# Towards Automated Cross-domain Exploratory Data Analysis through Large Language Models

Jun-Peng Zhu
East China Normal University & PingCAP

Boyan Niu
PingCAP, China

Peng Cai
East China Normal University, China

Zheming Ni
PingCAP, China

Jianwei Wan
PingCAP, China

Kai Xu
PingCAP, China

Jiajun Huang
PingCAP, China

Shengbo Ma
PingCAP, China

Bing Wang
PingCAP, China

Xuan Zhou
East China Normal University, China

Guanglei Bao
PingCAP, China

Donghui Zhang
PingCAP, China

Liu Tang
PingCAP, China

Qi Liu
PingCAP, China

## ABSTRACT

Exploratory data analysis (EDA), coupled with SQL, is essential for data analysts involved in data exploration and analysis. However, data analysts often encounter two primary challenges: (1) the need to craft SQL queries skillfully and (2) the requirement to generate suitable visualization types that enhance the interpretation of query results. Due to its significance, substantial research efforts have been made to explore different approaches to address these challenges, including leveraging large language models (LLMs). However, existing methods fail to meet real-world data exploration requirements primarily due to (1) complex database schema, (2) unclear user intent, (3) limited cross-domain generalization capability, and (4) insufficient end-to-end text-to-visualization capability.

This paper presents TiInsight, an automated SQL-based cross-domain exploratory data analysis system. First, we propose a hierarchical data context (i.e., HDC), which leverages LLMs to summarize the contexts related to the database schema, which is crucial for open-world EDA systems to generalize across data domains. Second, the EDA system is divided into four components (i.e., stages): HDC generation, question clarification and decomposition, text-to-SQL generation (i.e., TiSQL), and data visualization (i.e., TiChart). Finally, we implemented an end-to-end EDA system with a user-friendly GUI in the production environment at PingCAP. We have also open-sourced all APIs of TiInsight to facilitate research within the EDA community. Through extensive evaluations by a real-world user study, we demonstrate that TiInsight offers remarkable performance compared to human experts. Additionally, TiSQL achieves an execution accuracy of 86.3% on the Spider dataset when using GPT-4. It also attains an execution accuracy of 60.98% on the Bird test dataset.

**PVLDB Reference Format:**
Jun-Peng Zhu, Boyan Niu, Peng Cai, Zheming Ni, Jianwei Wan, Kai Xu, Jiajun Huang, Shengbo Ma, Bing Wang, Xuan Zhou, GuangLei Bao, Donghui Zhang, Liu Tang, Qi Liu. Towards Automated Cross-domain Exploratory Data Analysis through Large Language Models. PVLDB, 18(12): 5086 - 5099, 2025.
doi:10.14778/3750601.3750629

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://www.tiinsight.chat/.

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750629

## 1 INTRODUCTION

### 1.1 Problem Statement

Exploratory data analysis (EDA) [36, 50, 51, 53, 82–85], coupled with SQL, assumes a crucial role for data analysts engaged in data exploration and analysis. The EDA utilizes SQL to construct queries that extract vital information from databases. Generally, it involves "hands-on" interaction with a dataset, where users iteratively apply analysis actions (e.g., filtering, aggregations, sorting, visualizations), generating statistical insights, and constructing comprehensive views and reports. The EDA process facilitates a deeper understanding of the data, revealing latent patterns and trends that offer valuable insights to guide subsequent decision-making endeavors. In particular, the user explores a dataset D = {databases, schema, tuples}, where *databases* refer to the data sources the user wants to explore, which may encompass multiple databases; *schemas* refer to the database schema, encompassing tables, indexes, views, and other related elements; *tuples* represent individual records of data. The data analyst then performs a series of data analysis operations $s_1, dv_1, s_2, dv_2, \ldots, s_n, dv_n$, where $s_i$ indicates an SQL statement and $dv_j$ represents the visualization of the results. The user determines the next steps after executing $s_t$ and $dv_t$. The EDA faced two key technical challenges: (1) proficiency in SQL and (2) generating appropriate visualization types.

### 1.2 Limitations of Prior Art

There are many state-of-the-art (SOTA) approaches to address both challenges. Specifically, regarding SQL proficiency, there are numerous SOTA text-to-SQL approaches, which effectively reduce the complexity of crafting SQL queries for data analysts. However, in the EDA context, SOTA text-to-SQL approaches still have their limitations:

**(1) Complex database schema**. In real-world EDA scenarios, the data to be explored is typically stored in databases with complex schema [32, 43, 79]. For example, in financial data analysis, each database contains numerous wide tables with hundreds of columns [32, 79]. The schema complexity far exceeds that of existing benchmarks, such as Spider [78]. Existing text-to-SQL methods struggle to handle such complex scenarios, as evidenced by low accuracy [79] on this dataset. Moreover, large language model-based methods must consider context window limits [54, 55, 70] when constructing prompts, significantly impacting accuracy in complex scenarios. In the worst-case scenario, a query involving numerous tables and hundreds of columns may exceed the context window limits during prompt construction, preventing the method from functioning properly. On the other hand, the large number of tables and columns increases the context length, which can significantly decrease accuracy. In real-world customer scenarios at PingCAP, users define numerous abbreviations that may refer to a business, a location, or other entities. This further increases the complexity of the schema, requiring more domain knowledge.

**(2) Unclear user intent**. In real-world EDA scenarios, users often find it challenging to express their thoughts in natural language, which makes it difficult to convey their intent [32, 36]. This challenge is evident in users' inability to formulate clear intent parameters. For example, in the customer scenarios of PingCAP, customers often ask, "What is the growth rate?". This query may lack a time parameter; a more precise formulation would be, "What is the growth rate for PC in the current year?". Additionally, user queries frequently include abbreviations [79]; for example, "DoD analysis for daily bills.". In the worst-case scenario, users may struggle to clearly articulate their intent at the beginning of a data analysis task. In many cases, the user's natural language (NL) intentions cannot be captured by a single SQL statement [32, 59, 72]. Accurately addressing these queries requires not only the semantic parsing capabilities of large language models but also robust data analysis skills to infer intent beyond the explicit query [36]. Existing SOTA text-to-SQL approaches struggle to generate SQL in this context, let alone address accuracy issues [32, 33, 59].

**(3) Limited cross-domain generalization**. Existing text-to-SQL methods [43, 79] are designed for specific domains, which results in poor execution accuracy when applied to a different data domain, requiring fine-tuning for each domain. In an end-to-end enterprise-grade system, this becomes virtually impossible. Furthermore, fine-tuning introduces substantial overhead in terms of time, space, and cost. Consequently, integrating SOTA text-to-SQL approaches into an end-to-end EDA system proves challenging. For example, in the customer scenarios of PingCAP, there are applications in finance, retail, and other industries, but SOTA methods lack cross-domain generalization capabilities. Fine-tuning for different business contexts is required, making it impractical. On the other hand, fine-tuning requires well-labeled data [79], but many industries lack adequate labeled datasets. This further restricts the adaptability of fine-tuning methods in text-to-SQL. *It is important to note that current SOTA methods are heavily benchmark-oriented and perform poorly in real-world cross-domain text-to-SQL challenges.*

Beyond these limitations in text-to-SQL, there are also challenges in data visualization for EDA systems:

**(1) Insufficient end-to-end automated text-to-visualization capability**. The existing EDA systems typically offer three primary visualization methods: ① manual visualization [1, 10, 13, 67], where the user selects visualization types for a given dataset on selected attributes. ② natural language description [1, 36], where the user specifies visualizations, such as "help me create a line chart to visualize the sales of PC machines". These methods typically either specify the type of visualization in text or directly generate visualizations from text, whereas our scenario requires a process involving text-to-SQL execution followed by data-to-chart generation. ③ table-to-chart [48, 81], where the system automatically generates appropriate charts based on the dataset, a process that is often highly complex. The majority of research in this area concentrates on exploring multi-dimensional data. Most importantly, these methods generally visualize attributes pre-selected by the user or autonomously suggest interesting attributes from the dataset. This approach differs significantly from our scenario.

**(2) Overly complex table-to-chart recommendation process**. Existing methods [48, 71, 81] frequently employ complex algorithms, such as reinforcement learning [81], significantly increasing system complexity. Some research [41, 73] requires users to master specific visualization query languages, which significantly restricts the applicability of these methods in end-to-end systems. In numerous user studies, we observed that simple visualization types, such as pie, bar, and line charts, were preferred during data exploration, while more complex visualizations hindered users' ability to extract insights. In particular, data analysts have accumulated numerous heuristic rules for data visualizations, but these lack a straightforward way of applying them to visualization recommendations. The advent of LLMs presents new opportunities for rule-based visualization.

## 1.3 Key Technical Challenges

Enabling an end-to-end SQL-based automated EDA system offers significant benefits for data exploration and analysis. However, there is no free lunch in this context. There are three major **challenges** that need to be addressed:

**C1. How to mitigate the impact of a complex database schema and unclear user intent?** This is primarily due to the prevalence of complex database schemas and ambiguous user intentions in real-world EDA scenarios, significantly reducing EDA task accuracy. However, addressing these challenges without incurring additional costs remains highly difficult.

**C2. How to improve generalization and adaptability across different data domains?** In industrial practice at PingCAP, various business data scenarios, such as finance and retail, are commonly encountered. EDA methods are typically tailored to specific data domains, leading to the absence of a universal EDA system that can adapt to diverse scenarios. Therefore, developing automated EDA systems that provide generalization and adaptability remains a significant challenge.

**C3. How to design an automated EDA system that translates NL into visualization results with minimal human intervention?** Efficient end-to-end exploratory analysis requires the design of a dedicated system. Uncontrollable natural language inputs, unknown exploratory datasets, and complex domain-specific knowledge amplify the complexity of designing EDA systems. It is a key research focus to investigate a feasible automated EDA pipeline in data analysis.

## 1.4 Proposed Approach

In this paper, we propose TiInsight, a SQL-based, enterprise-grade, multi-stage automated cross-domain exploratory analysis system utilizing large language models. To address the first two challenges, we propose HDC, a hierarchical data context approach that leverages LLMs to summarize the database schema context, which is vital for open-world EDA systems to generalize across exploratory data domains. First, TiInsight summarizes database-related context information using HDC, such as column descriptions and table summaries. The primary data entities representing the database are subsequently abstracted. Ultimately, a database summary is generated that encompasses the entity descriptions, the tables associated with those entities, the key attributes of the entities, and the columns to which these entities may be mapped. These data contexts are stored in vector databases (e.g., ChromaDB [3], Pinecone [12], TiDB Vector [17]). Drawing on this data context, TiInsight also recommends several exploratory questions to facilitate the user's understanding of the data.

TiInsight is divided into four components (i.e., stages): HDC generation, question clarification and decomposition, text-to-SQL generation (i.e., TiSQL), and data visualization (i.e., TiChart). Subsequently, TiInsight processes the natural language questions posed by users. Initially, it clarifies the user's questions within the hierarchical data context, generating a precise data exploration question. It then determines whether the question can be answered with a single SQL statement. If not, it decomposes the complex question into multiple sub-questions and provides a detailed description of each sub-question. These clarification questions are subsequently provided to the TiSQL, which generates the corresponding SQL statement and integrates it with the self-refinement chain for further improving text-to-SQL accuracy. Finally, TiInsight generates the visualization chart utilizing the integrated TiChart. An end-to-end EDA system with a user-friendly GUI has been developed and deployed in the production environment at PingCAP.

## 1.5 Key Contributions and Outline

To summarize, this paper makes the following contributions:

(1) We investigate how to design key components in an SQL-based automated EDA system and analyze why existing methods are inadequate for enterprise-grade automated EDA systems.

(2) We propose a hierarchical data context approach called HDC. TiInsight with HDC is designed to facilitate data exploration of complex enterprise-grade datasets and enable cross-domain data analysis. Furthermore, we introduce a map-reduce framework into TiInsight to facilitate large-scale and complex database schemas during the HDC generation. The generated data context is stored in vector databases. (§3)

(3) Building on HDC, we propose a cross-domain text-to-SQL method named TiSQL (§4). The TiSQL explores various prompting techniques to enhance the accuracy of text-to-SQL generation. Building on this foundation, we introduce TiChart (§5), which implements a rule-based visualization recommendation.

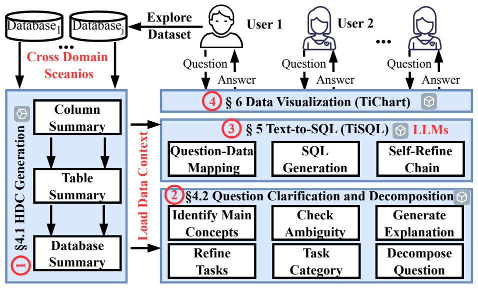


**Figure 1: Overall architecture of TiInsight.**

(4) We implement and deploy an automated end-to-end EDA system with a user-friendly GUI in the production environment at PingCAP. Simultaneously, we have also open-sourced all APIs[1] of TiInsight to facilitate research within the EDA community.

(5) We evaluate the performance of TiInsight (§6). The experimental results demonstrate that TiInsight achieved remarkable performance. In particular, TiSQL achieves 86.3% execution accuracy in the Spider dataset using GPT-4 without requiring additional model training or fine-tuning. It also demonstrates state-of-the-art performance on the Bird dataset.

## 2 THE OVERVIEW OF TIINSIGHT

This section gives a high-level introduction of TiInsight. As shown in Figure 1, TiInsight has four functional components to facilitate cross-domain exploratory data analysis tasks.

① **HDC generation**. We perform cross-domain exploratory data analysis. To address **C1** and **C2**, we design HDC generation, which uses LLMs to generate the hierarchical data context of the database schema efficiently. The LLMs are powerful for summarizing text content from massive datasets. It can capture the most significant content from the database schema to guide the subsequent exploratory process. This component allows us to interpret the data provided by the user across various hierarchies of the database schema. In Section §3.1, we introduce how to summarize the information provided by the database schema from different perspectives, including columns, tables, and databases, and how to address complex database schemas of real-world scenarios.

② **Question Clarification and Decomposition**. We decode and clarify the questions of different users. To address **C1** and **C2**, we further implement a solution to clarify questions using large language models. Specifically, this component resolves ambiguous user intentions and provides possible explanations. Subsequently, we refine the current user task. Simultaneously, TiInsight assesses whether the user task should be decomposed into a series of sub-tasks. Section §3.2 introduces how large language models can effectively clarify and decompose tasks.

③ **Text-to-SQL (TiSQL)**. We employ TiSQL to translate user questions into SQL statements. First, TiSQL needs to map the user question to specific tables and columns while providing the values for SQL conditional statements. In this process, we propose both coarse-grained and fine-grained mapping methods. Subsequently, TiSQL employs the large language model to generate SQL statements. We carefully design the prompt of TiSQL. However, the SQL statements

[1] https://docs.pingcap.com/tidbcloud/use-chat2query-api

generated in this step still contain errors. Finally, TiSQL employs a self-refine chain to enhance the generated SQL. In Section §4, we elaborate on using a large language model combined with prompt engineering to generate SQL statements without fine-tuning.

④ **Data Visualization (TiChart)**. In TiInsight, we utilize TiChart to visualize the user's query results to the fullest extent. A key challenge in the data visualization process arises from the complexity of user tasks, which can generate numerous sub-tasks, leading to uncertainty in the final data query (e.g., attributes and data). This greatly amplifies the complexity of visualizing the resulting data. To address this challenge, TiChart combines a rule-based approach with an LLM to generate clear and easy-to-understand visualizations. In Section §5, we provide a detailed discussion on how rules can be integrated into LLMs to improve the generation of visualization results.

## 3 HIERARCHICAL DATA CONTEXT AND QUESTION CLARIFICATION AND DECOMPOSITION

In this section, we first present a detailed implementation of the hierarchical data context (§3.1). Then, we give the design for the clarification and decomposition of the questions (§3.2).

### 3.1 Hierarchical Data Context

The hierarchical data context (i.e., HDC) forms the foundation of TiInsight, with all subsequent component designs built upon it. The hierarchical data context is designed primarily to improve the understanding of the database schema, which includes column information, table information, table-to-table relationships, and overall database details. This design is derived from experience during the proof-of-concept (POC) stage at PingCAP. In PingCAP's data analysis business operations, data analysts first collect and organize relevant database schema information when encountering unfamiliar datasets to gain a better understanding of the database. We leveraged this experience and fully automated the process using LLMs. The HDC comprises three components: column summary (§3.1.1), table summary (§3.1.2), and database summary (§3.1.3).

#### 3.1.1 *Column Summary.*

Columns (a.k.a., fields or attributes) are the fundamental components of a database schema. In real-world scenarios, two main challenges arise with columns: (1) column names are often abbreviated, and (2) there can be a very large number of columns, sometimes reaching thousands, and each column contains a lot of data. To address the first technical challenge, we propose building an embedding service to store domain knowledge (a.k.a., terms) in a vector database. When the LLMs generate a summary of column descriptions, they first retrieve relevant domain knowledge from the vector database, dynamically constructing prompts as supplementary explanations.

A column-by-column summarization approach would result in high latency due to repeated calls to the large language model. Additionally, using the LLM for each column individually is inefficient, leading to significant token wastage and increased costs. To address the second challenge, we propose a parallel, grouping-based approach to mitigate this issue by vertically partitioning the table. First, HDC groups columns into fixed sets (i.e., vertical partitioning). The column number of each group is empirically determined based on the context window limitations of LLMs. For example, empirically, groups contain 40 columns for GPT-4 and 80 columns for other models (i.e., GPT-4o, GPT-4o mini, and the Claude series LLMs in this paper) in the PingCAP production environment. It is noted that the value is model-specific. Each group is subsequently allocated threads from a thread pool before the LLM initiates the

**Algorithm 1:** Table Description Map-Reduce Framework.

```
Function Map(table, t):
    batch_table_description = [ ];
    blocks = split(table);
    for b in blocks do
        batch_column_description =
          retrieve_column_description(b);
        mPrompt =
          preparePrompt(batch_column_description, table);
        mResult = LLM(mPrompt, t);
        batch_table_description.append(mResult);
    return batch_table_description;
Function Reduce(batch_table_description, table, t,
  max_tokens):
    if len(batch_table_description) == 1 then
        rPrompt =
          preparePrompt(batch_table_description[0], table);
        return LLM(rPrompt, t);
    new_batch_table_description = [ ];
    current_batch = [ ];
    for description in batch_table_description do
        if get_token_count(current_batch) +
          get_token_count(description) > max_tokens then
            rPrompt = preparePrompt(current_batch, table);
            reduced_result = LLM(rPrompt, t);
            new_batch_table_description.append(reduced_result);
            current_batch = [ ];
        current_batch.append(description);
    if current_batch then
        rPrompt = preparePrompt(current_batch, table);
        reduced_result = LLM(rPrompt, t);
        new_batch_table_description.append(reduced_result);
    return Reduce(new_batch_table_description, table, t,
      max_tokens);
Function MRTableDes(table, t, max_tokens):
    mResult = Map(table, t);
    rResult = Reduce(mResult, table, t, max_tokens);
    save_table_description(table, rResult);
```

column description process. Comments may be included in the schema information provided by the schema for each column. For example, the *date* column might include a description of the date format, such as YYYY-mm-dd. After the aforementioned process, we append the column comments to each column description. Additionally, for each column, we randomly select several rows (i.e., three rows in this paper) to include in the column description, aiding the LLM in understanding the column's value type and other relevant information. Finally, the LLM receives the column content and dynamically constructs the CoT prompt to facilitate the column summary.

*3.1.2 Table Summary.* Next, we utilize the column summary obtained in Section §3.1.1 to summarize the information for each table in a table-by-table manner. The table summary consists primarily of two components: the table description and the table relationship.
**Table Description**. In real-world scenarios, tables may contain thousands of columns, each potentially with millions of rows. Encoding the table data within a prompt is impractical, so we encode only the column summaries. However, large and wide tables cannot encode all columns in a single prompt, as this often exceeds the context window limits of LLMs. Consequently, context window limits are a significant challenge in table summarization for LLMs. In this paper, we propose a map-reduce framework to summarize large tables. Algorithm 1 provides a detailed implementation of the table summary obtained using the map-reduce framework. First, it divides the wide table into multiple blocks using a vertical partitioning of columns. Then, it creates a summary for each block (lines 1-9). In this process, we retrieve the column descriptions corresponding to each block from the vector database. The prompt is constructed dynamically using this information (line 6). Finally, it uses the reduce mode to generate the table description (lines 10-27). Additionally, we save the context information of the table description obtained from the exploration to the vector database (line 31). The map-reduce framework offers two key benefits: (1) It prevents exceeding the context window limits of the LLM, and (2) It supports multi-threads in parallel summarization across different stages.

To create a comprehensive table description, we need to gather additional information, including the primary key, key attributes, table type, table entity, and natural language description. Specifically, the LLM may identify multiple primary keys, and we prompt it to select the most likely one. This primary key should better reflect the importance and uniqueness of the table. Key attributes play a vital role in understanding the purpose and content of the table. We prompt the LLM to identify and list a maximum of five key attributes from the table presented in this paper. In addition, we categorize the table type as either dimension, bridge, or fact based on its usage and role in data analysis. This classification is especially crucial for the OLAP analysis. Finally, we identify the main entity that the table focuses on.
**Table Relationship**. Table relationships identify all referential integrity relationships between a specified table and others, which helps to optimize complex queries. We need to identify the following elements: (1) the referencing table, (2) the referenced table, (3) the foreign key column, (4) the primary key column, and (5) the nature (i.e., relationship type) of the inferred relationship corresponding to the foreign key column (e.g., 1: 1, 1: N). Accurately identifying these relationships in real-world scenarios remains challenging due to the large number of tables and the complexity of their relationships. For example, in PingCAP's business, a single database may have thousands of tables, each containing thousands of columns.

**Algorithm 2:** Table Relationship.

```
1 Function TSTabRel(table, t, similar_count):
2     doc = retrieve_table_description(table);
3     relevant_tables, relevant_tables_description =
        coarse_grained_search(doc, similar_count);
4     prompt = preparePrompt(table, table.columns,
        relevant_tables_description);
5     rel = LLM(prompt, t);
6     save_table_relationship(table, rel);
```

**Algorithm 3:** Entity Extraction.

```
1 Function Entity(schema, t, N):
2     candidate_tables = topN(schema, N);
3     prompt = preparePrompt(candidate_tables,
        candidate_tables_summary);
4     entities = LLM(prompt, t);
5     save_entity_context(database, entities);
```

In this paper, based on the table descriptions, we propose a two-stage approach for identifying the table relationships to address the above challenges, as shown in Algorithm 2. To determine the referential integrity of a specific table with other tables, in **stage 1 (i.e., coarse-grained search)**, we retrieve *similar_count* tables from the vector database using the specified table description (lines 2-3). Typically, the number of *similar_count* is empirically set to 20 at the PingCAP production environment. This value was selected primarily due to the context window limitation of the LLMs. We acknowledge that these values are model-specific. Therefore, these parameters should be adjusted based on the target model's specifications. In **stage 2 (i.e., fine-grained explore)**, we input the specified table along with the table retrieved in stage 1 into the LLMs for fine-grained identification by CoT prompt (lines 4-5). We iterate over each table to establish an integrity relationship for all tables within the database. Given $n$ tables, the time complexity to explore all tables for each specified table is $O(n^2)$. Using a coarse-grained search reduces this complexity, allowing us to avoid exploring every table and achieve a time complexity of $O(n)$.

*3.1.3 Database Summary.* To facilitate efficient data exploration across different databases, we propose extracting representative **entities** for each database, enabling a **database summary** based on these abstracted entities. A database consists of numerous tables with complex relationships. Following the influence maximization principle [46], a table with a higher number of relationships to other tables is considered more important within the database. Conversely, for a table with very few relationships, its properties are generally insignificant. Therefore, we propose an entity extraction method that leverages the number of relationships between tables, as outlined in Algorithm 3. The algorithm first selects the top N tables (line 2) with the highest number of relationships from the current database. In the PingCAP business, the value of $N$ is empirically set to 20. This is primarily based on observations of PingCAP's business. We find that 20 tables with the largest number of relationships are sufficient to represent the core information of a database. It is worth noting that users of TiInsight should adjust this value based on the complexity of their specific business context. A prompt is then constructed using these tables and their summaries, and the LLM is employed to infer their entities (lines 3-4). Finally, the inferred entities are saved in the vector database (line 5).

For the extracted entity, we employ a CoT prompt with the LLM to infer: (1) a larger entity name that accurately represents the grouped tables and reflects their theme; (2) a larger entity summary emphasizing its purpose and the insights it contributes to a holistic understanding of the data source; (3) key attributes related to the larger entity, ensuring they are derived from the key attributes of the grouped tables; and (4) a list of the names of tables involved in the larger entity.

Finally, we infer the database summary from the entities using the LLM with a CoT prompt, which includes the following information:

- **Purpose**: Infer the purpose of the data source. Do not include details about entities.
- **Domain**: Determine the domain of the data source.
- **Business Impact**: Describe in simple terms how the data source aids in business operations, analytics, or decision-making processes.
- **Real-World Example**: Provide an example of real-world scenarios in which the data source can play a pivotal role.
- **User-Friendly Description**: Combine the results from the previous steps into a user-friendly description. Do not emphasize what it includes in the description.
- **Summary**: Write a summary of the data source based on the user-friendly description, highlighting key points, and in no more than 10 words.

## 3.2 Question Clarification and Decomposition

**Question Clarification**. Upon data import into TiInsight, our system initiates a rapid analysis to establish a hierarchical data context, enabling the user to explore the dataset effectively. A key challenge for TiInsight is that user questions can be semantically unclear. For example, in the user scenario of PingCAP, a common question like "Which product is the best?" may refer to sales, customer satisfaction, or other metrics. To address this ambiguity, we propose clarifying user questions using a hierarchical data context and establishing clear exploration intentions. Another key issue is that user questions may contain abbreviations, which can impact TiInsight to explore the dataset accurately. To address this issue, TiInsight establishes an embedding service that stores domain-specific knowledge within a vector database based on the retrieval-augmented-generation (RAG) mechanism.

TiInsight employs CoT prompts to direct LLMs in performing systematic clarifications. The main steps for question clarification are: **(1) Identifying main concepts**: analyze the task context to determine the primary concepts or variables mentioned. **(2) Checking for ambiguity**: review the identified concepts or variables to assess potential ambiguity based on the HDC in Section §3.1. **(3) Generating explanations**: for each unclear concept or variable, generate a relevant explanation grounded in the provided information. **(4) Refining the task**: use the generated explanations to clarify the task, ensuring it communicates the intended outcome and resolves any ambiguities. **(5) Providing a detailed description**: provide a detailed description of the refined task, including the objective, expected outcomes, and any relevant considerations or requirements. To demonstrate how HDC is applied in a CoT prompt, we present a segment of the prompt used for question clarification in Figure 2. Specifically, *summary* (line 3) denotes the database summary derived from HDC, while *tables_schema* (line 5) refers to the HDC related to the tables and columns within the current database. The domain knowledge, i.e., *relevant_chunked_texts* (line 6), *instructions* (line 6), and *term_sheet* (line 8), is retrieved from the vector database. It is especially important to note that TiInsight allows users to upload knowledge in specific domains by OpenAPI [58].

```
1  Act as a data analyst, your objective is to clarify the ↘
       task requirements more thoroughly in the context of ↘
       the information provided.
2  ### Database summary ###
3  %(summary)s
4  ### Key entities in the database ###
5  %(tables_schema)s
6  %(relevant_chunked_texts)s%(instructions)s
7  ---
8  Task: %(task)s%(term_sheet)s
9  ---
10 Let's think step by step as follows:
```

**Figure 2: A segment of the prompt for question clarification.**

**Question Decomposition**. Real-world questions are often too complex for TiInsight to answer fully in a single step, requiring the question to be broken down into sub-questions. A key challenge in decomposition for LLM is the lack of domain-specific knowledge. To address this, we propose an approach based on few-shot examples by similarity-based retrieval to guide LLMs in generating more effective decomposition plans. The main steps include (1) identifying potential subtasks that contribute to solving the task; (2) refining the potential subtasks to ensure they are well-defined with clear conditions for completion; and (3) for each refined subtask, providing a more detailed description that outlines the specific objective, expected outcomes, and any relevant considerations or requirements. To demonstrate the few-shot combined with HDC (i.e., line 10), we present a segment of the prompt in Figure 3. Building on this, we select decomposition task results with high user satisfaction as few-shot examples to further improve the performance of TiInsight in decomposition tasks. TiInsight employs the CoT prompt to guide LLMs in achieving question decomposition.

## 4 TISQL: TEXT-TO-SQL BASED ON HDC

In real-world scenarios, a database typically includes thousands of tables, each with hundreds or thousands of columns, and each

```
Act as a data scientist with access to a cleaned database
     of relevant data. Your objective is to break down the
      given task into smaller, actionable subtasks for
     systematic execution. Here is the information
     provided:
### Database summary ###
%(summary)s
### Key entities in the database ###
%(tables_schema)s
%(instructions)s
---
Task: %(clarified_task)s
---
%(few_shot_examples)s
Let's think step by step as follows:
```

**Figure 3: A prompt segment of question decomposition.**

**Algorithm 4:** Tables and Columns Filter.

```
1  Function TabColFilter(clarified_task, N):
2      relevant_tables = [ ];
3      necessary_tables_columns = [ ];
4      relevant_tables = retrieves(clarified_task, N);
5      blocks = split(relevant_tables);
6      map_result = Map(blocks, clarified_task);
7      necessary_tables_columns = Reduce(map_results);
8      return necessary_tables_columns;
9  Function Map(blocks, clarified_task):
10     results = [ ];
11     for block in blocks do
12         prompt = preparePrompt(block.HDC, clarified_task);
13         result = LLM(prompt, t);
14         results.append(result);
15     return results;
16 Function Reduce(map_result):
17     necessary_tables_columns = [ ];
18     for result in map_results do
19         if result not in necessary_tables_columns then
20             necessary_tables_columns.append(result);
21     return necessary_tables_columns;
```

table holds millions of records. Encoding all of this information into a prompt would exceed the context window limits of LLM. Therefore, in text-to-SQL tasks, it is essential to streamline the database schema, as an excessive number of schemas can cause errors during the schema linking [42] process. To address these challenges, we propose a schema filtering framework based on the map-reduce paradigm to filter tables and columns. The schema filtering algorithm is presented in Algorithm 4.

```
1  Act as an expert in text-to-SQL, your objective is to map
        a given task to relevant tables and columns based on
        the provided database schema and table relationships.
         Below is the information you have:
2  ### Table Schema ###
3  %(tables_schema)s
4  ### Table relationships ###
5  %(tables_relationship)s
6  %(instructions)s
7  ---
8  Task: %(clarified_task)s%(term_sheet)s
9  ---
10 Let's think step by step as follows:
```

**Figure 4: A segment of the prompt for schema filtering.**

As shown in Algorithm 4, TiSQL begins with a coarse-grained search, retrieving the top $N$ relevant tables from the vector database based on the clarified question (i.e., clarified_task) and cosine similarity (line 4). In PingCAP practice, this parameter is set to 30. This is a business-driven decision. In PingCAP's business, a single SQL statement typically involves no more than 30 tables. Then, TiSQL performs fine-grained filtering to exclude irrelevant tables and columns. To prevent the associated table and column summaries from exceeding the LLM's context limit, TiSQL divides these tables into more granular groups of related tables (line 5). Subsequently, map-reduce is employed to process the corresponding block utilizing idle threads (lines 9-21). In the map phase, the prompt (line 12) is dynamically constructed using the table and column summaries of the current group and the clarified task. The LLM, combined with the CoT prompt, is then used to identify the relevant tables and required columns (line 13). Figure 4 presents the prompt segment used in schema filtering. It is evident that, in this process, both the table schema (line 3) and the table relationships (line 5) among them are required as context. In the reduce phase, we apply the reduce mode for each fine-grained table group to merge the retrieved tables and columns (lines 16-21). TiSQL utilizes both the HDC and the map-reduce framework to streamline the LLM prompt, significantly reducing the complexity of schema linking. In particular, TiSQL does not separate the table and column recall into two stages. This approach allows TiSQL to leverage additional context from table and column summaries to better map the exploratory tasks of users.

After retrieving the relevant tables and columns, TiSQL enters the SQL generation phase, where it uses these tables and columns, along with question-SQL few-shot examples from the vector database, to build the CoT prompt and generate SQL statements dynamically. The specific steps include: **(1) Clarify the Task**: Thoroughly understand the task's specific goal, expected outcomes, and any relevant conditions. Ensure a clear understanding of the task's requirements. **(2) Identify keywords**: Extract all relevant keywords associated with tables and columns from the clarified task. **(3) Map keywords to columns and tables**: Match the extracted keywords to specific columns and tables within the schemas, determining the appropriate column and table names of each keyword. **(4) Generate SQL query**: Using the results of the mapping in step (3) and the relationships in the identified table, construct an SQL query that meets the task requirements. If no SQL query can be generated,

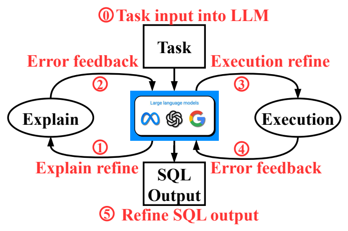


**Figure 5: The self-refine chain of TiSQL.**

leave the output empty. **(5) Rewrite SQL query**: Following the generation of the SQL query in step (4), refine it using rewriting rules: apply the *table.column* format to prevent ambiguity when multiple tables are included, verify that table and column references are accurate and correspond to the correct tables, and enclose all table and column names in backticks to ensure proper referencing and prevent syntax errors. These steps are encoded with HDC in the CoT prompt to guide the LLM in generating SQL.

The SQL statements generated from the above procedure may still contain errors; therefore, we propose a self-refinement chain, as illustrated in Figure 5. After generating SQL statements from the LLM, the results are initially fed into an explain refine process (①), where potential errors can be identified (②). The *EXPLAIN* statement is a widely used tool in databases that provides feedback on potential issues in SQL statements without having to execute them. For example, attempting to parse a non-existent table, view, or column results in an error from the database. Similarly, an invalid expression or an incorrect parameter in a query can lead to an error in the *EXPLAIN* output. For example, executing "EXPLAIN SELECT * FROM users WHERE id = 'abc'::integer;" returns the error "invalid input syntax for integer: 'abc'.". Such errors are frequently encountered in real-world scenarios. This method enables the identification and enhancement of SQL statements without the need to execute the query. After refining the SQL statements using the explain statement, certain errors may only become apparent during execution; therefore, we integrate execution refinement into the TiSQL refinement chain (③). Execute the SQL statements and relay any feedback from error information to the LLM for correction (④). The final SQL output is generated after iterating the self-refinement chain (⑤). This process prompts the LLM to self-reflect and learn from its errors, enhancing the quality of its output without requiring additional training on a wide range of text-to-SQL tasks.

## 5 TICHART: RULE-BASED VISUALIZATION

Data visualization is a key component of TiInsight, aiding users in comprehending data exploration results more effectively. In previous data analysis systems, most users needed to use d3js [10] and other tools for data visualization. However, these systems lacked an end-to-end solution with minimal human intervention. To address these limitations, we propose a rule-based data visualization approach. In the PingCAP business scenario, we collected chart visualization preferences of users and combined these rules with LLM recommendations to generate visualization output.

First, we identify the data type of each column by analyzing the sample data results. The type of data is a crucial factor in determining both visualization methods and accuracy. Different data types are suited to various visualization methods, each capable of effectively conveying patterns and information within the data. A thoughtful choice of data type can enhance the user's intuitive understanding of the core information within the data.

Second, based on the data type analysis, select an appropriate chart type from the available options to visualize the resulting data for the specified task. For example, the pie chart is suitable for displaying the proportion or distribution of categorical data. Specify 'label' for the categories and 'value' for their corresponding values. The line chart is appropriate for visualizing the relationship between two continuous variables across a constant range. The user must specify the 'x' and 'y' columns and optionally include a 'pivot_column'. The bar chart is used to compare discrete categories or groups. Specify 'x' for the categories and 'y' for the values. Optionally, a 'pivot_column' may display multiple series for each category. TiChart encodes these rules within the CoT prompt, guiding the LLM to select the appropriate type of visualization chart.

Finally, LLMs verify whether the chosen chart type is appropriate among the provided options. If inappropriate, replace it with a suitable chart type from the available options. If no suitable visual chart is available, TiChart presents the information in tabular form. TiChart integrates the above steps to achieve automated chart generation through a CoT-prompted LLM.

## 6 EXPERIMENTAL EVALUATION

In this section, we empirically evaluate the design of TiInsight using representative benchmarks. In particular, we focus mainly on the following research questions (RQs).

- **RQ1:** How accurate is TiSQL in performing text-to-SQL tasks? We focus on performance, specifically referring to execution accuracy (EX). In the SQL-based EDA system, the accuracy of SQL significantly influences the overall performance of the EDA system. (§6.2)
- **RQ2:** How effective is TiInsight in supporting user tasks? This performance includes **relevance** - Do TiInsight accurately identify and address the core aspects of the given analysis objective?, **completeness** - Does TiInsight provide necessary information to address the given analysis objective?, and **understandability** - Does TiInsight response easily interpretable, clear, and understandable for users, even with complex analysis objectives?. (§6.3)
- **RQ3:** How does latency between different components affect the effectiveness of the design in achieving the analysis objectives? (§6.4)
- **RQ4:** How do different LLMs compare in terms of costs? We focus on balancing accuracy and cost in the production environment at PingCAP. (§6.5)
- **RQ5:** In business customer production environments, how does TiInsight perform in terms of accuracy and latency? (§6.6)

### 6.1 Experimental Setup

*6.1.1 LLMs.* We support prompt-based LLMs, including gpt-4-0613 (referred to as GPT-4) [54], gpt-4o-2024-08-06 (referred to as GPT-4o) [9], gpt-4o-mini-2024-07-18 (referred to as GPT-4o mini) [8], claude-3-opus-20240229 (referred to as Claude-3 Opus) [19], claude-3-sonnet-20240229 (referred to as Claude-3 Sonnet) [19], and claude-3-haiku-20240307 (referred to as Claude-3 Haiku) [19]. We utilize

Table 1: The EX on the Spider test set.

| Classification | Methods | Test EX (%) |
|---|---|---|
| Rule-based | Duoquest [20] | 63.5 |
| PLM-based | BRIDGE v2 + BERT [47] | 64.3 |
| | T5-3B [64] | 70.1 |
| | T5-3B-PICARD [64] | 75.1 |
| | RESDSQL-3B + NatSQL [44] | 79.9 |
| LLM-based | DIN-SQL + CodeX [59] | 78.2 |
| | C3 + ChatGPT [28] | 82.3 |
| | DIN-SQL + GPT-4 [59] | 85.3 |
| | DAIL-SQL + GPT-4 [33] | **86.2** |
| | DAIL-SQL + GPT-4 + SC [33] | **86.6** |
| | **TiSQL + GPT-4 (ours)** | **86.3** |

Table 2: The EX on the Spider test set using different LLMs.

| Methods | Test EX (%) | | | | |
|---|---|---|---|---|---|
| | easy | medium | hard | extra | all |
| TiSQL + Claude-3 Opus | 91.1 | 86.8 | 80.8 | 72.5 | 84.1 |
| TiSQL + Claude-3 Haiku | 84.5 | 84.0 | 61.6 | 55.7 | 74.6 |
| TiSQL + Claude-3 Sonnet | 90.0 | 83.8 | 67.0 | 65.3 | 78.4 |
| TiSQL + GPT-4o | **91.9** | 86.2 | 79.9 | 73.1 | 83.9 |
| TiSQL + GPT-4o mini | 91.3 | 68.1 | 73.0 | 59.4 | 72.8 |
| **TiSQL + GPT-4** | 89.8 | **89.0** | **83.6** | **78.7** | **86.3** |

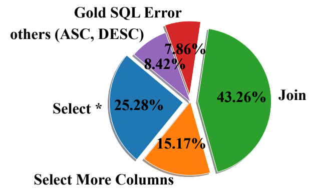


Figure 6: Spider dev result analysis.

the *text-embedding-ada-002* model of OpenAI to generate embeddings. We set the *temperature* parameter to 0, a common configuration for the product that ensures the stability of the output. Furthermore, we set the *top_p* parameter to 1. For LLMs of OpenAI, the *frequency_penalty* and *presence_penalty* are both set to 0.

*6.1.2 Environment.* We conducted extensive experiments with two pods on AWS Kubernetes. Each pod is configured with 2000 millicores of CPU and 4096 MiB of memory. The CPU model is Intel(R) Xeon(R) Platinum 8223CL, running at 3.00 GHz. The pods run Debian GNU/Linux 12 (Bookworm).

*6.1.3 Benchmark Datasets.* In our experimental setup, we utilize three datasets, including a micro-benchmark, Financial [4–7, 11, 14, 16], and two macro-benchmarks, Spider [15, 78], and Bird [2, 45]. The Spider and Bird are among the most widely used benchmarks for text-to-SQL tasks. We maintain the Financial dataset on the TiDB cloud and automate the daily update of the database at 0:00 by retrieving the latest data. The Financial dataset is the most frequently utilized by PingCAP during the proof-of-concept (POC) stage for EDA.

## 6.2 Performance of TiSQL (RQ1)

In this section, we evaluate the performance of TiSQL using the popular text-to-SQL benchmarks, Spider and Bird. We employ the execution accuracy (EX) [78] metric for the Spider. The EX measures NL2SQL performance by checking if predicted and ground-truth SQL queries produce identical result sets. The calculation of EX is presented in detail in [78]. We utilize the EX metric and the reward-based valid efficiency score (R-VES) [45] for the Bird benchmark. R-VES evaluates models based on two criteria: SQL query accuracy and runtime performance. The calculation of R-VES is presented in detail in [45]. The experimental evaluation of TiSQL is available at https://github.com/tidbcloud/tiinsight.

In Table 1, we compare the EX metric in TiSQL with those of the rule-based, PLM-based, and LLM-based text-to-SQL approaches using the Spider test dataset. The results indicate that TiSQL with GPT-4 (i.e., TiSQL + GPT-4) achieves SOTA performance, reaching 86.3%. The DAIL-SQL employs the GPT-4 model along with the self-consistency SQL refinement, achieving an accuracy of 86.6%. In comparison, TiSQL is only 0.3% lower in execution accuracy than DAIL-SQL + GPT-4 + SC. However, the self-consistency approach is time-consuming and incurs significantly higher costs than the original DAIL-SQL method. Consequently, self-consistency is not employed in TiSQL. In the published DAIL-SQL, the original DAIL-SQL remains the primary focus; however, our TiSQL achieves a performance improvement of over 0.1% compared to the original. It is noted that TiSQL does not undergo any fine-tuning process, yet its execution accuracy still exceeds that of the SOTA PLM-based method. This is primarily due to HDC, which aims to maximize the use of database schema information, thereby achieving full utilization of schema information in TiSQL. This further demonstrates the strong contextual learning and summarization capabilities of LLMs. TiSQL is designed to leverage the full capability of LLMs. On the other hand, TiSQL employs a combination of coarse-grained and fine-grained filtering methods for tables and columns to achieve more accurate recall. Ultimately, the refinement chain of TiSQL enhances execution accuracy.

In Table 2, we give the EX of various LLMs utilized in TiSQL on the Spider test dataset. The results indicate that TiSQL achieves the highest execution accuracy when utilizing GPT-4. And, in tasks of varying difficulty, TiSQL using GPT-4 has achieved SOTA execution accuracy. It is important to note that when TiSQL using the GPT-4o model, its execution accuracy outperforms all current PLM-based methods, and its execution accuracy is 2.4% lower than that of TiSQL + GPT-4.

In Table 3 and Table 4, we present the EX and R-VES results for the Bird dataset. We found that TiSQL outperforms DAIL-SQL + GPT-4 in all metrics. This demonstrates that the TiSQL outperforms DAIL-SQL + GPT-4 in handling complex text-to-SQL scenarios. The performance on R-VES is slightly inferior to that of SFT CodeS-15B. However, it is important to note that SFT CodeS-15B requires complex fine-tuning, which restricts its ability to generate SQL across diverse data domains. The results indicate that TiSQL demonstrates SOTA performance. This finding is consistent with the results obtained from the Spider dataset.

We analyze all SQL statements generated on the Spider dev dataset and identify 178 errors as false negatives, as illustrated in Figure 6. In some cases, joining more tables accounted for the largest proportion (about 43.26%), leading to more information being included in the final output. There are errors in the gold SQL and differences in sorting order and sorting sequence (among others). Some *SELECT* statements retrieve additional columns, or

Table 3: The EX on dev/test set of Bird. Results labeled with '-' indicate that we did not submit these models to https://bird-bench.github.io/ for testing; consequently, we did not report these values.

| Methods | Dev EX (%) | Test EX (%) |
|---|---|---|
| **Baselines Methods** | | |
| T5-3B [64] | 10.37 | 11.17 |
| PaLM-2 [45] | 27.38 | 33.04 |
| CodeX 175B [45] | 34.35 | 36.47 |
| ChatGPT [45] | 37.22 | 39.30 |
| ChatGPT + CoT [45] | 36.64 | 40.08 |
| Claude-2 [45] | 42.70 | 49.02 |
| GPT-4 [45] | 46.35 | 54.89 |
| DIN-SQL + GPT-4 [59] | 50.72 | 55.90 |
| DAIL-SQL + GPT-4 [33] | 54.76 | 57.41 |
| TA-SQL + GPT-4 [62] | 56.19 | 59.14 |
| SFT CodeS-7B [43] | 57.17 | 59.25 |
| MAC-SQL + GPT-4 [72] | 57.56 | 59.59 |
| DTS-SQL + DeepSeek 7B [60] | 55.8 | 60.31 |
| SFT CodeS-15B [43] | 58.47 | 60.37 |
| **Ours** | | |
| TiSQL + Claude-3 Opus | **59.2** | - |
| TiSQL + Claude-3 Haiku | 49.2 | - |
| TiSQL + Claude-3 Sonnet | 53.2 | - |
| TiSQL + GPT-4o | 57.6 | - |
| TiSQL + GPT-4o mini | 54.2 | - |
| **TiSQL + GPT-4 (ours)** | **59.1** | **60.98** |

Table 4: R-VES on the test set of Bird.

| Methods | Test EX (%) |
|---|---|
| GPT-4 [45] | 51.75 |
| Mistral [45] | 52.59 |
| DIN-SQL + GPT-4 [59] | 53.07 |
| DeepSeek [45] | 53.25 |
| DAIL-SQL + GPT-4 [33] | 54.02 |
| SFT CodeS-7B [43] | 55.69 |
| SFT CodeS-15B [43] | **56.73** |
| **TiSQL + GPT-4 (ours)** | **56.06** |

Table 5: The EX on the Spider test without schema linking.

| Methods | Test EX (%) ↓ |
|---|---|
| TiSQL + GPT-4o mini -w/o Schema Linking | 72.8 (0.0 ) |
| TiSQL + Claude-3 Haiku -w/o Schema Linking | 74.3 (↓ -0.3) |
| TiSQL + Claude-3 Sonnet -w/o Schema Linking | 77.5 (↓ -0.9) |
| TiSQL + GPT-4o -w/o Schema Linking | 82.1 (↓ -1.8) |
| **TiSQL + GPT-4 -w/o Schema Linking** | **84.2** (↓ -2.1) |

even all columns, none of which were specified in the original question. These cases can be considered valid SQL statements in exploratory data analysis.

We also explored the effect of schema linking on the execution accuracy of TiSQL, with the results reported in Table 5 and Table 6. The experimental results indicate that removing schema linking leads to a decrease in execution accuracy for TiSQL. Furthermore, it is important to note that in industrial scenarios involving numerous databases, tables, and columns, schema linking plays a crucial role

Table 6: The EX on Bird dev without schema linking.

| Methods | Dev EX (%) ↓ |
|---|---|
| TiSQL + Claude-3 Haiku -w/o Schema Linking | 49.2 (0.0 ) |
| TiSQL + Claude-3 Sonnet -w/o Schema Linking | 51.7 (↓ -1.5) |
| TiSQL + GPT-4o mini -w/o Schema Linking | 53.9 (↓ -0.3) |
| TiSQL + GPT-4o -w/o Schema Linking | 56.7 (↓ -0.9) |
| **TiSQL + GPT-4 -w/o Schema Linking** | **57.8** (↓ -1.3) |

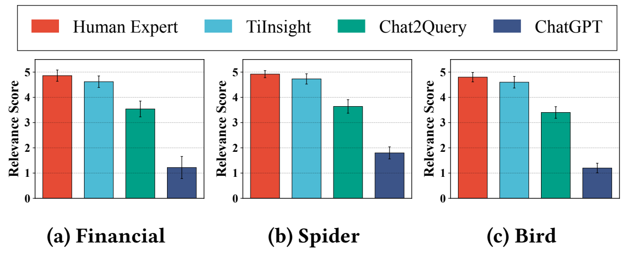


Figure 7: Relevance scores of the user study.

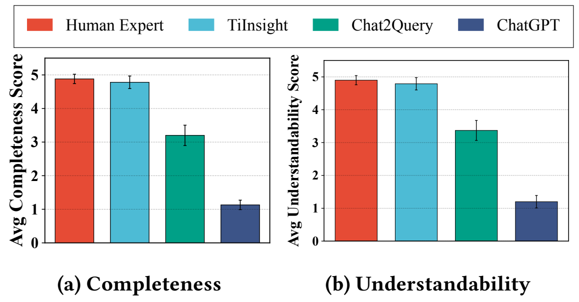


Figure 8: Average completeness and understandability scores.

in filtering out unnecessary schemas. This is primarily due to the current limitation in the context window size of LLMs.

## 6.3 User Study of TiInsight (RQ2)

We invited 20 participants for this user study. These individuals have data analysis experience, typically perform data analysis tasks, and are familiar with SQL. Initially, TiInsight recommends three exploration questions to the participants, allowing them 60 minutes to explore the three datasets, including Financial, Spider, and Bird. Participants are asked to rate the system's relevance, completeness, and understandability on a scale from 1 to 5.

We evaluate TiInsight along with the following baselines: (1) **Chat2Query** [82], which we use to explore the three datasets. Chat2Query is an exploratory data analysis system that recommends appropriate data visualization representations during exploration. (2) **ChatGPT** [55], where we encode metadata information for the datasets in the prompt. The ChatGPT version we use cannot produce suitable visualization results, so we ask it to recommend an appropriate chart type. It utilizes a CoT prompt in [85]. (3) **Human Expert**, who uses the same exploration questions as the previous two methods. The data analysis expert manually generates the exploration results and visualizes them to establish an upper limit for the exploration results.

During the rating process, participants are specifically reminded of the following questions:

**Q1:** When tasked with analyzing datasets, do these systems accurately comprehend the user's questions? In other words, must

you repeatedly prompt the system to correct questions in data exploration throughout the process?
**Q2:** We require all systems to generate appropriate SQL statements during the exploration process, which may necessitate breaking down the task into sub-problems. Are the decomposed subproblems reasonable? Do the SQL statements generated through this process meet your expectations? Does it make you feel "wow"? We believe that the "wow" factor encourages users to explore further.
**Q3:** Do you believe that the charts recommended by the system can help you understand the results of SQL queries more intuitively?
**Q4:** Do you believe that you gained a deeper understanding of this dataset during the data exploration process? This indicates that you acquired more insights through your exploration.
**Q5:** Do you find yourself attempting to ask additional questions after completing a given task? To what extent are you interested in further exploring the dataset?

In this experiment, we employ user studies to evaluate user satisfaction and experience, complementing rather than replacing objective performance evaluation. The accuracy of SQL queries has been rigorously evaluated through objective EX and R-VES (i.e., objective performance indicators). The user study complements these objective metrics by evaluating system usability from a user experience perspective. While certain metrics, such as the "wow" factor, are subjective, subjective evaluation measures have been widely recognized in the human-computer interaction (i.e., HCI) field as crucial elements for understanding actual system usability. As a result, EX, R-VES, and user study constitute a comprehensive evaluation framework. The practice of PingCAP demonstrates that user experience quality is as important as technical performance indicators. Additionally, it is important to note that, currently, no open-source benchmark exists for the objective evaluation of EDA systems.

The relevance scores are shown in Figure 7 for the three datasets. This result represents the average scores from all participants, accompanied by a 95% confidence interval. The human expert obtained the highest relevance score, which aligns with our expectations, as human experts are more familiar with the dataset and possess superior knowledge of data analysis. The results also show that TiInsight is comparable to the human expert. This can be attributed to the design of TiInsight, which centers around one fundamental question: How can human experts conduct data exploration? Therefore, we propose the hierarchical data context. Throughout the data exploration process, TiInsight gains more insights from datasets than other methods.

Chat2Query is slightly worse than TiInsight, yet significantly better than ChatGPT. Although Chat2Query is an advanced data exploration approach, it is limited by its inability to decompose complex questions into multiple sub-questions, restricting users from gaining deeper insights. Additionally, it produces a lower SQL "wow" factor compared to TiInsight. The SQL generated by TiInsight exhibits a more sophisticated structure compared with Chat2Query and frequently leverages table relationships to optimize complex join queries. Participants widely agreed that the ability to adjust the data visualization results generated by Chat2Query was a key feature.

ChatGPT exhibited the worst, as users could not execute SQL statements directly during data exploration, necessitating extra effort to verify the generated SQL. It cannot generate visualization charts and only provides text descriptions of the chart content, requiring users to utilize visualization tools to create them manually. As SQL execution is not possible, Chat2Query is unable to recommend appropriate visualization types based on data exploration results. Another significant issue is that it often recommends complex visualization results, which can hinder user comprehension. During data exploration, users often need to prompt the LLM to understand the data analysis objective accurately, and the hallucination problem in ChatGPT is particularly pronounced. This further emphasizes the importance of carefully designing the prompt for TiInsight. After engaging with our specified analysis goals, users often show little interest in pursuing further explorations.

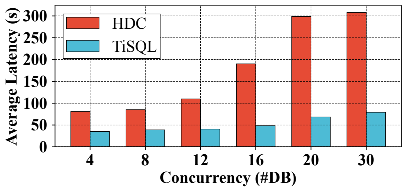


**Figure 9: Average latency for HDC and TiSQL.**

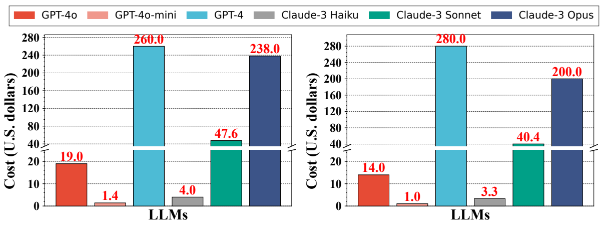
Figure 8 presents user scores for completeness and understandability, with each score reflecting the average from all participants across the three datasets. The same conclusion can be drawn regarding completeness and understandability. This further demonstrates that TiInsight has been well-received by users from multiple perspectives.

## 6.4 Experiments on Latency (RQ3)

We evaluate the average latency of various components of TiInsight on the Spider dataset. For users, average latency serves as a crucial indicator of system performance. Figure 9 depicts the average time required to generate the HDC for a database and generate and execute an SQL during concurrent exploration of varying databases. The concurrency involves using N databases for HDC and SQL generation and execution. The average latency of HDC generation increases with the number of explored databases. This is primarily because as the number of databases increases, more tables and columns must be explored to generate hierarchical data contexts. The average latency of SQL generation and execution by TiSQL is also increasing, primarily because TiSQL must filter a greater number of tables and columns. In our experiment, SQL generation takes less than 5 seconds using the GPT-4 model, while other models take even less time. It is also observed that a single HDC generation process requires more time than that of one text-to-SQL generation and execution. However, it is important to note that HDC generation occurs only once during the entire data exploration process; thus, this time is distributed across multiple analysis targets, resulting in an acceptable average latency. The design of TiInsight adopts an asynchronous mode, wherein upon submission of the analysis target, TiInsight promptly returns a bound ID to the user. At the same time, a background thread handles the binding task. This design enhances the overall user experience.

## 6.5 Experiments on Cost (RQ4)

In Figure 10, we analyze the average cost of TiInsight when utilizing various LLMs across different datasets. The prices of the different LLMs in [85]. The results indicate that the cost of utilizing the



**(a) Spider test** **(b) Bird dev**

**Figure 10: The cost of different LLMs for TiInsight.**

GPT-4 model is significantly higher (about $300) than other models. Specifically, the cost is $0.03 per 1, 000 token for input and $0.06 per 1, 000 tokens for output. This is primarily due to the high costs associated with both input and output tokens for the GPT-4 model. GPT-4o and GPT-4o mini exhibit significantly lower costs across various datasets, primarily due to the lower cost of their tokens compared to GPT-4. At the same time, the use of a prompt caching mechanism reduces costs by 50% compared to requests without caching. In particular, for GPT-4o, the cost is $1.25 per 1M cached input tokens, and for GPT-4o mini, it is $0.075 per 1M cached input tokens. However, GPT-4 does not support prompt caching. The Claude series models require a specific API setting and code adjustments to enable this prompt caching feature, which was not configured in our experiment. This highlights the overall trade-off between cost and accuracy in TiInsight.

### 6.6 TiInsight in Production (RQ5)

As of January 2025, TiInsight supports two business customers, including finance, and retail. At PingCAP, we have deployed TiInsight on TiDB Cloud for cloud customers and implemented it on TiDB Dedicated for large business customers. Based on customer feedback, TiInsight achieved over 85% SQL execution accuracy when using GPT-4 as the backbone model. SQL queries of any complexity are generated within 5 seconds. Customers reported that when using GPT-4o mini as the backbone, TiInsight generated SQL with poor accuracy and demonstrated limited comprehension across multiple conversation rounds. This is primarily due to the limited context understanding of the GPT-4o-mini, so we recommend using the GPT-4o or GPT-4 model in production environments. After conducting a thorough year-long observation of the production environment, we concluded that TiInsight significantly improved the accuracy of EDA and text-to-SQL tasks while minimizing human resource requirements, particularly in complex cross-domain data analysis scenarios. Consequently, TiInsight effectively addresses enterprise data analysis and text-to-SQL needs, resolving previous customer concerns regarding PingCAP's automated data exploration and SQL generation capabilities, thereby enhancing the business value.

## 7 RELATED WORKS

**Data Exploration**. Data exploration is a time-consuming task. Numerous research and commercial systems have been developed to facilitate this task [1, 13, 18, 24, 25, 27, 29, 31, 37, 40, 49–53, 56, 57, 68, 82]. Some research [1, 13, 40] focuses on the interface within the data exploration process, implementing interactive methods to assist non-expert users. Some research [24, 31] offers user-friendly data exploration interfaces for non-expert users via query-by-example. Several end-to-end EDA systems [29, 52, 53] utilize deep reinforcement learning to enhance data exploration. Some research [27, 49, 51, 68] focuses on the automatic discovery of insights in multidimensional data. ChatGPT [55] can also be utilized for data exploration. Chat2Query [82] is an SQL-based automated exploratory data analysis system that utilizes LLMs.

**Text-to-SQL**. Several state-of-the-art methods [38, 39] for text-to-SQL are broadly classified into rule-based, neural network (NN)-based, pre-trained language model (PLM)-based, and large language model (LLM)-based approaches. (1) **Rule-based Approaches**. Duoquest [20], Athena [63] and Athena++ [65] are three representative rule-based approaches. These methods rely on pre-defined rules or semantic parsers. (2) **NN-based Approaches**. Some research [21–23, 35, 75, 77, 80] utilize neural networks to learn the mapping from natural language to SQL queries, with the sequence-to-sequence method being the most representative. (3) **PLM-based Approaches**. With the emergence of pre-trained language models like BERT [26], research on text-to-SQL has gradually shifted from neural networks (NN) to pre-trained language models (PLM). Numerous PLM-based text-to-SQL methods [34, 44, 47, 64] have garnered significant attention from both the natural language processing and database communities. (4) **LLM-based Approaches**. With the emergence of LLMs, LLM-based text-to-SQL methods [28, 30, 33, 43, 59, 79] have garnered increasing attention from both the natural language processing and database communities.

**Data Visualization**. Data visualization [61, 66, 74] is a crucial component of the data analysis workflow and has received significant attention from both the data analysis and human-computer interaction communities. SEEDB [71] is a visualization recommendation engine to facilitate fast visual analysis. Voyager 2 [73] is a mixed-initiative system that blends manual and automated chart specification to help analysts engage. DeepEye [48] is a novel system for automatic data visualization. Table2Charts [81] learns common patterns from a large corpus of (table, chart) pairs using Q-learning. Lux [41] is an always-on framework for accelerating visual insight discovery in the dataframe workflows. Sevi [69] is a data visualization system that enables Speech2Text and Text2Vis. HAIChart [76] is a reinforcement learning-based framework designed to iteratively recommend effective visualizations.

## 8 CONCLUSION

In this paper, we propose TiInsight, a SQL-based multi-stage automated exploratory data analysis system through LLMs to facilitate real-world cross-domain scenarios. We propose the HDC to achieve cross-domain data analysis. We propose a series of algorithms combined with the map-reduce framework to enable efficient parallel HDC generation. Building on this foundation, we propose an efficient text-to-SQL method TiSQL and a rule-based data visualization method TiChart. Extensive experiments conducted on various evaluation datasets demonstrate the effectiveness of TiInsight. We open-source TiInsight and its APIs to facilitate research within the data analysis community.

## ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their insightful comments. This work was supported in part by the National Natural Science Foundation of China under Grant No. U22B2020. Peng Cai is the corresponding author.

## REFERENCES

[1] [n.d.]. Act on AI-powered insights in your flow of work built on the Salesforce Platform with Agentforce. Retrived in May, 2024 from https://www.tableau.com/.

[2] [n.d.]. BIRD-SQL: A Big Bench for Large-Scale Database Grounded Text-to-SQLs. Retrived in May, 2024 from https://bird-bench.github.io/.

[3] [n.d.]. Chroma - the open-source embedding database. Retrived in May, 2024 from https://github.com/chroma-core/chroma.

[4] [n.d.]. Crude Oil WTI Futures. Retrived in May, 2024 from https://www.investing.com/commodities/crude-oil-historical-data.

[5] [n.d.]. Dow Jones Industrial Average. Retrived in May, 2024 from https://fred.stlouisfed.org/series/DJIA.

[6] [n.d.]. Federal Funds Effective Rate. Retrived in May, 2024 from https://fred.stlouisfed.org/series/FEDFUNDS.

[7] [n.d.]. Gold futures. Retrived in May, 2024 from https://www.investing.com/commodities/gold-historical-data.

[8] [n.d.]. GPT-4o mini: advancing cost-efficient intelligence. Retrived in May, 2024 from https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

[9] [n.d.]. Hello GPT-4o. Retrived in May, 2024 from https://openai.com/index/hello-gpt-4o/.

[10] [n.d.]. The JavaScript library for bespoke data visualization. Retrived in May, 2024 from https://d3js.org/.

[11] [n.d.]. NASDAQ Composite Index. Retrived in May, 2024 from https://fred.stlouisfed.org/series/NASDAQCOM.

[12] [n.d.]. Pinecone serverless lets you deliver remarkable GenAI applications faster. Retrived in May, 2024 from https://www.pinecone.io/.

[13] [n.d.]. Power BI: Uncover powerful insights and turn them into impact. Retrived in May, 2024 from https://www.microsoft.com/en-us/power-platform/products/power-bi.

[14] [n.d.]. Shanghai Shenzhen CSI 300. Retrived in May, 2024 from https://www.investing.com/indices/csi300.

[15] [n.d.]. Spider: Yale Semantic Parsing and Text-to-SQL Challenge. Retrived in May, 2024 from https://yale-lily.github.io/spider.

[16] [n.d.]. Unemployment Rate. Retrived in May, 2024 from https://fred.stlouisfed.org/series/UNRATE.

[17] [n.d.]. Vector Search (Beta) Overview. Retrived in May, 2024 from https://docs.pingcap.com/tidbcloud/vector-search-overview.

[18] Sihem Amer-Yahia. 2024. Intelligent Agents for Data Exploration. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4521–4530.

[19] AI Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card* 1 (2024).

[20] Christopher Baik, Zhongjun Jin, Michael Cafarella, and HV Jagadish. 2020. Duoquest: A dual-specification system for expressive SQL queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2319–2329.

[21] Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 4560–4565.

[22] Zhi Chen, Lu Chen, Yanbin Zhao, Ruisheng Cao, Zihan Xu, Su Zhu, and Kai Yu. 2021. ShadowGNN: Graph Projection Neural Network for Text-to-SQL Parser. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 5567–5577.

[23] DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2021. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases. *Computational Linguistics* 47, 2 (2021), 309–332.

[24] Daniel Deutch and Amir Gilad. 2016. QPlain: Query by explanation. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 1358–1361.

[25] Daniel Deutch, Amir Gilad, Tova Milo, Amit Mualem, and Amit Somech. 2022. FEDEX: An Explainability Framework for Data Exploration Steps. *Proceedings of the VLDB Endowment* 15, 13 (2022), 3854–3868.

[26] Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[27] Rui Ding, Shi Han, Yong Xu, Haidong Zhang, and Dongmei Zhang. 2019. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In *Proceedings of the 2019 international conference on management of data*. 317–332.

[28] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306* (2023).

[29] Ori Bar El, Tova Milo, and Amit Somech. 2020. Towards Autonomous, Hands-Free Data Exploration.. In *CIDR*.

[30] Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining Small Language Models and Large Language Models for Zero-Shot NL2SQL. *Proceedings of the VLDB Endowment* 17, 11 (2024), 2750–2763.

[31] Anna Fariha and Alexandra Meliou. 2019. Example-driven query intent discovery: abductive reasoning using semantic similarity. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1262–1275.

[32] Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, Alex Van Grootel, Brandon Chow, Kai Deng, Katherine Lin, Marcos Campos, K. Venkatesh Emani, Vivek Pandit, Victor Shnayder, Wenjing Wang, and Carlo Curino. 2024. NL2SQL is a solved problem... Not!. In *CIDR*. https://www.cidrdb.org/cidr2024/papers/p74-floratou.pdf

[33] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363* (2023).

[34] Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Sam Madden, and Xiaoyong Du. 2023. Few-shot Text-to-SQL Translation using Structure and Content Prompt Learning. *SIGMOD* (2023), 1–28.

[35] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

[36] Xinyi He, Mengyu Zhou, Xinrun Xu, Xiaojun Ma, Rui Ding, Lun Du, Yan Gao, Ran Jia, Xu Chen, Shi Han, et al. 2024. Text2analysis: A benchmark of table question answering with advanced data analysis and unclear queries. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 18206–18215.

[37] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. 2015. Overview of data exploration techniques. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 277–281.

[38] George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *The VLDB Journal* (2023), 1–32.

[39] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *Proceedings of the VLDB Endowment* 13, 10 (2020), 1737–1750.

[40] Tim Kraska. 2018. Northstar: An Interactive Data Science System. *Proceedings of the VLDB Endowment* 11, 12 (2018).

[41] Doris Jung-Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A Hearst, et al. 2021. Lux: always-on visualization recommendations for exploratory dataframe workflows. *Proceedings of the VLDB Endowment* 15, 3 (2021), 727–738.

[42] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 13067–13075.

[43] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–28.

[44] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 13076–13084.

[45] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems* 36 (2024).

[46] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. 2018. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* 30, 10 (2018), 1852–1872.

[47] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 4870–4888.

[48] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018. Deepeye: Towards automatic data visualization. In *2018 IEEE 34th international conference on data engineering (ICDE)*. IEEE, 101–112.

[49] Pingchuan Ma, Rui Ding, Shi Han, and Dongmei Zhang. 2021. Metainsight: Automatic discovery of structured knowledge for exploratory data analysis. In *Proceedings of the 2021 international conference on management of data*. 1262–1274.

[50] Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han, and Dongmei Zhang. 2023. InsightPilot: An LLM-empowered automated data exploration system. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 346–352.

[51] Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han, and Dongmei Zhang. 2023. XInsight: eXplainable Data Analysis Through The Lens of Causality. *SIGMOD* (2023), 1–27.

[52] Tova Milo and Amit Somech. 2018. Deep reinforcement-learning framework for exploratory data analysis. In *Proceedings of the first international workshop on exploiting artificial intelligence techniques for data management*. 1–4.

[53] Tova Milo and Amit Somech. 2020. Automating exploratory data analysis via machine learning: An overview. In *SIGMOD*. 2617–2622.

[54] OpenAI. [n.d.]. GPT-4. Retrived in May, 2024 from https://openai.com/research/gpt-4.

[55] OpenAI. 2022. Introducing ChatGPT. Retrived in May, 2024 from https://openai.com/blog/chatgpt,.

[56] Biao Ouyang, Yingying Zhang, Hanyin Cheng, Yang Shu, Chenjuan Guo, Bin Yang, Qingsong Wen, Lunting Fan, and Christian S Jensen. 2024. RCRank: Multimodal Ranking of Root Causes of Slow Queries in Cloud Database Systems. *Proceedings of the VLDB Endowment* 18, 4 (2024), 1169–1182.

[57] Jinglin Peng, Weiyuan Wu, Brandon Lockhart, Song Bian, Jing Nathan Yan, Linghao Xu, Zhixuan Chi, Jeffrey M Rzeszotarski, and Jiannan Wang. 2021. Dataprep. eda: Task-centric exploratory data analysis for statistical modeling in python. In *Proceedings of the 2021 International Conference on Management of Data*. 2271–2280.

[58] PingCAP. [n.d.]. Use Knowledge Bases. Retrived in June, 2025 from https://docs.pingcap.com/tidbcloud/use-chat2query-knowledge.

[59] Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems* 36 (2024).

[60] Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed Text-to-SQL with Small Large Language Models. *arXiv preprint arXiv:2402.01117* (2024).

[61] Xuedi Qin, Yuyu Luo, Nan Tang, and Guoliang Li. 2020. Making data visualization more efficient and effective: a survey. *The VLDB Journal* 29, 1 (2020), 93–117.

[62] Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. *arXiv preprint arXiv:2405.15307* (2024).

[63] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R Mittal, and Fatma Özcan. 2016. ATHENA: an ontology-driven system for natural language querying over relational data stores. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1209–1220.

[64] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 9895–9901.

[65] Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. Athena++ natural language querying for complex nested sql queries. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2747–2759.

[66] Leixian Shen, Enya Shen, Yuyu Luo, Xiaocong Yang, Xuming Hu, Xiongshuai Zhang, Zhiwei Tai, and Jianmin Wang. 2022. Towards natural language interfaces for data visualization: A survey. *IEEE transactions on visualization and computer graphics* 29, 6 (2022), 3121–3144.

[67] Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics* 8, 1 (2002), 52–65.

[68] Bo Tang, Shi Han, Man Lung Yiu, Rui Ding, and Dongmei Zhang. 2017. Extracting top-k insights from multi-dimensional data. In *Proceedings of the 2017 ACM international conference on management of data*. 1509–1524.

[69] Jiawei Tang, Yuyu Luo, Mourad Ouzzani, Guoliang Li, and Hongyang Chen. 2022. Sevi: Speech-to-visualization through neural machine translation. In *Proceedings of the 2022 International Conference on Management of Data*. 2353–2356.

[70] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[71] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. 2015. Seedb: Efficient data-driven visualization recommendations to support visual analytics. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, Vol. 8. NIH Public Access, 2182.

[72] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2024. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. arXiv:2312.11242 [cs.CL]

[73] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 chi conference on human factors in computing systems*. 2648–2659.

[74] Aoyu Wu, Yun Wang, Xinhuan Shu, Dominik Moritz, Weiwei Cui, Haidong Zhang, Dongmei Zhang, and Huamin Qu. 2021. Ai4vis: Survey on artificial intelligence approaches for data visualization. *IEEE Transactions on Visualization and Computer Graphics* 28, 12 (2021), 5049–5070.

[75] Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2016. Sequence-based structured prediction for semantic parsing. In *Annual meeting of the Association for Computational Linguistics (ACL)*. 1341–1350.

[76] Yupeng Xie, Yuyu Luo, Guoliang Li, and Nan Tang. 2024. HAIChart: Human and AI Paired Visualization System. *arXiv preprint arXiv:2406.11033* (2024).

[77] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436* (2017).

[78] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. 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*. Association for Computational Linguistics.

[79] Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. 2024. FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis. *arXiv preprint arXiv:2401.10506* (2024).

[80] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103* (2017).

[81] Mengyu Zhou, Qingtao Li, Xinyi He, Yuejiang Li, Yibo Liu, Wei Ji, Shi Han, Yining Chen, Daxin Jiang, and Dongmei Zhang. 2021. Table2Charts: recommending charts by learning shared table representations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2389–2399.

[82] Jun-Peng Zhu, Peng Cai, Boyan Niu, Zheming Ni, Kai Xu, Jiajun Huang, Jianwei Wan, Shengbo Ma, Bing Wang, Donghui Zhang, Liu Tang, and Qi Liu. 2024. Chat2Query: A Zero-Shot Automatic Exploratory Data Analysis System with Large Language Models. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 5429–5432. https://doi.org/10.1109/ICDE60146.2024.00420

[83] Jun-Peng Zhu, Peng Cai, Kai Xu, Li Li, Yishen Sun, Shuai Zhou, Haihuang Su, Liu Tang, and Qi Liu. 2024. AutoTQA: Towards Autonomous Tabular Question Answering through Multi-Agent Large Language Models. *Proceedings of the VLDB Endowment* 17, 12 (2024), 3920–3933.

[84] Jun-Peng Zhu, Peng Cai, Kai Xu, Li Li, Yishen Sun, Shuai Zhou, Haihuang Su, Liu Tang, and Qi Liu. 2025. UNITQA: A Unified Automated Tabular Question Answering System with Multi-Agent Large Language Models. In *Companion of the 2025 International Conference on Management of Data*. 279–282.

[85] Jun-Peng Zhu, Boyan Niu, Peng Cai, Zheming Ni, Jianwei Wan, Kai Xu, Jiajun Huang, Shengbo Ma, Bing Wang, Xuan Zhou, et al. 2024. Towards Automated Cross-domain Exploratory Data Analysis through Large Language Models. *arXiv preprint arXiv:2412.07214* (2024).