. By selecting text2SQL example pairs similar to the user question, we can provide additional references about the schema, domain knowledge, and even help resolve literal issues.

Figure 1 illustrates the impact of example selection on NL2SQL generation. Our findings indicate that semantically similar examples within the prompt context lead to significant accuracy improvements. While even random examples from the same database schema provide some benefit, likely by exposing semantic relationships between question types and query structures, similarity-based example selection yields superior results. This improvement suggests that relevant examples convey valuable, schema-specific knowledge that enhances the capabilities of off-the-shelf LLMs. The highest accuracy is observed when the ground truth example is included, indicating that LLMs are adept at recognizing and leveraging correct examples but are less proficient at generating accurate queries without such guidance. However, it is important to note that even with ground truth examples, perfect accuracy is not achieved. This

implies a potential for LLM overconfidence, where pre-training biases can lead to incorrect generations despite the availability of correct examples.

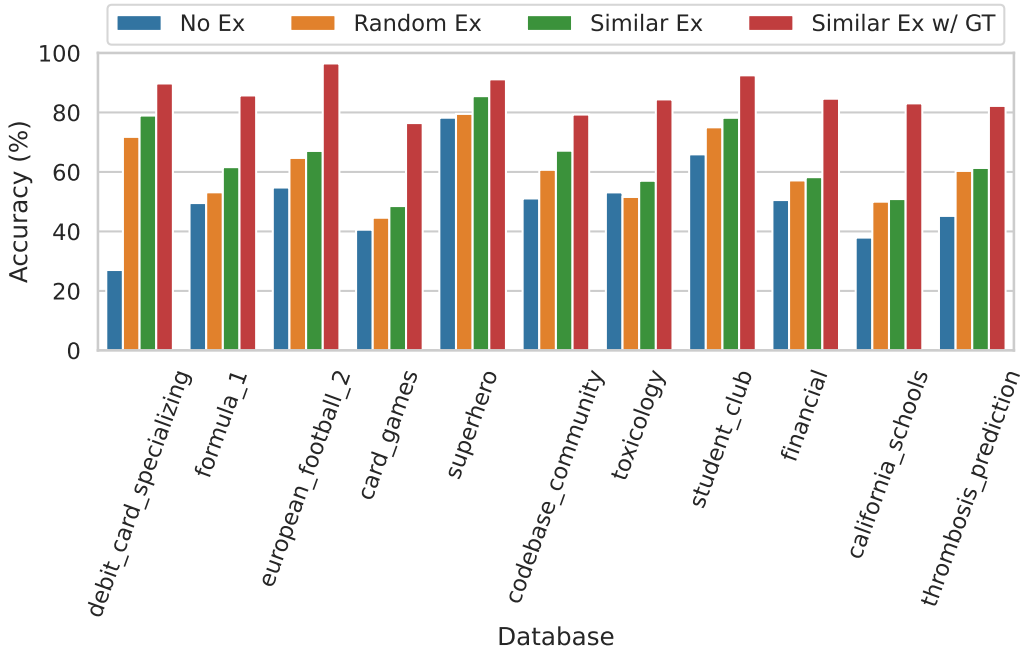

Figure 1: Generation accuracy with different ICL strategies on different databases from BIRD Bench. "No Ex": No examples are provided in the prompt. "Random Ex": 10 random examples from the same database schema are included in the prompt. "Similar Ex": The top 10 examples most similar to the question, based on question embedding similarity, are provided in the prompt. "Similar Ex w/ GT": The ground truth example is provided along with 9 other similar examples based on question embedding similarity.

2.3 Self-consistency for NL2SQL

Another technique that we see in the latest state-of-the-arts in the BIRD benchmark [13] leaderboard is self-consistency [14, 23]. The idea is to generate multiple output candidates via repeated runs and the most common (majority voting) or the most likely (specialized picker) output is selected. In [14], we use three different NL2SQL pipelines to generate a more diverse set of candidate pool, which result in a higher overall accuracy paired with a fine-tuned picker LLM model. While the strong results demonstrate how the stochastic nature of LLM generation can be exploited to address more ambiguous user queries and tasks, the ambiguous user queries and complex data schema still remain as unsolved challenges. Self-consistency also raises concerns about the increased number of LLM calls and the cost and latency of NL2SQL output generation. Therefore, improving the efficiency of these multi-step LLM interactions is a critical research priority.

2.4 Long context for NL2SQL

Much of LLM-based NL2SQL research and solutions assumed a limited context size (typically smaller than 8k tokens, which is just enough hold a few select table schema) and also degrading performance over increasing amount of contextual information [24]. As such, many NL2SQL pipelines focused on retrieving a handful of top-K table schema along with 3-5 examples or chain-of-thought (CoT)

demonstrations [15, 25]. Alternatively, in a recent study [20] we explored the potential of leveraging the long context LLMs, like *gemini-1.5* with 2-million tokens context limit, with lots of extra contextual information(e.g., entire DB table schema, hundreds of examples for ICL). In particular, we explored including entire database schemas, hints, tens of example queries, sample column values, as well as entire distinct values of columns. We study in detail the utility and cost of various long context-based strategies in [20]. The results indicate that the long context model exhibits a very strong retrieval capability over the extended context window for NL2SQL and is robust to extra irrelevant information in the prompt. This means that NL2SQL generation does not need to be impaired by the imperfect performance of retrieval services – we can retrieve more to ensure higher recall at the cost of lower precision.

3 Using Traditional ML for Latency-sensitive Database Problems

Recently, machine learning models have been used effectively in many database problems[26], showing a lot of promise. These early works show that there are many database problems where ML can benefit database systems, including run-time prediction [27], and query optimization [28]. All such internal database optimization tasks have in common that they have strict latency requirements as they happen at query time and they are data dependent. Cardinality estimation (CE), as one of the key building block in modern databases for optimizing performance, has been studied extensively as a target for many ML based approaches [29, 30].

Traditional CE techniques used in modern database systems have well-known limitations, as they make simplistic data modeling assumptions, such as data uniformity and independence of columns in the tables. Recent research has explored learned CE, employing machine learning for improved accuracy. Learned CE model following an instance-based approach have been shown to improve upon the state-of-art techniques used by database systems today [29]. Instance-based models are trained and evaluated on a specific database instance, requiring retraining for each new dataset. This category encompasses workload-driven models that are trained using a representative set of queries executed on the target database, incurring significant overhead and data-driven models that learn the data distribution within the database, avoiding query execution but still requiring retraining upon data updates. Despite their better accuracy, such learned CE models have not been adapted in practice, due to their high training overheads, among other reasons. As a result, pre-trained CE models, which are trained on a corpus of diverse datasets on transferable features, are highly desirable to alleviate the training costs and increase adaption.

Zero-shot models: This emerging paradigm aims for generalizability across diverse datasets without retraining. Pre-trained on a variety of databases, these models leverage transferable features to adapt to unseen data and are more robust to data modifications. We developed such pre-trained CE estimation models, which we describe next.

3.1 Cardinality Estimation Model

Recognizing the inherent graph representation of SQL queries, we explore the following two model architectures for the cardinality estimation task: Graph Neural Network (GNN) and Graph Transformer.

The design of the GNN model is based on the GNN cost model from [31]. Each node's features are passed through a separate Multi-Layer Perceptron (MLP) based on the node type. The GNN then mirrors the query execution order to propagate information in a topological order from leaf to the root node. Finally a learned graph-level embedding from the root node is used to predict query cardinality.

In addition to GNN, we build a Graph Transformer model which extends the attention mechanism to graph-structured data. Inspired by Graphormer [32], we make several modifications on top of the traditional transformer architecture to better capture graph complexity. These changes include

heterogeneous input embedding to accommodate node-type specific feature types, shortest-path spatial encoding as an attention bias to capture structural relationships between nodes, directional causal masking to enforce topological ordering, and a virtual node connected to all other nodes used for a comprehensive graph-level representation for the final cardinality prediction.

3.2 CardBench

To train and test the pre-trained CE models, we need a new benchmark. Existing benchmarks for cardinality estimation models often rely on limited datasets, like the Join Order Benchmark (JOB) with its single IMDb dataset. While JOB offers realistic data distributions, a single dataset is not sufficient for zero-shot training. Recent efforts have introduced new benchmarks with additional datasets, but they still fall short in representing the wide range of real-world data required for training and testing pre-trained models.

To close this gap, we have developed and open-sourced CardBench[33][1], a benchmark containing thousands of queries on 20 distinct databases, and scripts to compute data summary statistics and generate queries. CardBench contains 20 distinct datasets and thousands of queries of different complexities. Our goal is to foster further research in the area of learned CE, with a focus on enabling the training and testing of pre-trained zero-shot CE models.

CardBench datasets were chosen to be diverse, complex and cover a wide range of data distributions to stress CE models. All the datasets are publicly available or are based on publicly available datasets (we list the sources in the CardBench github repository). In comparison to existing CE benchmarks, CardBench includes a much higher number of datasets and training data (i.e., queries and cardinalities) to experiment with and compare different types of learned CE approaches. Our training data are SQL queries represented as graphs annotated with dataset statistics and the query cardinality as the label (Table 2, right). As shown in Figure 2, we represent the SQL query as a graph. The graph nodes are annotated with relevant the features described in [33]. The query graph has table nodes (in blue), column nodes (in green), operator nodes (in red), predicates (in yellow) and correlations across column of the same table (white circles with red outline).

CardBench includes three sets of training data, Single Table, Binary Join and Multi-Join. Single Table queries filter a single table using 1 to 4 filter predicates. Binary Join queries join two tables that are also filtered with 1 to 3 filter predicates per table. Multi Join queries perform 0 to 8 joins and also apply 0 to 2 filter predicates per table. The three training dataset configurations can demonstrate the challenges of more complex queries on CE models. Next, we present our experiments using Cardbench, with the two pre-trained CE estimation models that we trained.

3.3 Experiments with CardBench

In this section, we present our empirical findings, evaluating the accuracy of various model configurations for cardinality estimation. We focus on the *Binary Join* CardBench dataset that contain queries with 0 to 1 joins, and 1-3 predicates on each table, as often the majority of queries fall in this category (in the recently released dataset by Amazon [34] more than 90% of the queries have 0 to 1 joins).

We use the q-error, which is a common metric in database systems research for evaluating the accuracy of CE models[30]. It calculates the relative deviation of the predicted cardinality from the true cardinality for a given query. The q-error for a single query is calculated as follows:

[1] `https://github.com/google-research/google-research/CardBench_zero_shot_cardinality_training`

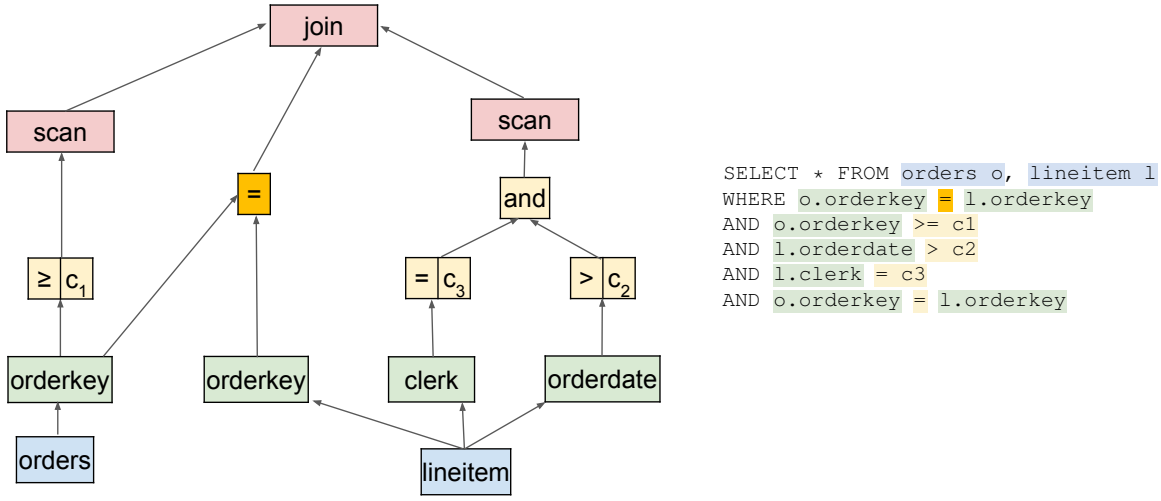

Figure 2: Foundation database models can generalize across tasks and datasets which is very different from current approaches where we need to train multiple models either per task (Zero-shot) or per dataset (Multi-Task) or even per combination (Instance-specific).

$$\text{q-error} = \max\left(\frac{\hat{y}}{y}, \frac{y}{\hat{y}}\right) \in [1, +\infty), \tag{1}$$

where y and $\hat{y}$ represent true and predicted cardinalities, respectively.

We assess CE performance using three distinct ML model configurations: instance-based, zero-shot, and fine-tuned.

Instance-based Models: In this configuration, we train and evaluate individual models using a single dataset. For each dataset, queries are randomly partitioned into training and validation sets with 85:15 train-validation split. An additional 500 queries are reserved as a standalone test set to evaluate final model accuracy.

Zero-shot Models: This configuration investigates the generalizability of cardinality prediction models to unseen datasets, i.e., data from another dataset apart from the training data. We train and validate the model on queries from 19 datasets, maintaining the 85:15 train-validation split. The model is then tested on the 20th dataset (not used in training). This test set contains the same queries used in the instance-based setting to ensure a fair comparison.

Fine-tuned Models: This configuration investigates whether fine-tuning a pre-trained zero-shot model improves accuracy and sample efficiency. We fine-tune the model initially trained on 19 datasets using the 20th dataset. The fine-tuned model's accuracy is then evaluated using the same 500 holdout queries from this 20th dataset.

The average training and inference times under each configuration are 2. We observe that training the zero-shot model over extended epochs can often result in lower accuracy due to over-fitting to the training set. Therefore, we cap the maximum number of training epochs for zero-shot training to 20, and set a maximum training epochs of 100 for the other two configurations. The graph transformer model size is 33.6MB, with 8.4M parameters in total, the GNN model size is 7.5MB, with 1.88M parameters in total.

We compare the learned cardinality approaches against a the cardinality estimation of PostgreSQL[2].

[2]https://www.postgresql.org/

Model	Training Time	Inference Time
Instance-Based (GNN)	1.3hr	
Zero-Shot (GNN)	1.5hr	35ms
Fine-Tuned (GNN)	11min	
Instance-Based (Transformer)	11.1hr	
Zero-Shot (Transformer)	11.1hr	97ms
Fine-Tuned (Transformer)	2.1hr	

Table 2: Training and Inference times for difference model types. The inference time of the zero-shot, fine-tuned and instance-based configurations of the same model type are the same since they share the same architecture and model size. For training a V100 GPU was used.

To get the estimates we load the CardBench data in PostgreSQL and use the *explain* command, similar to [31]. We call this baseline *Postgres*.

Cardinality prediction for queries with binary joins is challenging due to cross-table distribution and multi-column correlation, which are difficult to model, significantly impact join cardinality. Table 3 shows the P50, P75, P90 and P95 q-errors for queries with binary joins. The baseline algorithm exhibits significant inaccuracy in binary join query cardinality estimation, with a median q-error of 55.54 and P95 q-error of 4.4×10^6, respectively.

Learning-based models significantly outperform the baseline though, particularly at the 95th percentile. Instance-based models, which learn cross-table distributions within a dataset, achieve low median q-errors of 1.16 (GNN) and 1.20 (transformer). While their P95 q-errors are higher (37.55 for GNN, 43.96 for transformer), the results suggest that estimating join cardinality via learned distributions is feasible. However, out-of-distribution cardinality estimation for binary joins proves more challenging, as zero-shot models experience a dramatic increase in q-errors (up to 20x at the median, 5300x at P95). However, even a small amount of data used for fine-tuning improves the accuracy of pre-trained models considerably. Hence, we conclude that pre-trained models with fine-tuning is a viable approach to cardinality estimation.

Model	Q_{err}^{50}	Q_{err}^{75}	Q_{err}^{90}	Q_{err}^{95}
Postgres	5.29	21.03	38557.10	411511.45
GNN Instance	1.17	1.82	10.92	37.55
GNN Zeroshot	22.66	102.12	3430.23	16271.84
GNN Finetune	**1.32**	**2.45**	**13.62**	**109.77**
Transformer Instance	1.20	1.99	11.93	43.96
Transformer Zeroshot	24.88	264.06	6000.99	228499.79
Transformer Finetune	79.52	1.57	4.19	47.14

Table 3: Average q-error percentiles for Binary Join Queries, the best accuracy are in bold.

4 Foundation DB Models

As discussed in the previous section, the state-of-the-art is one-off models that need to be trained individually per task and even per dataset, which causes extremely high training overheads. Hence, a new learning paradigm is needed that moves away from such one-off models towards generalizable

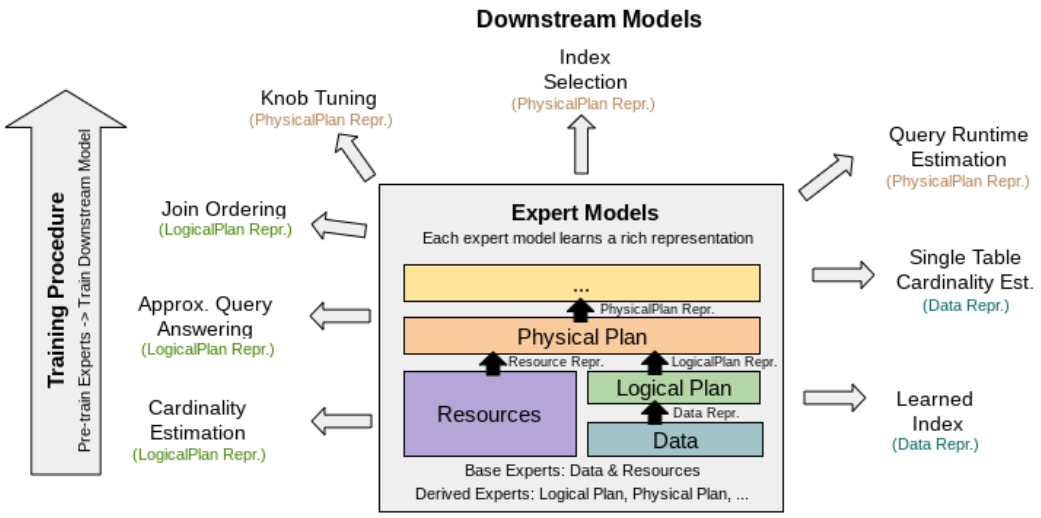

Figure 3: Foundation database models build on a mixture pre-trained experts where some experts learn representations independently (e.g., the data expert) and experts that enrich representations (e.g., the logical plan and physical plan expert). Shallow downstream models take these representation as input and solve a particular database task.

models that can be used with only minimal overhead for an unseen dataset on a wide spectrum of tasks. Recent advances in LLMs have proven that generalizable (i.e., foundation) models for text, coupled with fine-tuning, can be used to solve a wide variety of NLP problems [35]. In this section, inspired by LLMs, we propose a new direction which we call *foundation database models*[36], which are pre-trained in both task-agnostic and dataset-agnostic manner, making it possible to use the model with low overhead to solve a wide spectrum of downstream tasks on unseen datasets. In [36], we propose a vision for foundation database models, and describe our initial prototype and findings.

We make the observation that a set of foundational experts – data, resource, logical and physical query plan – could be used to solve many database problems, including cardinality estimation, index selection, run-time estimation, materialized view selection, partitioning and clustering key selection, etc. (see . Figure 3).

The insight to realize such a foundation model for database problems is that we use a mixture of pre-trained expert models as shown in Figure 3, which enable generalizability along the two dimensions: (1) To generalize across datasets, we provide a data expert that learns to summarize databases into learned embeddings which represent the characteristics of a given dataset. (2)Foundation database models come with additional pre-trained experts that enrich the data expert to solve a wide range of downstream tasks with only low overheads.

We pre-train expert models in a modular manner, each with a set of inputs that could be raw data, engineered features or learned embedding vectors from other expert models. This is where we differ from foundation language models that are trained end-to-end.

For our initial validation, we developed three expert models: the data expert, a logical plan expert, and a physical plan expert. The data expert does not use database specific information such as particular constants in the data or attribute and table names to learn the embedding, but it uses a new transferable-encoding of databases which allows the data expert to learn general data characteristics of a database such as data distributions and correlations across columns. The logical plan expert takes embeddings produced by the data expert as input, in addition to other features, and learns how various query operators modify their input data. The logical plan representation as such enriches the data

representation and can be used to solve tasks such as cardinality estimation or even approximate query answering where this information is needed. Finally, the physical plan expert also learns how logical operators are executed using various implementations.

In [36], we report our initial findings for cardinality and run-time estimation. For cardinality estimation at P50, our model achieves a q-error of 2.12 without fine-tuning, and 1.69 with fine-tuning compared to the q-error of 1.98 for Postgres, for a workload of queries with up to 2 joins and 5 filters. Our model does even better at the tail (P95), with q-error of 92.92 without fine-tuning and 26.08 with fine-tuning compared to the q-error of 294.15 for Postgres. Once again, we observe that fine-tuning is essential to achieve high accuracy.

We believe our initial results are quite promising, and more work is needed to investigate more experts, and a wider spectrum of downstream database problems. We also argue that different pre-training tasks need to be explored to identify the most effective way of training these models.

5 Concluding Remarks

In this paper, we described our experiences and insights using LLMs for natural language to SQL generation. While significant progress has been possible with the use of LLMs, there is still a knowledge gap in understanding enterprise schemas and data semantics for high accuracy.

We also argued using traditional ML models for solving latency sensitive regression problems, such as cardinality estimation. We presented two approaches, both inspired by LLMs. The first one pre-trains a model using a number of diverse datasets, and using database agnostic input features, and is able to generalize to an unseen dataset with fine-tuning, using small amounts of training data. The second one takes this one step further and argues for foundational database models that are generalizable both across datasets as well as across multiple database problems.

We posit that both foundational database models and LLMs hold significant promise for future research and the development of novel data management solutions.

Acknowledgement

We would like to thank to our many collaborators, who helped with various parts of the work reported in this paper; including Shobha Vasudevan, Chelsea Lin, Jiaxun Wu, Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Sercan Arik, Brenton Milne, Reza Sherkat, Per Jacobsson, Aleksandras Surna, and Mahadevan Sankar.

References

[1] L. Patel, S. Jha, C. Guestrin, and M. Zaharia, "Lotus: Enabling semantic queries with llms over tables of unstructured and structured data," *arXiv preprint arXiv:2407.11418*, 2024.

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

[3] A. Biswal, L. Patel, S. Jha, A. Kamsetty, S. Liu, J. E. Gonzalez, C. Guestrin, and M. Zaharia, "Text2sql is not enough: Unifying ai and databases with tag," *arXiv preprint arXiv:2408.14717*, 2024.

[4] M. Urban and C. Binnig, "Caesura: Language models as multi-modal query planners," in *Conference on Innovative Data Systems Research*, 2024.

[5] A. Narayan, I. Chami, L. Orr, S. Arora, and C. Ré, "Can foundation models wrangle your data?" *arXiv preprint arXiv:2205.09911*, 2022.

[6] B. Haynes, R. Alotaibi, A. Pavlenko, J. Leeka, A. Jindal, and Y. Tian, "Geqo: Ml-accelerated semantic equivalence detection," *Proceedings of the ACM on Management of Data*, vol. 1, no. 4, pp. 1–25, 2023.

[7] M. Lukasik, H. Narasimhan, S. Kumar, F. Yu, and A. Menon, "Regression aware inference with llms," in *EMNLP Findings*, 2024.

[8] A. Floratou, F. Psallidas, F. Zhao, S. Deep, G. Hagleither, W. Tan, J. Cahoon, R. Alotaibi, J. Henkel, A. Singla, A. V. Grootel, B. Chow, K. Deng, K. Lin, M. Campos, K. V. Emani, V. Pandit, V. Shnayder, W. Wang, and C. Curino, "Nl2sql is a solved problem... not!" in *Conference on Innovative Data Systems Research*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:266729311

[9] A. Wong, L. Pham, Y. Lee, S. Chan, R. Sadaya, Y. Khmelevsky, M. Clement, F. W. Y. Cheng, J. Mahony, and M. Ferri, "Translating natural language queries to sql using the t5 model," in *2024 IEEE International Systems Conference (SysCon)*. IEEE, 2024, pp. 1–7.

[10] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, "Tabert: Pretraining for joint understanding of textual and tabular data," *arXiv preprint arXiv:2005.08314*, 2020.

[11] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *arXiv preprint arXiv:1709.00103*, 2017.

[12] K. Maamari, F. Abubaker, D. Jaroslawicz, and A. Mhedhbi, "The death of schema linking? text-to-sql in the age of well-reasoned language models," *arXiv preprint arXiv:2408.07702*, 2024.

[13] 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*, vol. 36, 2024.

[14] M. Pourreza, H. Li, R. Sun, Y. Chung, S. Talaei, G. T. Kakkar, Y. Gan, A. Saberi, F. Özcan, and S. Ö. Arık, "Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql," *arXiv preprint arXiv:2410.01943*, 2024.

[15] M. Pourreza and D. Rafiei, "Din-sql: Decomposed in-context learning of text-to-sql with self-correction," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[16] H. Li, J. Zhang, C. Li, and H. Chen, "Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 11, 2023, pp. 13 067–13 075.

[17] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018.

[18] H. A. Caferoğlu and Ö. Ulusoy, "E-sql: Direct schema linking via question enrichment in text-to-sql," *arXiv preprint arXiv:2409.16751*, 2024.

[19] S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini, and A. Saberi, "Chess: Contextual harnessing for efficient sql synthesis," *arXiv preprint arXiv:2405.16755*, 2024.

[20] Y. Chung, G. T. Kakkar, Y. Gan, B. Milne, and F. Ozcan, "Is long context all you need? leveraging llm's extended context for nl2sql," *arXiv preprint arXiv:2501.12372*, 2025.

[21] M. Reid, N. Savinov, D. Teplyashin, D. Lepikhin, T. Lillicrap, J.-b. Alayrac, R. Soricut, A. Lazaridou, O. Firat, J. Schrittwieser *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[22] Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, T. Liu, B. Chang, X. Sun, L. Li, and Z. Sui, "A survey on in-context learning," 2024. [Online]. Available: https://arxiv.org/abs/2301.00234

[23] Y. Gao, Y. Liu, X. Li, X. Shi, Y. Zhu, Y. Wang, S. Li, W. Li, Y. Hong, Z. Luo, J. Gao, L. Mou, and Y. Li, "Xiyan-sql: A multi-generator ensemble framework for text-to-sql," *arXiv preprint arXiv:2411.08599*, 2024. [Online]. Available: https://arxiv.org/abs/2411.08599

[24] 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*, vol. 12, pp. 157–173, 2024. [Online]. Available: https://aclanthology.org/2024.tacl-1.9

[25] L. Nan, Y. Zhao, W. Zou, N. Ri, J. Tae, E. Zhang, A. Cohan, and D. Radev, "Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies," *arXiv preprint arXiv:2305.12586*, 2023.

[26] G. Li, X. Zhou, and L. Cao, "Machine learning for databases," in *Proceedings of the First International Conference on AI-ML Systems*, 2021, pp. 1–2.

[27] T. Kraska, U. F. Minhas, T. Neumann, O. Papaemmanouil, J. M. Patel, C. Ré, and M. Stonebraker, "Ml-in-databases: Assessment and prognosis." *IEEE Data Eng. Bull.*, vol. 44, no. 1, pp. 3–10, 2021.

[28] C. Anneser, N. Tatbul, D. Cohen, Z. Xu, P. Pandian, N. Laptev, and R. Marcus, "Autosteer: Learned query optimization for any sql database," *Proceedings of the VLDB Endowment*, vol. 16, no. 12, pp. 3515–3527, 2023.

[29] P. Negi, Z. Wu, A. Kipf, N. Tatbul, R. Marcus, S. Madden, T. Kraska, and M. Alizadeh, "Robust query driven cardinality estimation under changing workloads," *Proceedings of the VLDB Endowment*, vol. 16, no. 6, pp. 1520–1533, 2023.

[30] X. Wang, C. Qu, W. Wu, J. Wang, and Q. Zhou, "Are we ready for learned cardinality estimation?" *arXiv preprint arXiv:2012.06743*, 2020.

[31] B. Hilprecht and C. Binnig, "One model to rule them all: Towards zero-shot learning for databases," in *12th Conference on Innovative Data Systems Research, CIDR 2022, Chaminade, CA, USA, January 9-12, 2022*. www.cidrdb.org, 2022. [Online]. Available: https://www.cidrdb.org/cidr2022/papers/p16-hilprecht.pdf

[32] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" *Advances in neural information processing systems*, vol. 34, pp. 28 877–28 888, 2021.

[33] Y. Chronis, Y. Wang, Y. Gan, S. Abu-El-Haija, C. Lin, C. Binnig, and F. Özcan, "Cardbench: A benchmark for learned cardinality estimation in relational databases," *arXiv preprint arXiv:2408.16170*, 2024.

[34] A. van Renen, D. Horn, P. Pfeil, K. E. Vaidya, W. Dong, M. Narayanaswamy, Z. Liu, G. Saxena, A. Kipf, and T. Kraska, "Why tpc is not enough: An analysis of the amazon redshift fleet," in *VLDB 2024*, 2024. [Online]. Available: https://www.amazon.science/publications/why-tpc-is-not-enough-an-analysis-of-the-amazon-redshift-fleet

[35] A. Narayan, I. Chami, L. J. Orr, and C. Ré, "Can foundation models wrangle your data?" *Proc. VLDB Endow.*, vol. 16, no. 4, pp. 738–746, 2022.

[36] J. Wehrstein, C. Binnig, F. Ozcan, S. Vasudevan, Y. Gan, and Y. Wang, "Towards foundation database models," in *Conference on Innovative Data Systems Research*, 2025. [Online]. Available: https://vldb.org/cidrdb/2025/towards-foundation-database-models.html

Customizing Operator Implementations for SQL Processing via Large Language Models

Immanuel Trummer
Cornell University
itrummer@cornell.edu

Abstract

Recent advances in generative AI enable code synthesis at unprecedented levels of accuracy. In the context of data management, this opens up exciting opportunities to automatically customize core components of database management systems with the help of large language models such as OpenAI's GPT model series. This paper describes the vision of GenesisDB, a framework powered by large language models, that uses code synthesis to customize relational operators according to user specifications. The paper describes the first, simple prototype of GenesisDB that generates operator implementations in the Python programming language. The first experimental results demonstrate that the prototype is able to generate implementations for most relational operators, required for running standard benchmarks such as TPC-H. At the same time, the experiments reveal a multitude of research challenges that need to be solved to make this approach practical.

1 Introduction

Database management systems typically use a code base that evolves slowly over time, supported by large teams of software developers. End users may still influence system behavior via tuning knobs. However, the scope of such modifications is limited and insufficient for the examples outlined next.

Example 1: A user, trying to formulate a complex SQL query, would like to add custom SQL debugging support. For instance, this may take the form of customized output, including input/output samples, printed after each operation in the query plan.

Example 2: A developer wants to integrate SQL processing into a Python application that uses specialized in-memory data structures. To avoid conversion overheads and external dependencies, the developer would like to generate custom implementations of relational operators that are tailored to existing data structures.

Example 3: A database vendor would like to explore different types of progress updates while data is being processed. This can take the form, e.g., of a progress bar or of notifications detailing the amount of data processed at certain time intervals. The vendor would like to test various different variants in a user study before fully implementing the most popular version.

Example 4: The teacher of an introductory database class would like to create a specialized processing engine that generates educative content on used operators while processing. This output could take the form of operator descriptions, links to associated SQL tutorials or book chapters, or information on the complexity of each operator as it is processing.

Existing systems are typically unable to support all of the aforementioned specializations via parameter settings. Hence, supporting the example scenarios requires code changes of existing systems or implementing a new system from scratch. Such changes are beyond the capabilities of lay users without significant expertise in coding and databases. Even for experienced developers, some of the aforementioned changes may represent significant endeavors that consume large amounts of time.

This paper proposes the use of generative AI to customize core components of database management systems, thereby reducing or, in the best case, even eliminating customization overheads for developers and end users. As a proof of concept, it also reports on the first experimental results of a prototype system, GenesisDB, that uses generative AI to synthesize relational operator implementations.

The enabling technology of GenesisDB are large language models, based on the Transformer architecture. This architecture, together with pre-training approaches that leverage large amounts of unlabeled samples, has recently led to fundamental advances in the domain of natural and formal language processing. The newest generation of such models is often used without task-specific training, merely by specifying generation tasks as part of the input text (the so-called "prompt") while optionally providing a few solution samples. This is the approach taken by GenesisDB: it employs OpenAI's GPT models to synthesize custom code implementing relational operators.

GenesisDB comes with a set of prompt templates that describe the desired behavior of standard operators, as well as input and output formats. These prompt templates contain placeholders that can be substituted by user instructions (in natural language), thereby enabling customization. GenesisDB uses GPT to synthesize code for each operator. Then, it validates implementations by a sequence of more and more complex tests. Such tests include SQL queries, processed using newly generated operator implementations as well as a reference system (currently Postgres) to verify result consistency. Failed test cases initiate an automated debugging approach. Here, GenesisDB analyzes dependencies between operators and test cases as well as a probabilistic model to identify likely faulty operators. Next, GenesisDB tries to re-synthesize code for those operators, varying the prompt structure as well as generation settings to obtain a variety of alternative implementations to test. Specifically, GenesisDB may use code generated previously for other operators which passes a large number of test cases (and is therefore likely to be correct) as samples directly in prompts. This increases the chance to generate better code for other operators.

At run time, GenesisDB uses traditional query planning to translate queries into sequences of operator invocations. This process does not involve code synthesis by models and is therefore fast and robust. GenesisDB translates query plans into code that references custom operator implementations. While their implementation is dynamically synthesized, their input/output signature is known a-priori. More precisely, for each operator, the name of the function implementing it as well as the names of its parameters are known a-priori. On the other side, the type of each input parameter can be influenced by user commands. Despite of that, as GenesisDB uses a dynamically typed language for query execution, the code referencing custom operators does not require customization. Hence, GenesisDB synthesizes code only before run time. At run time, it uses a static component generating query-specific code that invokes previously synthesized operator implementations. Note that generated operator implementations can also be used outside of GenesisDB, e.g. in the context of an existing application storing data in specialized data structures.

GenesisDB is currently an early prototype and is restricted to generating operator code in the Python programming language. Despite its early stages, the experiments show that the current version already synthesizes correct operator implementations for a majority of relational operators. While the data processing performance is currently not competitive, the proof-of-concept results show that the resulting implementations can process even queries of elevated complexity of the TPC-H benchmark. In summary, the original scientific contributions in this paper are the following:

- The paper introduces the vision of using generative AI to synthesize customizable code for core components of database management systems.
- The paper describes an early prototype of GenesisDB, a code synthesis framework using OpenAI's GPT models, that follows this approach.
- The paper presents first experimental results, revealing the general feasibility of the approach while also pointing out limitations.
- The paper discusses next steps and future research challenges, based on observations with the current prototype.

The remainder of this paper is organized as follows. Section 2 provides background information and discusses prior, related work. Section 3 describes an early prototype of GenesisDB that is used for the experiments in the following section. Section 4 reports on proof-of-concept experiments, performed with the current prototype. Finally, Section 5 discusses next steps and research challenges that need to be solved to make the approach practical.

2 Background and Prior Work

The last few years have seen transformative advances in the domain of language processing, including natural as well as formal languages (while less relevant for this publication, recent Transformer models [13] can also be used for data of other modalities than text, such as images). These advances are due to novel neural network architectures, in particular the Transformer model [22, 24], as well as to the successful application of transfer learning methods in training. Transformer models, among other advantages, facilitate the creation of large models with hundreds of billions of trainable parameters. When trained on generic tasks (e.g., predicting the next token) on sufficiently large amounts of unlabeled training data, such models require little to no specialization to solve new tasks [2]. Transformer models can be used for code synthesis [4, 12], a feature exploited by GenesisDB. Trained on large amounts of code from repositories such as GitHub, such models complete prompts, i.e. short text documents containing partial code or natural language instructions, into fully specified code in general-purpose programming languages. In certain settings, their performance is nowadays comparable to the one of human developers [8].

These developments motivate the vision of database components, synthesized via language models. GenesisDB is a prototype implementing this approach. GenesisDB is based on OpenAI's GPT models [12]. Similar models power GitHub's CoPilot [5], an auto-completion tool offered on the GitHub platform. Unlike CoPilot, GenesisDB is however specialized to generating code for relational operators that, when combined, form a complete SQL processing engine. GenesisDB features prompt templates, specialized for generating relational operators, as well as strategies needed to create a complex system from parts generated in different code synthesis steps. The size of a typical engine, generated by GenesisDB, exceeds the code size typically generated in a single model invocation. GenesisDB relates most to CodexDB [20], a system synthesizing code for processing SQL queries that additionally implements natural language instructions. While CodexDB synthesizes code for processing *one* specific SQL query, GenesisDB aims at generating a system that can process *all* queries without run time code synthesis.

GenesisDB also relates to other recent applications of language models and, more broadly, machine learning in the context of database management systems. These include, for instance, prior work on natural language query interfaces [6, 10, 19, 23, 25], as well as approaches for entity matching [9], data wrangling [11, 16], database auto-tuning [18, 21], and data integration [3]. Several recent publications use language models or, generally, machine learning to implement relational operators directly [3, 7, 15, 17],

Table 1: Operators synthesized by GenesisDB.

Group	Operators
Tests	CheckColumnType, CheckTableType
I/O	LoadTable, StoreTable
Basic	GetColumn, GetValue, CreateTable, GetNull, IsNull, IsEmpty, SetColumn, AddColumn, FillIntColumn, FillFloatColumn, FillBoolColumn, FillStringColumn, NrRows, Map, ToInt, ToFloat, ToBool, ToString, Substring, Limit
Arithmetic	Addition, Subtraction, Multiplication, Division, Floor
Boolean	LessThan, GreaterThan, LessOrEqual, GreaterOrEqual, Equal, NotEqual, Case, And, Or, Not, Filter
Aggregates	Sum, Min, Max, Avg, Count, SumGrouped, MinGrouped, MaxGrouped, AvgGrouped
Complex	Sort, InnerJoin, LeftOuterJoin, RightOuterJoin, CartesianProduct

leading to approximate processing. Instead, GenesisDB uses machine learning before run time to synthesize code for processing.

3 Prototype Overview

Figure 1 shows an overview of the GenesisDB prototype. Section 3.1 discusses query processing. Section 3.2 describes how GenesisDB synthesizes its core engine using generative AI.

3.1 Query Processing

The **Processing** sub-system (left side in Figure 1) processes SQL queries and returns results to users. GenesisDB is an analytical SQL processing engine and supports all queries of the TPC-H benchmark (but no transactions). As discussed in more detail later, it uses relational operators for processing that are synthesized according to user instructions. A GenesisDB database is represented as a directory, containing a file with SQL commands creating the corresponding schema, as well as one .csv file for each table in the database (containing the data). The current prototype is restricted to processing data stored in .csv files. Expanding the scope to other data formats, likely enabling more efficient processing, is part of the future work plans.

The database catalog contains information on the database schema and file locations, extracted from the database directory. It is used to inform the query parser as well as the query planner. Currently, GenesisDB uses a simple, rule-base query planner, implemented by the Apache Calcite library [1]. The planner generates a logical query plan, determining the sequence of operations without selecting between operator implementations. Using cost-based optimization is challenging as the implementation (and, therefore, cost function) of operators changes dynamically. Learned cost models requiring a few samples

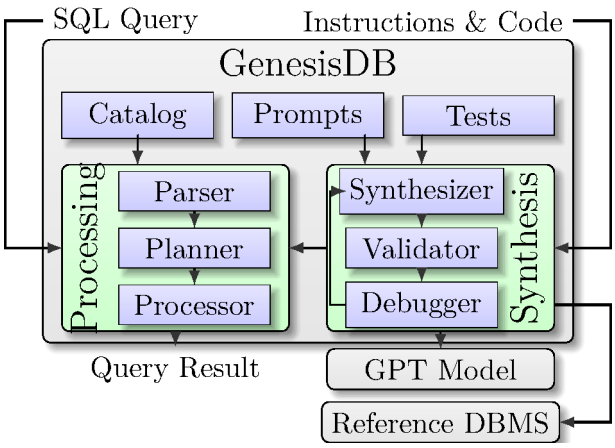

Figure 1: Overview of GenesisDB system.

Table 2: GenesisDB accepts natural language instructions or code snippets for the following SQL engine properties.

Property	**Semantics**
Prefix	Common prompt prefix (e.g., imports)
Table	How to represent relations
Column	How to represent columns
Null	How to represent the SQL NULL value
Boolean	How to represent Boolean values
Integer	How to represent integer values
Float	How to represent float values
String	How to represent string values
Suffix	Common prompt suffix (e.g., behavior)

may help in the future.

The SQL processor translates logical query plans into a sequence of steps, using a set of 54 operators, shown in Table 1. These operators are more fine-grained than the ones typically used in database engines (e.g., introducing separate operators for aggregation with and without grouping as well as for different join types). The goal of using relatively specialized operators is to make operator synthesis easier, reducing the number of cases that need to be handled per function. GenesisDB generates code for processing queries, combining synthesized code implementing operators with code invoking the latter to execute the query plan. In the current prototype, all code is formulated in Python (Version 3.8). Generating code in lower-level languages such as C is planned for future versions.

3.2 Operator Synthesis

The distinguishing feature of GenesisDB, compared to prior systems, is that it synthesizes operator implementations using generative AI. This opens up possibilities to customize generated components according to user instructions. In principle, such instructions may change the in-memory representation of tables, columns, or fields, as well as inject custom behavior during processing. For instance, this may include generating output that is useful for debugging or training, executing custom performance or data analysis, or writing checkpoints according to user instructions. GenesisDB generates code via

```
def not_equal(column_1, column_2):
    """ True where column_1 <> column_2.

    1. Return [Null] for rows where one input row is [Null].
    2. Return true iff first row <> second row otherwise.
    3. Ensure that the output is [Column].

    Args:
        column_1: [Column].
        column_2: [Column].

    Returns:
        [Column] containing Boolean values.
    """
```

Figure 2: Prompt template for generating NotEqual operator. The template contains placeholders (marked in color) that can be customized by users.

prompting [14], i.e. by submitting small text documents describing the generation task to generative models. Users influence the generation process by providing substitutes for placeholders in prompt templates or custom text that is added as a prefix or suffix to default prompts. Table 2 shows an overview of all the prompt components that can be influenced by users. An example of a prompt template containing placeholders from Table 2 follows.

Example 5: Figure 2 shows an example prompt template, used to generate the "NotEqual" operator. Prompt parts marked up in color represent snippets that can be customized by users.

Algorithm 1 shows high-level pseudo-code for the synthesis process. Given user instructions, a list of operators to synthesize, a set of test cases to validate synthesized operators, and, optionally, default implementations for each operator, it returns a custom engine (i.e., operator implementations) that follows the input instructions. The full list of operators, as well as their semantics and function signatures, remain fixed (to enable the query processor to use them appropriately to realize query plans). If desired, synthesis may only focus on a subset of operators (while using default implementations for others). To validate generated implementations, the current prototype uses a set of 172 test cases by default (users can easily add new test cases that are automatically used during synthesis). Test cases are either realized as SQL queries (then, GenesisDB compares query results generated by custom operators to the ones generated by a reference system) or as small code samples referencing operators (here, GenesisDB ensures that all assertions hold).

GenesisDB first sorts operators using a simple heuristic (based on the length of the associated prompt), prioritizing operators that are potentially easier to synthesize. As discussed more in the following sections, operator order matters as it influences, for instance, the order in which tests are performed. Also, code synthesized for operators that are ordered earlier may be used as a sample when synthesizing operators that appear later. After sorting, GenesisDB synthesizes the first version of the engine by generating one implementation for each required operator ($E.code[o]$ denotes the implementation of operator o).

Next, GenesisDB validates generated code via test cases and tries to fix problems via re-synthesis. This process continues until the generated engines passes all test cases or until a user-defined timeout is reached. Testing via Function RunTests stops at the first failed test case. The result of testing is a summary, reporting passed and failed tests. GenesisDB uses that summary to identify likely faulty

Algorithm 1 Generating SQL execution engines based on natural language instructions.

```
1: // Returns SQL processing engine using code synthesized
2: // according to natural language user instructions U or default
3: // implementations D to implement operator list O, validated
4: // via test cases T.
5: function GENESIS(U, O, T, D)
6:     // Choose order in which operators will be validated
7:     O ←SORTOPERATORS(O)
8:     // Synthesize initial code for each operator
9:     E ←ENGINE.INIT
10:    for o ∈ O do
11:        c ←SYNTHESIZE(U, E, O, o)
12:        E.code[o] ← c
13:    end for
14:    // Run tests and re-synthesize faulty operators
15:    while Not all tests pass and no timeout do
16:        // Validate synthesized engine
17:        r ←RUNTESTS(O, E, T)
18:        // Find operators likely to have bugs
19:        F ←FAULTYOPERATORS(r)
20:        // Try replacing faulty operators
21:        E ←FIXOPERATORS(U, O, D, E, F, r.tests)
22:    end while
23:    return E
24: end function
```

operators, and then tries to fix them. Fixing an operator involves re-synthesizing its code, possibly with a different prompt and different synthesis settings. If this approach fails to resolve previous problems, using a limited number of tries, GenesisDB uses default operator implementations instead. Those default operators should use the same data representation as the desired target engine (in order to be compatible). They may however not implement any custom behavior, requested by the user. Therefore, GenesisDB tries to minimize the number of default operators used. If default operators are not specified, GenesisDB ultimately notifies the user, hinting at operators that likely need replacement. After providing the corresponding code, the synthesis process can be restarted.

4 Proof-of-Concept Experiments

The goal of the experiments is to test whether the prototype is able to synthesize operator implementations that can process complex SQL queries.

4.1 Experimental Setup

All experiments are executed on a t2.2xlarge EC2 instance, featuring eight virtual CPUs, 32 GB of main memory, and 500 GB of EBS storage. The instance is running Ubuntu and GenesisDB uses Python 3.8 to execute queries. Postgres 10.22 is used to generate reference results for all test cases. All test queries are executed on a TPC-H database with a low scaling factor of 0.01 to speed up test evaluation. The following experiments use GPT-3 Codex for code synthesis. This model is relatively small, compared

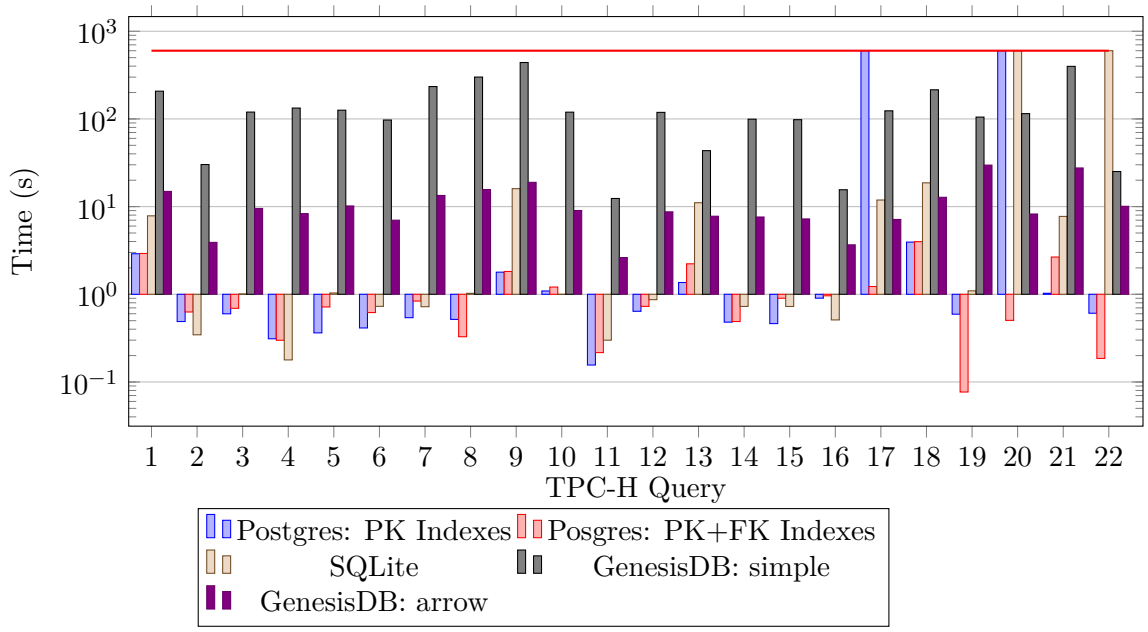

Figure 3: Performance on TPC-H queries with scaling factor 1. The red line marks the timeout of 10 minutes.

to recently released models, but has been specialized for code generation. GenesisDB retries operator synthesis five times in case of errors. It uses 172 test cases for validation, including all TPC-H queries, executed on a TPC-H database with a scaling factor of 0.01.

Tables 3 and 4 show values used to substitute placeholders in prompt templates in two separate experiments. The first experiment synthesizes a system that exploits standard Python classes for representing data in main memory. The second experiment generates an engine that relies on PyArrow to represent and process data. To increase the chances of successful generation, each prompt template was prefixed by the implementation of a simple operator (incrementing the values in a column by one) that follows the requested style (i.e., in the first experiment, the increment operator uses Python lists as input, in the second experiment it operates on Pyarrow data structures). Whenever GenesisDB is unable to synthesize an operator after at most five tries, a corresponding operator implementation is added manually. The ratio of successfully generated operators is one of the evaluation metrics discussed in the next subsection.

4.2 Experimental Results

Using Python data structure, GenesisDB is able to synthesize correct operator implementations in 89% of cases. On the other hand, GenesisDB was only able to generate correct implementations for 48% of operators when customizing synthesis for PyArrow data structures. Generating operators and running tests took about four hours for the first experiment and about three hours for the second experiment.

Figure 3 shows performance results for TPC-H queries, comparing GenesisDB (in arrow configuration and with simple Python lists as in-memory data representation) to Postgres 10.22 with primary key or primary and foreign key indexes and SQLite 3.36. Clearly, traditional database management systems achieve optimal performance. On the other hand, there are a few queries where the PyArrow implementation performs better than some of the traditional systems. Overall, while performance improvements are a primary goal of future work, the results show that the prototype can synthesize operator imple-

Table 3: Settings used to synthesize a simple SQL engine.

Property	**Value**
Prefix	import functools import operator import streamlit as st import time
Table	a list of rows where each row is a list
Column	a list
Null	None
Boolean	bool
Integer	int
Float	float
String	str

Table 4: Settings to synthesize SQL engine using Pyarrow.

Property	**Value**
Prefix	import pyarrow as pa import pyarrow.compute as pc import pyarrow.csv as csv
Table	a pyarrow Table
Column	a pyarrow Array
Null	None
Boolean	pa.bool_()
Integer	pa.int64()
Float	pa.float64()
String	pa.string()

mentations enabling processing of complex SQL queries. More importantly, the results provide evidence that customization has a significant impact on the properties of the generated implementations, notably run time.

5 Future Work

The proof-of-concept experiments demonstrate that language models can synthesize code for a majority of relational operators. At the same time, they provide the first evidence for the possibility of customizing the generated operator implementations, leading to significantly different behavior in the generated systems. While these first results are promising, they also reveal important limitations that need to be addressed in future work.

First, the performance results demonstrate the limitations of using Python for implementing relational operators. The choice of the Python programming language for those first experiments is motivated by several factors. First of all, Python is advertised as one of the languages in which GPT models are most proficient in (due to a large amount of corresponding training data in the corpora used for pre-training). At the same time, high-level programming languages such as Python enable relatively concise pieces of code that implement standard operators. Generating shorter pieces of code is oftentimes

easier for language models, compared to generating the more verbose operator implementations that are typical for traditional database management systems. However, as revealed in the experiments, the performance penalty of using Python is enormous, motivating future research that aims at generating code for relational operators in lower-level languages such as C.

Generating code in lower-level programming languages leads to new research challenges. First of all, whereas the current implementation aims to generate entire operator implementations in case of suspected errors, this may become inefficient in the case of larger operator implementations. Instead, it seems prudent to restrict re-synthesis efforts to parts of operator implementations that are likely to be incorrect. Here, language models can help to identify code parts, based on an analysis of error messages or inconsistent results, that likely require re-generation. Second, whereas the current implementation merely provides a description of desired operator behavior to the language model, future implementations could provide more information, facilitating the synthesis task. For instance, it might be possible to provide language models with a simple operator implementation as part of the prompt input, thereby facilitating the task of generating more sophisticated versions. Also, instead of generating operators from scratch, it may be easier to "morph" an operator implementation in multiple steps, running automated tests after each transformation step.

A complementary avenue to improve the performance of the generated implementations is to bias synthesis, based on automated performance tests. For instance, given user-defined performance goals, the system could regenerate operators whenever performance goals are not met. Of course, doing so introduces additional constraints that may make it harder to generate operator implementations that pass all correctness and performance tests.

Another source of future research challenges lies in increasing the success ratio of code synthesis. As shown in the experiments, the current prototype is not able to generate correct code in all cases. As a first step, a study evaluating the cost-quality tradeoff achieved by different OpenAI models (e.g., the recently released OpenAI o1 model) or models of other providers is interesting. Another possibility for future improvements is to replace generic language models by versions that are specialized for the task of generating relational operators. For instance, many of OpenAI's GPT model variants enable users to apply fine-tuning, meaning to re-train models on a corpora that is more representative of the specific tasks they are targeted at. In the specific scenario investigated in this paper, relevant training data could incorporate code from (open-source) database management systems. As the set of relational operators is fairly standardized across different database management systems, fine-tuning on existing code likely provides valuable information to the language model for operator synthesis. At the same time, this could enable GenesisDB to integrate advanced techniques that have been proven to improve performance for relational data processing, thereby benefitting performance as well.

Finally, future research could focus on improving the interaction between the user and the system. Language models cannot currently provide guarantees on generating accurate code. Therefore, coding assistants are typically considered tools for human-in-the-loop software development, rather than purely automated tools. As shown in the experiments, GenesisDB requires help to synthesize complete sets of operators as well. While some of the aforementioned approaches may push the boundaries in terms of the success ratio in code synthesis, it still seems likely that GenesisDB requires guidance from users to deal with complex scenarios. However, the time of users is precious which motivates research on how to get the most valuable insights from users with limited time investments from their side. For instance, users with an IT developer background could provide sample code that is requested by the language model. Here, the goal would be to carefully select which samples to request, maximizing the benefit for code synthesis. Alternatively, users without an IT background could still be helpful in verifying whether or not generated engine implementations satisfy the features requested via customization. Again, minimizing the number of questions addressed to users will make the system more practical.

6 Conclusion

This paper introduces the vision of highly customizable database management systems. Given recent developments in the domain of generative AI, it becomes possible to generate core components of database execution engines via generative AI. This opens up new possibilities for customization, according to user specifications.

This paper described the first prototype of GenesisDB, a code synthesis framework powered by language models that implements this approach. First experimental results show that GenesisDB is able to generate correct code for a majority of operator implementations. On the other hand, the current prototype is only able to generate correct code for a subset of operators and the performance of the generated code is not yet satisfactory. This opens up various opportunities for future research.

Acknowledgement

This project is supported by NSF CAREER grant IIS-2239326 ("Mining Hints from Text Documents to Guide Automated Database Performance Tuning").

References

[1] E. Begoli, J. Camacho-Rodríguez, J. Hyde, M. J. Mior, and D. Lemire. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *SIGMOD*, pages 221–230, 2018.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, pages 1877–1901, 2020.

[3] Z. Chen, J. Fan, S. Madden, and N. Tang. Symphony: Towards Natural Language Query Answering over Multi-modal Data Lakes. In *CIDR*, pages 1–7, 2023.

[4] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. PaLM: Scaling Language Modeling with Pathways. *CoRR*, abs/2204.0:1–87, 2022.

[5] G. D. Howard. GitHub Copilot: Copyright, Fair Use, Creativity, Transformativity, and Algorithms *. pages 1–13, 2021.

[6] G. Karagiannis, M. Saeed, P. Papotti, and I. Trummer. Scrutinizer: A Mixed-Initiative Approach to Large-Scale, Data-Driven Claim Verification. *PVLDB*, 13(12):2508–2521, 2020.

[7] T. Kraska, M. Alizadeh, A. Beutel, E. H. Chi, J. Ding, A. Kristo, G. Leclerc, S. Madden, H. Mao, and V. Nathan. SageDB: A learned database system. In *CIDR*, pages 1–10, 2019.

[8] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, and R. Leblond. Competition-Level Code Generation with AlphaCode. *DeepMind Technical Report*, pages 1–73, 2022.

[9] Y. Li, J. Li, Y. Suhara, A. Doan, and W. C. Tan. Deep entity matching with pre-trained language models. *Proceedings of the VLDB Endowment*, 14(1):50–60, 2020.

[10] Y. Luo, N. Tang, G. Li, C. Chai, W. Li, and X. Qin. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In *SIGMOD*, pages 1235–1247, 2021.

[11] A. Narayan, I. Chami, L. Orr, and C. Ré. Can Foundation Models Wrangle Your Data? *PVLDB*, 16(4):738–746, 2022.

[12] OpenAI. https://openai.com/blog/openai-codex/, 2021.

[13] OpenAI. GPT-4 Omni Release, 2024.

[14] T. L. Scao and A. M. Rush. How Many Data Points is a Prompt Worth? In *NAACL*, pages 2627–2636, 2021.

[15] S. Suri, I. Ilyas, C. Re, and T. Rekatsinas. Ember: No-Code Context Enrichment via similarity-based keyless joins. *PVLDB*, 15(3):699–712, 2021.

[16] N. Tang, J. Fan, F. Li, J. Tu, X. Du, G. Li, S. Madden, and M. Ouzzani. Rpt: Relational pre-trained transformer is almost all you need towards democratizing data preparation. *PVLDB*, 14(8):1254–1261, 2021.

[17] J. Thorne, M. Yazdani, M. Saeidi, F. Silvestri, S. Riedel, and A. Halevy. From natural language processing to neural databases. *Proceedings of the VLDB Endowment*, 14(6):1033–1039, 2021.

[18] I. Trummer. The Case for NLP-Enhanced Database Tuning: Towards Tuning Tools that "Read the Manual". *PVLDB*, 14(7):1159–1165, 2021.

[19] I. Trummer. BABOONS: Black-box optimization of data summaries in natural language. *PVLDB*, 15(11):2980 – 2993, 2022.

[20] I. Trummer. CodexDB: Synthesizing Code for Query Processing from Natural Language Instructions using GPT-3 Codex. *PVLDB*, 15(11):2921 – 2928, 2022.

[21] I. Trummer. DB-BERT: a Database Tuning Tool that "Reads the Manual". In *SIGMOD*, pages 190–203, 2022.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All You Need. In *Advances in Neural Information Processing Systems*, pages 5999–6009, 2017.

[23] N. Weir, A. Crotty, A. Galakatos, A. Ilkhechi, S. Ramaswamy, R. Bhushan, U. Cetintemel, P. Utama, N. Geisler, B. Hättasch, S. Eger, and C. Binnig. DBPal: Weak Supervision for Learning a Natural Language Interface to Databases. pages 1–4, 2019.

[24] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-Art Natural Language Processing. In *EMNLP*, pages 38–45, 2020.

[25] V. Zhong, C. Xiong, and R. Socher. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR*, abs/1709.0(1):1–12, 2017.

iDataLake: An LLM-Powered Analytics System on Data Lakes

Jiayi Wang, Guoliang Li, Jianhua Feng
Department of Computer Science, Tsinghua University, Beijing, China
{liguoliang@tsinghua.edu.cn}

Abstract

Existing data lakes struggle to effectively analyze heterogeneous data – like unstructured, semi-structured, and structured data – because of difficulties in heterogeneous data linking, semantic understanding, and execution pipeline orchestration. In this paper we highlight challenges for supporting data analytics on data lakes, and present a system `iDataLake` to address these challenges, which takes natural language query as input, orchestrates a pipeline for the query and outputs the results for the query. We first define semantic operators to support heterogeneous data analytics on data lakes. Then we introduce our data embedding method for the alignment of multi-modal data and introduce how to efficiently discover those data relevant to the query from the data lake. Next, we introduce the pipeline orchestration method, which converts input natural language queries into executable pipelines built from predefined semantic operators. By executing the pipeline over the discovered data, the data analytics queries can be efficiently answered with high accuracy and low cost.

1 Introduction

Data lakes are increasingly becoming the storage paradigm for managing large volumes of heterogeneous data, including structured, semi-structured, and unstructured data. Structured data, such as relational tables, is well-suited to traditional analytics methods like SQL. However, semi-structured data (e.g., JSON, XML) and unstructured data (e.g., text documents) present significant challenges because of their inherent complexity (e.g., nested structure or or complete lack of structure) and the absence of predefined schemas. Despite these challenges, performing analytics on such diverse datasets offers tremendous value, enabling organizations to derive deeper insights and make data-driven decisions across various domains and data formats.

Example 6: Consider a query: "Identify the top-5 directors whose movies in the 1980s received the highest ratio of positive reviews." This query is highly relevant for film study. However, answering this query poses significant challenges due to the fragmented nature of the required data. For instance, structured information, such as production years, is typically stored in structured tables, while review content resides in unstructured text documents. Additionally, resolving the query necessitates multi-step reasoning, including data integration, sentiment analysis, and ranking, which further complicates the process.

While SQL is effective for querying structured data, it cannot directly handle semi-structured and unstructured data, which lack the well-defined schemas required for SQL. Furthermore, SQLs cannot express semantic predicates, which limits the ability to process data in a way that reflects its inherent meaning. Although some approaches often attempt to extract structured information from unstructured data, such as transforming textual content into tables with the help of machine learning models [1, 2],

this extraction process inevitably leads to data loss and may compromise the accuracy and depth of the analysis. Therefore, traditional methods fall short in handling the diverse nature of data lakes, even when unstructured data is converted into a structured format.

Recent advances in large language models (LLMs) have shown a great opportunity in addressing these challenges. LLMs possess advanced semantic understanding capabilities, enabling them to handle unstructured data (e.g., text) more effectively than traditional methods [3, 5–8]. Their ability to process and generate meaningful representations of natural language data makes them a powerful tool for performing analytics over semi-structured and unstructured datasets [9, 10]. However, integrating LLMs into data analytics workflows for data lakes remains an open research problem, as it faces the following challenges.

C1. Heterogeneous Data Modeling and Linking. Data analytical queries often involve diverse subsets of correlated data. Accurately identifying and linking the relevant data for an input query is crucial for both the accuracy and efficiency of the analytics process. Inaccurate data linking can lead to incorrect results and significant delays. Developing effective methods to semantically model and link heterogeneous data types effectively is essential for accurate and efficient data analytics.

C2. Semantic Data Processing. Semantic understanding is essential for processing semi-structured and unstructured data, as it allows the system to interpret the inherent meaning of the data rather than relying on rigid schemas and exact textual matching. Effectively leveraging the advanced semantic capabilities of LLMs to process and analyze these data types (e.g., semantic filtering, semantic grouping) is a critical challenge.

C3. Automatic Pipeline Orchestration. It is crucial to generate an execution plan for an analytical query on data lakes. However, orchestrating an efficient and accurate query execution pipeline in a multi-modal data lake is complex. Unlike traditional databases with deterministic query plans based on relational algebra and predefined data schemas, data lakes require flexible and adaptive approaches to handle diverse data types. Automating the generation and orchestration of such pipelines to ensure efficiency and accuracy remains a key challenge.

C4. Efficient Pipeline Execution. Executing pipelines in a multi-modal data lake involves balancing accuracy and efficiency. Enumerating all potential execution pipelines may yield high accuracy but incur significant computational costs. Conversely, executing a single pipeline can improve efficiency but may reduce accuracy, since semantic operator execution over heterogeneous data may fail or meet unforeseen errors. Designing an adaptive plan selection process that balances both accuracy and efficiency dynamically during execution is a challenging but essential task.

In response to these challenges, we propose `iDataLake`, a novel LLM-powered analytics system designed to handle data analytics queries over multi-modal data lakes. `iDataLake` addresses the aforementioned challenges with the following contributions:

(1) **Unified Embedding-Based Data Linking (for C1)**: We introduce a unified embedding approach to efficiently link heterogeneous data types within a multi-modal data lake. We embed different types of data into a shared semantic embedding space and align embeddings of relevant data through a contrastive learning method. This alignment ensures that semantically relevant data is accurately identified and retrieved even in heterogeneous formats, thus improving both the accuracy and efficiency of query responses.

(2) **Semantic Operators for Data Analytics (for C2)**: We present a set of semantic operators tailored for data analytics over multi-modal data lakes. These operators are designed to perform statistical and semantic analysis across structured, semi-structured, and unstructured data, enabling `iDataLake` to handle a wide range of complex queries that traditional systems struggle with.

(3) **Pipeline Orchestration Algorithm (for C3)**: We propose an iterative two-stage algorithm for automatic pipeline orchestration. It iteratively selects an appropriate operator to reduce the query and gradually form the execution pipeline. In each step, it filters out irrelevant operators using a

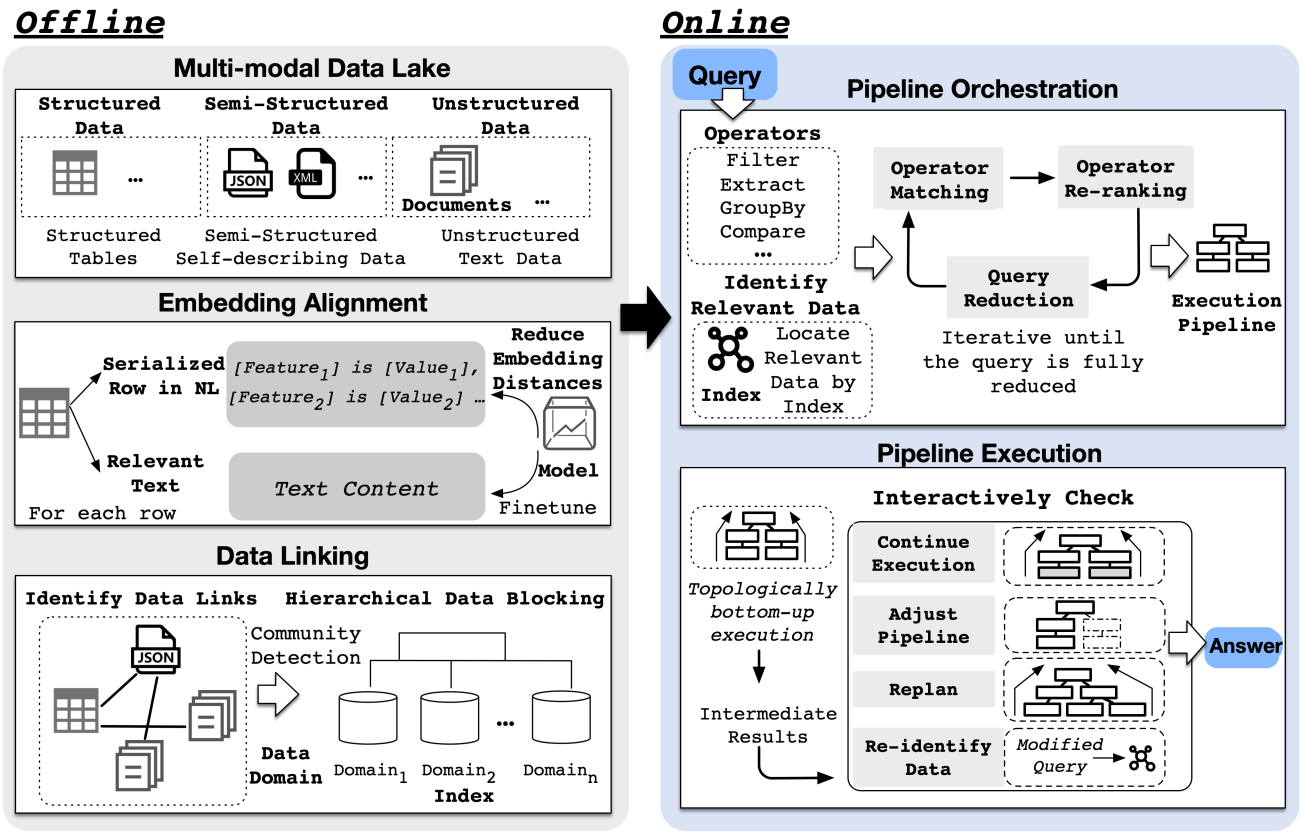

Figure 1: Framework of iDataLake.

low-cost approach and focuses on selecting from the remaining operators and organizing the relevant operators based on the query and available data. This ensures efficient and accurate query processing in a heterogeneous data environment.

(4) **Dynamic Pipeline Adjustment (for C4)**: Our system incorporates dynamic pipeline adjustment during query execution, allowing it to adapt to intermediate results. This flexibility enables iDataLake to optimize performance by skipping unnecessary computations and adjusting to unforeseen intermediate results, ensuring both efficiency and robustness.

In general, iDataLake represents a significant step forward in enabling high-accuracy, practical data analytics on data lakes. Unlike previous approaches that rely on lossy data extraction or are limited by SQL's rigid schema, iDataLake employs the semantic understanding capability of LLMs effectively to provide a more holistic and efficient solution.

2 LLM-Powered Analytics System on Data Lakes

We first introduce the system architecture of iDataLake (Figure 1), and then present the components in iDataLake.

2.1 iDataLake Architecture and Workflow

As illustrated in Figure 1, iDataLake is designed to process data lakes containing structured (e.g., relational tables), semi-structured (e.g., JSON, XML), and unstructured data (e.g., text documents).

Table 1: Summary of Semantic Operators

Operator	Category	Description
OrderBy	Transformation	Sort data according to specified criteria.
Compare	Transformation	Compare two input values and return the one that meets the criteria.
Refine	Transformation	Adjust text for clarity or precision, aiding subsequent processing.
Translate	Transformation	Convert text between languages using LLM or external tools.
Transform	Transformation	Convert data from one format to another, e.g., table to text.
Validate	Transformation	Verify the accuracy of generated information through external sources.
Scan	Retrieve	Load data files and enumerate their elements.
Filter	Retrieve	Remove irrelevant data based on specified criteria.
Augment	Retrieve	Fetch relevant data from external sources to enhance LLM responses.
Extract	Extraction	Extract relevant information from documents/tuples, similar to projection.
Conceptualize	Extraction	Identify key concepts to simplify complex queries.
Generate	Generation	Produce coherent text based on input, often for final responses.
Explain	Generation	Provide explanations or justifications for decisions.
GroupBy	Partition	Organize data into groups for computing summary statistics.
Cluster	Partition	Cluster similar data together.
Classify	Partition	Categorize or label entities using LLM or external ML models.
Link	Link	Identify and link related data, such as tables and documents in the data lake.
SetOP	Aggregation	Perform set operations, e.g., Union, Intersection, Complement.
Integrate	Aggregation	Combine information from multiple sources into a cohesive response.
Aggregate	Aggregation	Compute aggregation results, such as sum or average, from data.
Summarize	Aggregation	Condense text into shorter summaries for improved context consumption.

During offline pre-processing, `iDataLake` constructs an index tailored to the characteristics of the data, capturing inherent correlations across various data modalities. To enable efficient data linking and retrieval, `iDataLake` employs a unified embedding approach. Data of different types is transformed into a shared semantic embedding space, aligning heterogeneous data for seamless integration. Then using community detection algorithms, `iDataLake` discovers hierarchical clustering relationships within the data, partitioning the data into domain-specific clusters. This design ensures that relevant data can be located efficiently for any given query. Moreover, a vector index is built on top of embeddings to support efficient embedding retrieval. To address the out of distributions (OOD) issues among different data types, we also build OOD vector index to facilitate cross-modal retrieval.

For online analytics, `iDataLake` introduces a suite of semantic operators specifically designed for multi-modal data processing. When a query is received, `iDataLake` first utilizes the data linking index to identify relevant data within the data lake. The pipeline orchestration component then constructs an execution plan by iteratively selecting and applying appropriate semantic operators. The execution plan is executed by the query execution engine, which dynamically adjusts the pipeline based on intermediate results to maintain robustness and efficiency. Once the entire pipeline is executed, `iDataLake` generates the final result and returns it to the user. By following logically reasonable pipelines constructed over query-relevant data, `iDataLake` ensures high accuracy in its results.

2.2 Semantic Operators

Traditional relational operators are insufficient for unstructured data analytics, as they lack semantic processing capabilities and require data to adhere to strict schemas. To address this limitation, we introduce a set of semantic operators that serve as the fundamental building blocks for performing analytics over multi-modal data lakes. These operators are designed to handle diverse data types, bridge the gap between heterogeneous data sources, and enable joint query execution based on semantic

understanding rather than relying on predefined schemas.

2.2.1 Logical Operators

The logical operators in `iDataLake` are categorized into seven types, each designed to address a specific aspect of the analytics process. Table 1 summarizes these operators.

Data Transformation Operators. These operators transform data to generate intermediate representations or perform computations. Both input and output are typically lists of data. Examples include:

OrderBy: Sort input data based on specified criteria and return the ordered results.

Transform: Convert data from one format to another, such as transforming a table into a text paragraph describing its content.

Data Retrieval Operators. These operators retrieve relevant data from external sources. The input is typically a query or criteria, and the output is typically a list of data. Examples include:

Filter: Remove irrelevant data from specified data based on user-specific criteria and output the result.

Augment: Fetch additional data relevant to the query from external sources to enhance LLM responses.

Data Extraction Operators. These operators extract relevant information from certain data sources. The input is generally a list of data and the output is also a list. Examples include:

Extract: Extract specific information from documents or tuples, similar to the projection operator in relational algebra.

Conceptualize: Identify key concepts from input query, e.g., converting a description to its corresponding term, to simplify complex queries and improve understanding.

Data Generation Operators. These operators generate target content based on given requirements and input information. The input is generally text paragraphs, and the output is the target text result. Examples include:

Generate: Produce coherent text based on input, often used for final responses.

Explain: Provide explanations or justifications for decisions. The explanations of different reasoning paths are generally integrated to generate the final answer.

Data Partition Operators. These operators partition data into meaningful subsets for further analysis. The input is generally a list of data, and the output is a list of data lists. Examples include:

GroupBy: Organize data into groups for computing summary statistics or comparing information across groups.

Cluster: Cluster data together based on similarity metrics using clustering algorithms or LLM identification.

Data Linking Operators. These operators identify and link related data across different sources. The input is typically a set of data elements, and the output is a linked structure. An example is:

Link: Identify and link related data, such as tables and documents in a data lake, supporting different granularities (e.g., table level, tuple level, paragraph level, and document level).

Data Aggregation and Integration Operators. These operators integrate information from multiple sources or compute aggregate insights. The input is typically a list of data, and the output is a single value or cohesive response. Examples include:

Aggregate: Compute statistical aggregation results, such as sum or average, from data.

Summarize: Condense text into shorter summaries to reduce context consumption for LLMs.

2.2.2 Physical Operators

Each logical operator has multiple physical implementations tailored to different data types, corresponding to different physical operators. Based on the involvement of LLMs in the analytics process, physical

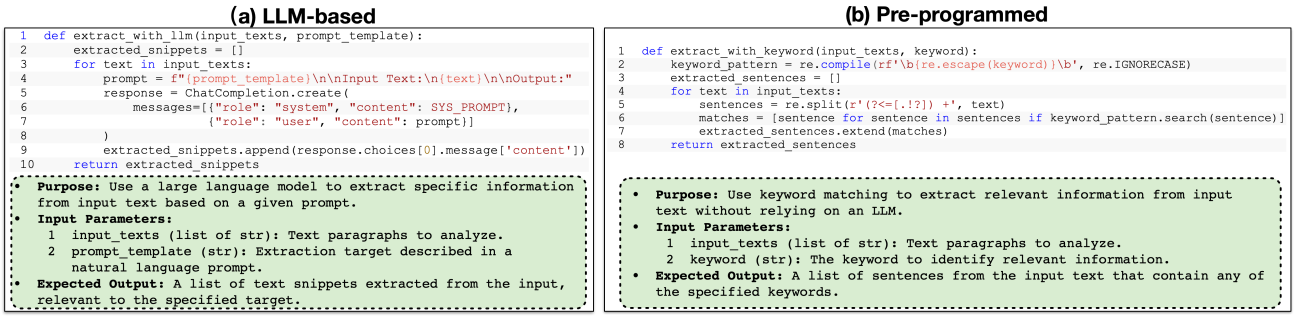

Figure 2: Example physical operators for the Extract operator.

operators are classified into two categories:

LLM-Based Physical Operators: These implementations leverage LLMs for tasks requiring semantic understanding by formulating appropriate prompts. For example, as shown in Figure 2(a), the *Extract* operator can utilize an LLM to identify and extract text snippets by providing prompts that specify the extraction target, such as *information related to sports*.

Pre-Programmed Physical Operators: These implementations rely on pre-defined logic to process data, similar to the physical operators in relational databases. For example, as shown in the *Extract* operator can extract information using keyword matching or regular expressions to extract information that follows certain patterns.

In order to select the appropriate physical operator for each logical operator in the plan, `iDataLake` maintains detailed text descriptions for each physical operator, including its purpose, input parameters, and expected outputs. If such descriptions are unavailable, `iDataLake` invokes the LLM to generate them with few-shot examples of existing operators automatically. With this information, the overall pipeline can be constructed accurately by LLMs based on actual needs.

To support user-defined functions (UDFs), we invoke LLMs to generate codes for UDFs.

2.2.3 Adding Other Operators

In practical applications, existing operators may not satisfy specific analytical requirements. `iDataLake` supports the seamless addition of new operators to address such needs. Only the following information is required to add a physical operator: (1) The programmed implementation of the physical operator for execution. (2) A detailed text description of the operator, including its purpose, input parameters, and outputs.

If these descriptions are incomplete, `iDataLake` can call the LLM to generate them. By defining these elements, `iDataLake` ensures that the newly added operators integrate seamlessly into the analytics framework.

2.3 Embedding and Linking

As introduced in Section 1, a key challenge for data analytics over data lakes is to efficiently identify the data that needs processing, as processing the complete data lake, extensively large in volume, is impractical. The effectiveness of identifying relevant data is critical since all subsequent operations are performed on this subset. Therefore, the identification process must achieve both high recall and high precision. High recall with low precision retains excessive irrelevant data, leading to inefficiencies. Conversely, high precision with low recall excludes relevant data, rendering correct results unattainable due to incomplete information.

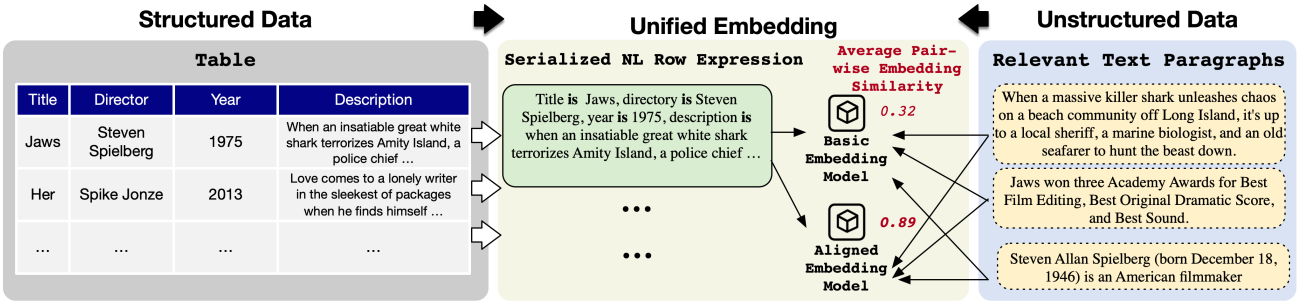

Figure 3: Example of embedding alignment between tables and text.

To address this, we propose a bi-encoder-based method that transforms diverse data types into a unified embedding space, where the similarity between embedding vectors serves as a metric to identify relevant data across modalities. However, training separate embedding models for different data types inevitably leads to misaligned embeddings. For example, embeddings for text and table data, although of the same dimension, are inherently unaligned. Achieving semantic alignment between these embeddings is a recognized challenge [11]. To the best of our knowledge, we are the first to semantically align embeddings for table data and text data.

Existing methods extend text embedding models to structured table data by serializing table rows into natural language (NL) sentences using formatting rules such as *[Feature1] is [Value1], [Feature2] is [Value2], ...* or *[Feature1]: [Value1], [Feature2]: [Value2], ...*, where *[Feature]* denotes the column name and *[Value]* denotes the cell value. Figure 3 illustrates this approach. However, directly computing embeddings for table rows from their serialized NL expressions is inaccurate, as the serialized format differs significantly from standard NL expressions, leading to embedding similarities that fail to reliably identify relevant data.

Embedding Alignment Between Different Types of Data. To overcome this limitation, we introduce a fine-tuning process that aligns text embeddings with embeddings derived from serialized table data. We collect a corpus of tables paired with ground-truth relevant text paragraphs from diverse sources. During fine-tuning, we employ a contrastive learning framework: table rows paired with their ground-truth text paragraphs are treated as positive examples, while table rows paired with unrelated text paragraphs serve as negative examples. Using these training pairs, we compute the Multiple Negatives Ranking Loss [12], which minimizes embedding distances for positive pairs and maximizes embedding distances for negative pairs.

The fine-tuning process proceeds iteratively, adjusting the embedding space to bridge the semantic gap between table and text data. Upon convergence, embeddings achieve meaningful alignment, enabling similarity measures in the unified space to accurately identify relevant data across data types. This alignment ensures both high recall and high precision in retrieval tasks, significantly enhancing system performance.

Data Linking. The vast size of data lakes results in an enormous number of data embeddings, posing significant challenges for efficiently locating relevant data. To address this, we observe that relevant data often forms natural clusters. Specifically, subsets of data relevant to one query are likely to be relevant or irrelevant to other queries together, as they often represent related entities. Leveraging this insight, we propose a preprocessing approach that identifies and organizes these clustering relationships within the data lake. This process divides the data into distinct blocks, where related data locate in the same block and the relevant blocks are linked.

To achieve this, we employ the Louvain community detection algorithm [13], which identifies hierarchical clustering relationships in the data. The algorithm starts by constructing a graph where

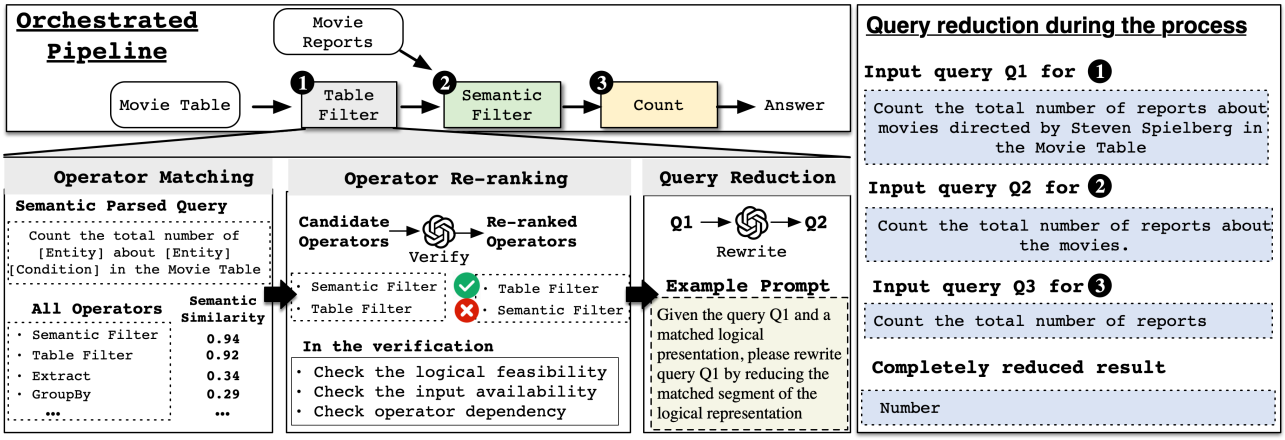

Figure 4: An example pipeline orchestration process, detailing the selection of the Table Filter operator.

nodes represent data items and edges represent embedding similarities. It iteratively optimizes modularity to detect communities, i.e., subsets of data points with high pair-wise similarities, producing a hierarchy of clusters. These clusters are then used to organize the data into blocks, with each block representing semantically similar data. When a query is issued, we first identify the relevant blocks and then locate specific data within each block. This hierarchical clustering approach reduces the data needing consideration, enhancing both retrieval efficiency and query accuracy.

2.4 Pipeline Orchestration

Orchestrating an accurate pipeline for query execution over multi-modal data lakes presents unique challenges due to the heterogeneity of data types and the absence of unified data schemas. Unlike traditional query execution in relational databases, which operates over structured data with predefined schemas and deterministic query plans, multi-modal data lakes require a flexible and adaptive pipeline orchestration mechanism capable of handling diverse data modalities and complex analytics tasks. In this section, we introduce the pipeline orchestration method employed in `iDataLake`, which iteratively selects and applies appropriate operators to reduce the query until it is fully solved. This method combines the semantic understanding capability of LLMs with a cost-efficient, two-stage operator matching strategy that minimizes LLM invocations to reduce cost and improve efficiency.

Overview of the Orchestration Process. The orchestration process in `iDataLake` follows an iterative workflow aimed at progressively reducing the query. Each iteration involves three key steps: (1) identifying suitable physical operators that can solve part of the query (coarse-grained matching); (2) validating the applicability and availability of the matched operators (fine-grained re-ranking), and (3) applying a feasible operator to simplify the query, solving part of the query logically. This process repeats until the query is fully resolved, which is verified by the LLM. We next introduce each step in detail.

Operator Matching. Efficiently matching a query with suitable physical operators is critical for effective pipeline orchestration. A naive approach of directly prompting an LLM with descriptions of all operators is infeasible due to constraints such as limited LLM context length and inaccurate selection among numerous options [4]. To address these issues, `iDataLake` employs a two-stage operator matching method:

Coarse-Grained Matching. This stage quickly eliminates irrelevant operators using low-cost checks. The key idea is that operators are likely to be applicable only if their use cases align with the query. However,

the specific values in the NL expressions are not helpful for the matching and can even mislead the matching. Therefore, to facilitate matching, both the query and operator use cases are transformed into *"logical representations"* by replacing concrete values with semantic placeholders such as [Entity] and [Condition]. The transformation is performed by instructing LLMs to conduct semantic parsing with few-shot examples. After computing these logical representations, the relevance between query and operators can be measured by the similarity of their semantic embedding vectors. For instance, the query *"Select the documents that are related to swimming"* corresponds to the logical representation of *"Select the documents that [Condition]"*, which has high semantic embedding similarity with the logical representation of a use case for the *Filter* operator *"Select the documents that satisfy [Condition]"* and thereby the matching can be conducted correctly.

Logical representations of operator use cases and their semantic embeddings can be precomputed. The logical representation of the query can be computed online. These embeddings enable efficient similarity-based comparisons, filtering out irrelevant operators accurately without involving the LLM.

After coarse-grained filtering, the remaining operators are further evaluated using the LLM. In this stage, the LLM verifies the applicability of each operator through prompts and few-shot examples. For applicable operators, the LLM determines the required inputs and checks their availability in the current context. This step relies an intermediate variable list maintained throughout the orchestration process, ensuring accurate and context-aware operator selection.

Fine-Grained Re-ranking. After coarse-grained filtering, the remaining operators are further evaluated using the LLM. In this stage, the LLM is instructed to verify the applicability of each operator through prompts and few-shot examples. For applicable operators, the LLM determines the required inputs and checks their availability in the current context. Specifically, an intermediate variable list is maintained throughout the whole orchestration process. The list is initialized with the whole data lake. During the orchestration, intermediate results are generated after the application of each operator. For each intermediate data, the LLM is instructed to generate a short text description of it. The generated text descriptions will then be added to the prompt to guide the LLM to select the appropriate inputs. Meanwhile, based on the dependency relationship between the inputs and outputs of the operators in the pipeline, the dependency between operators can also be determined. For each operator, only the dependency relationships with the operators whose outputs are prerequisites are necessary. In this way, a directed acyclic graph, i.e., a DAG-format pipeline can be constructed that can obtain much higher efficiency than sequential pipelines.

Query Reduction. Once an applicable operator is selected, it is applied to simplify the query. To ensure accurate query reduction, the LLM is instructed to conduct this reduction. This involves resolving part of the query logically. A concrete example of this process is illustrated in Figure 4.

Iterative Query Decomposition. The process iterates in a DFS manner until the query is fully reduced to a solved form. Each iteration involves selecting an appropriate operator, validating its applicability, and applying it to reduce the query. The resulting plan is represented as a directed acyclic graph (DAG), where nodes correspond to the selected physical operators, and edges represent dependencies between them. Notably, this orchestration method can generate multiple candidate pipelines by exploring multiple paths in the DFS process.

By combining coarse-grained filtering with embedding-based similarity and fine-grained LLM-driven operator selection, `iDataLake` achieves both high efficiency and accuracy in operator matching. This iterative reduction process enables `iDataLake` to systematically simplify complex queries over multi-modal data lakes while leveraging LLMs only when inevitable. When the query is completely solved by this iterative process, `iDataLake` constructs an efficient plan to solve complex queries effectively.

2.5 Pipeline Execution

Unlike traditional databases, where execution plans are deterministic to guarantee correct results, `iDataLake` needs to be able to adapt dynamically to handle the inherent unpredictability of both unstructured data and complex queries. This section describes how `iDataLake` executes interactively, adjusting its pipeline in response to intermediate results and unforeseen conditions. As illustrated in Figure 1, `iDataLake` can decide to continue execution, adjust the pipeline, replan, or re-identify relevant data based on the state of intermediate results.

Determining Operator Inputs. Similar to the orchestration phase (Section 2.4), identifying the correct inputs for each operator is critical during execution. In the orchestration phase, the relationships between operators' inputs and outputs are established. During execution, `iDataLake` determines operator inputs according to these identified relationships. Additionally, after each intermediate execution step, the LLM generates a brief textual description of the results. If an input determined during orchestration is unavailable or unsuitable during execution, `iDataLake` dynamically re-evaluates the input selection by repeating the orchestration process. This re-evaluation uses the generated descriptions of intermediate results as part of the prompt to guide the LLM in selecting appropriate inputs for the operator.

Parallel Topological Execution. To maximize execution efficiency, `iDataLake` follows a bottom-up, parallelized execution strategy based on the pipeline's topological order. Operators are executed in parallel once their prerequisites are met, continuing until all operators have been executed and the final result is produced. After each operator executes, `iDataLake` verifies whether its intermediate result aligns with expectations. Discrepancies can occur due to issues in the initially discovered relevant data or the incorrectness of the orchestrated pipeline. Therefore, if discrepancies are found, `iDataLake` first re-examines the relevant data identification steps, querying the LLM to rewrite the original query in an alternative form and using the rewritten query to re-identify relevant data. If the re-identified data closely matches the initial results, the system triggers a dynamic adjustment to the pipeline, as described next.

Pipeline Adjustment During Execution. When an operator fails to produce the expected result, `iDataLake` dynamically adjusts the execution plan without restarting the entire planning and execution process. Early stopping conditions, identified by the LLMs, can prune unnecessary operations, enhancing the execution efficiency.

Incremental Execution. As discussed in Section 2.4, multiple candidate plans are generated during the planning phase. Although only one plan is executed, many of these plans share a common sequence of initial operators due to their generation by a depth-first search (DFS) strategy. When an operator fails, `iDataLake` selects the plan with the longest matching sequence of initial operators and resumes execution from the last successful operator. This prevents the need to regenerate the entire plan. However, if there is minimal overlap between plans or no matching plans, `iDataLake` initiates replanning.

Replanning for Adjustment. If no plan shares a sufficient initial sequence, `iDataLake` replans from the point of failure. This is faster than starting from scratch (replanning for the initial query), as the current query has already been partially simplified by the earlier execution. If no suitable pipeline can be found, `iDataLake` backtracks to the last successful operator, repeating this process until it reaches the root (i.e., backtracks to the initial query).

Pruning Based on Insights from Data Analysis. Analyzing unstructured data can uncover relationships such as equivalencies or exclusivities between entities or conditions that were not initially identified during the planning phase. These insights enable the optimization of query execution by eliminating redundant operations. For instance, if two conditions are found to be equivalent, applying both filters becomes unnecessary, and one can be skipped to improve efficiency. Conversely, if conditions are mutually exclusive, the query can be simplified or terminated early, avoiding unnecessary data processing. Consider the query: *List the mathematicians who have won the Fields Medal at the age of 50 or older.* By

analyzing the Fields Medal's eligibility criteria, it becomes evident that the conditions *"won the Fields Medal"* and *"at the age of 50 or older"* are mutually exclusive, as the award is restricted to individuals under 40. This insight allows the query execution to be halted immediately, saving computational resources by avoiding processing large datasets for a result that is guaranteed to be empty.

3 Challenges and Opportunities

Scalability and Efficiency. A major challenge in implementing LLM-powered analytics on a large data lake is the inherent high cost and low computation efficiency of LLMs. Multi-modal data lakes consist of structured, semi-structured and unstructured data, which require distinct processing pipelines. Efficiently querying such vast datasets with LLMs necessitates reducing invocation costs without sacrificing performance. Scaling LLMs to handle such vast quantities of heterogeneous data, demands careful optimization of both the model's computation and the overall query pipeline architecture. For example, ordering the operators in the pipeline based on cardinality estimation [14, 15] is crucial for efficiency. Research into more efficient LLM architectures, such as fine-tuning models for domain-specific tasks or incorporating hybrid models with different sizes for tasks of different complexity [17], could mitigate the scalability issue. Additionally, caching intermediate results in a distributed system or identifying and indexing critical data components, such as coresets [18, 19] could improve the system's efficiency in handling large datasets.

Interpretability and Transparency. LLMs are often criticized for their "black box" nature, making it difficult to understand how they generate results or why they make particular decisions. This lack of transparency poses challenges for users who need to trust and interpret the outcomes of their data analytics queries. In the context of multi-modal data lakes, where complex relationships between structured and unstructured data must be identified and analyzed, ensuring that the system's reasoning is interpretable becomes even more critical. Approaches to improving the interpretability of LLMs, such as attention mechanisms and counterfactual reasoning, can enhance the transparency of the system. By providing insights into the specific data features or relationships that influence the model's decision-making, these methods could increase user confidence in the results. Moreover, incorporating explainability into the pipeline design could facilitate the adoption of LLM-powered analytics in sensitive or regulated industries.

Evaluation and Benchmarking. The evaluation of LLM-driven analytics over multi-modal data lakes remains an open challenge, particularly in comparing the effectiveness of different methodologies. Metrics for traditional data analytics, such as precision, recall, or F1 score, may not adequately capture the nuances of integrating structured and unstructured data sources. Furthermore, due to the large-scale and dynamic nature of the data lake, it is difficult to develop comprehensive benchmarks that can consistently evaluate the model's performance across different types of queries and use cases. Developing new benchmarking methodologies tailored for multi-modal data lake analytics would be an essential step forward. These benchmarks could focus on both the accuracy of the results and the system's ability to integrate and analyze heterogeneous data sources. Additionally, creating standardized test datasets for data lake environments could encourage further research and provide a basis for consistently comparing different models and systems.

Model Adaptation to Domain-Specific Needs. LLMs are typically trained on a wide array of general-domain data, which may not always capture the specific nuances of particular industries or use cases, such as legal, healthcare, or finance. As a result, LLMs may not fully comprehend domain-specific terms, contexts, or relationships without further domain-specific fine-tuning. For example, the data linking may become inaccurate for personal domain data, even after the alignment finetuning. Domain adaptation of LLMs is a promising avenue for improving system performance in specialized applications.

By leveraging transfer learning or few-shot learning techniques, LLMs can be fine-tuned to perform better on industry-specific tasks. Furthermore, the system can be augmented with domain-specific knowledge bases or ontologies, which can enhance the model's understanding of complex, industry-relevant concepts.

4 Conclusion

In this paper, we present `iDataLake`, a novel LLM-powered analytics system designed to address the challenges of data analytics in multi-modal data lakes. By integrating large language models with advanced semantic operators, embedding-based data linking, and dynamic pipeline orchestration, `iDataLake` provides a unified framework for querying and analyzing diverse data types, including structured, semi-structured, and unstructured data. `iDataLake` introduces several key innovations including semantic operators tailored to the unique requirements of multi-modal analytics, unified embedding-based data linking for aligning heterogeneous data types in a common semantic space, dynamic pipeline adjustment to adapt to evolving query execution requirements and interactive and incremental plan execution to ensure robust and efficient query handling.

Acknowledgment. This paper was supported by National Key R&D Program of China (2023YFB4503600), NSF of China (61925205, 62232009, 62102215), Shenzhen Project (CJGJZD20230724093403007), Zhongguancun Lab, Huawei, and Beijing National Research Center for Information Science and Technology (BNRist).

References

[1] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, Christopher Ré. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes *VLDB*, 2024.

[2] Xueqing Wu, Jiacheng Zhang, Hang Li Text-to-table: A new way of information extraction *ACL*, 2022.

[3] Matthias Urban, Carsten Binnig. Efficient Learned Query Execution over Text and Tables [Technical Report] *Corr*, 2024.

[4] Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, Minlie Huang. Large Language Models Are Not Robust Multiple Choice Selectors *ICLR*, 2024.

[5] Samuel Madden, Michael J. Cafarella, Michael J. Franklin, Tim Kraska. Databases Unbound: Querying All of the World's Bytes with AI *VLDB*, 2024.

[6] Liana Patel, Siddharth Jha, Parth Asawa, Melissa Pan, Carlos Guestrin, Matei Zaharia. LOTUS: Enabling Semantic Queries with LLMs Over Tables of Unstructured and Structured Data *CoRR*, 2024.

[7] Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, Sepanta Zeigham, Aditya G. Parameswaran, Eugene Wu. Towards Accurate and Efficient Document Analytics with Large Language Models *CoRR*, 2024.

[8] Jiayi Wang, Guoliang Li. AOP: Automated and Interactive LLM Pipeline Orchestration for Answering Complex Queries *CIDR*, 2025.

[9] Matthias Urban, Carsten Binnig. CAESURA: Language Models as Multi-Modal Query Planners *CIDR*, 2024.

[10] Zui Chen, Zihui Gu, Lei Cao, Ju Fan, Samuel Madden, Nan Tang. Symphony: Towards natural language query answering over multi-modal data lakes *CIDR*, 2023.

[11] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever Learning Transferable Visual Models From Natural Language Supervision *ICML*, 2021.

[12] Nils Reimers and Iryna Gurevych. *EMNLP-IJCNLP*, 2019.

[13] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte and Etienne Lefebvre. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[14] Jiayi Wang, Chengliang Chai, Jiabin Liu, Guoliang Li FACE: a normalizing flow based cardinality estimator *Proc. VLDB Endow.*, 2022.

[15] Jiayi Wang, Chengliang Chai, Jiabin Liu, Guoliang Li Cardinality estimation using normalizing flow *VLDB J.*, 2024.

[16] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, Alfons Kemper. Learned Cardinalities: Estimating Correlated Joins with Deep Learning *CIDR*, 2019.

[17] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Gerardo Vitagliano. Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing *CIDR*, 2025.

[18] Jiayi Wang, Chengliang Chai, Nan Tang, Jiabin Liu, Guoliang Li. Coresets over multiple tables for feature-rich and data-efficient machine learning *Proc. VLDB Endow.*, 2023.

[19] Chengliang Chai, Jiayi Wang, Nan Tang, Ye Yuan, Jiabin Liu, Yuhao Deng, Guoren Wang. Efficient Coreset Selection with Cluster-based Methods *KDD*, 2023.

[20] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data *AISTATS*, 2017.

[21] Cynthia Dwork, Frank McSherry, Kobbi Nissim, Adam D. Smith. Calibrating Noise to Sensitivity in Private Data Analysis *TCC*, 2006.

Large Language Models as Pretrained Data Engineers: Techniques and Opportunities

Yin Lin, Bolin Ding, Jingren Zhou
Alibaba Group
{yin.lin, bolin.ding, jingren.zhou}@alibaba-inc.com

Abstract

The advent of large language models (LLMs) has yielded promising results across various fields. With their ability to understand, retrieve, synthesize information, and perform advanced reasoning, LLMs have shown significant potential for facilitating and automating data management. We propose that the extensive knowledge embedded within these models enables them to serve as pre-trained data engineers, enhancing both the effectiveness and scope of data tasks.

In this paper, we illustrate an architectural framework for integrating LLMs into data engineering workflows, aiming to enhance the applicability and usability of LLM-assisted data engineering solutions. We explore key techniques and opportunities across three critical stages: (a) data wrangling, to simplify and optimize data preparation and transformation; (b) analytical querying, to extend querying capabilities and interfaces in data systems; and (c) table augmentation, to enhance the original tables with additional data, improving the performance of data-centric tasks like machine learning.

1 Introduction

Large Language Models (LLMs) have recently demonstrated remarkable progress across a variety of tasks, including natural language processing, question answering, code generation, and information retrieval [3, 8, 25, 57, 60, 75]. These models, trained on extensive data corpora, encompass vast amounts of integrated knowledge and possess advanced reasoning capabilities [79–81]. Moreover, they have shown strong cross-task generality on a wide range of natural language tasks.

Building on these impressive capabilities, the rapid development of LLMs has motivated data researchers to reconsider traditional challenges in data management [21, 54, 72]. Data engineering, which includes the preparation, analytical querying, and enrichment of data, demands specialized domain expertise and is often time-consuming, lacking a one-size-fits-all solution due to the complexity of data-related tasks [65]. For example, considering a data imputation task, imputing the categories of items might require row-by-row interactions to understand contextual nuances, whereas imputing the Body Mass Index (BMI) can be completed by using standardized formulas that rely on height and weight. Similarly, to answer a user question such as "*which item is the most positively reviewed*" might require semantic parsing of each row, whereas answering "*what are the monthly sales for MacBook Pro*" may involve translating the question into an aggregate query to obtain the result. Pioneering studies [8, 21, 52, 85] have illustrated that well-crafted *prompts* can effectively guide LLMs to achieve state-of-the-art performance in specific data engineering tasks, such as data imputation, entity matching, and Text2SQL. However, integrating LLMs into complex and heterogeneous data engineering workflows remains a challenging endeavor.

First, designing a single prompt for each distinct data preparation task is suboptimal, as data wrangling generally necessitates a multi-step process involving diverse tasks. Employing domain-specific

solutions customized for each data set and problem may not be uniformly effective. Secondly, while LLMs have empowered data querying with strong semantic reasoning capabilities and world knowledge [79–81], challenges arise in designing effective interfaces for integrating LLMs into data systems to facilitate efficient analysis. Recent efforts have focused on developing SQL-like [4, 45, 68, 86] and Pandas-like [58] programming interfaces to improve LLM usability across diverse systems. Additionally, to assist non-experts who may lack the ability to write code and queries, facilitating the translation of natural language questions into executable queries [64, 77, 82] is also critical for enhancing accessibility. Lastly, to enhance the performance of data-centric applications such as machine learning, augmenting the original tabular data can be highly beneficial. However, challenges remain to effectively utilize LLMs for this purpose. LLMs have the potential to enrich original datasets by retrieving relevant information [20, 33, 34] and performing feature engineering [31, 46] to generate useful features.

In this paper, we introduce an architectural framework for integrating the powerful capabilities of LLMs into data engineering, focusing on three key stages: *data wrangling*, *analytical querying*, and *table augmentation for machine learning*. Within each stage, we review current research and demonstrate how LLMs can facilitate and potentially expand the scope of data-related tasks.

In particular, we elaborate on three systems we have recently developed: UniDM [61], which proposes a unified framework for data wrangling; DAIL-SQL [22], a Text2SQL solution that systematically examines both in-context learning and supervised fine-tuning to optimize performance; and SMARTFEAT [46], which automates feature engineering for machine learning by identifying and applying appropriate transformations on the original table. Furthermore, we envision future directions to further extend LLM-assisted data engineering applications, including (1) developing automated, unified systems for efficient end-to-end data preparation, (2) enhancing querying capabilities for unstructured data and world knowledge in data systems through flexible querying interfaces, and (3) constructing automated solutions for machine learning data processing.

2 Integrating LLM Modules in Data Engineering

Data engineering often contains complex and labor-intensive tasks that traditionally require substantial engineering effort. Figure 1 outlines an architectural framework for integrating LLM modules into data engineering workflows, centered on three stages: 1) *data wrangling*, (2) *analytical querying*, and (3) *table augmentation for machine learning*.

Data Wrangling. Real-world data are often heterogeneous, incomplete, or contain errors, rendering them unsuitable for direct analytical or modeling tasks. Data wrangling typically encompasses tasks such as entity matching [19], error detection [1, 30], and data imputation [26], enabling the integration, cleaning, and transformation of raw data into structured formats suited to downstream applications. To facilitate complex data wrangling, the framework incorporates LLMs into *data wrangling operators*. In these operators, LLMs are prompted with the *task parameters*, such as the task name, task query, and relevant *contextual inputs* from the dataset. The LLMs are then applied to each row to predict the next word for completing tasks [54, 61, 72], or to the entire dataset to generate code [13] to process data more efficiently.

Analytical Querying. Data engineers and analysts often query datasets to extract meaningful insights. The framework extends the traditional definition of relational querying by incorporating LLM capabilities, which enable querying not only of structured data but also of unstructured textual content, documents, and world knowledge through various querying interfaces. Operators leveraging LLMs facilitate access to extensive corpora of unstructured text [68, 86] and can even function as knowledge storage mechanisms [7]. The framework describes multiple interfaces for utilizing LLMs in analytical querying, including *SQL interfaces* [4, 5, 15, 45, 68, 70, 86], *programming interfaces* [4, 58],

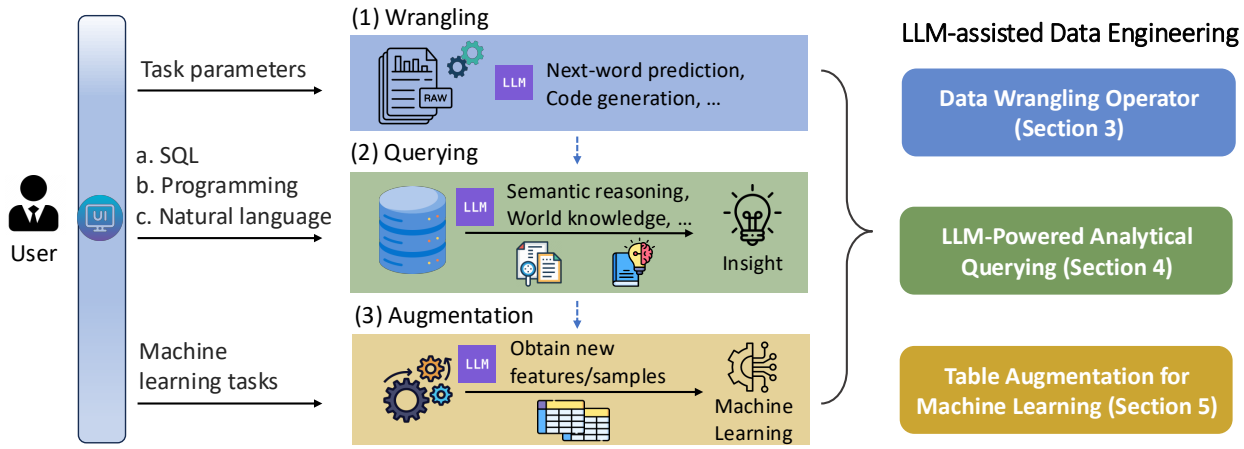

Figure 1: Integrating LLM-based modules into data engineering workflows: (1) facilitating data wrangling tasks, (2) supporting complex analytical querying through multiple interfaces, and (3) augmenting original tabular data for improved machine learning performance.

and *natural language interfaces* [7, 48], in which LLMs are commonly embedded as SQL extensions or as Pandas-like data science APIs. They can also translate natural language (NL) into logical operators [47, 49, 62, 64, 77, 82] and enable interactive, conversational NL-based interfaces [7, 48].

Table Augmentation for Machine Learning. Table augmentation can enrich original datasets with new features or samples, thereby improving performance in data-centric tasks such as machine learning. However, this process often relies heavily on domain expertise and demands substantial manual effort from data scientists. Given a machine learning task, the framework leverages LLMs to support two types of table augmentation [14]: generation-based augmentation [31, 46], which uses LLMs to iteratively generate semantically meaningful features for original datasets, and retrieval-based augmentation [27, 34], which retrieves additional external data.

In the following sections, we review the detailed techniques for integrating LLM modules in each stage. We then discuss unique opportunities to expand the scope and improve the usability of LLM-assisted data engineering solutions.

3 Data Wrangling with LLMs

Data wrangling involves a series of data preparation steps that transform datasets into formats suitable for analysis and modeling. To leverage LLMs in facilitating complex data preparation tasks, we first formalize these tasks using a generalized *data wrangling operator*, defined as follows:

Definition 3.1 (Data Wrangling Operator) *Let $\mathcal{D}$ be a dataset requiring data wrangling and let T denote the task parameters. An LLM-based data wrangling operator opr_{LLM} is defined as a function*

$$\mathsf{opr}_{LLM} : (\mathcal{D}, T) \rightarrow \mathcal{D}_o,$$

where LLM is the selected large language model. Given T and $\mathcal{D}$, the operator constructs prompts to instruct the LLM. The outputs generated by the LLM is then applied to $\mathcal{D}$, resulting in a clean and structured dataset $\mathcal{D}_o$.

Data Preparation Tasks. Data wrangling is rarely a single-step process; it usually involves multiple individual preparation steps, each implemented by a data wrangling operator. These tasks can be

categorized into three broad categories based on their purposes: *data integration* [17], *data cleaning* [63], and *data transformation* [29, 35]. Data integration involves discovering and combining data from various sources, which typically includes tasks such as schema matching [71], entity resolution [10, 44, 78], and join/union discovery [20, 33, 34]. Data cleaning aims to remove or replace inaccurate data values with more accurate ones. It typically includes tasks such as deduplication, error detection [1, 30], and data imputation [26]. Data transformation involves restructuring and filtering data, including tasks such as changing schema [35], and removing irregular rows or values [26]. Recent research has explored the use of LLMs to address one or more of these tasks, showing promising results and, in some cases, achieving state-of-the-art performance.

Prompts. LLMs have shown remarkable proficiency in text-intensive tasks. Researchers have also demonstrated that LLMs trained to predict the next word (e.g., "*Mary has a little*" – "*lamb*") can be adapted to data-related tasks by providing natural language descriptions of those tasks [54]. For example, to perform entity matching, one might ask an LLM "*Are iPhone 16 Pro and iPhone 16 Pro Max the same?*" and obtain the answer "*No.*". Moreover, models such as GPT-3 [9], GPT-4 [57], and LLaMA [75] also possess robust code-generation capabilities like human programmers.

Therefore, within data wrangling operators, the LLM is guided by *prompts* derived from task parameters, such as the task name, task query, and contextual inputs extracted from the data. These prompts can be structured in various forms to perform row-wise executions using the ability of LLMs in performing next-word prediction. For example, for a data imputation task, the prompt can be formatted such as a *question* (e.g., "`What is the timezone of Copenhagen?`"), a *cloze* (e.g., "`Copenhagen is in the ____ timezone.`"), or a *completion* (e.g., "`The timezone of Copenhagen is...`"). Furthermore, prompts can also be designed to synthesize code for domain-specific solutions [13, 52, 55], such as "write a Python regular expression to extract dates in the format 'YYYY-MM-DD'." The operator then incorporates a parser to extract outputs from LLMs and may apply additional transformations to the original datasets, thereby producing the output data.

3.1 Unifying LLM-based Data Wrangling

While the integration of LLMs into data wrangling has demonstrated exceptional performance in various data preparation tasks [44, 54, 72, 84], these approaches often rely on specifically tailored prompt designs for each task, which limits their usability as end-to-end platforms that encompass multiple data preparation steps.

In this subsection, we explore the generality of data preparation tasks based on Definition 3.1. We propose that disparate data preparation tasks could be unified, applying a general procedure to solve different tasks. To this end, we introduce a unified system, UniDM [61], which generalizes data preparation tasks into three main steps as shown in Figure 2. The core objective of UniDM is to unify diverse data tasks by developing a general mechanism that transforms task parameters T, and contextual inputs derive from $\mathcal{D}$ into a prompt understandable by LLMs to complete the task.

The first step, **context retrieval**, extracts the relevant context from the dataset $\mathcal{D}$ necessary for solving the task. Prior approaches often require users to manually specify the context [6, 54] or rely on attribute similarity to identify useful records [2, 51]. However, these methods can be less effective when dealing with a diverse range of complex data preparation tasks. In contrast, UniDM introduces an automated retrieval process that leverages LLMs through two specialized prompt templates. These templates guide LLMs to determine the relevant *attributes* and *rows* in $\mathcal{D}$ required for completing the task T. For example, consider a data imputation task shown in Figure 2, where the goal is to impute the timezone for Copenhagen. In this case, the first prompt identifies the "`country`" attribute as essential for inferring the timezone and the second prompt selects a subset of relevant records to guide this task (e.g., via few-shot prompting). One such record might be: "`city: Alicante, country:`

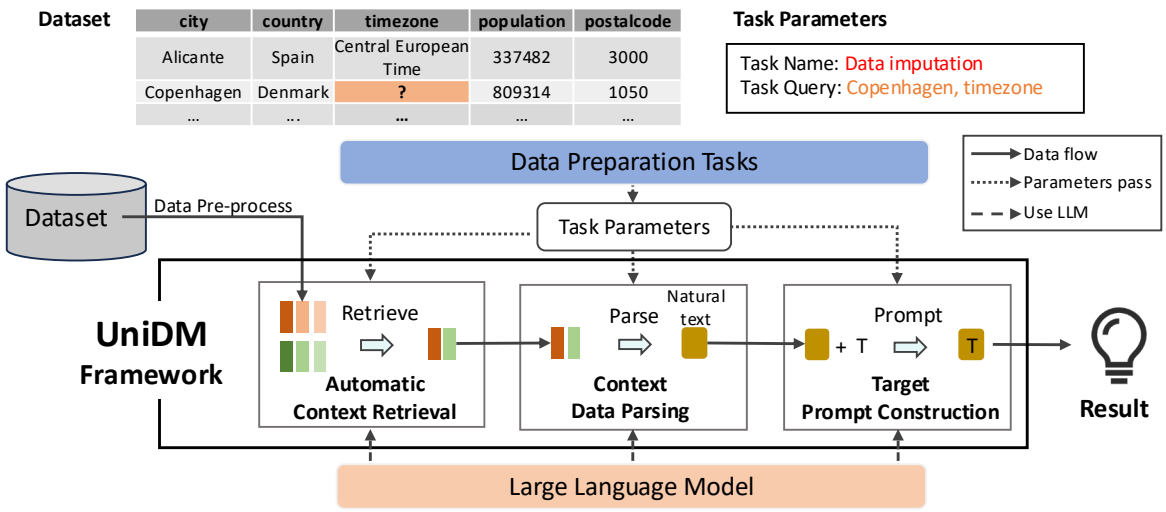

Figure 2: An overview of the UniDM framework.

`Spain, timezone: Central European Time`".

The second step, **context data parsing**, transforms the retrieved context from a tabular format into a textual representation, enabling LLMs to better capture the semantics. Unlike conventional methods that *serialize* input rows into a simple text string [54, 72], UniDM employs a prompt template to instruct LLMs to create a semantically rich text representation of the context. For example, the tabular record "`city: Alicante, country: Spain, timezone: Central European Time`" is transformed into: "`Alicante is a city in Spain and is in the Central European timezone.`".

The final step is **target prompt construction**, where UniDM synthesizes the task parameters T, including the task name, task query, and the contextual inputs of dataset $\mathcal{D}$ obtained from the previous two steps, into a final prompt for the LLMs to generate results. The construction of this final prompt is also relied on LLMs. For the data imputation task, the final prompt could take the form of a cloze-style prompt: "`This task involves imputing the missing value ...; The context is: Alicante is a city in Spain and is in the Central European timezone ...; Copenhagen is a city in Den- mark and is in the ____ timezone.`" This prompt is then used to generate the final result for the data preparation task.

By abstracting these steps, UniDM unifies disparate data preparation tasks, offering a systematic approach to leverage LLMs effectively and generalizing across multiple task types. In a similar vein, another recent framework, CHORUS [38], presents a unified approach for synthesizing data discovery and exploration tasks. Unlike UniDM's three-step procedure, CHORUS decomposes prompts into six fixed components and employs specialized templates and context retrieval methods to automatically construct the final prompt. It has shown highly effective performance for data discovery tasks such as table-class detection, column-type annotation, and join-column prediction.

3.2 LLM-as-a-Compiler for Data Wrangling

While UniDM offers a unified framework for automating data wrangling effectively, a limitation lies in its reliance on *row-wise execution*, which can become inefficient and costly for large datasets due to the need for LLM invocations on each record. To overcome this limitation, more recent work, SEED [13], proposes an improved approach by introducing an optimizer that automatically selects from four LLM-assisted modules: *CodeGen*, which generates code; *CacheReuse*, which reuses previous LLM query results; *ModelGen*, which distills an LLM into a smaller machine learning model; and *LLM*, which

directly generates answers. By combining these modules, SEED automatically produces a hybrid data wrangling solution that achieves performance comparable to row-wise execution while dramatically reducing the number of LLM calls.

4 Data Analytics with LLMs

Traditional relational queries enable users to perform scalable and accurate analyses on structured data through formal querying languages. However, much of the information that data users want to query [50] resides not only in structured datasets, but also in various types of unstructured sources such as text, documents, and even in the vast world knowledge encoded within LLMs. Moreover, traditional systems often assume users have the technical expertise to write complex queries, which can be a barrier for non-expert users. For example, a restaurant owner may want to analyze customer reviews (e.g., *identifying the most positively reviewed dishes in Japanese restaurants in New York City*) but may lack the skills to write complex queries or code.

LLMs have demonstrated the ability to understand, extract, and answer questions using unstructured data and world knowledge through methods such as retrieval-augmented generation (RAG) [40]. Recent research [7] has demonstrated the promising potential of integrating LLM capabilities into data systems to combine strong semantic reasoning and LLM knowledge with the efficient computational query execution capabilities of traditional relational data systems.

In this section, we provide an expanded definition of analytical querying that extends the relational querying definition $Q : (\mathcal{D}_s, \mathsf{q}) \rightarrow \mathcal{R}$, where q represents a query in the formal query language, and $\mathcal{D}_s$ is the structured dataset, and $\mathcal{R}$ is the query result. The LLM-powered analytical querying definition includes support for unstructured data, integrates knowledge from LLMs, and accommodates diverse interfaces, as follows:

Definition 4.1 (Analytical Querying) *Let $\mathcal{D}_s$ denote the structured dataset, $\mathcal{D}_u$ denote the unstructured text data that may be leveraged for question answering, and $\mathcal{D}_{LLM}$ denote the world knowledge encoded within a large language model LLM. Given a user question q^+, an LLM-powered analytical querying is expressed as*

$$Q_{LLM} : (\mathcal{D}_s, \mathcal{D}_u, \mathcal{D}_{LLM}, \mathsf{q}^+) \rightarrow \mathcal{I},$$

where the output $\mathcal{I}$ represents the insights derived from the query result.

In Definition 4.1, the LLM-powered analytical querying not only enables analysis of structured data $\mathcal{D}_s$ but also enables the extraction and processing of information from unstructured text data $\mathcal{D}_u$ to answer the question. It also facilitates querying access and exploits the LLM knowledge through $\mathcal{D}_{LLM}$. In addition, the querying interface is not restricted to formal query languages; instead, the user question q^+ supports various formats according to the data system and user requirements, including SQL queries [4, 45, 68, 86], data science code snippets [4, 58], and natural language inputs [7, 47, 48, 62, 64, 77, 82]. It may also incorporate a conversational agent to translate query results into more easily interpretable insights in natural language [7, 48].

To support this extended analytical querying, recent research efforts have focused on enhancing or developing next-generation data systems [49] that leverage the capabilities of LLMs. In the following subsections, we will explore the specific mechanisms through which these systems achieve the goal:

- **Building declarative querying interfaces.** These interfaces enable users to ask questions based on their needs rather than focusing on the technical execution details [49]. They can express their queries in a more intuitive and user-friendly manner, which the systems then translate into the necessary operations.

- **Translating natural language questions into queries.** These systems may leverage semantic parsers driven by LLMs to convert user questions expressed in natural language into executable queries [48], which bridges the gap between user intent and system execution.
- **Expending the set of logical operators and enabling optimizations.** These systems offer an enriched set of logical operators that augment traditional relational operations [58]. By incorporating LLM-powered semantic data processing and information extraction capabilities, these operators work with traditional relational operators to facilitate optimization and effective query planning.

4.1 Interfaces for Analytical Querying

In real-world analytical querying applications, there are various design considerations when integrating LLMs into data analytics systems. State-of-the-art research primarily focuses on three types of user interfaces: *SQL interface*, *programming interface* and *natural language interface*.

SQL Interface. To incorporate user questions into executable operations within database management systems (DBMS), modern DBMS vendors have explored integrating LLMs as extensions of SQL user-defined functions (UDFs). For example, systems like BigQuery [70], Databricks [15], and Redshift [5] uses AI modules to perform information extraction based on the prompted user question. However, these LLM UDFs typically support only row-by-row execution, making them inefficient and prohibitively expensive for large datasets.

To address the challenge of querying both structured and unstructured data, several systems have proposed transforming unstructured data and LLM knowledge into a structured format compatible with SQL. For example, ZenDB [45] automatically extracts semantic hierarcal structure from text documents allowing users to impose a schema on their documents and query the document with SQL interface. Evaporate [4] generates structured views of data from input documents by employing efficient entity extraction techniques on semi-structured data. GALIOS [68] enables users to query large language models via an SQL interface, executing some parts of the query plan with prompt-based interactions to retrieve data from the LLM. Similarly, HQDL [86] extends SQL beyond the data at hand, using LLMs to answer questions that require additional context.

Programming Interface. Another approach to integrating LLMs into analytical querying is through programmatic interfaces provided by data science platforms, such as packages in Python, which enable developers to construct flexible data analysis pipelines. For example, Palimpzest [4] allows users to declaratively query text and images using an API, similar to querying tables in a relational database. The LOTUS system [58] provides a Pandas-like interface enriched with *semantic operators*, such as semantic filtering, ranking, and aggregation. This enables developers to make use of both traditional relational functionalities and advanced LLM-driven semantic reasoning capabilities in data analytics.

Natural Language Interface. Recent advancements have demonstrated that LLMs can translate natural language questions into executable relational queries with high accuracy [47, 62, 64, 77, 82]. Research has also explored developing interactive or conversational agents that provide end-to-end query support for non-expert users. For example, SUQL [48] builds a conversational interface to answer questions over semi-structured data by executing queries derived from user utterances that correspond to specific query intentions. Similarly, TAG [7] proposes a general-purpose query model that translates natural language inputs into queries and generates answers in natural language based on the query results. Although natural language interfaces greatly improve usability for non-expert users, they inherently carry ambiguities [41]. Therefore, we believe that data systems should not rely solely on natural language as the querying interface. An ideal design would map natural language inputs to expressive SQL-like or programming languages, ensuring accurate and efficient execution within the data system. Systems

such as SUQL and TAG exemplify this approach by first interpreting and generating SQL queries from natural language questions, then executing these queries and presenting the results in natural language.

4.2 Text2SQL Semantic Parser

A critical challenge in bridging user intentions to executable database queries lies in accurately translating natural language questions into executable queries. While Text2SQL has long been a focus in both natural language processing and database research, LLMs have recently emerged as a transformative paradigm for this task [47, 62, 77].

In this subsection, we present DAIL-SQL [22] which provides an integrated tool for improving LLM-based Text2SQL solutions. It systematically explores two key directions: *prompt engineering for in-context learning* on closed-source LLMs and *supervised fine-tuning (SFT)* on open-source LLMs.

Prompt Engineering. Effective prompt design is crucial to promote accurate SQL generation from LLMs [9]. DAIL-SQL examines both question representation and example selection to optimize prompt design.

- *Question representation:* This involves representing user questions and database information in the most informative format (e.g., natural text, code-like schemas). We extensively evaluate five widely adopted approaches [12, 47, 53, 56, 73]. We find that certain representations yield higher execution accuracy than others. In particular, using the OpenAI demonstration prompts [56] or the code representation [53] that includes comprehensive schema details tends to perform better. DAIL-SQL adopts the code representation and finds that including supplementary details, such as foreign key information and implication rules like the "*with no explanation*" implication, also helps improve execution accuracy.

- *Example selection and organization:* The in-context learning ability of LLMs [18] is also helpful in improving Text2SQL performance with carefully selected examples. DAIL-SQL demonstrates that few-shot prompting is most effective when examples are similar to the user's query. To identify effective examples, DAIL-SQL masks domain-specific terms in both the natural language questions and the pre-generated SQL for the user query and candidate examples. It then ranks the candidates based on question and query similarity. Experimental results indicate that selecting examples based on both question and SQL similarity consistently outperforms random selection. Furthermore, while including full example details may yield optimal performance, we find that for powerful LLMs like GPT-4, retaining only the question-SQL mapping is an efficient and effective approach.

Supervised Fine-Tuning (SFT). Beyond prompting, DAIL-SQL explores fine-tuning open-source LLMs (e.g., LLaMA variants [67, 73–76, 87]) to improve Text2SQL performance. Empirical results show that supervised fine-tuning is essential to achieve competitive accuracy. With carefully curated training data (e.g., from historical Text2SQL workloads), DAIL-SQL shows that fine-tuned open-source models exhibit strong potential for Text2SQL tasks. Unlike in-context learning with closed-source LLMs, fine-tuned open-source LLMs do not learn from contextual examples, e.g., queries in the current data analytical pipeline, but we can always applying in-context learning on fine-tuned LLMs to incorporate the learned experience and characteristics of the workload in an online data analytical pipeline.

To evaluate the performance of Text2SQL techniques, a variety of benchmarks have been proposed, such as Spider [83] and Bird [42], along with metrics like execution accuracy and query efficiency. DAIL-SQL demonstrates that both prompt engineering and supervised fine-tuning can significantly enhance Text2SQL's execution accuracy, providing valuable insights for data systems aiming to translate natural language questions into precise and reliable SQL queries. More recently, XiYan-SQL [23], a

state-of-the-art Text2SQL system, further improved the benchmark performance by integrating schema linking techniques to retrieve only relevant columns for a given natural language question, designing a more efficient schema representation, and implementing an ensemble strategy for LLM in candidate generation and selection.

However, Text2SQL research has primarily focused on relational queries over structured data, which covers only a fraction of the questions posed by real-world users [7]. The incorporation of LLMs into data systems brings significant opportunities to leverage world knowledge and semantic reasoning, thereby extending relational queries through a broader set of logical operators, as we will describe next.

4.3 Logical Operators

Executing analytical queries relies on a set of logical operators and their corresponding physical execution to complete the tasks. Integrating LLMs into data analytics does not mean overwriting traditional relational operators. SQL and data science operators remain essential for supporting scalable and efficient queries over structured data. The objective of developing LLM-powered *logical operators* is to augment these existing operators with AI-based transformations.

Recent research has primarily focused on extending SQL operators because they can be seamlessly integrated with database management systems and have the potential to leverage existing query engines and optimizing compilers. For example, SUQL [48] augments SQL with free-text primitives (`SUMMARY` and `ANSWER`) powered by LLMs. These primitives enable the system to provide row-by-row summaries and answer user questions on unstructured text data, while automatic optimizations are considered to reduce the number of LLM calls to improve efficiency. HQDL [86] treats an LLM as a virtual table that can be queried for information unavailable in structured data, and TAG [7] integrates LLM-based retrieval capabilities into table querying. In contrast, LOTUS [58] expands Pandas' operator set with semantic operators (e.g., `sem_map`, `sem_join`, and `sem_filter`), facilitating both logical query plan optimization and operator execution optimization. We see significant potential for developing additional operators. A key design criterion is to ensure these operators are expressive and comprehensive while enabling execution optimizations. For instance, strategies such as model selection or prompt engineering can optimize LLM performance, while using code generation instead of row-wise execution can improve efficiency.

Another crucial aspect is planning the query execution to provide answers to user questions. For instance, when a user poses a question in natural language, it's vital to generate a query execution plan that may involve extracting unstructured information or utilizing LLM knowledge. Traditional Text2SQL approaches, as discussed, are insufficient for generating queries and utilizing the optimizers in DBMS for the expanded set of operators. Utilizing the reasoning capabilities of LLMs to decompose questions into executable operations and enable optimization of query execution plans is important. Current methods often require users to specify updated schemas [45] or explicitly define the query plan [4, 58], and incorporate rule-based optimization techniques like operator reordering, predicate pushdown, and lazy evaluation to improve performance.

5 Table Augmentation for Machine Learning with LLMs

Machine learning (ML) is a pivotal data-centric application that plays a critical role in numerous decision-making processes [16, 39, 66]. It derives sophisticated patterns from historical data to produce predictive outcomes. However, real-world data are often unsuitable for direct ML training due to limited data entries and potentially insufficient features [14]. Acquiring more high-quality tabular data with well-suited features is essential for improving ML performance. However, this process often relies heavily

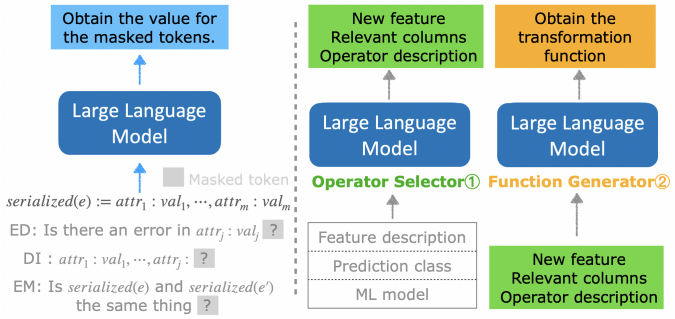

Figure 3: SMARTFEAT overview: advancing from row-level to feature-level generation.

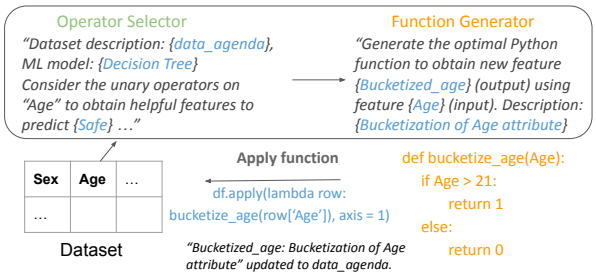

Figure 4: Example: constructing *Bucketized Age*.

on domain expertise, demanding considerable human effort [59] and remaining a persistent challenge in both database and data science research [11].

Numerous approaches have been proposed to automate the ML data processing pipeline [88], yet they often depend on machine learning or deep learning recommendations, which require substantial data collection of training, or employ rule-based systems, thereby limiting their applicability. Recent advancements in LLMs present significant opportunities to enhance tabular data augmentation process. Existing work broadly falls into two categories: *generation-based* approaches, which generate new data based on the original table, and *retrieval-based* approaches, which identify and extract relevant data from external sources. Building on the general definition of table augmentation tasks [14], we formally define the LLM-assisted table augmentation task for ML as follows:

Definition 5.1 (Table Augmentation for Machine Learning) *Let $\mathcal{T}_{\mathcal{A}}$ be the original table with feature set $\mathcal{A}$, and $\mathcal{P}$ be the data pool available for augmentation. Consider a downstream machine learning model $f_{\boldsymbol{\theta}}$, where $\boldsymbol{\theta}$ represents the model parameters. The objective of the table augmentation task is to transform $\mathcal{T}_{\mathcal{A}}$ into an augmented table $\mathcal{T}'_{\mathcal{A}_{new}}$ to enhance the performance of $f_{\boldsymbol{\theta}}$. Formally, the table augmentation task is defined as an augmentation function assisted with an large language model LLM:*

$$\text{aug}_{LLM} : (\mathcal{T}_{\mathcal{A}}, \mathcal{P}, f_{\boldsymbol{\theta}}) \rightarrow \mathcal{T}'_{\mathcal{A}_{new}}, \ \ s.t., \ \mathbb{E}(f_{\boldsymbol{\theta}}^{\mathcal{T}'_{\mathcal{A}_{new}}}) < \mathbb{E}(f_{\boldsymbol{\theta}}^{\mathcal{T}_{\mathcal{A}}}),$$

where $\mathbb{E}(f_{\boldsymbol{\theta}}^{\mathcal{T}_{\mathcal{A}}})$ and $\mathbb{E}(f_{\boldsymbol{\theta}}^{\mathcal{T}'_{\mathcal{A}_{new}}})$ represent the empirical errors of the machine learning model trained on $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{T}'_{\mathcal{A}_{new}}$, respectively.

5.1 Generation-based Table Augmentation

An effective approach to improving tabular datasets is *feature engineering*, which involves augmenting existing features to create more relevant inputs for machine learning models. This process often leads to substantial performance enhancements [88].

We introduce SMARTFEAT [46], a system that leverages LLMs to automate the feature engineering pipeline. Compared with traditional rule-based automated feature engineering tools, SMARTFEAT leverages the reasoning capabilities of LLMs to search for meaningful and interpretable features. Furthermore, instead of generating new feature values through row-by-row LLM executions, which could be computationally expensive, SMARTFEAT identifies promising features upfront and then synthesizes the transformation code to compute these features efficiently at scale. SMARTFEAT operates through an iterative search process, progressively refining and expanding the feature set. As shown in Figure 3, the system consists of two main components:

Table 1: Comparison of average AUC values (↑) for different ML models: SMARTFEAT vs. baseline methods.

Methods	Diabetes	Heart	Bank	Adult	Housing	West Nile Virus	Tennis
Initial AUC	82.20	67.38	91.46	76.81	86.72	78.96	77.93
SMARTFEAT	**86.76 (+4.3%)**	**72.15 (+7.0%)**	91.47 (≈)	**87.00 (+13.3%)**	**92.19 (+6.3%)**	**82.12 (+4.0%)**	87.39 (+9.5%)
CAAFE	-	69.67 (+3.4%)	**91.73 (+0.3%)**	83.10 (+8.2%)	92.15 (+6.3%)	80.11 (+1.8%)	**88.50 (+13.6%)**
Featuretools	82.24 (≈)	66.78 (-0.9%)	91.04 (-0.5%)	73.85 (-3.9%)	79.47 (-8.1%)	73.12 (-7.4%)	81.29 (+4.3%)
AutoFeat	75.24 (-8.4%)	64.92 (-3.7%)	-	-	77.63 (-10.5%)	70.90 (-10.2%)	71.73 (-8.0%)

- *Operator selector* ①: This component takes as input (a) dataset feature descriptions, (b) the prediction task (e.g., classification), and (c) the downstream machine learning model. It applies operator-guided feature generation using various operator prompt templates, such as unary, binary, group-by–aggregate, and extractor operators. The operator selector interacts with the LLM to determine suitable operators to apply and outputs the *name of the new feature*, the *relevant columns* for computing the new feature, and a *descriptive explanation* of the feature.

- *Function generator* ②: Based on the outputs of the operator selector, this component generates an executable transformation function. The function is applied to the original dataset to compute the values of the new feature, and the augmented dataset and feature descriptions are updated for further iterations. Including detailed feature descriptions during this process is highly beneficial in guiding function generation. When no suitable function can be derived (e.g., extracting the capital city for each country), SMARTFEAT resorts to row-level LLM generation to obtain feature values, leveraging the common knowledge and reasoning abilities of LLMs.

In each iteration, the operator selector first chooses a semantically meaningful operator, and then the function generator obtains the transformation to compute feature values. For example, as shown in Figure 4, given the ML prediction task, model selection, and feature description, the operator selector chooses a unary operator to bucketize the `Age` column. It generates a new feature name (`Bucketized_age`), a description (e.g., "`Bucketization of Age attribute`"), and identifies the relevant column(s). The function generator then translates this information into executable code, applies the transformation to the dataset, and adds the resulting feature to the feature set.

The feature generation process begins by exploring potential unary operators on the original features. Using a prompt template, the operator selector iterates over each original feature, prompting LLMs to propose potentially beneficial unary operations (e.g., bucketization or scaling). Once an operator is selected, the function generator retrieves the corresponding transformation function and applies it to the dataset. Building on the original and unary features, SMARTFEAT prompts LLMs to suggest binary and group-by-aggregate operators that may further enhance the data set. Finally, the process considers extractors that can operate on multiple inputs to generate additional features.

In addition to SMARTFEAT, another feature engineering tool, CAAFE [31], also uses LLMs to produce Python code for feature engineering. Unlike SMARTFEAT, which utilizes a pre-defined operator-guided search for new features, CAAFE employs chain-of-thought instructions [79] to guide a series of intermediate steps to generate new features.

Lastly, we compare the performance of SMARTFEAT and CAAFE against traditional automated feature engineering tools based on *expansion-selection* methods [37]: Featuretools [36], which exhaustively generates features using predefined operators and applies feature selection, and AutoFEAT [32], which constructs a large set of nonlinear features followed by a search algorithm to select an effective subset. Using these tools, we processed seven datasets from *Kaggle*, performed classification, and evaluated the Area Under the ROC Curve (AUC) as the primary performance metric across four ML classification

models: *linear regression*, *GaussianNB*, *random forest*, and *extra tree*. The average AUC of the four models is presented in Table 1.

The results show that the LLM-based approaches significantly outperform traditional methods. SMARTFEAT enhances the original AUC score by up to 13.3% on the *Adult* dataset, while CAAFE improves by 8.2%. We observe that CAAFE recommends a smaller set of features, and the operator-guided search in SMARTFEAT generates a more comprehensive feature set. The new features generated by Featuretools and AutoFEAT are agnostic to the dataset context and prediction task, and thus exhibit comparatively lower performance. However, this evaluation is limited to publicly available datasets, which LLMs might have encountered during training, potentially leading to overly optimistic results. Evaluating the performance of LLM-assisted feature engineering on private datasets remains a topic for future exploration.

5.2 Retrieval-based Table Augmentation

When generating additional information directly from the original table is insufficient, an alternative approach to data augmentation involves performing dataset searches and utilizing external text to enrich the data [14].

LLMs have demonstrated promising potential in identifying joinable [34] and unionable [20, 33] structured data, facilitating data discovery of related tables to broaden available information. Additionally, unstructured sources such as Wikipedia can be harnessed by performing entity linking [69] and extracting relevant content. For instance, FeSTE [27] combines web search with fine-tuned BERT model to supplement data with Wikipedia-derived information, demonstrating the potential of LLMs for retrieval-based augmentation.

6 Concluding Remarks

In this paper, we explored three essential stages of the data engineering workflow: data wrangling, analytical querying, and table augmentation for machine learning, which often require significant effort from data engineers. Traditional automated approaches, which rely on rule-based algorithms or machine learning models, can struggle with complex scenarios or demand significant human involvement for training dataset collection. Recently, large language models (LLMs) have shown growing promise in automating these tasks due to their extensive training data corpora and cross-task generality. LLMs hold potential to function as pretrained data engineers, facilitating the automation of data engineering workflows, optimizing performance, and expanding the scope of data tasks.

For data wrangling, we formally defined an LLM-based data wrangling operator, which uses task parameters and dataset context to construct prompts that guide LLMs in performing data preparation tasks. We introduced a unified system, UniDM, for LLM-based data wrangling, which automates prompt construction to handle different tasks in the multi-step wrangling process. However, a limitation of UniDM as we discussed is its reliance on row-level execution, which can be inefficient and costly for large datasets. To this end, we encourage ongoing efforts to explore approaches to utilize LLMs more efficiently for data wrangling. Looking ahead, we envision end-to-end data preparation platforms powered by LLMs, where users provide raw data, and necessary operators and optimizations are determined automatically to minimize costs and improve the wrangling quality.

For analytical querying, we are able to extend traditional relational queries by leveraging LLMs' capabilities to query unstructured data and LLM's knowledge via various interfaces (e.g., natural languages). To better interpret users' questions, we presented a Text2SQL framework, DAIL-SQL, which incorporates techniques including prompt engineering and supervised fine-tuning to improve the execution accuracy of translated SQL queries. We also discussed the importance of developing

LLM-powered logical operators to expand the analytical capabilities of databases. We anticipate that future data systems will integrate these operators to harness LLMs' knowledge and reasoning ability. These systems should also support query planning and optimization, and offer both natural-language interfaces and declarative programming interfaces for analytical queries.

To support data processing for machine learning, we demonstrated that tabular data augmentation through generation- or retrieval-based approaches can boost downstream prediction performance. We introduced an automated feature engineering tool, SMARTFEAT, which employs operator-guided search to enable LLMs to generate meaningful and interpretable new features. We also discussed retrieval-based approaches using LLMs for searching and leveraging external text to enrich datasets. In the future, we can utilize LLMs to autonomously plan and execute table augmentation pipelines for generating more meaningful features and accessing additional data, while this may require careful pre- and post-processing to ensure efficiency and reliability.

There are several additional topics we touched upon that are worth further discussion. For example, in this paper, to optimize the performance of LLMs, we discussed approaches based on prompt engineering and supervised fine-tuning, which are also used to explore different avenues: Table-GPT [43] proposes fine-tuning LLMs with real tables to improve their ability to process two-dimensional data structures, while Jellyfish [84] explores instruction tuning to enhance LLMs as universal data processing solutions that better adhere to human instructions. In addition, reinforcement fine-tuning [28] has shown potential, particularly for reasoning-intensive models such as OpenAI's o1-mini, enabling stronger generalization and reducing reliance on large-scale labeled data. More recently, Deepseek-R1 [24], an open-source LLM based on reinforcement learning, has demonstrated strong reasoning capabilities without relying on supervised data. Collectively, these advancements in post-training techniques show significant potential to boost the performance of LLM-assisted data analytical tasks and open up new possibilities for addressing complex data engineering challenges. Moreover, current evaluations are predominantly based on public data sets, which can lead to overly optimistic results if LLMs have seen these datasets during training. Evaluating LLMs on private data lakes and unseen tasks is essential for assessing their generality and robustness.

References

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. Detecting data errors: Where are we and what needs to be done? *Proc. VLDB Endow.*, 9(12):993–1004, 2016.

[2] Mohammad Shahmeer Ahmad, Zan Ahmad Naeem, Mohamed Y. Eltabakh, Mourad Ouzzani, and Nan Tang. Retclean: Retrieval-based data cleaning using foundation models and data lakes. *CoRR*, abs/2303.16909, 2023.

[3] Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernández Ábrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan A. Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vladimir Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, and et al. Palm 2 technical report. *CoRR*, abs/2305.10403, 2023.

[4] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *Proc. VLDB Endow.*, 17(2):92–105, 2023.

[5] Blessing Bamiduro and Anusha Challa. Large language models for sentiment analysis with amazon redshift ml (preview). `https://aws.amazon.com/blogs/big-data/large-language-models/-for-sentiment-analysis-with-amazon-redshift-ml-preview/`.

[6] Felix Bießmann, Tammo Rukat, Philipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. Datawig: Missing value imputation for tables. *J. Mach. Learn. Res.*, 20:175:1–175:6, 2019.

[7] Asim Biswal, Liana Patel, Siddarth Jha, Amog Kamsetty, Shu Liu, Joseph E. Gonzalez, Carlos Guestrin, and Matei Zaharia. Text2sql is not enough: Unifying AI and databases with TAG. *CoRR*, abs/2408.14717, 2024.

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[10] Ursin Brunner and Kurt Stockinger. Entity matching with transformer architectures - A step forward in data integration. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, pages 463–473. OpenProceedings.org, 2020.

[11] Chengliang Chai, Jiayi Wang, Yuyu Luo, Zeping Niu, and Guoliang Li. Data management for machine learning: A survey. *IEEE Trans. Knowl. Data Eng.*, 35(5):4646–4667, 2023.

[12] Shuaichen Chang and Eric Fosler-Lussier. How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings. *CoRR*, abs/2305.11853, 2023.

[13] Zui Chen, Lei Cao, Sam Madden, Ju Fan, Nan Tang, Zihui Gu, Zeyuan Shang, Chunwei Liu, Michael J. Cafarella, and Tim Kraska. SEED: simple, efficient, and effective data management via large language models. *CoRR*, abs/2310.00749, 2023.

[14] Lingxi Cui, Huan Li, Ke Chen, Lidan Shou, and Gang Chen. Tabular data augmentation for machine learning: Progress and prospects of embracing generative AI. *CoRR*, abs/2407.21523, 2024.

[15] Databricks. Ai functions on databricks. `https://docs.databricks.com/en/index.html`.

[16] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Trans. Neural Networks Learn. Syst.*, 28(3):653–664, 2017.

[17] AnHai Doan and Alon Y. Halevy. Semantic integration research in the database community: A brief survey. *AI Mag.*, 26(1):83–94, 2005.

[18] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. A survey for in-context learning. *CoRR*, abs/2301.00234, 2023.

[19] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[20] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. *Proc. VLDB Endow.*, 16(7):1726–1739, 2023.

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

[22] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145, 2024.

[23] Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. Xiyan-sql: A multi-generator ensemble framework for text-to-sql. *CoRR*, abs/2411.08599, 2024.

[24] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[25] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: retrieval-augmented language model pre-training. *CoRR*, abs/2002.08909, 2020.

[26] Mazhar Hameed and Felix Naumann. Data preparation: A survey of commercial tools. *SIGMOD Rec.*, 49(3):18–29, 2020.

[27] Asaf Harari and Gilad Katz. Few-shot tabular data enrichment using fine-tuned transformer architectures. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 1577–1591. Association for Computational Linguistics, 2022.

[28] Hesam Sheikh Hassani. What is openai's reinforcement fine-tuning? `https://www.datacamp.com/blog/reinforcement-fine-tuning`.

[29] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek R. Narasayya, and Surajit Chaudhuri. Transform-data-by-example (TDE): an extensible search engine for data transformations. *Proc. VLDB Endow.*, 11(10):1165–1177, 2018.

[30] Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 829–846. ACM, 2019.

[31] Noah Hollmann, Samuel Müller, and Frank Hutter. Llms for semi-automated data science: Introducing CAAFE for context-aware automated feature engineering. *CoRR*, abs/2305.03403, 2023.

[32] Franziska Horn, Robert Pack, and Michael Rieger. The autofeat python library for automated feature engineering and selection. In *Machine Learning and Knowledge Discovery in Databases - International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part I*, volume 1167 of *Communications in Computer and Information Science*, pages 111–120. Springer, 2019.

[33] Xuming Hu, Shen Wang, Xiao Qin, Chuan Lei, Zhengyuan Shen, Christos Faloutsos, Asterios Katsifodimos, George Karypis, Lijie Wen, and Philip S. Yu. Automatic table union search with tabular representation learning. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 3786–3800. Association for Computational Linguistics, 2023.

[34] Zezhou Huang, Jiaxiang Liu, Haonan Wang, and Eugene Wu. The fast and the private: Task-based dataset search. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*. www.cidrdb.org, 2024.

[35] Zhongjun Jin, Michael R. Anderson, Michael J. Cafarella, and H. V. Jagadish. Foofah: Transforming data by example. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 683–698. ACM, 2017.

[36] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Campus des Cordeliers, Paris, France, October 19-21, 2015*, pages 1–10. IEEE, 2015.

[37] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 979–984. IEEE Computer Society, 2016.

[38] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. CHORUS: foundation models for unified data discovery and exploration. *Proc. VLDB Endow.*, 17(8):2104–2114, 2024.

[39] Konstantina Kourou, Themis P Exarchos, Konstantinos P Exarchos, Michalis V Karamouzis, and Dimitrios I Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, 13:8–17, 2015.

[40] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[41] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. The dawn of natural language to SQL: are we fully ready? [experiment, analysis & benchmark]. *Proc. VLDB Endow.*, 17(11):3318–3331, 2024.

[42] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[43] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. Table-gpt: Table fine-tuned GPT for diverse table tasks. *Proc. ACM Manag. Data*, 2(3):176, 2024.

[44] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep entity matching with pre-trained language models. *Proc. VLDB Endow.*, 14(1):50–60, 2020.

[45] Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, Sepanta Zeighami, Aditya G. Parameswaran, and Eugene Wu. Towards accurate and efficient document analytics with large language models. *CoRR*, abs/2405.04674, 2024.

[46] Yin Lin, Bolin Ding, H. V. Jagadish, and Jingren Zhou. SMARTFEAT: efficient feature construction through feature-level foundation model interactions. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*. www.cidrdb.org, 2024.

[47] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *CoRR*, abs/2303.13547, 2023.

[48] Shicheng Liu, Jialiang Xu, Wesley Tjangnaka, Sina J. Semnani, Chen Jie Yu, and Monica Lam. SUQL: conversational search over structured and unstructured data with large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 4535–4555. Association for Computational Linguistics, 2024.

[49] Samuel Madden, Michael J. Cafarella, Michael J. Franklin, and Tim Kraska. Databases unbound: Querying all of the world's bytes with AI. *Proc. VLDB Endow.*, 17(12):4546–4554, 2024.

[50] Steve McDowell. Komprise unleashes fresh insights about your unstructured data. `forbes.com/sites/stevemcdowell/2023/03/09/komprise-unleashes-fresh-insights-about-your-unstructured-data/`.

[51] Yinan Mei, Shaoxu Song, Chenguang Fang, Haifeng Yang, Jingyun Fang, and Jiang Long. Capturing semantics for imputation with pre-trained language models. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 61–72. IEEE, 2021.

[52] Xupeng Miao, Zhihao Jia, and Bin Cui. Demystifying data management for large language models. In Pablo Barceló, Nayat Sánchez-Pi, Alexandra Meliou, and S. Sudarshan, editors, *Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago AA, Chile, June 9-15, 2024*, pages 547–555. ACM, 2024.

[53] Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies. *CoRR*, abs/2305.12586, 2023.

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

[55] Nhan Nguyen and Sarah Nadi. An empirical evaluation of github copilot's code suggestions. In *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*, pages 1–5. ACM, 2022.

[56] OpenAI. Sql translate. `https://platform.openai.com/examples/default-sql-translate?spm=teamfile.libs.0.0.affb3f1dDq3ukY`.

[57] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

[58] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. Semantic operators: A declarative model for rich, ai-based analytics over text data. *CoRR*, abs/2407.11418, 2024.

[59] Norman W. Paton, Jiaoyan Chen, and Zhenyu Wu. Dataset discovery and exploration: A survey. *ACM Comput. Surv.*, 56(4):102:1–102:37, 2024.

[60] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2463–2473. Association for Computational Linguistics, 2019.

[61] Yichen Qian, Yongyi He, Rong Zhu, Jintao Huang, Zhijian Ma, Haibin Wang, Yaohua Wang, Xiuyu Sun, Defu Lian, Bolin Ding, and Jingren Zhou. Unidm: A unified framework for data manipulation with large language models. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org, 2024.

[62] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. Evaluating the text-to-sql capabilities of large language models. *CoRR*, abs/2204.00498, 2022.

[63] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, 10(11):1190–1201, 2017.

[64] Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X. Sean Wang. PURPLE: making a large language model a better SQL writer. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*, pages 15–28. IEEE, 2024.

[65] El Kindi Rezig, Lei Cao, Michael Stonebraker, Giovanni Simonini, Wenbo Tao, Samuel Madden, Mourad Ouzzani, Nan Tang, and Ahmed K. Elmagarmid. Data civilizer 2.0: A holistic framework for data preparation and analytics. *Proc. VLDB Endow.*, 12(12):1954–1957, 2019.

[66] Rashida Richardson, Jason M Schultz, and Kate Crawford. Dirty data, bad predictions: How civil rights violations impact police data, predictive policing systems, and justice. *NYUL Rev. Online*, 94:15, 2019.

[67] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950, 2023.

[68] Mohammed Saeed, Nicola De Cao, and Paolo Papotti. Querying large language models with SQL. In *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*, pages 365–372. OpenProceedings.org, 2024.

[69] Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Trans. Knowl. Data Eng.*, 27(2):443–460, 2015.

[70] Abirami Sukumaran. Llm with vertex ai only using sql queries in bigquery. `https://cloud.google.com/blog/products/ai-machine-learning/llm-with-vertex-ai-only-using-sql-queries-in-bigquery`.

[71] Edhy Sutanta, Retantyo Wardoyo, Khabib Mustofa, and Edi Winarko. Survey: Models and prototypes of schema matching. *International Journal of Electrical and Computer Engineering (IJECE)*, 6(3):1011–1022, 2016.

[72] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Samuel Madden, and Mourad Ouzzani. RPT: relational pre-trained transformer is almost all you need towards democratizing data preparation. *Proc. VLDB Endow.*, 14(8):1254–1261, 2021.

[73] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[74] The Vicuna Team. Vicuna: An open-source chatbot impressing gpt-4 with 90`https://lmsys.org/blog/2023-03-30-vicuna/`.

[75] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[76] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023.

[77] Immanuel Trummer. From BERT to GPT-3 codex: Harnessing the potential of very large language models for data management. *Proc. VLDB Endow.*, 15(12):3770–3773, 2022.

[78] Pengfei Wang, Xiaocan Zeng, Lu Chen, Fan Ye, Yuren Mao, Junhao Zhu, and Yunjun Gao. Promptem: Prompt-tuning for low-resource generalized entity matching. *Proc. VLDB Endow.*, 16(2):369–378, 2022.

[79] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[80] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[81] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[82] Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander R. Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter S. Lasecki, and Dragomir R. Radev. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 1962–1979. Association for Computational Linguistics, 2019.

[83] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 3911–3921. Association for Computational Linguistics, 2018.

[84] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. Jellyfish: A large language model for data preprocessing. *CoRR*, abs/2312.01678, 2023.

[85] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. Large language models as data preprocessors. In *Proceedings of Workshops at the 50th International Conference on Very Large Data Bases, VLDB 2024, Guangzhou, China, August 26-30, 2024*. VLDB.org, 2024.

[86] Fuheng Zhao, Divyakant Agrawal, and Amr El Abbadi. Hybrid querying over relational databases and large language models. *CoRR*, abs/2408.00884, 2024.

[87] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[88] Marc-André Zöller and Marco F. Huber. Benchmark and survey of automated machine learning frameworks. *J. Artif. Intell. Res.*, 70:409–472, 2021.

LLM-Powered Proactive Data Systems

Sepanta Zeighami, Yiming Lin, Shreya Shankar, Aditya Parameswaran
UC Berkeley
{zeighami, yiminglin, shreyashankar, adityagp}@berkeley.edu

Abstract

With the power of LLMs, we now have the ability to query data that was previously impossible to query, including text, images, and video. However, despite this enormous potential, most present-day data systems that leverage LLMs are reactive, reflecting our community's desire to map LLMs to known abstractions. Most data systems treat LLMs as an opaque black box that operates on user inputs and data as is, optimizing them much like any other approximate, expensive UDFs, in conjunction with other relational operators. Such data systems do as they are told, but fail to understand and leverage what the LLM is being asked to do (i.e. the underlying operations, which may be error-prone), the data the LLM is operating on (e.g., long, complex documents), or what the user really needs. They don't take advantage of the characteristics of the operations and/or the data at hand, or ensure correctness of results when there are imprecisions and ambiguities. We argue that data systems instead need to be *proactive*: they need to be given more agency—armed with the power of LLMs—to understand and rework the user inputs and the data and to make decisions on how the operations and the data should be represented and processed. By allowing the data system to parse, rewrite, and decompose user inputs and data, or to interact with the user in ways that go beyond the standard single-shot query-result paradigm, the data system is able to address user needs more efficiently and effectively. These new capabilities lead to a rich design space where the data system takes more initiative: they are empowered to perform optimization based on the transformation operations, data characteristics, and user intent. We discuss various successful examples of how this framework has been and can be applied in real-world tasks, and present future directions for this ambitious research agenda.

1 Introduction

The database community has long acknowledged the need to store, process, and query data in various degrees of structure, from relational, to semi-structured, and more recently, to unstructured data, including text, images, and video. Recent developments in AI models, and LLMs in particular, have unlocked the ability to better process and make sense of unstructured data, in addition to structured data, for tasks including information extraction [1, 2], summarization [3], data cleaning [4], dataset search [5], and data integration [6]. LLMs also enable us to better understand users, manifesting in rapid progress in benchmarks on translating natural language queries into SQL [7, 8]. LLMs truly have the potential to disrupt our entire field [9].

All of this progress in harnessing LLMs for data management—for processing both unstructured and structured data—is valuable. However, our belief is that we are still not leveraging the full potential of LLMs for data management. In most data systems that leverage LLMs for data processing, including those proposed in recent work [2, 10, 11], LLM operations are treated as a given, i.e., as black-box invocations on monolithic user inputs and data, where, akin to other types of UDFs, the data system doesn't attempt to fully understand the underlying data, user intent, or constituent operations, and just does as they are told. We call such data systems **reactive**, in that *they passively execute user-specified*

operations, without understanding the underlying intent, the semantics of the operations, or the data on which it should be applied. Reactive data systems are fundamentally limited in their ability to accurately and efficiently address user needs. If the LLM operations as expressed in the user query have low accuracy or throw an error, reactive data systems will faithfully pass the burden of low accuracy or errors back to the user, without attempting to proactively correct for this. To understand the limitations of reactive database systems, consider the following example.

Example 7: *(Police Misconduct)* At UC Berkeley, we are co-leading an effort, along with journalists and public defenders, to build a state-wide police misconduct and use-of-force database*. As part of this effort, our collaborators have gathered, through public records requests, millions of documents detailing incidents across 700 agencies. Each incident can be split across multiple files, and can name several officers. Each file itself can have many sub-documents, including officer testimonies, medical examiner reports, eyewitness reports, and internal affairs determinations. The officers themselves may be part of several incidents. Journalists and public defenders are interested in both investigating the behavior of individual officers, as well as broader systemic patterns, all in an effort to ensure greater accountability. In such a setting, a reactive data system would encounter various difficulties, as follows:

- *(Difficulties with the Data).* Suppose a journalist is interested in understanding the medical impacts of use of force. This is typically detailed in the medical examiner report within the broader use-of-force document. Simply providing the LLM the entire document as is (often hundreds of pages) can lead to the LLM making errors [12]; instead, by decomposing the document into specific semantically meaningful portions and focusing LLM attention on those portions can both improve accuracy and reduce cost. Another alternative is RAG (Retrieval-Augmented Generation) on pages or chunks, but RAG once again doesn't try to proactively identify the meaning of the documents, or chunks, leading to low accuracies. Here, *treating unstructured data as a black box monolith*, as in present-day reactive systems, is problematic.

- *(Difficulties with the Operations).* The journalists have identified dozens of fields of interest in the incidents, including, but not limited to: dates, people mentioned, locations, use of firearms, drug use, use of batons and K9 units, among others. Some fields are dependent on other fields, e.g., whether there was disciplinary action is contingent on whether there was an internal affairs investigation. Simply specifying all of the fields to be extracted as is in a single prompt (as a map operation or equivalently, a projection) can lead to the LLM making errors on some of them; instead, by decomposing this operation into smaller "well-scoped" operations, we can ensure greater accuracy of LLM outputs. Here, *treating the operations as a black box*, without understanding their semantics, as in present-day reactive systems, is problematic.

- *(Difficulties with the User Intent).* Suppose a journalist is interested in exploring the documents for mentions of a specific officer, "John Smith". While a reactive data system would faithfully return mentions of John Smith, if any, it would omit mentions of officers where the first name is an initial, i.e., "J. Smith", as well as mentions where the middle initial is present, e.g, "John M. Smith". One could certainly change the query by requiring a semantic match instead of an exact match—but the journalist would have no way of knowing that such mentions exist in the first place. A better approach would be to provide, as feedback to the journalist, what the query does not currently cover (but could), so that they can make a more informed choice about what it is they are actually after. Here, *treating the user intent as given*, as is done in present-day reactive data systems, is problematic.

*https://bids.berkeley.edu/california-police-records-access-project

In all three instances, we find that present-day data systems, especially those that harness LLMs to help make sense of unstructured data, are reactive: they treat the data, user query, and operations as black-box indivisible monoliths. Instead, we argue that data systems should be *proactive*: rather than treating LLM invocations on data as a given, such data systems should *posess the agency to understand user intent, transformation operations, and the underlying data*—and to make decisions on how to best reconfigure the data, operations, and user input to suit the analysis need. In the example above, this may include, for example, uncovering underlying layout or patterns in unstructured documents, decomposing (or fusing) operations into semantically equivalent but more accurate ones, or going above and beyond immediate user input to determine the actual underlying user intent, in concert with the user. We argue for *truly harnessing the power of LLMs, to understand and make sense of both structured and unstructured data, rather than simply treating them as black box unstructured data processors*. We believe the three axes of understanding (1) user intent, (2) data operations and (3) the data itself are key to the data systems' ability to accurately and efficiently processes structured and unstructured data.

In the following, we will put forth our vision for *proactive data systems—systems that more effectively harness LLMs for structured and unstructured data processing by improving our understanding of data, intent, and operations.* While our vision is ambitious and expansive, our early work has already shown promise:

- Our work has shown how understanding unstructured document collections better, especially those that obey similar templates, can pay rich dividends in both cost and accuracy for document processing [1, 13].
- Our work has shown how a better understanding of error-prone LLM-based unstructured data processing operators, as well as the ability to decompose or rewrite these operators can lead to data processing pipelines that are a lot more accurate [14, 15].
- Our work has also shown that tailoring our responses to the underlying user intent, especially as part of a dialog with the user, rather than just strictly adhering to the user request as stated, can be very helpful, as evidenced in tasks that range from data visualization to dataset search [16–18].

Our experience is grounded in our police misconduct analysis application, as well as our other work in understanding where and how LLMs go wrong, and how we may be able to avoid these mistakes [19, 20].

By moving beyond simply executing user instructions "as is" and treating LLM invocations as a black box, the effective offline and online execution space of proactive data systems is effectively unbounded and open-ended. For example, a user task for extracting information from documents can be decomposed in a potentially unbounded number of different ways, with different accuracies. Simply leveraging an LLM to, in turn, do this query planning and optimization for us can lead to suboptimal results. Instead, in this vision paper, we discuss various recipes for proactive systems to help make sense of each of our three axes of data, intent, and operations, such as performing decompositions and rewrites to improve accuracy when performing a given operation, finding structure in unstructured data to better understand the data and answer queries on it, and by adjusting data and query representations based on user feedback to better align with user intent. We discuss future directions along each axis to build better proactive database systems.

2 Typical User Workflows with Proactive Data Systems

A proactive data system understands, at a deeper semantic level, user intent, the specific operations it performs, and the data it performs operations on. Typical user workflows with such a system are similar to traditional database systems, where the user first provides (or ingests) the data, potentially alongside

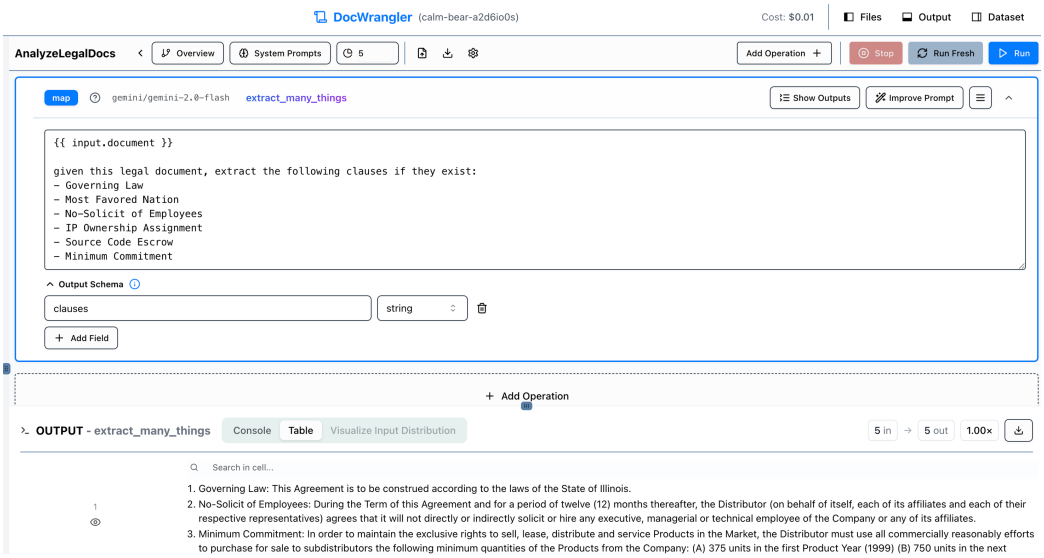

Figure 1: A potential interface for the data system. The user provides a task to be performed on a set of documents, here a collection of legal documents.

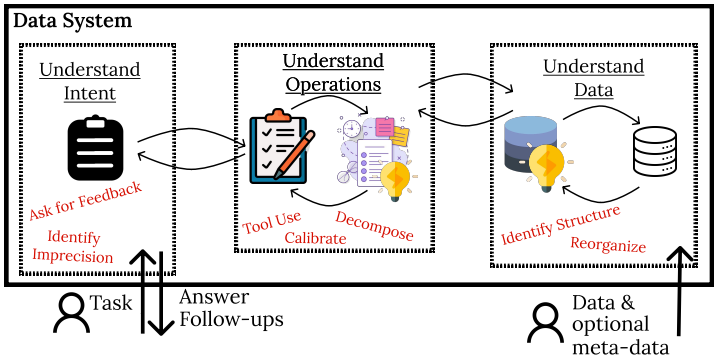

Figure 2: A Proactive Data System (red = LLM-powered)

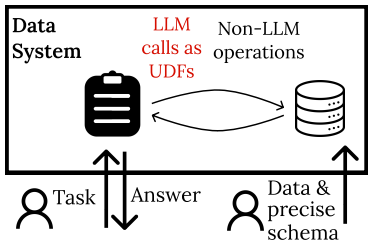

Figure 3: A Reactive Data System

additional descriptive information about its content or schema, and then proceeds to execute queries on this data. We describe this workflow in more detail next, while acknowledging that a range of design choices may all be appropriate, depending on the use cases.

Data Definition. The user first ingests or registers their data (e.g., a PDF document collection of incident reports, police officer employment CSVs, as well as audio/video of incidents in the police misconduct case), along with metadata if any. Unlike traditional database systems, this metadata is optional, and the system is free to proactively understand the structure and semantics of the data. That said, any user-provided specification or description can help improve accuracy and better specialize the system for specific use-cases of interest. In Example 7, such a specification can be similar to the first paragraph of Example 7 as plain text, or could include definitions about technical terms or additional background providing domain knowledge, e.g., defining police misconduct. We note that such information is often needed to allow users to query the data meaningfully even in relational databases when using text-to-SQL [21]. We also envision that in certain use-cases, the users may proactively identify the entities of interest for downstream querying, even if they don't register the attributes they may care about in the future. For example, in the police misconduct setting, the users may want to indicate that they intend to analyze information about incidents, police officers, and agencies. Armed with all of this

information, the system can proactively add indexes, reorganize the data, and extract certain fields, among other such offline actions, as we will describe in Section 4.

Task Specification. Users are then free to issue queries or tasks on the data, which can be specified in natural language, or a combination of natural language prompts associated with data processing operators, as shown in Figure 1 for a map operation on a collection of contracts, with a prompt extracting a number of legal clauses, within DocWrangler, our IDE for DocETL [14]. If the user instead chose to preregister a schema during the data definition stage, they are free to extend it with additional LLM-populated attributes and issue SQL queries based on these lazily populated attributes, as we do in ZenDB [1].

Task Execution. The system then performs the task on the underlying data. Rather than performing the task as is on predefined data monoliths, a proactive data system will try to understand the task and the data, performing reformulations of this task by decomposing the corresponding operators, as well as the data, all in an effort to maximize accuracy and minimize cost. For example, the system can decompose the user-provided task into smaller easier-to-do operations, and can leverage the semantics of the document(s) to intelligently focus the LLM's attention on relevant portions.

In Section 3 we discuss various ways for a proactive data system to understand user-defined operations and reformulate them, and we extend the discussion in Section 4 on how the system can similarly understand the data and perform data transformations to improve accuracy. In Section 5, we discuss how the data system can ensure the task was performed as the user intended. When it comes to unstructured data, we focus our attention mostly on documents for concreteness, though our general approach may be applicable to a variety of formats.

To further differentiate a proactive data system from a reactive one, Figs. 2 and 3 provide a high-level overview of different components in these systems. While a reactive data system considers tasks and data as is, and performs operations on the components as instructed, a proactive data system leverages LLMs to understand user intent, the operations it performs, as well as the data to ensure maximal accuracy at minimum cost.

3 Proactive Operation Understanding

In reactive data systems, the onus is on the user to author queries involving the "right" LLM-powered operators, with the system then determining how to execute these in conjunction with other relational operators. However, even in cases where users are able to specify clear, unambiguous operations, the granularity at which they specify them may not be optimal for execution. Fundamentally, this stems from a lack of understanding of what LLMs can do well versus what they can't—something most users are not aware of. In this section, we discuss various approaches to operation reformulation that can improve accuracy while maintaining or reducing computational costs. We first describe new operators that we may introduce, and then methods for assessing and improving cost and accuracy when leveraging new or existing operators.

3.1 How and Where to Introduce New Operators

We now describe ways to reformulate existing LLM-powered operations into different ones.

Decomposition into simpler LLM operations. In Example 7, we described how sometimes journalists want to extract dozens of fields from a given document (e.g., police officer names, descriptions of misconduct, locations, use of firearms, among others). The journalist may specify the entire list of fields along with instructions within a prompt. However, executing this as is in a single LLM call may lead to poor accuracies as LLMs often struggle to identify multiple concepts simultaneously [14]. A

proactive system can decompose such an operation into separate focused ones that each extract one type of information, improving accuracy. An LLM can be asked rewrite a prompt that says "extract fields f_1, ..., f_n from the following ..." into "extract field f_i from the following ...". In prior work, we have identified several new decomposition-based rewrite rules [14] that can lead to higher accuracies, when coupled with LLMs being used to instantiate the rewrites themselves. Recent work on Text-2-SQL also leverages similar ideas [22]: rather than attempting translation in "one-shot" (i.e., single LLM call), which often fails due to schema mismatches or poorly-named schemas [23], it is beneficial to parse the operation into smaller units, e.g., given "find all employees who joined before 2020", first identify the core concepts: employee records and hire dates, and treat each separately.

Leveraging non-LLM components. In certain cases, operations that are assigned to an LLM may be better done through other means, e.g., through SQL. For example, finding the average settlement amount for misconduct cases can be decomposed into an LLM operation to identify relevant cases and their settlement amounts, followed by a SQL aggregation to compute the average. We have employed similar techniques where we separate operations that require real-world reasoning (suited for LLMs) from mathematical computations (better handled by traditional database engines or calculators) in our work on ZenDB [1]. Determining how to do this automatically is challenging.

Leveraging reasoning or data feedback for reformulation. As described avove, a proactive data system must reformulate (e.g., decompose or rework) operations to execute them better. However, finding good reformulations is difficult. Current approaches to discovering good reformulations or decompositions are quite naive. Systems like DocETL [14] simply prompt LLMs to suggest rewrites, either taking the first suggestion or selecting from multiple candidates. One option is to use a powerful "reasoning" model like OpenAI's o1 model to rewrite the task, but this still fundamentally relies on one-shot prompting, and is unaware of how the rewrite will actually perform on the data. We need approaches that can learn what characteristics of the data make tasks challenging, what types of LLM errors occur in different contexts, and use this knowledge to guide reformulation—perhaps even in an agentic fashion.

Calibrating LLM outputs. Independent of which decomposition or reformulation is used, when LLMs are independently being applied to a set of items (documents, tuples), the outputs can often be inconsistent and non-calibrated. For example, if we ask LLMs to rate the severity of every incident in our document collection, it often gives all of them the same score, or worse, gives them scores that are only losely correlated with the severity. To remedy this lack of consistency, we can take various actions. We can leverage the LLMs themselves to pick representative examples that indicate the full range of the categories of interest, provided as few-shot examples. Or they can rework the prompt to describe in more detail the criteria used for evaluation—to ensure consistency. Finally, they can also restrict the space of possible outputs (e.g., when LLMs are asked to extract state information from a collection of US addresses, they may extract CA in some cases and California in others). While this doesn't change the semantic meaning of the operation, it can significantly improve LLM accuracy by providing better context and guidance, as has also been explored in work on prompt optimization [24].

3.2 Assessing and Improving Performance with Reformulation

Next, we describe ways to assess the benefits of reformulations, and reduce cost while preserving accuracy.

Leveraging LLMs to assess benefits. One question that naturally emerges when considering decomposing operations into smaller units is how to assess the benefits of such decompositions. While it is a-priori hard to tell whether a decomposition will help, we can run both the non-decomposed and decomposed variants on a sample. LLMs are much better at evaluating outputs than generating them, so LLMs can be used to tell which version performs better. For example, one can employ a "generate-fix

& rewrite-verify" pattern: generate an initial operation formulation, verify its correctness (e.g., through automated checks or LLM verification), and if verification fails, attempt alternative formulations [25]. This pattern, which we also use in DocETL [14], allows a system to systematically explore the space of possible formulations until finding one that passes verification, effectively optimizing for accuracy through trial and refinement. However, doing evaluation in a cost-effective manner remains a challenge.

Deferring to cheaper LLMs for the "easy bits". One concern with decomposition is that it may increase the cost of the overall pipeline, since one LLM call per document may now become multiple. One way to defray the cost is to couple decomposition with cost optimization: for the simpler newly decomposed operations, we can alternatively use cheaper and smaller models to handle them, and only use the more expensive model for the most complicated operations. For example, when we're trying to extract many fields from a police record document, we can use a cheaper model for extraction of locations and dates, while using a more expensive model for harder tasks such as determining the type of misconduct incident. While the idea of cheaper proxy models isn't new [10, 26], here, since the space of decompositions is infinite, and for each decomposition (or sequences thereof), we could use different models and different confidence thresholds, each with different cost-accuracy tradeoffs, the problem becomes a lot more compliacated. Additionally, unlike previous settings which focused primarily on tasks with well-defined accuracy metrics, we now must provide guarantees for open-ended generative operations—where quality is harder to quantify.

Expensive predicate ordering, but with synthesized predicates. For operations that involve subselecting documents based on certain criteria (all expressed together in one prompt), we can leverage existing related work on expensive predicate ordering [27, 28]; however, in our context, we can introduce an arbitrary number of new dependent predicates (that are potentially easier to check and therefore cheaper). For example, instead of using an expensive model to examine each police record document to extract medical impacts to the victims, if any, we can consider cheaper filters that are easier to check, for example, if the document contains any medical information at all. This check could potentially be done by a cheaper model and rule out a substantial fraction of the documents. Similarly, when decomposing a complex filter like "find incidents involving both use of force and drug use" into two filters, one for "use of force" and one for "drug use," the system can evaluate the more selective filter first to minimize expensive LLM calls.

4 Proactive Data Understanding

Proactive data systems take initiative to truly understand the data, rather than simply treating it as inputs to opaque UDF (here LLM) calls. It can leverage the provided data descriptions, as well as actual content, to create representations that are useful for downstream data processing tasks. The system can understand each document on its own (Section 4.1), understand relationships between documents or portions thereof (Section 4.2), or preprocess documents based on anticipated future tasks (Section 4.3).

4.1 Identifying Semantic Structure within a Document

Although documents may appear unstructured, they often are semantically structured. This structure may be implicit in the text, e.g., content in adjoining portions of the text is often related. They can also be explicit, e.g., tables or figures embedded within a PDF document. We discuss how to identify, extract, and leverage hidden structure from unstructured documents.

Leveraging implicit hierarchical structure. Portions of documents are often semantically related. A section or subsection within a document often contains information that is semantically related, while other parts are less related or unrelated. For example, the medical examiner report within a

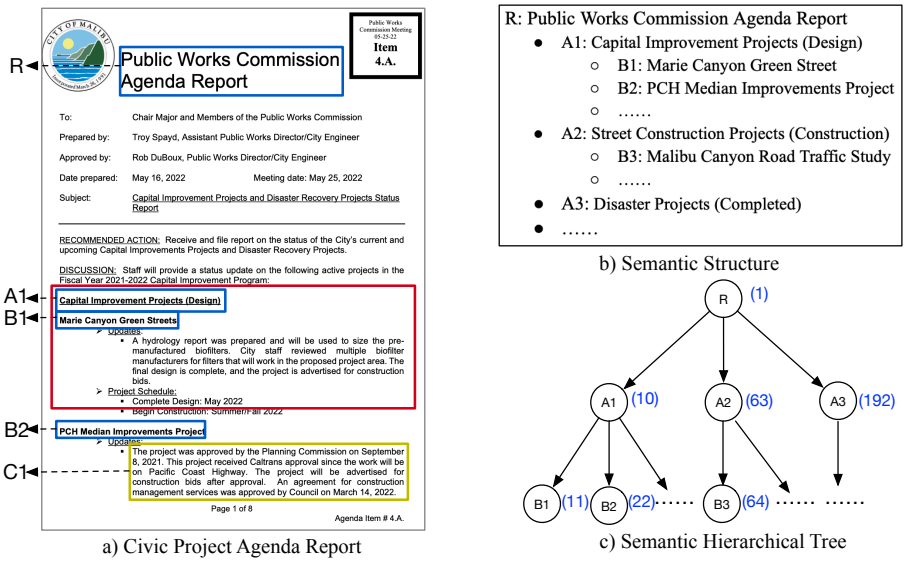

Figure 4: Semantic Hierarchy in a Civic Agenda Document (a) the Document itself (b) the Corresponding "Table of Contents" (c) the Corresponding Semantic Hierarchy.

broader police record PDF contains most of the medically relevant information about an incident, while the eyewitness report contains most of the relevant information from eyewitnesses. Identifying these subdivisions within a document and routing a query to the subdivision at the "right" granularity can lead to higher accuracy than both RAG or providing the entire document to an LLM [1]. This structure is best represented as a semantic hierarchy. There are various ways to construct such a hierarchy, including leveraging formatting information that distinguishes headers from other portions, or using an LLM to identify which phrases may be headers as we do in ZenDB [1]—see Figure 4 for an example. Another approach is to construct this semantic hierarchy on content alone, where summaries of related chunks are merged and recursively summarized [29]. Nonetheless, building semantic structures that are useful for downstream tasks remains a challenge, as different views of the document may be useful for different tasks, where even simple information such as location can have different connotations. For instance, when organizing police activities in a specific case based on location they occurred in, a user might be interested in geographical location of activities (i.e., at a specific address) while another user might be interested in types of locations (e.g., if the police activity was outdoor or inside). The system needs to consider various possible semantics of the data when identifying the semantic structure.

Leveraging explicit structure. Unstructured documents often contain structured portions, such as embedded tables and key-value pairs. Treating them as plain text for data processing is ineffective and error-prone. For example, if we're not careful in preserving visual information, a missing value in a key-value pair could lead to the next key being misinterpreted as the corresponding value. Moreover, depending on the approach used to query such tables, we may lose visual information (used to show table structure and group columns and rows), and be unable to effectively process numerical information. A proactive data system therefore will identify and extract such structured portions and represent them in a structured format, for example, as tabular data or key-value pairs in Figure 5, while preserving their context within the document (e.g., their location and semantic relationships to the rest of the document). Our recent tool, TWIX [13], proposes an efficient approach for automatically extracting structured portions from documents, using a combination of visual and LLM-based inference, while preserving this context for the extracted information. However, many challenges remain, such as accurately representing the semantic relationship between structured and unstructured document portions, e.g., to understand which queries should be answered based on the structured and unstructured portions, and how much background context is necessary to make sense of the structured portions.

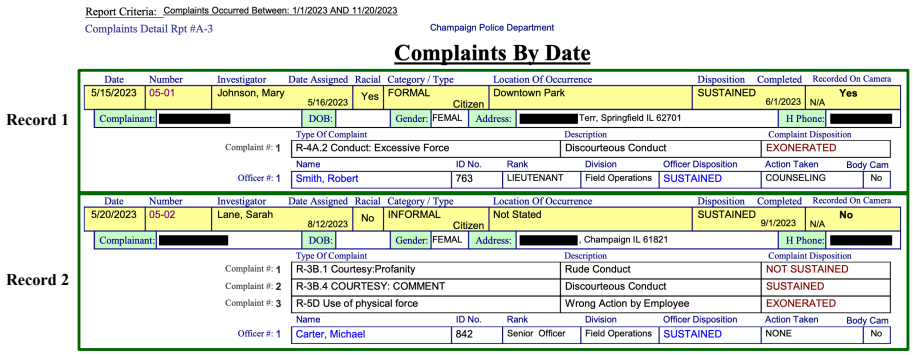

Figure 5: Tables and Key-Value Pairs in Use of Force Records.

4.2 Identifying Cross-Document Relationships

There are multiple reasons to perform cross-document organization.

Identifying documents that may be queried together. Beyond understanding structure within a single document, it is important to understand relationships across documents, since these related documents may often be queried together. In Example 7, the dataset, a single incident can span several PDF documents, often without such information being linked to each other. The data system needs to proactively identify relationships between such documents to organize the data prior to querying. This, for instance, can be done by clustering the documents. However, clustering is challenging, since the system needs to understand the documents to be able to cluster them properly. Simply embedding the documents, and clustering the embeddings does not work since the documents can vary considerably in length. Another approach is to leverage LLMs to check if two documents correspond to the same incident, but this is expensive, especially when there are $O(n^2)$ comparisons. We may be able to leverage LLMs to identify cheaper proxies or blocking rules (e.g., two documents may not be related unless the date ranges overlap) for this organization. In some settings, folder organization provides cues for identifying cross-document relationships (e.g., documents that are very "far apart" from a folder structure standpoint may be unlikely to be related).

Identifying shared templates across documents. A separate concern is to combine semantic hierarchy construction with cross-document relationships, so that we are able to identify shared "templates" across documents. These templates can both help scale up extraction across documents, but also help identify documents whose structure differs considerably. For example, journalists may want to identify incidents where there is an internal affairs report within a broader police record document, since these are ones where there is a corresponding disciplinary action.

4.3 Task-Aware Data Pre-Processing

The system can attempt to proactively find and organize portions of the data that will be useful to improve performance on a reasonable subset of data processing tasks downstream. Given that document collections can span in the millions, it can be expensive to do extensive processing of the data upfront, for instance, by populating a materialized view with all the attributes a user can ever hope to query; it can also be time-consuming to leave all the data processing to when the user issues a task. As such the system needs to decide how much preprocessing is beneficial upfront, and what to perform at query time. To strike a balance between the two extremes, one option is for the system to identify and/or extract data units that it deems to be useful in the future for a wide variety of queries. This can be done by understanding the semantics of the data. For instance, in Example 7, the system can decide that sections that describe police incidents at a high level (e.g., the internal affairs report) are typically useful for future task processing as they provide a comprehensive summary of most relevant aspects. The system can keep pointers to such sections as lightweight indexes, but leave more specific data processing

to when the user issues queries. Similarly, the system can do schema identification in advance to find what type of data is represented in the documents, and use the identified schema to answer queries. The system can decide whether to extract information upfront to populate the schema, or keep pointers to where the information can be found at query time. For instance, the system may choose to retain pointers to all portions that mention police officers in the document so as to accelerate analysis of those aspects downstream, without going all the way to populating a materialized view with officer attributes (since these can vary depending on user need).

5 Proactive User Understanding

Even with best-effort operation reformulation and data understanding capabilities, a fundamental challenge remains: the gap between what users specify and what they actually need. This challenge manifests in multiple ways—users may provide ambiguous specifications, fail to articulate implicit assumptions, or simply not know how to express their requirements fully [20, 30]. To bridge this gap between the user's intent and the operations performed, a proactive data system needs to be internally aware of this gap when executing the task (Section 5.1), leverage user feedback to bridge the gap (Section 5.2) and provide mechanisms for users to externally validate query results (Section 5.3).

5.1 Imprecision-Aware Processing

Unlike reactive data systems that only execute the query as stated, proactive data systems can leverage LLMs to help truly identify user intent, despite the user-provided tasks representing an imprecise or incomplete specification thereof. The system should therefore internally consider multiple possible user intents when processing tasks, potentially providing different answers that correspond to these different interpretations. This can be done to varying degrees. In the simplest form, the system can consider multiple interpretations of the statements provided by the user. In Example 7, the system can consider various spellings of the same name, either in the input provided by the user or derived from the data. Such attempts are similar to possible world notions in the database community [31], where the database can consider various possibilities for "fuzzy" data and queries.

A proactive system can take more aggressive steps to understand user intent. For example, the system can consider adding new predicates that the user may be interested in—e.g., if a user has previously asked several questions about police misconduct in a specific city but submits a new query without specifying location, the system might prioritize results from that city. This intent discovery can be data-driven—the system might determine that records from certain cities are more relevant or interesting and prioritize them in the output. Moreover, if the output for an operation is too large, the system can selectively display what it determines to be the most relevant answers or provide appropriate summaries or sample outputs.

The system can also anticipate user questions, for example, "why was a certain record not provided in the answer", and proactively relevant records that, while not strictly matching the query, might be of interest to the user. Additionally, the system can modify queries by dropping or relaxing certain predicates—e.g., if the user has specified a predicate that leads to empty results. Or, the system might expand query scope, for example, geographically, to include potentially interesting results (such as when a specific type of police misconduct, while not present in the queried city, occurred in neighboring jurisdictions).

5.2 Leveraging User Feedback

A proactive system can leverage feedback from users to clarify intent in a lightweight manner. This feedback serves two purposes: improving accuracy for the current task, and enhancing the system's understanding of user intent for future queries.

To improve the accuracy on the current task, the system can decide to ask follow-up questions [32]. This might include asking for clarification about task goals, gathering additional specifications, or presenting example results for users to indicate which best match their needs. The important challenge is balancing the need for clarity with minimizing user burden: for example, when processing police misconduct documents, rather than asking multiple detailed questions, the system might show representative document types and let users select which are most relevant—then apply this learning broadly across the document collection. Similarly, when encountering potential name variations in Example 7, the system might ask a single question about handling typos that can inform its overall matching strategy, rather than asking the user to confirm every typo correction. While LLMs offer promising capabilities for generating targeted feedback requests, automatically determining what feedback to request and when remains an open research challenge. Prior work on predictive interaction is highly relevant here [33].

User feedback can also be leveraged to improve system performance on `future` tasks. For example, if the user provides feedback that a certain document is relevant or not relevant to a task, the system can then update its data and task processing mechanism to take that feedback into account. This can, for instance, change operation rewrite rules used internally for processing (Section 3), modify data representation [16] or change how semantic structure is extracted from the data (Section 4). While this approach shares similarities with query-by-example systems that learn from user-provided examples, e.g., [34], it extends the concept more broadly——allowing the system to refine its understanding of user intent across a diverse range of tasks and feedback types.

5.3 Verifying Execution

A proactive system must provide users with the means to verify that their tasks were executed correctly. Verification is particularly important when the system makes autonomous decisions—for instance, when correcting potential typos in names, the system should clearly show which corrections were made to allow users to catch any incorrect modifications. If, during processing, the system encountered anomalous documents, it's best to indicate them as such to the users so that they don't pollute the rest of the analysis.

While the system can provide comprehensive execution traces, including details of LLM operations performed and data sources accessed [35], presenting this information in a user-friendly way remains challenging. Simply showing raw execution traces or complete datasets is overwhelming and impractical, as users cannot reasonably review large amounts of data to verify correctness. An interesting open challenge is to determine a small subset or explanation that conveys the same information as the entire provenance; we can always verify such explanations using an LLM.

6 Conclusion

The database community stands at a pivotal moment where LLMs offer unprecedented capabilities for processing both structured and unstructured data. In this vision paper, we proposed *proactive* data systems: systems that possess agency in understanding and optimizing data processing tasks. Unlike traditional *reactive* systems that treat LLMs as black-box UDFs operating on monolithic inputs, proactive systems go further in leveraging LLMs to aid data processing along three axes. We presented these axes—operations, data, and user intent—and demonstrated the potential of LLMs to help in

each one through our recent work on operator reformulation, document organization and analytics, and intent-aware optimization. Overall, proactive data systems can achieve both higher accuracy and lower costs than reactive systems that treat LLMs as black boxes.

References

[1] Y. Lin, M. Hulsebos, R. Ma, S. Shankar, S. Zeigham, A. G. Parameswaran, and E. Wu, "Towards accurate and efficient document analytics with large language models," *arXiv preprint arXiv:2405.04674*, 2024.

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

[3] J. Fang, C.-T. Liu, J. Kim, Y. Bhedaru, E. Liu, N. Singh, N. Lipka, P. Mathur, N. K. Ahmed, F. Dernoncourt, *et al.*, "Multi-llm text summarization," *arXiv preprint arXiv:2412.15487*, 2024.

[4] A. Narayan, I. Chami, L. J. Orr, and C. R'e, "Can foundation models wrangle your data?," *Proc. VLDB Endow.*, vol. 16, pp. 738–746, 2022.

[5] Z. Huang, J. Liu, H. Wang, and E. Wu, "The fast and the private: Task-based dataset search," *arXiv preprint arXiv:2308.05637*, 2023.

[6] M. Kayali, F. Wenz, N. Tatbul, and Ç. Demiralp, "Mind the data gap: Bridging llms to enterprise data integration," *arXiv preprint arXiv:2412.20331*, 2024.

[7] B. Li, Y. Luo, C. Chai, G. Li, and N. Tang, "The dawn of natural language to sql: Are we fully ready?," *arXiv preprint arXiv:2406.01265*, 2024.

[8] A. Floratou, F. Psallidas, F. Zhao, S. Deep, G. Hagleither, W. Tan, J. Cahoon, R. Alotaibi, J. Henkel, A. Singla, *et al.*, "Nl2sql is a solved problem... not!," in *CIDR*, 2024.

[9] R. C. Fernandez, A. J. Elmore, M. J. Franklin, S. Krishnan, and C. Tan, "How large language models will disrupt data management," *Proc. VLDB Endow.*, vol. 16, p. 3302–3309, jul 2023.

[10] L. Patel, S. Jha, P. Asawa, M. Pan, C. Guestrin, and M. Zaharia, "Semantic operators: A declarative model for rich, ai-based analytics over text data," *arXiv preprint arXiv:2407.11418*, 2024.

[11] E. Anderson, J. Fritz, A. Lee, B. Li, M. Lindblad, H. Lindeman, A. Meyer, P. Parmar, T. Ranade, M. A. Shah, B. Sowell, D. Tecuci, V. Thapliyal, and M. Welsh, "The design of an llm-powered unstructured analytics system," 2024.

[12] 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*, vol. 12, pp. 157–173, 2024.

[13] Y. Lin, M. Hasan, R. Kosalge, A. Cheung, and A. G. Parameswaran, "Twix: Automatically reconstructing structured data from templatized documents," *arXiv preprint arXiv:2501.06659*, 2025.

[14] S. Shankar, T. Chambers, T. Shah, A. G. Parameswaran, and E. Wu, "Docetl: Agentic query rewriting and evaluation for complex document processing," *arXiv preprint arXiv:2410.12189*, 2024.

[15] A. G. Parameswaran, S. Shankar, P. Asawa, N. Jain, and Y. Wang, "Revisiting prompt engineering via declarative crowdsourcing," *Cidr*, 2024.

[16] S. Zeighami, Z. Wellmer, and A. Parameswaran, "Nudge: Lightweight non-parametric fine-tuning of embeddings for retrieval," *arXiv preprint arXiv:2409.02343*, 2024.

[17] H. Li, N. Chalapathi, H. Qu, A. Cheung, and A. G. Parameswaran, "Inferring visualization intent from conversation," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pp. 1184–1194, 2024.

[18] M. Hulsebos, W. Lin, S. Shankar, and A. Parameswaran, "It took longer than i was expecting: Why is dataset search still so hard?," in *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics*, pp. 1–4, 2024.

[19] S. Shankar, H. Li, P. Asawa, M. Hulsebos, Y. Lin, J. Zamfirescu-Pereira, H. Chase, W. Fu-Hinthorn, A. G. Parameswaran, and E. Wu, "Spade: Synthesizing assertions for large language model pipelines," *arXiv preprint arXiv:2401.03038*, 2024.

[20] S. Shankar, J. Zamfirescu-Pereira, B. Hartmann, A. Parameswaran, and I. Arawjo, "Who validates the validators? aligning llm-assisted evaluation of llm outputs with human preferences," in *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–14, 2024.

[21] 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*, vol. 36, 2024.

[22] M. Pourreza, H. Li, R. Sun, Y. Chung, S. Talaei, G. T. Kakkar, Y. Gan, A. Saberi, F. Ozcan, and S. O. Arik, "Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql," *arXiv preprint arXiv:2410.01943*, 2024.

[23] K. Luoma and A. Kumar, "Snails: Schema naming assessments for improved llm-based sql inference," *Proc. ACM Manag. Data*, vol. 3, Feb. 2025.

[24] O. Khattab, A. Singhvi, P. Maheshwari, Z. Zhang, K. Santhanam, S. Vardhamanan, S. Haq, A. Sharma, T. T. Joshi, H. Moazam, *et al.*, "Dspy: Compiling declarative language model calls into self-improving pipelines," *arXiv preprint arXiv:2310.03714*, 2023.

[25] Y. Chung, G. T. Kakkar, Y. Gan, B. Milne, and F. Ozcan, "Is long context all you need? leveraging llm's extended context for nl2sql," *arXiv preprint arXiv:2501.12372*, 2025.

[26] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: optimizing neural network queries over video at scale," *Proc. VLDB Endow.*, vol. 10, p. 1586–1597, Aug. 2017.

[27] J. M. Hellerstein and M. Stonebraker, "Predicate migration: Optimizing queries with expensive predicates," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pp. 267–276, 1993.

[28] V. Raman, B. Raman, and J. M. Hellerstein, "Online dynamic reordering for interactive data processing," in *VLDB*, vol. 99, pp. 709–720, Citeseer, 1999.

[29] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning, "Raptor: Recursive abstractive processing for tree-organized retrieval," *arXiv preprint arXiv:2401.18059*, 2024.

[30] S. Papicchio, P. Papotti, and L. Cagliero, "Evaluating ambiguous questions in semantic parsing," in *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*, pp. 338–342, IEEE, 2024.

[31] D. Suciu, D. Olteanu, C. Ré, and C. Koch, *Probabilistic databases*. Springer Nature, 2022.

[32] Y. Li and D. Jobson, "Llms as an interactive database interface for designing large queries," in *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics*, pp. 1–7, 2024.

[33] J. Heer, J. Hellerstein, and S. Kandel, "Predictive interaction for data transformation," in *Conference on Innovative Data Systems Research (CIDR)*, 2015.

[34] A. Fariha, S. M. Sarwar, and A. Meliou, "Squid: Semantic similarity-aware query intent discovery," in *Proceedings of the 2018 International Conference on Management of Data*, pp. 1745–1748, 2018.

[35] W. C. Tan *et al.*, "Provenance in databases: Past, current, and future.," *IEEE Data Eng. Bull.*, vol. 30, no. 4, pp. 3–12, 2007.

Top Ten Challenges Towards Agentic Neural Graph Databases

Jiaxin Bai[††] Zihao Wang[*†] Yukun Zhou[†] Hang Yin[‡] Weizhi Fei[‡] Qi Hu[†]
Zheye Deng[†] Jiayang Cheng[†] Tianshi Zheng[†] Hong Ting Tsang[†] Yisen Gao[∨]
Zhongwei Xie[⊥] Yufei Li[⊤] Lixin Fan[¶] Binhang Yuan[†] Wei Wang[†]
Lei Chen[†] Xiaofang Zhou[†] Yangqiu Song[†]

[†] Department of Computer Science and Engineering, HKUST, Hong Kong, China
[‡] Department of Mathematical Sciences, Tsinghua University, Beijing, China
[¶] AI Group, WeBank [∨] Institute of Artificial Intelligence, Beihang University
[⊥] Wuhan University [⊤] Sichuan University
{jbai,zwanggc,yzhoufw,qhuaf,zdengah,jchengaj,tzhengad,httsangaj}@cse.ust.hk
{biyuan, weiwa, leichen, zxf, yqsong}@cse.ust.hk
lixinfan@webank.com {h-yin20, fwz22}@mails.tsinghua.edu.cn
yisengao@buaa.edu.cn zhongwei.xie@whu.edu.cn evangeline@stu.scu.edu.cn

Abstract

Graph databases (GDBs) like Neo4j and TigerGraph excel at handling interconnected data but lack advanced inference capabilities. Neural Graph Databases (NGDBs) address this by integrating Graph Neural Networks (GNNs) for predictive analysis and reasoning over incomplete or noisy data. However, NGDBs rely on predefined queries and lack autonomy and adaptability. This paper introduces Agentic Neural Graph Databases (Agentic NGDBs), which extend NGDBs with three core functionalities: autonomous query construction, neural query execution, and continuous learning. We identify ten key challenges in realizing Agentic NGDBs: semantic unit representation, abductive reasoning, scalable query execution, and integration with foundation models like large language models (LLMs). By addressing these challenges, Agentic NGDBs can enable intelligent, self-improving systems for modern data-driven applications, paving the way for adaptable and autonomous data management solutions.

1 Introduction

Graph databases like Neo4j [1], TigerGraph [2], and Azure Cosmos DB are useful tools for representing and querying interconnected data using nodes and edges. These databases are adept at handling the complex relationships inherent in graph-structured data, providing efficient mechanisms for storage and retrieval.

A Neural Graph Database (NGDB), as introduced in [3], represents a system architecture that merges the predictive capabilities of Graph Neural Networks (GNNs) with the rich data representation features of graph databases (GDBs). NGDBs enhance graph databases by leveraging GNNs for advanced machine-learning tasks while preserving and utilizing the information embedded within the graph data model.

However, methodologies for conducting inferences within this latent neural space are yet to be thoroughly explored. To address this gap, the integration of neural execution engines on top of neural

[†] Equal Contribution

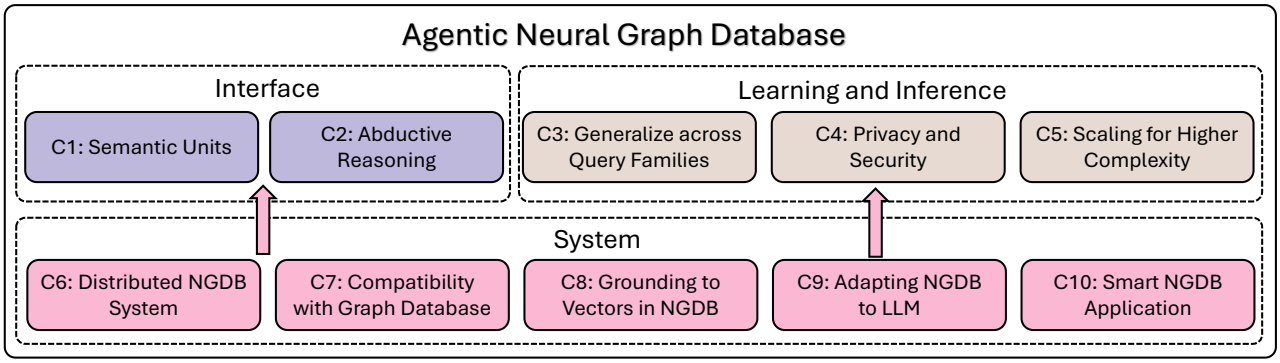

Figure 1: The top ten challenges in achieving Agentic NGDB. Its three perspectives include interface, learning, and system.

graph storage has been proposed [4]. By utilizing neural embeddings and neural networks, NGDBs enhance their ability to perform complex reasoning and more effectively infer hidden relationships, which are the capabilities that traditional graph databases lack. This fusion of symbolic graph representations with neural computation paves the way for more intelligent and adaptable data management systems to address contemporary applications' diverse demands. The process of "neuralization" is particularly beneficial for inferring missing information within the underlying graph data model, enriching the database with additional knowledge.

From a broader perspective, the principles of data management systems revolve around efficiently storing, retrieving, and managing data while providing a layer of abstraction to users. These systems aim to handle large volumes of data and complex operations, concealing the underlying complexities from end-users. Motivated by this principle, we propose the concept of **Agentic Neural Graph Databases (Agentic NGDBs)**, extending neuralization to further automate data and data management processes. Here, we summarize the challenges regarding the Agentic NGDB from the following three perspectives interface, learning, and system:

- **Interface**: The Agentic NGDB should automatically construct appropriate queries that generate useful answers for a given task in a specific context.
- **Learning and Inference**: Agentic NGDB should leverage neural networks to execute queries and derive meaningful answers as neural network predictions, even when the underlying data model is incomplete.
- **System**: The Agentic NGDB should remain compatible with existing graph databases, supporting most standard GDB operators. Additionally, it should function as an adaptor for foundation models, enhancing knowledge and reasoning capabilities. Furthermore, it must actively learn by constructing and executing appropriate `CREATE`, `UPDATE`, or `DELETE` queries in a given context.

There are significant challenges to achieving each of these aspects, as illustrated in Figure 1. We identify the most critical challenges for realizing these functionalities based on recent progress in the research community on logical query answering and logical hypothesis generation for relational graphs.

Interface The first significant challenge in the **Interface** component is addressing *fundamental semantic units* (**Challenge 1**) within the neural graph database's query and data model. Semantic units refer to the data types associated with nodes and edges, such as atomic IDs, text strings (e.g.,

entities and events), numbers, and dates. Constructing queries that effectively handle these diverse semantic units presents a significant obstacle. Beyond managing individual semantic units, another critical challenge lies in connecting these units to construct more complex queries. The **Interface** component also requires advanced *abductive reasoning capabilities* (**Challenge 2**). In Agentic NGDBs, abductive reasoning refers to identifying the optimal NGDB query that best explains or supports a specific task in a given context. This capability ensures that the database can adaptively generate meaningful, task-relevant queries. The generated queries can then be executed symbolically by a graph database or through neural execution within the system.

Learning and Inference Neural query execution is the core functionality of traditional NGDBs, referring to the ability to perform tasks or actions according to a predefined plan or strategy within the neural space [5] by learning and inferences. However, several practical challenges remain in this area. One major challenge is enhancing *inference capabilities for better generalization* (**Challenge 3**) across query families. This involves ensuring that NGDB systems can effectively handle and generalize across diverse query structures and types, even when presented with novel or complex combinations of queries. Another critical hurdle involves maintaining *data privacy and security* (**Challenge 4**). Due to their inherent vulnerability to extracting latent knowledge, NGDBs must safeguard sensitive information against advanced inference attacks, particularly in neural models. Robust privacy mechanisms are essential for building trust and ensuring security in NGDB applications. The *scaling laws of neural query execution* (**Challenge 5**) must be explored. Scaling laws in NGDBs describe how the system's performance improves as key factors, such as the number of model parameters, the size of the training dataset, and training costs, are increased. This concept is rooted in neural scaling laws observed in deep learning, where larger models generally lead to better performance, albeit at higher computational costs.

System The **System** component focuses on building a system on top of the learning and inference algorithms that can ensure continuous learning and adaptation within Agentic NGDBs. Efficiently processing and managing large-scale data while maintaining high performance becomes especially critical when dealing with massive datasets. This challenge is further amplified in distributed NGDB architectures, where optimizing query performance under read-intensive workloads and dynamically fluctuating demands is necessary. Ensuring elastic scalability and developing NGDBs that operate effectively as *distributed systems* (**Challenge 6**) are key to achieving these goals. These systems must be capable of improving themselves by writing and executing `CREATE`, `UPDATE`, and `DELETE` clauses or performing model editing directly within the neural latent space. The first aspect of this functionality involves ensuring *compatibility with graph database models* (**Challenge 7**). The fundamental CRUD (CREATE, READ, UPDATE, DELETE) actions are essential for managing and modifying persistent data elements in traditional graph databases. Seamlessly integrating these actions into NGDBs is necessary for enabling effective self-improvement. The second aspect involves *grounding vectors within NGDBs* (**Challenge 8**). For effective learning and adaptation, the system must accurately identify the locations of relevant knowledge and understand how reasoning is conducted within the latent neural space. Proper grounding ensures that modifications and updates align with the underlying knowledge representation. Moreover, the Agentic NGDB must be capable of *integrating with foundation models*, such as large language models (LLMs), to enhance its reasoning and knowledge capabilities (**Challenge 9**). The NGDB can provide more reliable and contextually accurate results for various tasks by leveraging foundation models' advanced natural language understanding and reasoning capabilities. Finally, we discuss the challenge of developing Smart Neural Graph Databases (NGDB) applications (**Challenge 10**), particularly Agentic NGDB. The challenges lie in creating systems that leverage their advanced functionalities across diverse applications.

Agentic NGDBs extend the capabilities of traditional NGDBs by incorporating autonomy, active learning, and adaptability. While NGDBs enhance graph databases by integrating Graph Neural Networks (GNNs) to perform advanced inference and reasoning and handle incomplete or noisy data, they rely heavily on predefined tasks and human-defined queries. In contrast, Agentic NGDBs introduce three core functionalities: automatic query construction tailored to specific tasks and contexts, neural query execution for predictive analysis, and continuous learning and adaptation through active updates to the knowledge base. In the following sections, we will individually discuss each identified challenge.

2 Challenge 1: Semantic Units

The NGDB primarily relies on relational graphs, where nodes and relations are the basic semantic units. Incorporating diverse semantic units, such as numbers and events, introduces complexity due to their intrinsic relationships. For example, numbers involve algebraic operations (e.g., addition, subtraction), while events involve temporal and causal relations. Addressing these complexities requires reasoning engines that can learn and process such relationships effectively.

Number literals (e.g., age, height) are critical for filtering and querying within NGDBs. Prior work includes methods like KBLRN [6], KR-EAR [7], and LitCQD [8], which improve reasoning by integrating numeric constraints into queries. Despite these advancements, challenges remain. These include developing advanced numerical operations and integrating neural-symbolic systems into NGDBs while ensuring compatibility with symbolic solvers for faithful reasoning. Existing approaches focus on entity-centric knowledge graphs, but event-centric knowledge graphs (EVKGs) like ATOMIC [9] and ASER [10] emphasize relationships between events (e.g., temporal and causal relations). Reasoning on EVKGs involves determining event occurrences and their sequences, which introduces unique challenges compared to entity-centric KGs. Recent work extends traditional reasoning by integrating temporal and occurrence constraints[11].

Moreover, beliefs, desires, and intentions (BDI) represent higher-level, abstract semantic units extending beyond simple entity-attribute relationships and eventualities. These elements are crucial for modeling human-like reasoning, decision-making, and behavior prediction. Beliefs refer to what an agent (human or system) assumes or holds to be true about the world. These can include factual statements like *It is raining outside* and subjective perspectives like *This movie is great.* In KGs, beliefs are often represented as knowledge nodes or statements that may vary across agents or contexts, allowing for personalization or multi-agent reasoning. Intentions represent the goals or purposes behind an agent's actions or decisions and as a bridge between beliefs and actions. Intentions are often implicit and must be inferred from user behavior or contextual information. KGs are typically modeled as motivational nodes or goals that guide reasoning about why an agent performs specific actions. For instance, *PersonX intends [to buy a gift for a friend]*, which could explain why *PersonX searches for [gift shops nearby].* On the other hand, desires represent an agent's wants, preferences, or needs, which may not always lead to concrete actions unless accompanied by intention. In knowledge graphs, desires are commonly expressed as preferences or motivational entities that influence behavior, such as *PersonX desires [to eat ice cream].* These three elements allow knowledge graphs to capture human motivations more comprehensively. These concepts are closely connected to the Theory of Mind (ToM), which refers to the ability to understand that other agents (humans, machines, etc.) possess their own beliefs, desires, and intentions that may differ from one's own. In the context of knowledge graphs, the Theory of Mind enhances reasoning about multi-agent knowledge by enabling the understanding of diverse perspectives. Theory of Mind also enables the inference of motivations by reasoning about the interplay between beliefs, desires, and intentions.

Integrating BDI and ToM in Agentic NGDB has practical applications across various domains. In e-commerce, systems like FolkScope [12], COSMO [13], and RIG [14] are the knowledge graphs that

leverage BDI to model user behavior, enabling personalized recommendations by linking user actions (e.g., purchases) with inferred desires and intentions. In commonsense reasoning, resources like ATOMIC use BDI to represent cause-effect relationships, allowing systems to reason about potential outcomes of actions. Multi-agent systems benefit from BDI-enhanced KGs by enabling cooperative and competitive interactions that account for multiple agents' goals, beliefs, and desires. Additionally, in natural language understanding, BDI helps interpret user intent in queries, conversations, and social media posts by associating semantic meanings with inferred motivations. We still need systematic storing and inference with these intention knowledge graphs.

3 Challenge 2: Abductive Reasoning with NGDB

Abductive reasoning, the process of inferring the most plausible explanations for observations, is a fundamental aspect of human cognition and artificial intelligence. In the context of knowledge graphs (KGs), abductive reasoning generates hypotheses to explain observations (entity sets) by leveraging structured relationships and entities. Complex Logical Query Answering (CLQA) has further advanced abductive reasoning by enabling multi-hop logical inferences over large, incomplete graphs. Neural Graph Databases (NGDBs) build on these advancements, offering a more flexible and robust framework for abductive reasoning.

Early methods for abductive reasoning in KGs relied on supervised learning and search-based techniques. Generative models, such as transformer-based architectures, were used to produce logical hypotheses. For example, [15] proposed a supervised generative model trained on datasets like FB15k-237 and WN18RR, which excelled in structural fidelity but struggled to generalize to unseen observations due to the limitations of supervised objectives. To address these limitations, reinforcement learning (RL) techniques were introduced. Reinforcement Learning from Knowledge Graph feedback (RLF-KG) employed proximal policy optimization (PPO) to generate hypotheses aligned with observed evidence. This approach improved explanatory power and generalizability, achieving significant gains in metrics like Jaccard similarity and Smatch scores across multiple datasets. NGDBs extend these methods by embedding knowledge graph data in a latent space, enabling flexible query processing and hypothesis generation. By leveraging latent embeddings, NGDBs can infer missing information and generate hypotheses for complex logical queries, even on incomplete graphs, outperforming traditional graph databases. NGDBs represent a significant step forward in abductive reasoning, synthesizing the strengths of CLQA and advanced generative models. However, several challenges must be addressed:

- **More Generalized Observation** In the current definition of abductive reasoning, the definition of the observations is a set of entities. However, observation can be further generalized to a context, for example, a conversation history in the conversational recommendation task setting, or a structured shopping session.

- **More Complex Structured Hypotheses** Existing abductive reasoning models on KGs primarily focus on conjunctive tree-formed queries. NGDBs, with their increased query expressiveness, require hypothesis generation models capable of handling more complex structured observations. For instance, hypotheses should accommodate EFO_k (existential first-order logic) and cyclic queries, expanding beyond the limitations of earlier models.

- **Graph-Based Hypothesis Generation Models** Traditional sequence-based models struggle to capture the structural complexity of logical hypotheses, which are fundamentally query graphs. These graphs exhibit features like permutation invariance of logical operators, requiring models explicitly designed to generate graph-structured hypotheses.

- **NGDB as a Reward Model for Reinforcement Learning** Previous RL-based methods, such as [15], relied on symbolic execution results from knowledge graphs to provide reward signals during hypothesis generation. However, these reward signals suffer from the incompleteness inherent to the open-world assumption. NGDBs can address this issue by serving as a more robust reward model, leveraging their latent embeddings and flexible query capabilities to improve hypothesis generation.

4 Challenge 3: Generalization across Query Families

Introducing neural modules in graph databases enables the generalization to the *knowledge* in databases. However, the development of neural modules is always entangled with their targeted query families, thus naturally biased toward them due to their inductive biases, emphasizing the challenge of generalizing towards different *query types*. Compared to classic database algorithms that support an entire query family as long as it is formally defined, neural modules still suffer a loss in performance for generalization even when the query family is fixed [16]. Readers are also referred to related surveys [17, 18].

4.1 Different Query Families and Their Neural Modules

Tree-formed Queries and Compositional Generalizability. The tree-formed query is a collective term that describes the whole query family that can be recursively defined in a tree structure, in which logical connectives and variables are carefully organized so that set operations can formally derive the answers [16], the set operations include set projection [19], intersection, union [20], complement[20] and set difference [21]. To tackle such kinds of queries, a line of research is known as query embeddings, where sets are modeled as embeddings, and set operations mentioned above are modeled directly by neural modules [21–23]. The set operations composition allows the models to generalize the entire tree-form query family. This connection between model design and query family is termed the compositional generalizability [16, 24], and the performance drop with the increasing of compositional levels is still universally observed and remains a challenge to address.

EFO-1 Queries and Query Graph. It is shown that tree-formed query family is constrained by certain assumptions and fails to represent the whole family of Existential First Order queries with one free variable (EFO-1 query) such as cyclic query [25]. To handle new graph-theoretic features which cannot be represented in tree-formed queries. One commonly adopted technique for EFO-1 queries is the DNF normal form or the UCQ query-solving strategy [20, 26], which solves the conjunctive query first and then takes the union of the answer set of each conjunctive query. A query graph [26] can naturally describe each conjunctive query. This formulation motivates graph-related search methods [25] or graph neural networks [26].

More Advanced Query Types. More advanced query families still exist, though the development of corresponding neural models on these topics is insufficient at the current stage. Thus, we discuss some of the challenges we might face in pursuit of more advanced queries in NGDB from the following aspects **(i) Multi-arity predicates:** The first challenge we may encounter is when the knowledge databases are constructed by $(n+1)$-ary tuples, the relation corresponds to n-ary predicate and a graph becomes a hypergraph [27]. **(ii) Support of functions** The corresponding research gap is the support for functions in the query – a function can output nodes, numbers, semantic units, or data of more advanced modality – for example, the `AVG` and `COUNT` functions in SQL but not in current CQA models. We have noted one preliminary research trying to fill this gap [8].

4.2 Minimal Assumption for Broad Generalization

Previous case studies showcase the close entanglement of the neural modules and the query types they support syntactically. In other words, the key to generalization is minimizing the query families' assumptions and the inductive biases of the neural part of NGDB. We present two types of methods with minimal assumptions.

Neuro-symbolic Methods. NGDB implies that the underlying database is a graph, meaning neural modules solely modeling the graph itself impose no assumptions on the query family it might support. Such neural modules include link predictors or knowledge graph embeddings that map a triple (s, p, o) of subject, predicate, and object into a score [28]. Therefore, the critical design task of NGDB with such modules is revising the algorithms into the neuro-symbolic forms with the scores produced by link predictors [25, 29]. An apparent and more decomposed approach is to derive an instance of a classic graph database using the link predictor, and all previous research in graph databases applies directly. Notably, the neuro-symbolic approach achieves the same level of generalizability in queries as the classic database research.

Sequence Models. Language models or sequence models are general-purpose models and thus further disentangle the inductive bias of neural modules and the specific task (queries in NGDB). Such models support the sequence inputs, which cover inputs from all possible kinds of query types. However, the performance on specific query types is transferred from designing neural architectures to curating the training datasets. The cost is transferred from the complex inference algorithms to the training phase [30].

4.3 Learning Aspects of Generalization

From the machine learning perspective, one new issue is uniformly improving the performance of all queries of a particular query family under the analogy of query types as tasks. The approach towards this goal also varies for different methods. For neuro-symbolic approaches, the generalization will be improved coherently as link prediction performance improves. For neural methods, the challenge of generalization is the same as multi-task learning. Query embeddings, as a particular case of neural methods, recent works propose adopting set operators with meta-learning, yielding the solution of meta operators [24].

5 Challenge 4: Privacy and Security

5.1 Database Privacy

Privacy in data storage refers to protecting sensitive information from unauthorized access and misuse [31]. Traditional databases are facing several privacy risks, which can be categorized into: (1) Unauthorized Access [32]: Unauthorized access to databases can result in large-scale data leakage, exposing sensitive personal information. (2) Insider Threats [33]: Employees with legitimate access may misuse their privileges, either intentionally or unintentionally compromising data privacy. (3) Data Inference Attacks [34]: Attackers can employ various techniques to deduce sensitive information from seemingly innocuous data.

To mitigate privacy risks, several protection methods have been developed: (1) Data Anonymization [35]: Techniques such as k-anonymity [36] and l-diversity [37] help mask individual identities within datasets, making it harder to trace data back to specific individuals. (2) Encryption [38]: Data encryption ensures that unauthorized parties cannot access sensitive information even if they breach a database. (3) Access Control [32]: Access control restricts data access to authorized users only, reducing the risk of

insider threats. (4) Differential Privacy [39]: This approach adds noise to data outputs, ensuring that the presence or absence of an individual in a dataset does not significantly affect the results of queries.

5.2 New Privacy Challenges in NGDBs

Graph databases, while offering advantages in managing complex relationships, introduce specific privacy risks: (1) Link Prediction Attacks [40]: Adversaries can use machine learning models to predict hidden relationships within the graph, potentially uncovering private connections. (2) Structural Attacks [41]: Even when the data content is anonymized, the graph's structure itself can reveal sensitive insights. The unique structure of graph data amplifies these risks, as the relationships between entities can reveal information that is not immediately apparent from isolated data points. Neural Graph Databases (NGDBs) represent a significant advancement in data management, combining the strengths of traditional graph databases with the capabilities of neural networks. The exploration of privacy issues in NGDBs remains largely underdeveloped, with significant gaps in research addressing potential vulnerabilities and mitigation strategies.

Potential Attacks. One of the primary strengths of NGDBs is their ability to generalize from incomplete data by inferring hidden relationships. While this capability can enhance data retrieval and knowledge discovery, it also poses significant privacy risks [42]: (1) Model Inversion Attacks [43]: Neural models can be susceptible to inversion attacks, where an adversary uses access to the model to recover the graph data used for NGDB training. (2) Membership Inference Attacks [44]: Attackers may infer whether a particular data point (node or edge) was part of the training data, revealing sensitive information in NGDBs. (3) Embedding Leakage [45]: The embeddings generated by NGDBs to represent nodes and relationships can leak sensitive information, as these embeddings often capture detailed structural and content-based features of the graph stored.

Promising Defenses. (1) Differential Privacy in NGDBs: Extending differential privacy techniques to protect neural graph databases is a key research direction. Adding noise to the model parameters or gradients during training can help mitigate membership inference and model inversion attacks [46, 47]. (2) Embedding Obfuscation: Techniques to obfuscate embeddings without losing their utility for answering complex queries need to be developed to prevent leakage of sensitive information [48]. (3) Private Distribute Training: Privacy problems in distributed NGDBs need further development [49]. Federated learning, including Secure Multi-Party Computation (SMPC) [50] and Homomorphic Encryption (HE) [51] techniques, can be adapted to NGDBs to ensure that data is processed without being revealed.

Evaluation Benchmarks. Another significant challenge in NGDBs is the evaluation of privacy protection efficacy. Assessing the effectiveness of privacy-preserving mechanisms requires robust benchmarks that can accurately measure both privacy protection and the quality of retrieved data. However, such benchmarks are currently lacking in the field. To address this challenge, standardized evaluation metrics and datasets should be developed that can facilitate comprehensive testing of privacy-preserving techniques in NGDBs. Establishing reliable benchmarks will provide insights into the strengths and weaknesses of different approaches, ultimately guiding future developments in privacy protection.

6 Challenge 5: Scaling for Higher Complexity

In deep learning, neural scaling law is an empirical law that describes the performance of neural models improves with the number of parameters, training dataset size, and training cost [52, 53]. During the development of the NGDB model, scaling is also a major thread, primarily encompassing the scaling of parameter number, query data size, and training costs. The query embedding methods and sequence models often scale the training costs in the training stage, including the model parameters and queries. In contrast, the neuro-symbolic methods often scale the computation cost over the test stage to improve

the performance. We mainly discuss how to scale these models further, particularly when the query structure becomes increasingly complex [25, 54] and the magnitude of the knowledge databases becomes very large [55]. Specifically, we introduce the complexity of these models in the training and inference stages and discuss their efficiency and scalability challenges.

Data Scaling in the Training Stage. Both query embedding and sequence models are trained from scratch, requiring many sampled queries as training data. The quality and size of these training queries are crucial, and they typically encompass various query types. The NGDB models generally use the same dataset, with the basic 1p query type enumerating the entire knowledge graph [20]. To incorporate new features such as negation [22], cyclic queries [25], and multivariable queries [54], it is essential to sample query types that include these features. Materializing training queries becomes infeasible as the knowledge graph grows, and sampling logical queries is incompatible with traditional single-hop frameworks based on graph partitioning. To address this challenge, SMORE [55] proposes a scalable framework that efficiently samples training data on the fly with high throughput. In contrast, neuro-symbolic methods primarily rely on pre-training for the knowledge graph completion task and depend on search algorithms to address general logical queries.

Test Time Scaling in Inference Stage. We first introduce the notion of query complexity and data complexity [56]. Data complexity captures the relation between the time complexity and the database size $|E|$ (number of the edges) when the query is fixed. In contrast, query complexity is assessed based on the size of the query $|Q|$ (number of the predicates) when assuming the database is fixed. When discussing the complexity, the query is restricted to tree-formed queries and EFO-1 queries that we have discussed before. The complexity of neural symbolic search is well studied. The complexity for tree-formed queries is $O(|Q||E|)$. Such approaches [25, 29] require $O(|Q|)$ search steps, while each step requires a search over the database, which is $O(|E|)$. For the general EFO-1 query, the cyclic query makes the general complexity particularly hard and results in $O(|E|^{|Q|})$ time, which is polynomial in data but exponential in query. One distinct feature of query embeddings and sequence models compared to neuro-symbolic methods is the disentanglement of the $|E|$ term and $|Q|$ term. Notably, the neural network encoder [26] or sequence model [30] work on the query directly, which is usually $O(|Q|)$ to encode query information and $O(|E|)$ to decode the answer by embedding comparison. However, this great advantage in inference time complexity of query embeddings and neural symbolic models comes from the additional and usually resource-consuming training procedures.

7 Challenge 6: Distributed NGDB System

7.1 Scenario Features and System Requirements

Scenario Features. NGDB is targeted at a scenario where users can simultaneously conduct graph data management and graph inference. We identify four features of such a scenario that significantly affect the system design. (1) Hybrid symbolic and neural operation [4]. Users can input queries requiring algebraic, neural, or hybrid computation; (2) Massive graph data and embeddings. Not only do the graph data of different domain knowledge exhibit tremendous scale [57], but also various types of embeddings [58] of these graph data further enlarge the volume; (3) Read intensive workload. During the serving stage, most of the graph data and embeddings are queried more frequently rather than updated [59]; (4) Dynamic workload fluctuation. Different parts of the graph data and embeddings are accessed in different time slots and the number of online users and frequency and data volume of one query fluctuate [60].

System Requirements. The neural graph database system should fulfill the following requirements to handle these features effectively and efficiently. (1) Co-located graph and embedding management.

The NGDB system should support symbolic graph data and neural embedding management. (2) High query performance. The latency of a single query and system throughput for numerous tenants serving massive data should be optimized. (3) Scalability.. The hardware resource management should be scalable to handle workload fluctuation, especially computational resources, cost-efficiently. Challenges are introduced to the system design of neural graph databases to implement these system features.

7.2 Challenges of System Design

User Interface Design. Existing vector databases provide SQL-like interfaces and parameterized API[61], while most of the interfaces mainly focus on relational data. Graph databases provide numerous interfaces[58], but there is little experience in combining neural operations into symbolic graph operations. It is essential to design highly expressive declarative user interfaces as well as programming interfaces.

Query-Oriented Distributed Storage. Due to the massive volume of graph data and corresponding embeddings, which is out of the capacity of standalone storage, distributed storage is an indispensable mechanism of NGDB. Under read-intensive workload, partitioning (or sharding), acting as a distributed index, tailored for most frequent and costly types of query could remarkably reduce the intermediate data transfer, consequently enhancing the overall latency and throughput[61]. Practices in graph database community[62–65] and vector database community concludes valuable principles and strategies on distributed storage and indexing of graph data and embedding separately. However, the hybrid storage of both data types is not explored, especially in circumstances where hybrid queries, requiring both symbolic and neural processes, are of evident importance. A typical example question is about whether embeddings and raw graph data shall be co-located. Although some open source graph database[66, 67] and vector databases[60, 68–70] could be utilized as standalone storage engine in NGDB, partitioning should be carefully designed under specific query workload.

Distributed Graph Computing and Inference. There are abundant works on distributed graph query and analysis with various algorithms in the graph database community[71, 72]. However, distributed system support for knowledge graph inference has not been adequately explored. Atom[73] points out a key observation that query embedding is the performance bottleneck, which shall be one of the considerations in NGDB query execution. On the base of these two kinds of computation optimization, when encountered with hybrid queries requiring both computation, query planning, and scheduling for maximized parallelism and minimized network communication overhead, still remain an unexplored direction. There are some preliminary practice cases in relational databases [74–76], which consider the optimization with neural operators but are still far from mature.

Elastic Scalability. To deal with dynamic workload fluctuation, fine-grained elasticity is of great importance to distributed NGDB systems, in which case on-demand resource provision helps reduce the cost[77] of NGDB service. Besides, not all the massive data are simultaneously accessed. There are evident biases and data heat shifts in database serving scenarios. Therefore, we argue that being cloud-native with elastic scalability is a crucial requirement for the NGDB system. Manu[60] detects such workload fluctuation in industrial applications, thus fully embracing the mechanisms of elastic scalability via dedicated abstraction of hardware resource management, including GPU, CPU, and disk. Besides, storage-computation-separation is essential for cloud-native databases[78]. It is essential to explore the combination of these separate practices. Additionally, the trade-off between latency and elasticity is a critical concern since practices in vector databases reveal that embedding management requires a large memory occupation.

8 Challenge 7: Compatibility of NGDB with Traditional Graph Database

Like graph databases, Neural Graph Databases (NGDB) are another way of the data model that derives the properties from the existing graphs, including nodes and edges, to represent entities and their relationships [4]. This structural consistency makes migrating and interoperating data between the two databases relatively easy. In terms of interfaces, NGDB can maintain support for standard graph query languages [66, 79] while providing vectorized query capabilities, allowing users to query and operate in familiar languages. In terms of operations, traditional CRUD operations remain fully functional, with the reasoning function of neural networks serving as enhanced features. For instance, conventional graph databases provide foundational support in query processing through mature storage and indexing technologies, while NGDB handles queries requiring missing link inference. Such compatibility design will enable a seamless system transition, where users can migrate to get NGDB capabilities without completely reconstructing existing applications. However, NGDB faces several challenges with traditional graph databases:

Novel Query Interface. Incorporating deep learning and graph neural networks extends beyond conventional graph database functionalities, requiring novel interfaces for deep learning-based queries and inference. This creates compatibility issues when attempting to reuse existing query languages, highlighting the need to develop new query languages or extend current ones [79, 80].

Performance-Consistency Trade-off. While traditional graph databases are optimized for storage and querying [81], they may struggle to meet performance requirements when handling large-scale graph-based deep-learning tasks. NGDB emphasizes representation learning on nodes and edges [4], requiring consideration of high-performance computing and distributed training paradigms. For instance, during conventional CRUD operations, NGDB may need to update node and relation embeddings, introducing additional computational overhead. Moreover, integrating neural components introduces temporal consistency challenges, where model updates may lead to temporary discrepancies between the base graph data and learned representations. Finding an optimal balance between consistency guarantees and computational efficiency remains a considerable challenge for NGDB systems.

9 Challenge 8: Grounding to Vectors with NGDB

Grounding natural language to knowledge bases has been extensively studied in conventional graph databases. Traditional approaches typically handle different grounding scenarios: hypothesis or query grounding (with free variables) [20, 82], and entity [83] or event [11, 84]). With the emergence of NGDBs, where structural information and semantic content are encoded as vectors, the grounding process faces new challenges and opportunities. Recent work [4] introduces a neural graph engine that learns query planning and execution strategies through interactions with Neural Graph Storage. However, grounding to general NGDBs still presents several unique challenges.

Semantic Granularity and Disambiguation. Semantic granularity and disambiguation pose fundamental difficulties. The grounding process must accurately translate natural language queries into appropriate vector representations while determining suitable levels of semantic granularity, such events, propositions, etc. [84, 85]. This challenge is compounded by the need to handle abstraction and polysemy when mapping linguistic elements to vector spaces, as meanings can vary significantly based on context.

Compositional Semantics and Reasoning. Second, compositional semantics and reasoning path selection present significant challenges. NGDBs must effectively represent complex multi-hop relations while maintaining transitivity and logical consistency in vector operations. The system needs to identify relevant paths in the vector space for query resolution, which becomes particularly challenging when dealing with multiple possible reasoning paths. In addition, determining appropriate termination criteria

for path exploration is crucial for both efficiency and accuracy.

Interpretation and Groundedness Evaluation. The third challenge is around interpretation and groundedness evaluation. The system is expected to reliably convert vector-based results back to natural language while providing clear explanations for its reasoning process. Additionally, it needs to report the level of groundedness for each grounding operation, ensuring semantic fidelity is maintained throughout the process. This is particularly important for applications requiring high precision and explainability.

10 Challenge 9: Adapting NGDB to LLM

This section explores the integration of Neural Graph Databases (NGDBs) with Large Language Models (LLMs) to enable joint reasoning and Retrieval-Augmented Generation (RAG). NGDBs can serve as retrieval modules for LLMs, leveraging structured data and reasoning capabilities to enhance generated outputs' accuracy, scalability, and contextual relevance. Joint learning of LLMs and NGDBs involves training these systems within a unified framework to combine natural language understanding with advanced logical reasoning.

NGDB-RAG: Definition and Components. NGDB-RAG (Neural Graph Database - Retrieval-Augmented Generation) is a system that integrates NGDBs with LLMs to enhance both retrieval and generation tasks. The NGDB-RAG system is composed of three main components. The first is the neural graph storage, which stores embeddings of nodes and edges in the graph. These embeddings capture both local and global structural relationships within the graph, providing a rich representation of the data. The second component is the neural query engine, which tries formulating and processing logical queries in the embedding space. This engine enables flexible modeling and supports logical operations such as conjunction, disjunction, and negation, allowing for robust retrieval even in incomplete or noisy graphs. The third component is integrating with LLMs, where NGDB reasoning results are incorporated into the language model. This integration can be achieved through text-based methods, by converting structured data into natural language, or through vector-based methods, by embedding structured data as vectors for direct input into the LLM.

Functionality of NGDB-RAG. NGDB-RAG enhances retrieval by utilizing the structured relationships in NGDBs to perform advanced reasoning tasks. Unlike traditional RAG systems that rely on document similarity, NGDB-RAG leverages the intricate dependencies within knowledge graphs to retrieve more accurate and contextually relevant information. In the generation process, NGDB-RAG integrates structured knowledge and reasoning capabilities from NGDBs to improve the generated text's factual accuracy and logical consistency while reducing hallucinations. Furthermore, NGDB-RAG is designed to handle large-scale graphs and supports various query types, including temporal, spatial, and numerical reasoning, ensuring scalability and expressiveness in practical applications.

Joint Learning Framework. The joint learning framework of NGDBs and LLMs employs a co-training approach where both systems share parameters or representation spaces to enable collaborative learning. Improvements in one component positively influence the other, creating a feedback loop that enhances the overall system. The combined training objective is expressed as: $L_{\text{total}} = L_{\text{LLM}} + \lambda L_{\text{NGDB}}$. In this equation, L_{LLM} represents the loss associated with the language model, typically the cross-entropy loss for next-token prediction. L_{NGDB} denotes the loss related to NGDB reasoning tasks, such as the error between predicted and true query answers. The hyperparameter λ controls the balance between the two loss components. The objective of this joint training is to improve the reasoning capabilities of the NGDB while enhancing the LLM performance.

Future work aims to develop the co-training framework further to enable simultaneous training of NGDB reasoning engines and LLMs, ensuring parameter sharing and collaborative learning. Efforts are also being made to refine the combined loss function to balance language modeling and reasoning

tasks better, enhancing both components' performance. Integration modules are being developed to incorporate NGDB reasoning results into LLMs through text-based and vector-based methods. These advancements are expected to create a unified system capable of performing advanced reasoning and generating high-quality, contextually accurate text.

11 Challenge 10: Smart Neural Graph Databases

Benefited from its rich functionalities, Agentic NGDB offers a wide range of applications across domains:

• **Autonomous Data Management**: Agentic NGDB can autonomously manage complex datasets, optimize query execution, and organize storage structures without human intervention. This is particularly useful in large-scale systems where manual optimization is impractical.

• **Personalized Recommendations**: Through continuous learning, Agentic NGDB can provide real-time personalized recommendations by analyzing user preferences and graph-based relationships. This is crucial in e-commerce and social networks, where tailored experiences drive user engagement [86].

• **Complex Event Processing**: Agentic NGDB is well-suited for handling complex event processing [11], where multiple events and data streams need to be analyzed in real-time. By leveraging their semantic understanding and neural inference, Agentic NGDB can identify correlations and patterns across seemingly unrelated events, making them valuable in cybersecurity, fraud detection, and IoT systems.

12 Conclusion

Agentic Neural Graph Databases (Agentic NGDBs) represent an advancement in data management, building on traditional graph databases and Neural Graph Databases (NGDBs) by introducing autonomy, continuous learning, and advanced reasoning.

This paper identifies ten key challenges to realizing Agentic NGDBs, including semantic representation, abductive reasoning, generalization across query types, scalability, privacy, and integration with foundation models like large language models (LLMs). Ensuring compatibility with traditional databases, grounding knowledge in vectors, and developing distributed systems are essential for achieving robust and scalable solutions.

By overcoming these challenges, Agentic NGDBs can transform modern data-driven applications. Their ability to autonomously generate and execute queries, support continuous learning, and integrate symbolic and neural reasoning offers new possibilities in autonomous data management, personalized recommendations, and complex event processing. These advancements promise to redefine how we manage, query, and reason over interconnected data for the future.

References

[1] Justin J Miller. Graph database applications and concepts with neo4j. In *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*, volume 2324, pages 141–147, 2013.

[2] Alin Deutsch, Yu Xu, Mingxi Wu, and Victor E. Lee. Tigergraph: A native mpp graph database. *ArXiv*, abs/1901.08248, 2019.

[3] Maciej Besta, Patrick Iff, Florian Scheidl, Kazuki Osawa, Nikoli Dryden, Michal Podstawski, Tiancheng Chen, and Torsten Hoefler. Neural graph databases. In Bastian Rieck and Razvan

Pascanu, editors, *Learning on Graphs Conference, LoG 2022, 9-12 December 2022, Virtual Event*, volume 198 of *Proceedings of Machine Learning Research*, page 31. PMLR, 2022.

[4] Hongyu Ren, Mikhail Galkin, Michael Cochez, Zhaocheng Zhu, and Jure Leskovec. Neural graph reasoning: Complex logical query answering meets graph databases. *arXiv preprint arXiv:2303.14617*, 2023.

[5] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[6] Alberto García-Durán and Mathias Niepert. Kblrn: End-to-end learning of knowledge base representations with latent, relational, and numerical features. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 372–381. AUAI Press, 2018.

[7] Yankai Lin, Zhiyuan Liu, and Maosong Sun. Knowledge representation learning with entities, attributes and relations. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 2866–2872. IJCAI/AAAI Press, 2016.

[8] Caglar Demir, Michel Wiebesiek, Renzhong Lu, Axel-Cyrille Ngonga Ngomo, and Stefan Heindorf. Litcqd: Multi-hop reasoning in incomplete knowledge graphs with numeric literals. In Danai Koutra, Claudia Plant, Manuel Gomez Rodriguez, Elena Baralis, and Francesco Bonchi, editors, *Machine Learning and Knowledge Discovery in Databases: Research Track - European Conference, ECML PKDD 2023, Turin, Italy, September 18-22, 2023, Proceedings, Part III*, volume 14171 of *Lecture Notes in Computer Science*, pages 617–633. Springer, 2023.

[9] Maarten Sap, Ronan Le Bras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A. Smith, and Yejin Choi. ATOMIC: an atlas of machine commonsense for if-then reasoning. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 3027–3035. AAAI Press, 2019.

[10] Hongming Zhang, Xin Liu, Haojie Pan, Yangqiu Song, and Cane Wing-Ki Leung. ASER: A large-scale eventuality knowledge graph. In Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen, editors, *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 201–211. ACM / IW3C2, 2020.

[11] Jiaxin Bai, Xin Liu, Weiqi Wang, Chen Luo, and Yangqiu Song. Complex query answering on eventuality knowledge graph with implicit logical constraints. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[12] Changlong Yu, Weiqi Wang, Xin Liu, Jiaxin Bai, Yangqiu Song, Zheng Li, Yifan Gao, Tianyu Cao, and Bing Yin. FolkScope: Intention knowledge graph construction for E-commerce commonsense discovery. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1173–1191, Toronto, Canada, July 2023. Association for Computational Linguistics.

[13] Changlong Yu, Xin Liu, Jefferson Maia, Tianyu Cao, Laurence (Yang) Li, Yifan Gao, Yangqiu Song, Rahul Goutam, Haiyang Zhang, Bing Yin, and Zheng Li. Cosmo: A large-scale e-commerce common sense knowledge generation and serving system at amazon. 2024.

[14] Jiaxin Bai, Zhaobo Wang, Junfei Cheng, Dan Yu, Zerui Huang, Weiqi Wang, Xin Liu, Chen Luo, Qi He, Yanming Zhu, et al. Intention knowledge graph construction for user intention relation modeling. *arXiv preprint arXiv:2412.11500*, 2024.

[15] Jiaxin Bai, Yicheng Wang, Tianshi Zheng, Yue Guo, Xin Liu, and Yangqiu Song. Advancing abductive reasoning in knowledge graphs through complex logical hypothesis generation. In *In Proceedings of the 62st Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics*, 2024.

[16] Zihao Wang, Hang Yin, and Yangqiu Song. Benchmarking the Combinatorial Generalizability of Complex Query Answering on Knowledge Graphs. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 1, December 2021.

[17] Zihao Wang, Hang Yin, and Yangqiu Song. Logical queries on knowledge graphs: Emerging interface of incomplete relational data. *IEEE Data Eng. Bull.*, 2022.

[18] Lihui Liu, Zihao Wang, and Hanghang Tong. Neural-symbolic reasoning over knowledge graphs: A survey from a query perspective. *arXiv preprint arXiv:2412.10390*, 2024.

[19] Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. *Advances in neural information processing systems*, 31, 2018.

[20] H Ren, W Hu, and J Leskovec. Query2box: Reasoning Over Knowledge Graphs In Vector Space Using Box Embeddings. In *International Conference on Learning Representations (ICLR)*, 2020.

[21] Lihui Liu, Boxin Du, Heng Ji, ChengXiang Zhai, and Hanghang Tong. Neural-Answering Logical Queries on Knowledge Graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1087–1097, 2021.

[22] Hongyu Ren and Jure Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems*, 33:19716–19726, 2020.

[23] Zihao Wang, Weizhi Fei, Hang Yin, Yangqiu Song, Ginny Wong, and Simon See. Wasserstein-fisher-rao embedding: Logical query embeddings with local comparison and global transport. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13679–13696, 2023.

[24] Hang Yin, Zihao Wang, and Yangqiu Song. Meta operator for complex query answering on knowledge graphs. *arXiv preprint arXiv:2403.10110*, 2024.

[25] Hang Yin, Zihao Wang, and Yangqiu Song. Rethinking existential first order queries and their inference on knowledge graphs. In *The Twelfth International Conference on Learning Representations*, 2024.

[26] Zihao Wang, Yangqiu Song, Ginny Y Wong, and Simon See. Logical message passing networks with one-hop inference on atomic formulas. *arXiv preprint arXiv:2301.08859*, 2023.

[27] Haoran Luo, Yuhao Yang, Gengxian Zhou, Yikai Guo, Tianyu Yao, Zichen Tang, Xueyuan Lin, Kaiyang Wan, and others. NQE: N-ary Query Embedding for Complex Query Answering over Hyper-relational Knowledge Graphs. *arXiv preprint arXiv:2211.13469*, 2022.

[28] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE transactions on knowledge and data engineering*, 29(12):2724–2743, 2017.

[29] Yushi Bai, Xin Lv, Juanzi Li, and Lei Hou. Answering Complex Logical Queries on Knowledge Graphs via Query Computation Tree Optimization. In *Proceedings of the 40th International Conference on Machine Learning*, pages 1472–1491. PMLR, July 2023. ISSN: 2640-3498.

[30] Jiaxin Bai, Tianshi Zheng, and Yangqiu Song. Sequential query encoding for complex query answering on knowledge graphs. *Transactions on Machine Learning Research*, 2023.

[31] Martin S Olivier. Database privacy: balancing confidentiality, integrity and availability. *ACM SIGKDD Explorations Newsletter*, 4(2):20–27, 2002.

[32] Elisa Bertino, Gabriel Ghinita, Ashish Kamra, et al. Access control for databases: Concepts and systems. *Foundations and Trends® in Databases*, 3(1–2):1–148, 2011.

[33] Ted E Senator, Henry G Goldberg, Alex Memory, William T Young, Brad Rees, Robert Pierce, Daniel Huang, Matthew Reardon, David A Bader, Edmond Chow, et al. Detecting insider threats in a real corporate database of computer usage activity. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1393–1401, 2013.

[34] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655, 2015.

[35] Suntherasvaran Murthy, Asmidar Abu Bakar, Fiza Abdul Rahim, and Ramona Ramli. A comparative study of data anonymization techniques. In *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, pages 306–309. IEEE, 2019.

[36] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International journal of uncertainty, fuzziness and knowledge-based systems*, 10(05):557–570, 2002.

[37] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *Acm transactions on knowledge discovery from data (tkdd)*, 1(1):3–es, 2007.

[38] Iqra Basharat, Farooque Azam, and Abdul Wahab Muzaffar. Database security and encryption: A survey study. *International Journal of Computer Applications*, 47(12), 2012.

[39] Cynthia Dwork. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer, 2006.

[40] Wanyu Lin, Shengxiang Ji, and Baochun Li. Adversarial attacks on link prediction algorithms based on graph neural networks. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 370–380, 2020.

[41] Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo. Structural attack against graph based android malware detection. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pages 3218–3235, 2021.

[42] Qi Hu, Haoran Li, Jiaxin Bai, Zihao Wang, and Yangqiu Song. Privacy-preserved neural graph databases. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1108–1118, 2024.

[43] Hao Fang, Yixiang Qiu, Hongyao Yu, Wenbo Yu, Jiawei Kong, Baoli Chong, Bin Chen, Xuan Wang, Shu-Tao Xia, and Ke Xu. Privacy leakage on dnns: A survey of model inversion attacks and defenses. *arXiv preprint arXiv:2402.04013*, 2024.

[44] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S Yu, and Xuyun Zhang. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys (CSUR)*, 54(11s):1–37, 2022.

[45] Congzheng Song and Ananth Raghunathan. Information leakage in embedding models. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 377–390, 2020.

[46] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Wenqi Wei, and Lei Yu. Effects of differential privacy and data skewness on membership inference vulnerability. In *2019 First IEEE international conference on trust, privacy and security in intelligent systems and applications (TPS-ISA)*, pages 82–91. IEEE, 2019.

[47] Yue Wang, Cheng Si, and Xintao Wu. Regression model fitting under differential privacy and model inversion attack. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[48] Qi Hu and Yangqiu Song. Independent distribution regularization for private graph embedding. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 823–832, 2023.

[49] Qi Hu, Weifeng Jiang, Haoran Li, Zihao Wang, Jiaxin Bai, Qianren Mao, Yangqiu Song, Lixin Fan, and Jianxin Li. Fedcqa: Answering complex queries on multi-source knowledge graphs via federated learning. *CoRR*, 2024.

[50] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78(110):1–108, 1998.

[51] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur)*, 51(4):1–35, 2018.

[52] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

[53] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[54] Hang Yin, Zihao Wang, Weizhi Fei, and Yangqiu Song. EFO_{k}-CQA: Towards Knowledge Graph Complex Query Answering beyond Set Operation, July 2023. arXiv:2307.13701 [cs].

[55] Hongyu Ren, Hanjun Dai, Bo Dai, Xinyun Chen, Denny Zhou, Jure Leskovec, and Dale Schuurmans. Smore: Knowledge graph completion and multi-hop reasoning in massive knowledge graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1472–1482, 2022. arXiv:2110.14890 [cs].

[56] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[57] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008.

[58] Arijit Khan. Knowledge graphs querying. *ACM SIGMOD Record*, 52(2):18–29, 2023.

[59] Yuanyuan Tian. The world of graph databases from an industry perspective. *ACM SIGMOD Record*, 51(4):60–67, 2023.

[60] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, et al. Manu: a cloud native vector database management system. *arXiv preprint arXiv:2206.13843*, 2022.

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

[62] Ali Ben Ammar. Graph database partitioning: A study. In *2016 7th International Conference on Information, Intelligence, Systems & Applications (IISA)*, pages 1–9. IEEE, 2016.

[63] Zhisong Fu, Zhengwei Wu, Houyi Li, Yize Li, Min Wu, Xiaojie Chen, Xiaomeng Ye, Benquan Yu, and Xi Hu. Geabase: A high-performance distributed graph database for industry-scale applications. *International Journal of High Performance Computing and Networking*, 15(1-2):12–21, 2019.

[64] Changji Li, Hongzhi Chen, Shuai Zhang, Yingqian Hu, Chao Chen, Zhenjie Zhang, Meng Li, Xiangchen Li, Dongqing Han, Xiaohui Chen, et al. Bytegraph: a high-performance distributed graph database in bytedance. *Proceedings of the VLDB Endowment*, 15(12):3306–3318, 2022.

[65] Ricky Sun and Jamie Chen. Design of highly scalable graph database systems without exponential performance degradation. In *Proceedings of the International Workshop on Big Data in Emergent Distributed Environments*, pages 1–6, 2023.

[66] Inc. Neo4j. Neo4j. `https://neo4j.com`, 2023. Accessed: 2024-10-28.

[67] Inc. TigerGraph. Tigergraph. `https://www.tigergraph.com`, 2023. Accessed: 2024-10-28.

[68] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2614–2627, 2021.

[69] Qdrant Team. Qdrant. `https://qdrant.tech`, 2023. Accessed: 2024-10-28.

[70] SeMI Technologies. Weaviate. `https://weaviate.io`, 2023. Accessed: 2024-10-28.

[71] Sarra Bouhenni, Said Yahiaoui, Nadia Nouali-Taboudjemat, and Hamamache Kheddouci. A survey on distributed graph pattern matching in massive graphs. *ACM Computing Surveys (CSUR)*, 54(2):1–35, 2021.

[72] Lingkai Meng, Yu Shao, Long Yuan, Longbin Lai, Peng Cheng, Xue Li, Wenyuan Yu, Wenjie Zhang, Xuemin Lin, and Jingren Zhou. A survey of distributed graph algorithms on massive graphs. *ACM Computing Surveys*, 57(2):1–39, 2024.

[73] Qihui Zhou, Peiqi Yin, Xiao Yan, Changji Li, Guanxian Jiang, and James Cheng. Atom: An efficient query serving system for embedding-based knowledge graph reasoning with operator-level batching. *Proceedings of the ACM on Management of Data*, 2(4):1–29, 2024.

[74] Ye Yuan, Bo Tang, Tianfei Zhou, Zhiwei Zhang, and Jianbin Qin. nsdb: Architecting the next generation database by integrating neural and symbolic systems. *Proceedings of the VLDB Endowment*, 17(11):3283–3289, 2024.

[75] Zhanhao Zhao, Shaofeng Cai, Haotian Gao, Hexiang Pan, Siqi Xiang, Naili Xing, Gang Chen, Beng Chin Ooi, Yanyan Shen, Yuncheng Wu, et al. Neurdb: On the design and implementation of an ai-powered autonomous database. *arXiv preprint arXiv:2408.03013*, 2024.

[76] Shu Liu, Asim Biswal, Audrey Cheng, Xiangxi Mo, Shiyi Cao, Joseph E Gonzalez, Ion Stoica, and Matei Zaharia. Optimizing llm queries in relational workloads. *arXiv preprint arXiv:2403.05821*, 2024.

[77] Huanchen Zhang, Yihao Liu, and Jiaqi Yan. Cost-intelligent data analytics in the cloud. *arXiv preprint arXiv:2308.09569*, 2023.

[78] Feifei Li. Cloud-native database systems at alibaba: Opportunities and challenges. *Proceedings of the VLDB Endowment*, 12(12):2263–2272, 2019.

[79] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1433–1445, New York, NY, USA, 2018. Association for Computing Machinery.

[80] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J. Smola, and Zheng Zhang. Deep graph library: Towards efficient and scalable deep learning on graphs. *CoRR*, abs/1909.01315, 2019.

[81] Renzo Angles, Marcelo Arenas, Pablo Barcelo, Peter Boncz, George Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan Sequeda, Oskar van Rest, and Hannes Voigt. G-core: A core for future graph query languages. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1421–1432, New York, NY, USA, 2018. Association for Computing Machinery.

[82] Wanjun Zhong, Jingjing Xu, Duyu Tang, Zenan Xu, Nan Duan, Ming Zhou, Jiahai Wang, and Jian Yin. Reasoning over semantic-level graph for fact checking. *arXiv preprint arXiv:1909.03745*, 2019.

[83] Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, 2014.

[84] Cheng Jiayang, Lin Qiu, Chunkit Chan, Xin Liu, Yangqiu Song, and Zheng Zhang. Eventground: Narrative reasoning by grounding to eventuality-centric knowledge graphs. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 6622–6642, 2024.

[85] Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. Dense x retrieval: What retrieval granularity should we use? *arXiv preprint arXiv:2312.06648*, 2023.

[86] Jiaxin Bai, Chen Luo, Zheng Li, Qingyu Yin, and Yangqiu Song. Understanding inter-session intentions via complex logical reasoning, 2024.

It's FREE to join!

TCDE

tab.computer.org/tcde/

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

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

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

Join TCDE via Online or Fax

ONLINE: Follow the instructions on this page:

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

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

Name _____

IEEE Member # _____

Mailing Address _____

_____

Country _____

Email _____

Phone _____

TCDE Mailing List

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

Membership Questions?

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

TCDE Chair

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

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

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