# DataFrame QA: A Universal LLM Framework on DataFrame Question Answering Without Data Exposure

**Junyi Ye** JY394@NJIT.EDU
**Mengnan Du** MENGNAN.DU@NJIT.EDU
**Guiling Wang** GUILING.WANG@NJIT.EDU
*Ying Wu College of Computing, New Jersey Institute of Technology, Newark, USA*

**Editors:** Vu Nguyen and Hsuan-Tien Lin

## Abstract

This paper introduces DataFrame question answering (QA), a novel task that utilizes natural language processing (NLP) models to generate Pandas queries for information retrieval and data analysis on dataframes, emphasizing safe and non-revealing data handling. Specifically, our method, leveraging large language model (LLM), which solely relies on dataframe column names, not only ensures data privacy but also significantly reduces the context window in the prompt, streamlining information processing and addressing major challenges in LLM-based data analysis. We propose DataFrame QA as a comprehensive framework that includes safe Pandas query generation and code execution. Various LLMs are evaluated on the renowned WikiSQL dataset and our newly developed UCI-DataFrameQA, tailored for complex data analysis queries. Our findings indicate that GPT-4 performs well on both datasets, underscoring its capability in securely retrieving and aggregating dataframe values and conducting sophisticated data analyses. This approach, deployable in a zero-shot manner without prior training or adjustments, proves to be highly adaptable and secure for diverse applications. Our code and dataset are available at https://github.com/JunyiYe/dataframe-qa.

**Keywords:** Large language models; DataFrame question answering.

## 1. Introduction

In the era of large language models (LLMs), table question answering (QA) with LLMs typically involves embedding the entire table into the prompt, along with the user's question and instructions Li et al. (2023); Chen (2022). This method is highly effective for querying small and simple tables or dataframes. However, since tables are inherently two-dimensional structures, they can quickly increase the size of the prompt with the addition of rows. This becomes particularly challenging with large dataframes such as those related to weather, traffic, and product sales, which can easily exceed the 4K or 8K content window limit of most models.

Unlike lengthy text content that can be efficiently managed using techniques such as Retrieval-Augmented Generation (RAG) Lewis et al. (2020), which allows the summarization of each chunk and integrates searching to generate a final answer (thus efficiently processing a large volume of text), large tables present a different challenge. Tables, due to their two-dimensional structure and data density, do not lend themselves to this kind of summarization and retrieval-based processing. This difference highlights the unique challenge of using LLMs to efficiently manage large table data, as opposed to handling extensive text content.

Despite many models (e.g., GPT-4 and Claude-3) now expanding their context window sizes to 16k, 32k, or more, several key challenges remain. First, the computational cost for processing tokens is high. For example, it costs $0.005 per 1K tokens using the latest GPT-4o model API. Large tables can rapidly increase costs by boosting token count. Second, embedding full datasets risks potential data leakage. In addition, recent studies indicate the problem of 'Lost in the Middle', where long prompts can decrease the model performance Liu et al. (2023). LLMs also struggle with mathematics, which becomes problematic when the query involves calculations Ouyang et al. (2022). Moreover, conventional querying often includes superfluous data beyond what is necessary to answer the question. These pose challenges in maintaining accuracy, efficiently managing computational resources and protecting sensitive information when applying LLMs to table-based QA.

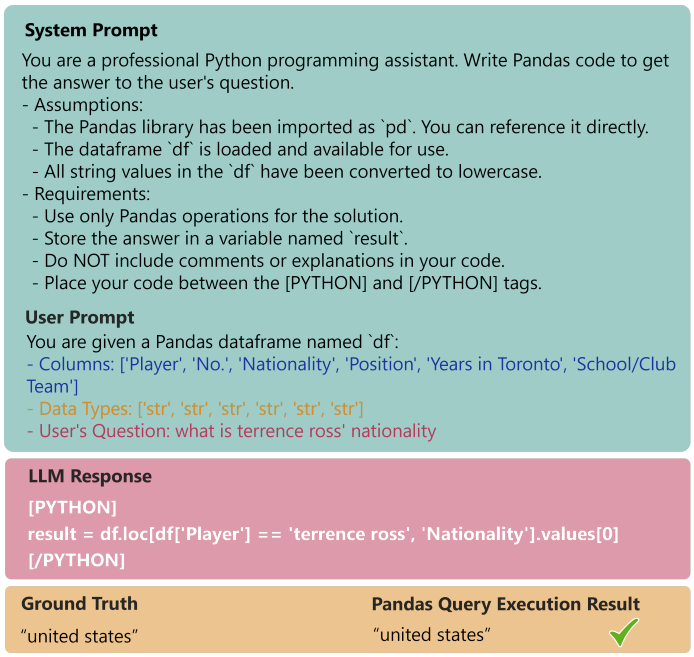


Figure 1: An example where a LLM (Chatgpt) can generate a correct Pandas query to answer user question using only the table header and column data types, without accessing the table values. Typically, the total number of tokens for DataFrame QA tasks, including both the prompt and the model output, stays below 250 tokens.

To address these challenges, we propose a new task and framework called DataFrame QA. This task aims to enhance the efficiency and security of querying the dataframe by using NLP models to generate Pandas queries. It employs a method that utilizes only table column names and data types, effectively reducing data leakage risks and minimizing the need for extensive context windows (Figure 1). Although various NLP models can be used for the DataFrame QA task, this study focuses on LLMs because of their superior code generation abilities, highlighting their appropriateness for complex data manipulation tasks without prior specific training. We modify the WikiSQL dataset and create a new

dataset, UCI-DataFrameQA for our task, conducting evaluations in a zero-shot manner using Llama2 Touvron et al. (2023), CodeLlama Roziere et al. (2023), GPT-3.5, and GPT-4 OpenAI (2023). Furthermore, our research involves a thorough analysis of the causes of errors and the inherent challenges associated with the DataFrame QA task, along with potential solutions. This investigation provides insights for future dataset expansions and improvements and for enhancing model performance. The major contributions of this work can be summarized as follows:

(1) We propose DataFrame QA, a new task and a general LLM framework for table information retrieval and data analysis, safeguarding against data leakage.

(2) Developed UCI-DataFrameQA, a new dataset leveraging GPT-4's capabilities, designed for diverse questions on DataFrame, setting a foundation for expansive future dataset creation.

(3) We evaluated several mainstream open-source and closed-source LLMs on two benchmark datasets, analyzed a wide range of error classes, and provided corresponding solutions.

## 2. Related Work

To the best of our knowledge, existing literature does not directly address the DataFrame QA task. The closest domain is Table QA, which is a specialized area of NLP that focuses on interpreting and answering queries based on tabular data. This field can be broadly divided into two key tasks:

### 2.1. Text-to-SQL

Text-to-SQL involves converting natural language questions into SQL queries that can be executed against relational databases. The aim is to accurately interpret the user's intent and translate it into syntactically and semantically correct SQL commands. Recent advancements in Text-to-SQL have primarily leveraged neural network-based approaches, including LLMs Ye et al. (2023); Ni et al. (2023), especially sequence-to-sequence models Liu et al. (2021); Xu et al. (2021); Herzig et al. (2020); Yu et al. (2018); Zhong et al. (2017). These technologies have demonstrated significant effectiveness in understanding diverse queries across various domains and in generating the corresponding SQL statements. Current Text-to-SQL technologies primarily rely on simple database schemas and basic queries, limiting their ability to handle complex, real-world database structures and advanced relational tasks.

### 2.2. QA on Semi-Structured Tables

This task focuses on accurately parsing HTML tables, which are often semi-structured and vary in format, to understand the context and relationships within the data and provide the correct answers Pasupat and Liang (2015). It requires advanced techniques in data extraction, contextual understanding, and natural language processing to effectively navigate the diverse structures and formats of HTML tables. Recent developments have utilized transformer-based models Xie et al. (2022); Pan et al. (2021); Glass et al. (2021); Yin et al. (2020) and LLMs Li et al. (2023), significantly improving the ability to process and interpret complex table structures and query contexts. These models have notably improved

the accuracy and efficiency of extracting information from semi-structured HTML tables, representing a substantial advancement in the field. However, these models face limitations when loading HTML/CSV-formatted tables, often showing limited proficiency in tasks such as identifying missing cells or finding column names, leading to low accuracy in specific tests Li et al. (2023). Including table values in input poses data privacy risks and challenges due to context window limitations and the handling of sensitive information.

## 3. Methodology

### 3.1. Problem Statement

The DataFrame QA task is defined as the process of leveraging NLP models to generate executable Pandas queries in response to natural language queries on a specified dataframe. This highlights the versatility of Pandas, which offers a wide array of data analysis operations, ranging from simple data retrieval to intricate statistical analyses.

The system is characterized by an input tuple $(S, H, C, Q)$, where:

- $S$ represents the system prompt, encompassing task description, output format, Python library constraints, and more.
- $H = \{h_1, h_2, ..., h_n\}$ denotes the dataframe headers.
- $C = \{c_1, c_2, ..., c_n\}$ covers additional column information, such as data types and column descriptions.
- $Q$ is the natural language query.

The system's output, $P'$, is a Pandas query generated by an NLP model $f$ in response to $Q$. The ground truth query, $P$, is the correct Pandas query that yields the answer to $Q$. Thus, the generated Pandas query is a function of the input, defined as:

$$P' = f(S, H, C, Q) \tag{1}$$
$$A' = \text{execute}(P', df) \tag{2}$$
$$A = \text{execute}(P, df) \tag{3}$$

$A'$ and $A$ are the results obtained by executing $P'$ and $P$ on the dataframe $df$ in a safe sandbox, respectively. The effectiveness of the DataFrame QA system is measured by how closely $A'$ approximates the ground truth answer $A$.

Dataframe QA Task marks a departure from traditional table question answering, which typically involves analyzing the relationship between questions and table contents. The DataFrame QA task, instead, concentrates on the analysis of dataframe structures and data types, deliberately omitting the scrutiny of actual data values. It evaluates NLP models' ability to understand dataframe headers and column metadata, and to convert this understanding into valid Pandas queries through natural language questions.

Moreover, the task utilizes the versatility of prompts to include additional information, such as dataset descriptions and specific constraints. This enhances contextual comprehension beyond what traditional fixed-input models offer. This approach facilitates a more

dynamic and informed model interaction, greatly expanding the potential of dataframe analysis in NLP.

### 3.2. Challenges

Key challenges of DataFrame QA include:

- **Interpreting User Questions:** Understanding how questions relate to the dataframe's structure and column types, requiring domain-specific knowledge and handling ambiguities.
- **Formulating Pandas Queries:** Proficiency in creating accurate Pandas queries that meet the technical and logical requirements of the task.
- **Following Instructions:** Adhering to given guidelines or constraints, ensuring that responses are technically correct and contextually suitable.

These challenges highlight the intricacies involved in the Dataframe QA task, underscoring the need for NLP models that are not only proficient in natural language processing but also capable of understanding and manipulating complex data structures.

### 3.3. DataFrame QA Framework

Figure 2 outlines our DataFrame QA framework, structured in three stages: Pandas Query Generation, Code Execution, and Result Evaluation.

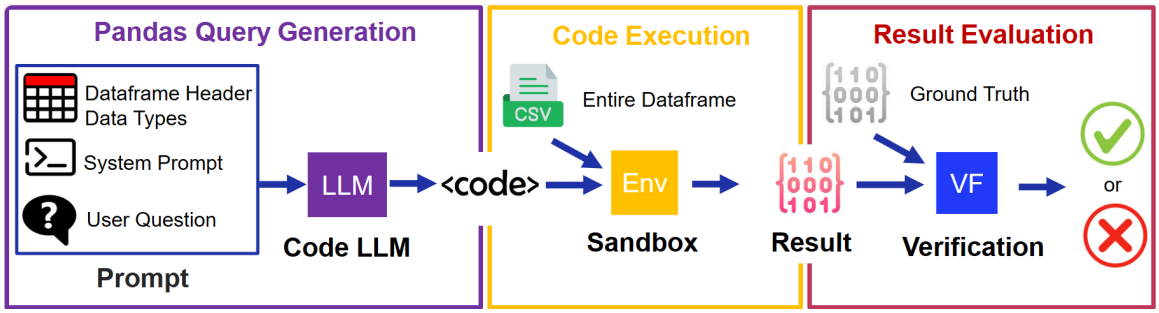


Figure 2: Framework of DataFrame QA. Note that, LLM in the figure can be replaced with any fine-tuned NLP model trained for the DataFrame QA task.

#### 3.3.1. Pandas Query Generation

In this initial phase, a LLM processes the prompt, comprising the dataframe header, column data types, system prompt (i.e. assumption and requirements), and user question, to generate a Pandas query. This design ensures data privacy by avoiding exposure to table values and leverages column data types to inform query selection.

#### 3.3.2. Code Execution

The generated query is executed within a controlled virtual environment, protecting against unauthorized operations. This environment is restricted to essential libraries (Pandas, NumPy, and Math), thereby enhancing security. The execution results, stored as Python objects, offer flexibility for further processing. For example, large table results can be provided as downloadable content, while Matplotlib plot objects are displayed directly.

#### 3.3.3. Result Evaluation

We compare the results of executed queries with ground truth answers, encountering challenges due to the diversity of data types involved. Our methodology standardizes results across numeric, string, and list/ndarray types to facilitate an accurate comparison. Note that Pandas queries often return series or dataframe objects, rather than direct answers to user questions, mirroring the characteristics of coding datasets commonly used in LLM training. To address this, we employ a relaxed evaluation criterion, considering the contents of series or dataframes correct if they include the answer. For pairs with a mismatch, we perform a manual comparison to ensure the accuracy and relevance of the results.

## 4. Experimental Settings

### 4.1. Dataset

To rigorously assess the proficiency of LLMs in generating Pandas queries for two distinct types of tasks, we have adapted the WikiSQL and UCI datasets to align with our research objectives.

| User Question | Pandas Query | Types |
|---|---|---|
| which province is bay of islands in? | `result = df.loc[df['Electorate']=='bay of islands', 'Province'].iloc[0]` | Retrieval |
| how many combined days did go shiozaki have? | `result = df.loc[df['Wrestler']=='go shiozaki', 'Combined days'].values[0]` | Aggregation |
| how does the average shell weight vary across different numbers of rings? | `result = df.groupby('Rings')['Shell_weight'].mean()` | Data Analysis |
| can you create a new column 'volume' as a product of length, diameter, and height, then find the average volume for each sex? | `df['Volume'] = df['Length'] * df['Diameter'] * df['Height']`<br>`result = df.groupby('Sex')['Volume'].mean()` | Data Analysis |

Table 1: Examples of Sample Questions and Corresponding Pandas Queries Categorized by Complexity Level. Retrieval/Aggregation queries can be resolved using single-step, SQL-like queries, whereas Data Analysis questions necessitate multi-step or complex Pandas operations.

#### 4.1.1. **Simple Query Dataset - WikiSQL**

WikiSQL Zhong et al. (2017), a benchmark in Text-to-SQL research, provides a test set comprising 15,878 table-question pairs, designed to evaluate natural language interfaces

with relational databases. We transformed these tables into dataframes, ensuring datatype consistency for each column. Additionally, we utilized the results of the SQL queries executed on these tables as the ground truth for our DataFrame QA framework.

A notable challenge in WikiSQL is the frequent lowercasing of entities in user questions, which can lead to ambiguities when formulating Pandas queries. To mitigate this issue, we standardized all user questions and dataframe strings to lowercase. Additionally, we explicitly instructed in the prompt that all strings within the dataframe are lowercased. This approach improves clarity and uniformity in LLM query processing, ensuring consistent interpretation and handling of string data.

The WikiSQL dataset predominantly features straightforward information retrieval questions (71%), solvable with single-step operations similar to basic SQL queries. The remaining 29% focus on aggregation tasks, including 12% MIN/MAX, 9% COUNT, and 8% AVG/SUM. These represent simpler and more direct query scenarios, which require basic dataframe operations.

#### 4.1.2. **Complex Dataset - UCI-DataFrameQA**

To develop a DataFrame QA dataset reflective of real-world scenarios, we adopted a comprehensive approach. We sourced diverse dataframes from the UCI dataset Newman et al. (1998), spanning various domains such as animals, automobiles, and medical fields, to simulate different societal contexts. Our methodology was designed to represent three real-life data interaction roles: **1) Data Scientists**, who delve into detailed data analysis queries for patterns, trends, and statistical insights; **2) General Users**, such as patients in medical datasets or customers in automobile datasets, seeking practical, consumer-oriented aspects of the data; **3) Data Owners**, like hospitals or companies, focusing on extracting business-oriented insights.

Utilizing GPT-4, we generated questions mirroring typical inquiries and challenges these roles face in real-life scenarios, thereby creating a comprehensive DataFrame QA dataset. In Table 1, we illustrate simple retrieval/aggregation questions from WikiSQL and complex data analysis questions generated in the UCI-DataFrameQA dataset, with the latter posing greater challenges.

For each of the 11 dataframes from the UCI dataset, GPT-4 generated 60 questions (20 per role), each with a corresponding Pandas query. The appendix details the prompts and provides examples of the question/Pandas query generation.

Following a meticulous manual review, we compiled a final set of 547 question/Pandas query pairs. This curation involved eliminating pairs with inaccurate matches, those requiring external libraries, or questions unsolvable with just the provided table headers. This rigorous selection process ensures the dataset comprises realistic and executable DataFrame QA scenarios.

Table 2 presents the distribution of generated question types within the UCI dataset, categorized by different user roles. Analysis of the dataset indicates that 22% of the questions are focused on basic retrieval/aggregation tasks, while a predominant 78% involves more advanced operations, such as grouping, correlation analysis, and sorting. Specifically, the Data Scientist role primarily concerns complex data analysis queries, whereas General Users tend to concentrate on more straightforward retrieval/aggregation questions. Questions from Data Owners exhibit a range of complexity, bridging the two extremes. The

| Role | Retrieval/Aggregation | Data Analysis |
|---|---|---|
| Data Scientist | 9 (5%) | 175 (95%) |
| General User | 69 (40%) | 105 (60%) |
| Data Owner | 42 (22%) | 147 (78%) |

Table 2: Distribution of Generated Question Types on UCI Dataset Across Different Roles.

dataset is a pivotal testbed for assessing LLMs' adeptness in handling advanced queries, showing their capacity to execute complex, multi-step data analysis.

This distinction in question complexity across two datasets provides an opportunity to evaluate LLMs' capabilities over a wide spectrum of query complexities, ranging from simple data retrieval to sophisticated data manipulation tasks.

### 4.2. Baselines

Our experimental baselines include:

**Llama2.** This advanced iteration of the Llama language model series offers configurations ranging from 7B to 70B parameters Touvron et al. (2023). With training on 2 trillion tokens, it is equipped with an expanded 4K token context window, enhancing its applicability across diverse NLP tasks.

**CodeLlama.** A specialized variant of Llama2, CodeLlama is tailored for coding-related tasks Roziere et al. (2023). It shows superior performance in coding benchmarks, benefitting from a 16K token window. We utilized its instruction models at 7B, 13B, and 34B parameter sizes, focusing on their ability to generate and interpret code.

**GPT-3.5, GPT-4.** These models are benchmarks in the LLM domain, showcasing exceptional performance across coding, general NLP tasks, and conversational capabilities. Their ongoing updates reinforce their status as leaders in AI-driven solutions. The model versions used in our experiments are gpt-3.5-turbo-0613 and gpt-4-0613.

### 4.3. Implementation Details

In our methodology, a consistent greedy decoding strategy was applied across all LLMs. For the deployment of Llama2 and CodeLlama, the official checkpoints available through the HuggingFace were used. The models were executed on NVIDIA A100 GPUs, with the number of units ranging from one to four, depending on the model size.

### 4.4. Evaluation Metric

The pass@1 score Chen et al. (2021) is a crucial metric in evaluating the performance of LLM in the context of code generation tasks. It measures the accuracy of the LLMs in providing the correct answer on their first attempt without any additional iterations or refinements. This score is especially important in scenarios where immediate and precise responses are required, reflecting the model's ability to accurately interpret and respond to complex queries in a single pass.

## 5. Experimental Analysis

This study delves into the DataFrame QA task, focusing on the capabilities and challenges of cutting-edge LLMs. Our investigation is centered around three pivotal research questions: **Q1. Are current state-of-the-art LLMs capable of effectively handling the DataFrame QA task? Q2. What factors influence the performance differences among various LLMs? Q3. What are the inherent challenges and potential solutions associated with DataFrame QA tasks?**

### 5.1. Q1: Efficacy of Leading LLMs in DataFrame QA Tasks

Our study assesses various LLMs' first-attempt accuracy (pass@1), with findings illustrated in Figures 3 and 4. Notably, GPT-4 exhibits high pass@1 accuracies—85.5% on WikiSQL and 97.2% on UCI-DataFrameQA, reflecting its adeptness in processing a wide array of queries, from simple data retrieval to complex analysis, without direct access to table data.

Analysis of GPT-4 on WikiSQL's test set reveals that 11.6% of queries pose challenges due to mismatched pairs (5%), ambiguity (5%), and quotation mark issues (1.6%), affecting execution. Addressing these issues, GPT-4's pass@1 accuracy could rise to 96.7%, aligning with its performance on UCI-DataFrameQA.

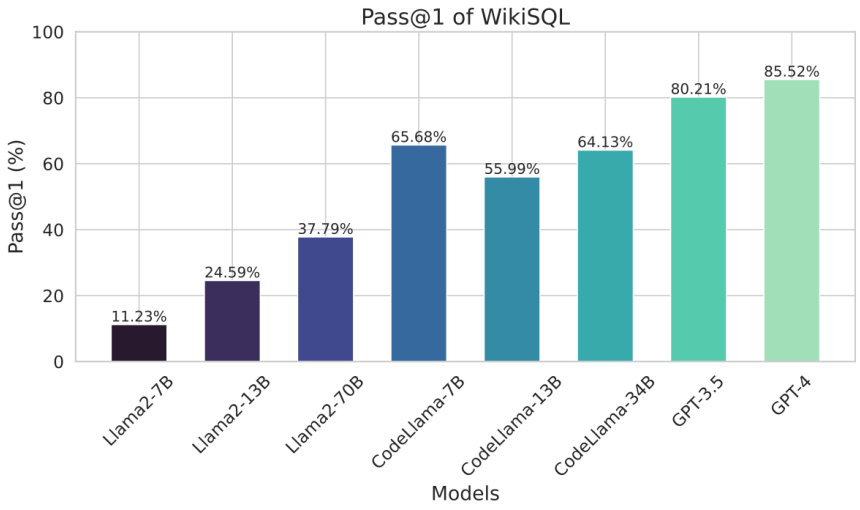


Figure 3: Performance of LLMs on WikiSQL.

**Scaling Laws in LLM Performance:** We observed clear performance stratification among models: Llama2 and CodeLlama, with the latter surpassed by the GPT series. GPT and Llama2 models adhere to scaling laws (Kaplan et al. (2020)), showing gains with increasing size. In contrast, CodeLlama models deviate from these laws, warranting further exploration in subsequent sections.

**Comparison with Text-to-SQL Models:** GPT-4, in comparison to specialized Text-to-SQL models such as TAPEX Liu et al. (2021) with an execution accuracy of 89.5% and SeaD+EG$_{CS}$ Xu et al. (2021) at 92.7%, exhibits slightly lower performance. This discrepancy is attributed to two main factors: **Task Complexity:** Generating Pandas queries is

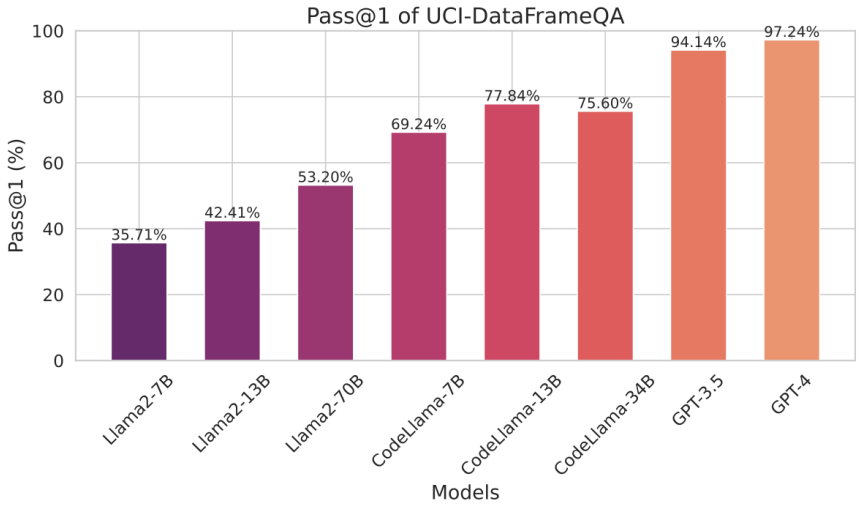


Figure 4: Performance of LLMs on UCI-DataFrameQA.

inherently more complex than structured SQL queries, given Pandas' wider operation range. **Zero-Shot Learning Approach:** Unlike TAPEX and SeaD+EG$_{CS}$, which are specialized for Text-to-SQL, GPT-4's zero-shot application, without specific fine-tuning, impacts its efficiency in DataFrame QA tasks.

### 5.2. Q2. Performance Variation Determinants

In delving into what drives performance differences in LLMs, we scrutinize failure cases on WikiSQL using GPT-3.5. Starting with 100 error samples, we categorized these using GPT-4 and further confirmed through manual verification, identifying eight distinct error types detailed in the Appendix. We then used GPT-3.5 to classify all error classes across model input, sample rows, generated queries, execution output, and expected results, where incorrect queries often span multiple error categories.

**Error Distribution Across LLMs:** Figure 5 presents a heatmap of error class distribution across LLMs. Predominant errors include String Matching and Comparison, Data Access and Bounds, and Query Condition and Value. These stem from issues like misidentified column names, case sensitivity, and misinterpretations of user questions and instructions, often due to a mismatch between instructions and their execution.

**Bias Between Instructions and Training Data:** In our analysis of CodeLlama models, we identified a notable bias reflecting discrepancies between instruction adherence and training data distribution, particularly evident in handling string queries. The 13B model registered 2368 instances, and the 34B model recorded 1175 instances of errors related to this bias, in stark contrast to the mere 175 instances encountered by the 7B model. These errors persisted even with explicit instructions to treat all strings as lowercase, underscoring a variance in the models' interpretation and compliance with these directives, deviating from the expected scaling law behavior in such tasks.

For instance, `df[df['Player']=='Terrence Ross']['Nationality'].values[0]` incorrectly queries the capitalized `'Terrence Ross'`, contrary to the directive

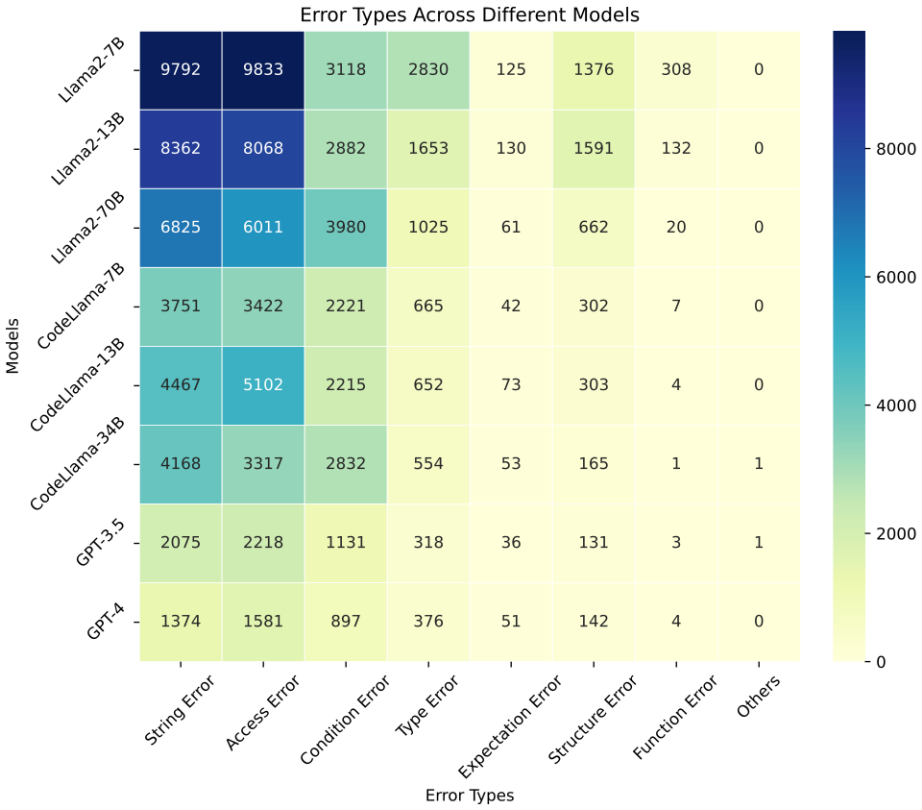


Figure 5: Distribution of error types among different LLMs on WikiSQL. Definitions of error types and prompts for error classification are put in Appendix.

of all strings in the dataframe being lowercase. This example highlights the bias issue: when querying dataframe values, especially with proper nouns like names and places, the model tends to capitalize them, showing a preference influenced by its training data over the provided instructions. This tendency exemplifies the broader challenge of aligning model behavior with specific directives, particularly when they counter the model's training data distribution.

These results reveal that DataFrame QA tasks test not only the coding abilities of LLMs but also their ability to adhere to instructions in the face of conflicts with training data distribution, along with their comprehensive skills in understanding user queries and dataframe headers.

### 5.3. Q3. Inherent Challenges and Solutions in DataFrame QA Tasks

In Table 3, we outline typical failure cases in DataFrame QA tasks, categorizing a spectrum of error types commonly encountered by LLMs. These include Value Retrieval, Column Reference, Instruction Misalignment, Aggregation, Function-Column Ambiguity, Insufficient Column Data/Format Information, Coding Syntax, and Hallucination Errors, each

Table 3: Typical Failure Cases in DataFrame QA Task. $Q$ : User Question , $P'$ : Generated Pandas Query, $P$ : Correct Pandas Query'

| **Error Type** | **Eample** |
|---|---|
| Value Retrieval Error | $Q$ : Which province is grey and bell electorate in?<br>$P'$ : `result = df[(df['Electorate']=='grey')\|(df['Electorate']=='bell')]['Province'].unique()`<br>$P$ : `result=df[(df['Electorate']=='grey and bell')]['Province'].unique()` |
| Column Reference Error | $Q$ : What is the variance in resting blood pressure (trestbps) among different heart disease severity levels?<br>$P'$ : `df.groupby('ca')['trestbps'].var()`<br>$P$ : `df.groupby('num')['trestbps'].var()` |
| Instruction Misalign-ment | $Q$ : What are the mean and median lengths of abalone by each sex category?<br>$P'$ : `import pandas as pd`<br>`# Group the dataframe by sex`<br>`...`<br>$P$ : `result = df.groupby('Sex')['Length'].agg(['mean', 'median'])` |
| Aggregation Error | $Q$ : what is the total amount of allied-unrelated where the component is human capital?<br>$P'$ `result = df[df['Component']=='human capital']['Allied-Unrelated'].sum()`<br>$P$ `result = df[df['Component']=='human capital']['Allied-Unrelated'].count()` |
| Function-Column Ambiguity Error | $Q$ : what is the average score when the swimsuit score is 8.503?<br>$P'$ : `result = df[df['Swimsuit']=='8.503']['Average'].mean()`<br>$P$ : `result = df[df['Swimsuit']=='8.503']['Average']` |
| Insufficient Column Data/For-mat Information | $Q$ : which team played on december 5?<br>$P'$ `df['Date'] = pd.to_datetime(df['Date'])`<br>`result = df[df['Date'] == 'december 5']['Team']`<br>$P$ : `result = df[df['Date'] == 'december 5']['Team']` |
| Coding Syntax Error | $Q$ : how does the average shell weight vary across different numbers of rings?<br>$P'$ : `result = df.groupby('Rings').mean()['Shell_weight']`<br>$P$ : `result = df.groupby('Rings')['Shell_weight'].mean()` |
| Hallucination Error | $Q$ : i'm interested in knowing the most common age of abalone. can you find that for me?<br>$P'$ : `result = df['Age'].mode()`<br>$P$ : `result = df['Rings'].mode()` |

posing distinct challenges. Detailed examples and solutions for varied failure types are comprehensively provided in the Appendix.

- **Value Retrieval Error:** This error occurs when Pandas queries fetch incorrect values due to ambiguous user questions. For instance, a query for `'grey and bell'` might be misinterpreted as separate searches for `'grey'` and `'bell'`. Also, LLMs might omit crucial characters like Roman numeral `'I'` or hyphens, mistaking them for textual errors.

  *Solution:* Utilizing quotation marks around specific query terms can significantly reduce these errors, clarifying the intended search as a singular entity.

- **Column Reference Error:** Errors occur when queries incorrectly target columns, often due to ambiguous names. This common challenge arises especially when LLMs lack domain-specific knowledge, leading to column misidentification.

  *Solution:* Providing clear column descriptions in prompts, such as specifying the roles of `'ca'` and `'num'` columns, can greatly reduce these errors.

- **Instruction Misalignment Error:** This error occurs when LLMs deviate from given instructions, often seen in superfluous import statements and comments, typical in training datasets but unnecessary for DataFrame QA tasks. It also includes issues like case sensitivity errors, where LLMs incorrectly handle capitalized names or terms.

  *Solution:* Precise directives in prompts, such as `'Do not import Pandas library'` and specific case sensitivity guidelines, help ensure LLMs' adherence to task-specific requirements.

- **Aggregation Error:** These errors occur when LLMs apply incorrect aggregation functions, often because questions contain words common to both operations and column names, like `'average'`.

  *Solution:* Clear column information and specific query formulations, such as stating the nature of `'allied-unrelated'` columns, guide LLMs to apply the correct aggregation method.

- **Function-Column Ambiguity Error:** This type of error manifests when there is confusion between column names and function names, leading to incorrect query execution.

  *Solution:* Renaming columns may not always work. Encapsulating column names in quotes in queries can distinguish them from function commands, aiding accurate interpretation.

- **Insufficient Column Data/Format Information Error:** These errors often occur due to mismatches between the LLM's assumptions about dataset structure and the actual data format, particularly in handling date-related queries.

  *Solution:* Specifying column formats, such as the format of `'Date'` column, in prompts ensures precise LLM data handling.

- **Coding Syntax Error:** Highlights differences in LLMs' coding capabilities, especially in structuring and executing dataframe queries.

  *Solution:* Choosing an advanced base LLM or training on DataFrame QA datasets enhances their query optimization and data handling skills.

- **Hallucination Error:** This type of error arises when LLMs create responses based on incorrect assumptions or non-existent data, often due to a lack of domain knowledge.

  *Solution:* Providing detailed data and column information in prompts, like explaining how abalone age is determined, helps LLMs bridge domain knowledge gaps and improve query accuracy.

In summary, DataFrame QA tasks present inherent challenges that broadly fall into two categories. The first pertains to the inherent capabilities of LLMs, particularly visible in issues like Instruction Misalignment and Coding Syntax Errors. GPT models, in particular, have a significantly lower error rate in these areas than other models, showcasing their superior ability to align with human instructions and coding accuracy.

The second category comprises challenges unique to the DataFrame QA environment. This includes complexities in question interpretation, table header naming, and domain knowledge gaps, which are central to the task-specific intricacies. Additionally, difficulties that arise from missing table value formats and value range specifications also contribute to this category, leading to errors in query processing. Tackling both the model-intrinsic limitations and these dataframe-specific complexities is vital for enhancing LLMs' performance in DataFrame QA tasks.

## 6. Conclusions

In summary, our introduction of a new DataFrame QA task and framework represents a significant advancement in the field. This zero-shot approach, which leverages dataframe headers and datatypes along with user questions and deliberately excludes table values, addresses data privacy concerns and minimizes extraneous data in prompts. Beyond this, DataFrame QA can further enrich the prompts with dataset descriptions and column data format details, aiding in clarifying the column meanings within dataframes. This method not only offers improved control over code execution outputs but also provides greater scalability compared to traditional Text-to-SQL tasks.

Through testing with advanced open-source and closed-source LLMs, we have analyzed error patterns, challenges and determined that the efficacy of DataFrame QA relies not only on the coding abilities of LLMs but also on their understanding of the relationship between user questions, dataframe columns, and provided instructions. In particular, the accuracy rate of GPT-4 largely consistent with practical applications.

## Acknowledgments

We would like to thank Shreyas Kulkarni for his contribution to the dataset building for this work. The work is in part supported by NSF #2310261 and Federal Highway Administration Exploratory Advanced Research (FHWA EAR) Grant 693JJ320C000021. The views and conclusions in this paper are those of the authors and should not be interpreted as representing any funding agencies.

## References

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Wenhu Chen. Large language models are few (1)-shot table reasoners. *arXiv preprint arXiv:2210.06710*, 2022.

Michael Glass, Mustafa Canim, Alfio Gliozzo, Saneem Chemmengath, Vishwajeet Kumar, Rishav Chakravarti, Avi Sil, Feifei Pan, Samarth Bharadwaj, and Nicolas Rodolfo Fauceglia. Capturing row and column semantics in transformer based question answering over tables. *arXiv preprint arXiv:2104.08303*, 2021.

Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*, 2020.

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.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv preprint arXiv:2310.09263*, 2023.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. Tapex: Table pre-training via learning a neural sql executor. *arXiv preprint arXiv:2107.07653*, 2021.

D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. Uci repository of machine learning databases, 1998. URL http://www.ics.uci.edu/~mlearn/MLRepository.html.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida Wang, and Xi Victoria Lin. Lever: Learning to verify language-to-code generation with execution. In *International Conference on Machine Learning*, pages 26106–26128. PMLR, 2023.

OpenAI. Gpt-4 technical report, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Feifei Pan, Mustafa Canim, Michael Glass, Alfio Gliozzo, and Peter Fox. Cltr: An end-to-end, transformer-based system for cell level table retrieval and table question answering. *arXiv preprint arXiv:2106.04441*, 2021.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*, 2015.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I Wang, et al. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966*, 2022.

Kuan Xu, Yongbo Wang, Yongliang Wang, Zujie Wen, and Yang Dong. Sead: End-to-end text-to-sql generation with schema-aware denoising. *arXiv preprint arXiv:2105.07911*, 2021.

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.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314*, 2020.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task. *arXiv preprint arXiv:1810.05237*, 2018.

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