Table Call: A New Paradigm for Table Question Answering

Anonymous EMNLP submission

Abstract

Large language models (LLMs) have exhibited strong semantic understanding capabilities in interpreting and reasoning for table question answering (TQA). However, they struggle with excessively lengthy or complex input tables, especially when dealing with disorganized or hierarchical structures. To address these issues, we propose a new paradigm for TQA, named **Table Call**, which leverages the tool-using capabilities of LLMs. Specifically, Table Call invokes different tools for various types of table questions, such as SQL, Python, and LLMs, to simplify table understanding. Moreover, to enhance table comprehension capabilities of the LLM, we propose a few-shot library updating technique where we use a dynamically updated library to provide better QA pairs for LLM prompting. Experimental results on both open-domain and specific-domain datasets demonstrate that our approach achieves state-of-the-art performance, significantly outperforming previous methods.

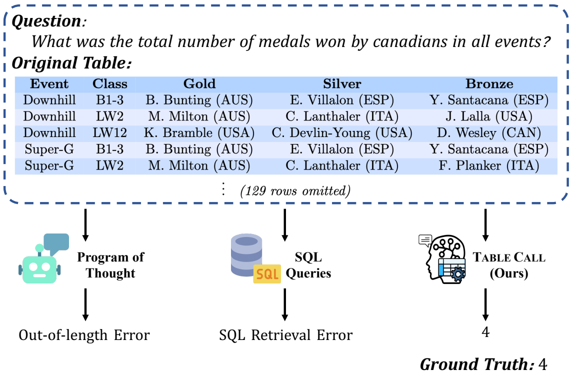

Figure 1: **Comparison between different TQA approaches when handling a lengthy, disorganized table.** LLM-based methods take the entire table as input, resulting in out-of-length errors. SQL-based methods are confused by single cells containing both the name and country abbreviation of the cyclist, leading to SQL retrieval errors. In contrast, our proposed Table Call excels at the TQA task, providing correct answers.

1 Introduction

Table Question Answering (TQA) (Berant et al., 2013; Pasupat and Liang, 2015; Herzig et al., 2020a; Yin et al., 2020) is a critical task in natural language understanding and information retrieval, gaining prominence in fields such as finance and education. TQA evaluates the ability to reason over structured or semi-structured table data, understand the textual content of tables, and integrate free-form natural language questions with table data. The complexity of TQA arises from the unordered nature of table cells and the substantial length of many tables, presenting unique challenges for effective analysis and comprehension.

Earlier SQL-based approaches for Table Question Answering (TQA) (Zhong et al., 2017; Yu et al., 2018) employ semantic parsing to convert natural language table questions into executable commands, such as SQL queries. These queries facilitate the retrieval and manipulation of table data to generate responses, allowing for quick database access without being limited by table length. Recently, large language models (LLMs) like GPT (Ouyang et al., 2022; OpenAI, 2023b; Achiam et al., 2023) and LLaMA (Touvron et al., 2023; MetaAI, 2024) have shown exceptional capabilities in language understanding and generation, providing greater robustness compared to traditional rule-based methods and pre-training fine-tuning paradigms. This has led to extensive research aimed at enhancing TQA using LLMs. Strategies include leveraging LLMs through in-context learning (Chen, 2023; Pourreza and Rafiei, 2024; Zhang et al., 2023; Ye et al., 2023; Chen et al., 2023; Wang et al., 2024) and employing multi-step reasoning via chain-of-thought (CoT) prompting (Zhang et al., 2023; Liu et al., 2023; Chen et al., 2023; Wang et al., 2024). Additionally, some approaches (Zhang et al., 2023) integrate LLMs with tools like SQL or Python to further improve TQA

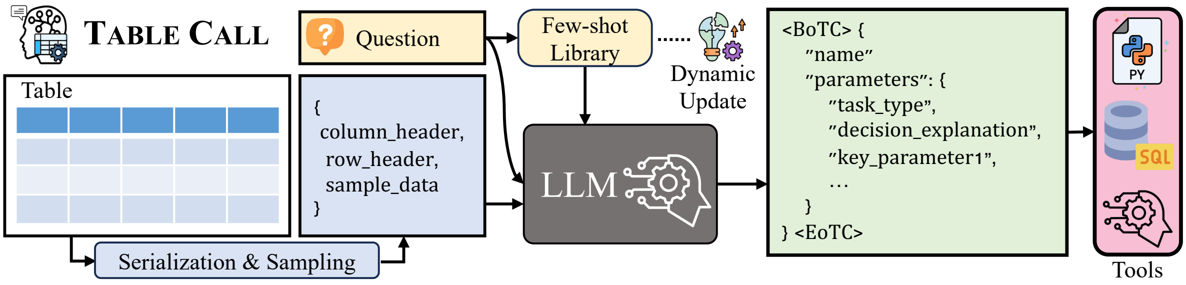

Figure 2: **Overview of TABLE CALL.** We initially serialize the table and sample the first three rows as input. By employing few-shot library updating technique, we guide the large language model to categorize the question types and extract key parameters. The resulting JSON-formatted output is then utilized with various tools.

performance.

While the above strategies are commonly used to handle TQA, they encounter several challenges, as shown in Figure 1: 1) SQL-based methods require converting the question into a precise SQL query, and their performance is critically influenced by the regularity of the table. 2) Large Language Models (LLMs) exhibit inadequate table comprehension capabilities when facing complex tables, such as disorganized or hierarchical tables. They tend to treat TQA as a uniform language task, neglecting the different types of table tasks and performing poorly in numerical reasoning, aggregation, comparison, and understanding of layout information. Moreover, LLMs struggle with lengthy tables that consume many tokens, leading to a decline in performance as the number of tokens increases.

To overcome the above problems, we propose a novel paradigm for table question answering called TABLE CALL, as depicted in Figure 2. Our approach combines the immunity of tools like SQL to table length limitations with the powerful comprehension and reasoning capabilities of LLMs. Unlike previous methods, our approach classifies question types and utilizes the appropriate tools, including SQL, Python, and LLMs, to address each corresponding question type. In the first phase of TABLE CALL, we categorize table-related questions and extract key information from both the tables and questions through few-shot library updating, while inputting sampled table data to avoid out-of-length errors. In the second stage, our model selects and leverages tools such as LLMs, SQL, and Python to more accurately answer specific questions.

Previous methods (Chen, 2023; Pourreza and Rafiei, 2024) have demonstrated the powerful capabilities of in-context learning in the TQA task. By adding few(1)-shot examples, LLMs can quickly learn to answer TQA questions. However, this approach relies heavily on the quality of the few-shot examples, as demonstrated by Nori et al. (2023). Our study also utilizes dynamic few-shot learning to enhance TQA performance. We propose a few-shot library updating technique based on dynamic few-shot learning (Nori et al., 2023) to enable LLMs to better understand and answer questions. We feed the output of an LLM, along with the table and the question, into another LLM acting as an evaluator. This evaluator assesses the quality of the generated output and checks whether the QA pair should be added to the few-shot library, thereby becoming part of future few-shot examples. By dynamically updating the few-shot library, we can provide better QA pairs as few-shot examples for LLM prompting.

The contributions of this paper can be summarized as follows:

- We present a novel method named TABLE CALL, classifying table problems into corresponding tasks and applying specific tools for each task. Without exceeding the token limits of LLMs, this approach can handle tables up to ten times longer for certain TQA tasks than common LLM-based methods.
- We incorporate few-shot library updating technique to generate better few-shot examples and enhance table comprehension capabilities and reduce hallucinations.
- Extensive experiments on pubic benchmark datasets WikiTableQuestions and AIT-QA, demonstrate that our proposed TABLE CALL outperforms the state-of-the-art methods.

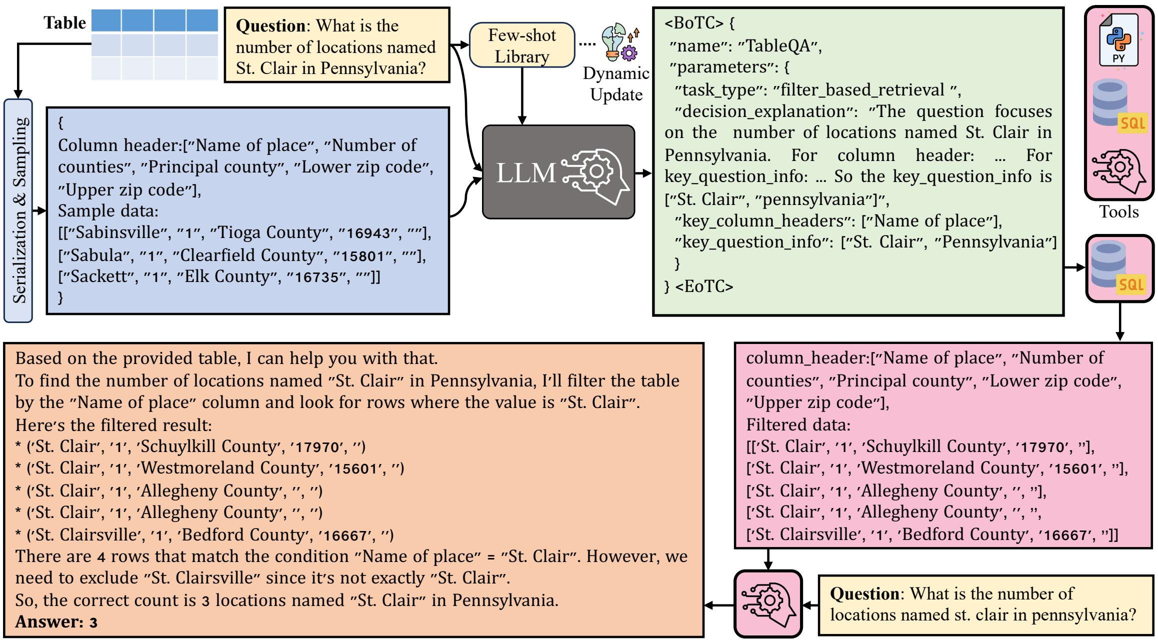

Figure 3: **Example of TABLE CALL processing a filter-based retrieval question.** The table length is 517. Baseline approaches suffer from the token length limits of LLMs, leading to out-of-length errors. However, our approach avoids such errors by initially inputting only sample data into the model.

2 Related Works

2.1 Table Question Answering

Table question answering (TQA) is a task of language reasoning from table data. It tests the ability to reason over structured or semi-structured data, understand textual table contents and fuse free-form natural language questions with table data.

Early works conducted semantic parsing through hand-crafted features and grammar rules to transform table questions into executable commands (Berant et al., 2013; Pasupat and Liang, 2015; Yin and Neubig, 2017; Zhong et al., 2017; Yu et al., 2018). However, these methods require converting the question into a strict SQL/Python query statement, and the regularity of the table influences the performance bottleneck critically.

Pretrained language models, trained on extensive tabular data, gain a general syntactic and semantic understanding of tables. Thus these models can encode tables and generate answers directly (Herzig et al., 2020a; Yin et al., 2020; Liu et al., 2022; Xie et al., 2022; Zhou et al., 2022; Deng et al., 2022; Zhong et al., 2022; Sundar and Heck, 2023; Yu et al., 2023). These methods have high training costs and lack interpretability however.

Some works have shown that adding few-shot learning to large language models (LLMs) significantly improves TQA accuracy (Chen, 2023; Pourreza and Rafiei, 2024). This capability of LLMs can also be applied to answering tabular questions. However, simply adding few-shot examples lacks interpretability and does not fully unleash the potential of LLMs. Subsequent works use various strategies to better guide LLMs in TQA interpretation and reasoning. ReAcTable (Zhang et al., 2023) generates intermediate data representations using external tools such as SQL and Python code executors, transforming TQA tasks into a more accessible format. Similarly, Binder (Cheng et al., 2023) splits the reasoning phase and uses external tools. Ye et al. (2023) generate sub-tables and sub-questions with SQL queries through in-context learning. Liu et al. (2023) aggregate textual and symbolic reasoning and use a mix self-consistency mechanism to get the answer. Chen et al. (2023) propose Program-of-Thoughts to generate step-by-step python code for complex numerical reasoning tasks. CHAIN-OF-TABLE (Wang et al., 2024) guides LLMs to iteratively generate operations and update the table, creating a table reasoning chain. Liu et al. (2024) construct an augmenting table with external information and then generate SQL queries over both tables to answer questions.

Some methods have been proposed to handle lengthy tables. Zhao et al. (2023) reconstruct hierarchical tables into a tree structure and employ multi-turn QA for long-text tables. Sui et al. (2024)

introduce predefined certain constraints to meet the LLM call request. Binder (Cheng et al., 2023) inputs only three tables rows for all table sizes. We draw inspiration from Binder to tackle the issue of lengthy tables causing the LLM to exceed its input length limits by inputting only the first three rows. During the tool phase, we adeptly resolve the issues of information loss caused by this truncated input method.

2.2 Function Calling

Function calling is a technology first introduced by OpenAI in June 2023 (OpenAI, 2023a). It connects large language models (LLMs) to external tools. Models are trained to both detect when a function should to be called (depending on the input) and to respond with JSON that adheres to the function signature. The basic sequence of steps for function calling is as follows: 1. Call the model with the user query and a set of functions defined in the functions parameter. 2. The model can choose to call one or more functions; if so, the content will be a stringified JSON object adhering to your custom schema. 3. Parse the string into JSON, and call the function with the provided arguments if they exist. 4. Call the model again by appending the function response as a new message, and let the model summarize the results back to the user.

3 Method

3.1 Overview

Figure 2 illustrates an overview of the proposed TABLE CALL. TABLE CALL receives a natural language query Q and a table T as inputs. Table T comprises column headers H_{column}, data D, and potentially row headers H_{row}. For hierarchical tables, T features multi-layered headers for both columns and rows. During the calling phase, we initially serialize the table T and sample the data D. A large language model (LLM), enhanced with few-shot library updating, is then employed to determine the task type of the question Q, generate pertinent key parameters, and provide explanations for its decision-making process. Based on the JSON-formatted output from the calling phase and the question, various tools are then used to generate the final results.

3.2 Calling Phase

3.2.1 Serialization and Sampling

Function calling (OpenAI, 2023a) involves the capability within an API to describe and invoke one or more functions, enabling the model to intelligently produce a JSON object with arguments that can be used to execute the specified functions. In this paper, we leverage this concept to guide large language models (LLMs) to classify task types of the question Q, and to generate corresponding key parameters for each task.

The input of the calling phase is a question Q, column headers H_{column}, the sample rows of data D_{sample}, and possibly row headers H_{row}. For hierarchical tables, the column headers H_{column} and the row headers H_{row} are nested. We simply flatten the header. This can retain the layout information of the table to the greatest extent, with the cost of taking up more token input. We refer to the initial three rows of data D fed into the model as D_{sample}, drawing inspiration from Binder (Cheng et al., 2023). Given the token limitations of the model, inputting only D_{sample} addresses out-of-length error associated with lengthy tables. Furthermore, D_{sample} facilitates the model's comprehension of the overall table structure and the data representation types present in the complete table.

3.2.2 Few-shot Library Updating

Incorporating few-shot learning, even with a single example, considerably enhances the reasoning capabilities of large language models (LLMs) (Chen, 2023; Pourreza and Rafiei, 2024). Nevertheless, for table-based questions that encompass multiple task types, it is crucial to supply better question-answer pairs that enhance the model's comprehension of both the table and the question. Inspired by Nori et al. (2023), we introduce few-shot library updating technique during the calling phase. This strategy can provide better QA pairs and aids in the precise classification of question tasks and the extraction of key parameters.

As illustrated in Figure 4, we utilize a few-shot library consisting of basic question-JSON pairs.

In the first stage, for any given question, we select k semantically similar few-shot examples using k-NN clustering within the embedding space. In Section 4.4, we discuss the impact of the choice of k on the outcomes.

In the second stage, these k few-shot examples, along with the column header H_{column}, the row header H_{row}, and the sample data D_{sample}, are input into the large language model (LLM) as prompts. This setup facilitates the generation of a JSON-formatted output that includes the task type, key parameters, and decision explanation. The task type and key parameters are subsequently used to invoke additional tools. Meanwhile, drawing from the Chain of Thought (CoT) approach (Wei et al., 2022), we prompt our LLM to generate a series of intermediate reasoning steps, termed decision explanation. We provide detail explanations in the Appendix A.1 on how to use prompts during the calling phase to generate JSON outputs.

We use an alternative LLM to evaluate the task-type and key parameters in the output. If the JSON-formatted output is accurate, we update the few-shot library by adding the new question-JSON pair. This iterative refinement ensures the continuous enhancement and relevance of our few-shot library, thereby improving the performance over time.

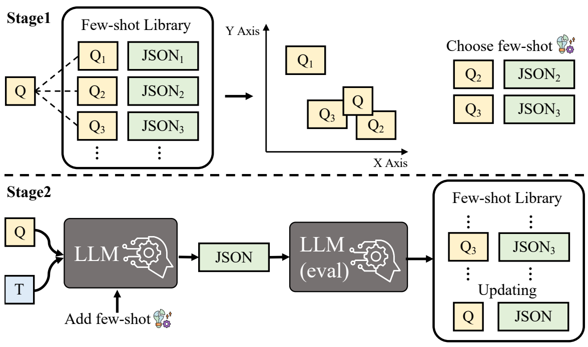

Figure 4: **Overview of our proposed few-shot library updating technique.** In the first stage, we select k few-shot examples by computing and compare the similarity. In the second stage, we update the few-shot library by judging the generating JSON-formatted output and incorporating the new question-JSON pair.

3.2.3 Task Type Classifier

In the realm of Table Question Answering (TQA), we encounter a diverse range of question tasks, each requiring distinct reasoning strategies. These tasks can be broadly categorized into five types: Direct Retrieval, Filter-Based Retrieval, Aggregation, Comparison, and Sequential/Relative Positioning.

Direct Retrieval requires identifying specific rows and columns using key information to directly access the answer within the table. This involves defining key column headers and key question information, which ideally allows tools to directly retrieve answers.

Filter-Based Retrieval retrieves data using specific criteria applied to one or more columns. This method differs from Direct Retrieval as it often involves complex query conditions that are not directly derivable from the sample data D_{sample}.

Aggregation tasks filter data on certain criteria before performing operations like summing, averaging, or counting. Parameters for Aggregation tasks include key column headers, key question information, and task-specific commands like SUM, AVG, or COUNT.

Comparison tasks involve data filtering and comparing values to identify extremes such as the highest or lowest values. Key parameters contain the key column header, key question information and comparison terms like 'highest'.

Sequential/Relative Positioning tasks focus on the sequence or relative positioning of table items, typically involving prepositions like 'after' or 'directly before' indicating a relational query concerning sequence. For these types of tasks, it's not possible to directly locate useful row information from D_{sample}. Therefore, the key parameters are the corresponding row information and relative positioning prepositions.

3.3 Tools

In the field of table question answering (TQA), integrating large language models (LLMs) with external tools is becoming increasingly prevalent (Zhang et al., 2023; Liu et al., 2024). In this paper, we employ a combination of distinct tools: SQL, Python, and large language models (LLMs), tailored to different task types within table question answering. For more detailed examples, please refer to Appendix A.2.

We input the complete data D into a SQL database to circumvent issues associated with exceeding the token limits.

For Direct Retrieval and Filter-Based Retrieval, by leveraging SQL, we can identify rows corresponding to the key question information and subsequently locate the relevant column using the key column header. If there is only one row filtered, we directly determine the answer. If there are more than one rows, we then use a task-based LLM as the reansoning tool to further reasoning and get the answer.

For Aggregation tasks, we first use SQL to identify related rows and columns. We then use a task-based LLM to combining the key task information and use Python shell to compute the final result.

For Comparison tasks, we similarly first use SQL and then input the question, filtered rows, and key task information into the LLM.

For Sequential or Relative Positioning tasks, we directly use the taskbased LLM as relying solely on the sample data D_{sample}, we cannot determine the sequential or relative positioning table item.

3.4 Handling Exceptions

Given our method involves converting strings to JSON and code, there is an inherent risk of encountering execution errors.

After the calling phase, we generate a JSON-formatted file. Typically, we use <BoTC> and <EoTC> as specific identifiers to locate the JSON output. However, even though few-shot library updating technique can guide the LLM to generate the JSON output, the high requirements for JSON formatting and the inherent randomness of LLM outputs can lead to errors in the generated JSON. Specifically, these errors can manifest as symbol misplacements or irrelevant responses.

- Symbol misplacements can cause JSONDecodeError, such as an extra or missing bracket. In such cases, we employ additional scripts to check and correct these errors.

-Irrelevant responses refer to situations where the LLM fails to correctly output key parameters, preventing the accurate selection of rows and columns based on these parameters.

When using tools based on the JSON-formatted file, different exceptions may arise:

- SQL exceptions occur when the SQL query requires a non-existing column header or row data required do not exist in the SQL database.

- Python exceptions are similar to JSON exceptions involving symbol errors, where the generated Python code may be non-standard.

To address these exceptions, we input the table data into an LLM, and a task-guided chain-of-thought LLM directly outputs the results.

4 Experiment

4.1 Experimental Setup

4.1.1 Datasets

We conduct extensive experiments on two datasets: the open-domain table question-answering dataset WikiTableQuestions (Pasupat and Liang, 2015) and the aviation-domain hierarchical table question-answering dataset AIT-QA (Katsis et al., 2022).

WikiTableQuestions consists of tables sourced from Wikipedia. Each task involves answering a question based on a given table. The dataset includes 2,108 tables on various topics and 22,033 questions of varying complexity. For our experiments, we use the test set, comprising 4,344 question-answer pairs. This dataset features complex questions that require multi-step reasoning and various data operations such as comparison, aggregation, and arithmetic computation.

AIT-QA is a question-answering dataset on hierarchical tables in the aviation industry, consisting of 116 tables with a total of 515 question-answer pairs. Tables in AIT-QA have a much more complex layout than Wikipedia tables, featuring hierarchical row and column headers and domain-specific terminology. Thus, AIT-QA serves as a valuable extension and supplement to WikiTableQuestions.

The two datasets encompass a wide variety of tables and questions that require multi-step reasoning and various data operations, including comparison, aggregation, arithmetic computation, and layout understanding.

4.1.2 Baselines

For the WikiTableQuestions dataset, we compare our method with training-based methods (Yin et al., 2020; Liu et al., 2022; Zhou et al., 2022; Jiang et al., 2022; Ni et al., 2023) and prompt-based methods (Cheng et al., 2023; Zhang et al., 2023; Ye et al., 2023; Wang et al., 2024; Liu et al., 2023).

For the AIT-QA dataset, we compare our method with the state-of-the-art method Zhao et al. (2023) and the methods in the original AIT-QA paper, including TABERT (Yin et al., 2020), TaPas (Herzig et al., 2020b) and RCI (Katsis et al., 2022).

4.1.3 Model

Previous prompt-based methods mainly employ GPT-3.5 (OpenAI, 2023b) as benchmarks. However, due to the per-second concurrency limits and overall resource constraints of GPT platforms, we opt to utilize open-source LLMs. Due to resource limitations, we randomly sampled one-third of AIT-QA (Katsis et al., 2022) for comparative experiments with both GPT-3.5-turbo and LLaMA3-8B (MetaAI, 2024). As shown in Table 1, the accuracy of both models was nearly identical.

Thus, we conduct experiments mainly with the LLaMA3-8B. LLaMA3-8B uses a tokenizer with a vocabulary of 128K tokens that encodes language more efficiently. LLaMA3-8B supports a maximum of 8,192 input tokens, while GPT-3.5-turbo supports up to 16,385 tokens. This means that the combined count of the input tokens and the generated tokens for LLaMA3-8B cannot exceed 8,192, or the model will return an out-of-length error.

Methods	Accuracy
gpt-3.5-turbo	77.5
LLaMA3-8B	78.4

Table 1: Model capabilities on the AIT-QA dataset.

4.1.4 Implementation Details

We compared random sampling and selecting the first three rows of table data and found no significant difference. Hence, we opt to sample the first three rows as input. In the calling phase, we use the LLaMA3-8B (MetaAI, 2024) as a evaluater to judge the quality of generated JSON output. We use SQLite (Consortium, 2024) to run SQL queries and use Python shell to run Python code.

4.1.5 Metrics

In this paper, we use accuracy between the model-predicted answer and the ground-truth answer to compare the response quality of TABLE CALL with the baseline approaches. In specific, we use the Flexible Denotation Accuracy (FDA), which compares results after removing units (years, $, etc).

4.2 Comparison with State-of-the-art Methods

Table 2 shows the comparison result on the WikiTableQuestions dataset (Pasupat and Liang, 2015). Our model is compared with both training-based-LLM method and prompt-based-LLM method, and achieves the state-of-the-art performance. The results indicate that TABLE CALL excels at answering multi-step reasoning questions on disorganized and lengthy tables.

The results on the AIT-QA dataset (Katsis et al., 2022) are shown in Table 3. Our method significantly outperforms other methods on every data subset of the AIT-QA dataset. The results show that TABLE CALL excels in complex table understanding. Unlike other methods that require serializing tables into a tree structure or a specific SQL sequence, we simply flatten the nested headers of the table without further operations. This highlights the universality and efficiency of our approach.

Methods	Accuracy
Training-based LLMs	
TABERT (Yin et al., 2020)	52.3
Tapex (Liu et al., 2022)	57.5
TaCube (Zhou et al., 2022)	60.8
OmniTab (Jiang et al., 2022)	62.8
LEVER (Ni et al., 2023)	65.8
Prompt-based LLMs	
Binder (Cheng et al., 2023)	64.6
ReAcTable (Zhang et al., 2023)	65.8
Dater (Ye et al., 2023)	65.9
CHAIN-OF-TABLE (Wang et al., 2024)	67.3
Mix SC (Liu et al., 2023)	73.6
Ours	**77.6**

Table 2: Accuracy on WikiTableQuestions.

Data subset	TABERT	TaPaS	RCI	LLMCTP	Ours
KPI-driven	41.4	48.3	60.0	74.5	**91.7**
Table-driven	31.1	50.0	48.6	71.8	**81.1**
Row header hierarchy	21.9	47.3	45.9	61.6	**82.2**
No row header hierarchy	38.8	50.4	54.2	81.8	**84.8**
Overall	34.0	49.3	51.8	76.3	**84.1**

Table 3: Accuracy on AIT-QA.

4.3 Result on Lengthy Tables

End-to-end TQA often fails or degrades in performance because it relies on the whole table as input for reasoning. Thanks to the strategic approach of only inputting the first three rows of the table during the calling phase and invoking different tools for various types of table questions, TABLE CALL excels at reasoning lengthy tables, effectively managing token limitations while still capturing essential data features. As depicted in Figure 5, the performance of LLaMA3-8B with chain-of-thought shows a sharp decline as the table size increases. In contrast, TABLE CALL maintains a consistently higher performance, exhibiting only minimal reductions.

4.4 Few-Shot Library Updating

A significant advantage of TABLE CALL is its adaptability. We can continually refine our model by updating our few-shot library during inference.

Table 6 shows the performance of TABLE CALL on WikiTableQuestions using 0-shot, 1-shot, and 3-shot, with and without updates to the few-shot library. We created two sizes of the original few-shot libraries, with the raw library built from selections from the training set. The 0-shot model under-

Type	Direct Retrieval	Filter-Based Retrieval	Aggregation	Comparison	Sequential/Relative Positioning	Overall
LLaMA3-8B	82.6	68.1	56.5	76.1	72.0	71.1
Ours	88.7	73.4	75.6	76.8	77.6	**77.6**

Table 4: Performance across different task types on the WikiTableQuestions dataset.

Type	Direct Retrieval	Filter-Based Retrieval	Aggregation	Comparison	Sequential/Relative Positioning	Overall
LLaMA3-8B	81.6	70.6	71.4	62.5	-	78.45
Ours	86.1	78.6	85.7	75	-	**84.1**

Table 5: Performance across different task types on the AIT-QA dataset.

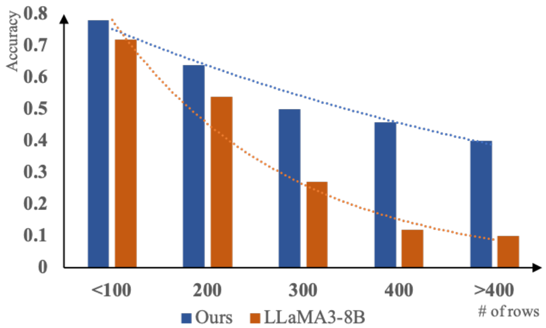

Figure 5: Lengthy Table Performance Comparison on the WikiTableQuestions dataset.

Strategy	Similarity	Accuracy
0-shot	-	56.2
Raw few-shot library with size 5		
1-shot		
w/o update	0.52	69.5
w/ update	0.56	73.9
3-shot		
w/o update	0.46	67.3
w/ update	0.52	75.3
Raw few-shot library with size 50		
1-shot		
w/o update	0.57	71.4
w/ update	0.62	76.8
3-shot		
w/o update	0.54	72.0
w/ update	0.61	77.6

Table 6: Comparison of few-shot strategies on the WikiTableQuestions dataset.

performs compared to the direct chain-of-thought approach with LLaMA-8B due to the stringent requirements for generating JSON-formatted outputs and the complexities of task-type classification and key parameter extraction. However, incorporating few-shot learning significantly enhances our model's ability to answer table-based questions. Continuously updating the few-shot library further improves accuracy and enhances the model's overall understanding abilities. Given the specific-domain nature of AIT-QA, with a 0.75 similarity between questions and library examples, we only need five examples in the raw library.

4.5 Task Type Classifier

As shown in Table 4 and Table 5, TABLE CALL categorizes question into five distinct task types on both WikiTableQuestions and AIT-QA datasets. We compared our method with LLaMA3-8B using chain-of-thought prompting. Benefiting from the TABLE CALL paradigm and few-shot library updating for task type classification, our approach consistently outperform the end-to-end LLaMA model across all tasks. Specifically, aggregation-type questions pose a dual challenge: selecting key rows and executing complex numerical computations. The direct end-to-end approach proves less effective. In contrast, our method not only classifies questions but also extracts key information from them and employs SQL to pinpoint relevant rows. Subsequent numerical computations are facilitated by a LLM within a Python shell. This process significantly enhances the interpretability and execution efficiency of the reasoning, effectively minimizing model hallucinations.

5 Conclusion

The proposed TABLE CALL is a novel method invoking different tools for table question answering in complex and lengthy tables. Unlike the existing methods, TABLE CALL introduces well-designed calling phase with few-shot library updating technique to classify tabular question types, enhancing table interpreting and reasoning. A large-scale empirical study on the WikiTableQuestions and AIT-QA datasets demonstrates that TABLE CALL achieves state-of-the-art performance in table question answering tasks.

Limitations

In our classification of tabular question types, we divide questions into five categories: Direct Retrieval, Filter-Based Retrieval, Aggregation, Comparison, and Sequential/Relative Positioning. This categorization is sufficient for the benchmarks used in this paper. However, we can further expand these categories to include more tasks, such as Table-to-Text.

In this paper, we employ SQL, Python, and large language models (LLMs) as tools. These tools are adequate for handling the vast majority of table tasks. Nonetheless, the tools in our method are extensible and can be integrated with any table processing or understanding approach, such as adding a voting mechanism among the tools.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

Wenhu Chen. 2023. Large language models are few (1)-shot table reasoners. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1120–1130.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding language models in symbolic languages. In *Proceedings of the Eleventh International Conference on Learning Representations*.

SQLite Consortium. 2024. Sqlite.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *ACM SIGMOD Record*, pages 33–40.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Mueller, Francesco Piccinno, and Julian Eisenschlos. 2020a. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020b. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333.

Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. Omnitab: Pretraining with natural and synthetic data for few-shot table-based question answering. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 932–942.

Yannis Katsis, Saneem Chemmengath, Vishwajeet Kumar, Samarth Bharadwaj, Mustafa Canim, Michael Glass, Alfio Gliozzo, Feifei Pan, Jaydeep Sen, Karthik Sankaranarayanan, et al. 2022. Ait-qa: Question answering dataset over complex tables in the airline industry. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*, pages 305–314.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: Table pre-training via learning a neural SQL executor. In *Proceedings of the International Conference on Learning Representations*.

Tianyang Liu, Fei Wang, and Muhao Chen. 2023. Rethinking tabular data understanding with large language models. *arXiv preprint arXiv:2312.16702*.

Yujian Liu, Jiabao Ji, Tong Yu, Ryan Rossi, Sungchul Kim, Handong Zhao, Ritwik Sinha, Yang Zhang, and Shiyu Chang. 2024. Augment before you try: Knowledge-enhanced table question answering via table expansion. *arXiv preprint arXiv:2401.15555*.

MetaAI. 2024. Introducing meta llama 3: The most capable openly available llm to date.

Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen-tau Yih, Sida I Wang, and Xi Victoria Lin. 2023. Lever: learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning*, pages 26106–26128.

Harsha Nori, Yin Tat Lee, Sheng Zhang, Dean Carignan, Richard Edgar, Nicolo Fusi, Nicholas King, Jonathan Larson, Yuanzhi Li, Weishung Liu, et al. 2023. Can generalist foundation models outcompete special-purpose tuning? case study in medicine. *Medicine*, pages 77–3.

OpenAI. 2023a. Function calling and other api updates.

OpenAI. 2023b. Gpt-3.5 turbo.

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

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1470–1480.

Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. In *Advances in Neural Information Processing Systems*, pages 24824–24837.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654.

Anirudh S Sundar and Larry Heck. 2023. ctbls: Augmenting large language models with conversational tables. In *Proceedings of the 5th Workshop on NLP for Conversational AI*, pages 59–70.

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

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, pages 24824–24837.

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. 2022. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 602–631.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. 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.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 440–450.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426.

Bowen Yu, Cheng Fu, Haiyang Yu, Fei Huang, and Yongbin Li. 2023. Unified language representation for question answering over text, tables, and images. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4756–4765.

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*, pages 3911–3921.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M Patel. 2023. Reactable: Enhancing react for table question answering. In *Proceedings of the VLDB Endowment*, page 1981–1994.

Bowen Zhao, Changkai Ji, Yuejie Zhang, Wen He, Yingwen Wang, Qing Wang, Rui Feng, and Xiaobo Zhang. 2023. Large language models are complex table parsers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14786–14802.

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

Wanjun Zhong, Junjie Huang, Qian Liu, Ming Zhou, Jiahai Wang, Jian Yin, and Nan Duan. 2022. Reasoning over hybrid chain for table-and-text open domain question answering. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 4531–4537.

Fan Zhou, Mengkang Hu, Haoyu Dong, Zhoujun Cheng, Fan Cheng, Shi Han, and Dongmei Zhang. 2022. Tacube: Pre-computing data cubes for answering numerical-reasoning questions over tabular data. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2278–2291.

A Appendix

A.1 Prompts for Calling

In our prompts, we incorporate in-context learning and chain-of-thought approaches to enhance the large model's ability to understand task type classification and extract key parameters. Figure 6 presents our prompt for defining and invoking the table call function. Figure 7 then shows the prompt for using input and few-shot examples. The combination of Figure 6 and Figure 7 constitutes the full text of the prompt for the calling phase.

A.2 Usage of Tools

In this paper, we propose 5 question task types: Direct Retrieval, Filter-Based Retrieval, Aggregation, Comparison, and Sequential/Relative Positioning. In Figure 4, the example shows how TABLE CALL uses tools for a question with filter-based retrieval type. We then present examples of the remaining four task types.

As shown in Figure 8, the large language model (LLM) identifies this question as a direct retrieval task and outputs the key column header and key question information in a JSON format. Subsequently, an SQL query is employed to fetch question-related rows, and the final answer is directly determined according to the key column name. This method, which solely relies on SQL, is the fastest. If multiple rows are retrieved, we then employ a method similar to that illustrated in Figure 4, using the LLM for further reasoning to produce the final result.

Figure 9 shows the process for the aggregation type. We first utilize SQL to extract rows and columns relevant to the question. Subsequently, we employ a LLM along with a Python shell to compute and derive the final result.

Figure 10 shows the process for the comparison type. In this example, similar to the case presented in Figure 4, we first use SQL to extract filtered rows and then apply a LLM to determine the final result. The distinction lies in the inclusion of a comparison term in the key parameters.

Figure 11 demonstrates that we utilize a task-guided Chain of Thought LLM to address questions involving sequential or relative positioning. This type of question necessitates the use of a complete table to determine the sequence or relative position of items. Similarly, in cases of exceptions, we input the table data into a LLM to directly provides the results.

```
Role : SYSTEM
Content : You are a proficient artificial intelligence assistant, specialized in performing interface
     parameter parsing tasks, locating and parsing parameters effectively.
Role : FUNCTION
Content : The available function calls are as follows. If function call is related to user's question, help
     me return function call and paremeters. Responses should be formatted as "<BoFC> JSONDICT <EoFC>",
     where <BoFC> and <EoFC> are specific identifiers, and JSONDICT is a JSON dictionary that can be parsed
     using json.loads(). JSONDICT consists 2 keys: "name" and "parameters". The "name" key is the name of
     the function call, and the "parameters" key is a dictionary where "arg" is the name of a function call
     parameter and "value" is the value of that parameter. The value must be explicitly mentioned by the
     user or default parameter value). For example, The return of function call should be like "<BoFC> {{"
     name": "function call name", "parameters": {{"arg1": "value1", "arg2": "value2"...}}}} <EoFC>".
Role : FUNCTION
Content : {
    "name": "TableQA",
    "description": "Choose the task type of the question. Based on the task type, retrieve key parameters
         from the user's question.
    This function analyzes the question to identify key terms and phrases, matching them with available
         column headers to determine the most applicable ones. It also compares these terms with items in
         the sample data list to select pertinent key question info. Note that the sample data list is only
         a preview, illustrating the table's structure and data types, not the complete dataset. The actual
         table contains many more rows and additional data not shown in the sample, ensuring comprehensive
         data retrieval.",
    "parameters": {
        "type": "object",
        "properties": {
            "task_type": {
                "type": "string",
                "description": "According to the question, choose the task type from the following list: ["
                     Direct Retrieval", "Filter-Based Retrieval", "Aggregation", "Comparison", "Sequential/
                     Relative Positioning"]. Describe each task type and corresponding key parameters:..."},
            "decision_explanation":{
                "type": "string",
                "description": "Provide a detailed rationale for selecting specific parameters."},
            "key_parameter1": {...},
            "key_parameter2": {...},
            ...},},
    "required": ["task_type", "decision_explanation", "key_parameter1", "key_parameter2", ...]}
```

Figure 6: The prompt for define table call.

```
Role : USER
Content : Example:
    Input:{"column header list": [EXAMPLE_TABLE_COLUMN_HEADER_HERE]},
        "row header list": [EXAMPLE_TABLE_ROW_HEADER_HERE],
        "sample data list": [EXAMPLE_TABLE_SAMPLE_DATA_HERE]}
    Question: [EXAMPLE_QUSTION_HERE].
    Expected response: [EXAMPLE_RESPONSE_HERE].
Role : USER
Content : The sample data list consists of initial few rows of the table, providing a preview to assist in
     understanding the data and the overall structure of the table. Please note that the actual table
     contains many more rows.
    Input:{"column header list": [TABLE_COLUMN_HEADER_HERE],
        "row header list": [TABLE_ROW_HEADER_HERE],
        "sample data list": [SAMPLE_DATA_HERE]}
    Question:[QUSTION_HERE]
    Expected response:
```

Figure 7: The prompt of the user input.

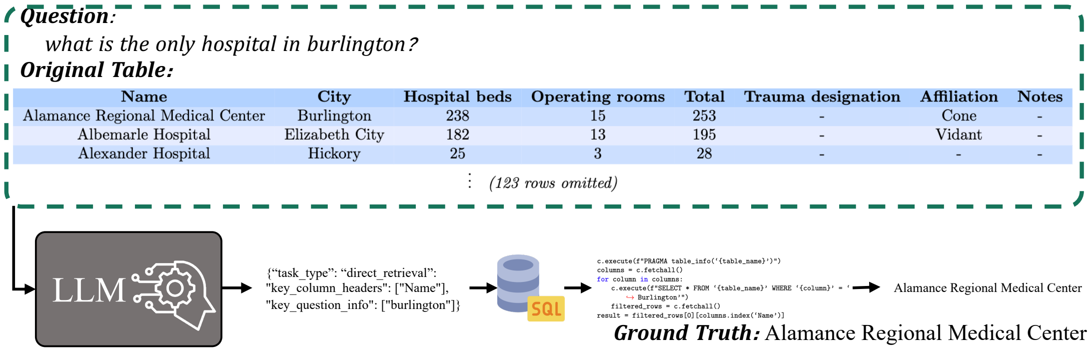

Figure 8: The example of TABLE CALL for a question with a direct retrieval type.

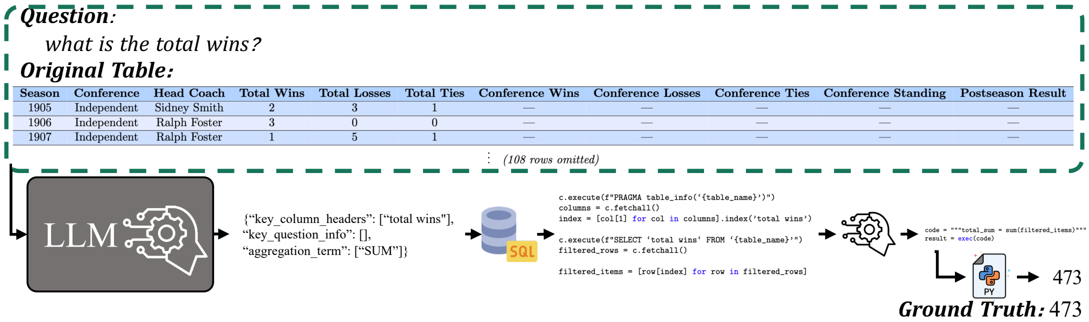

Figure 9: The example of TABLE CALL for a question with an aggregation type.

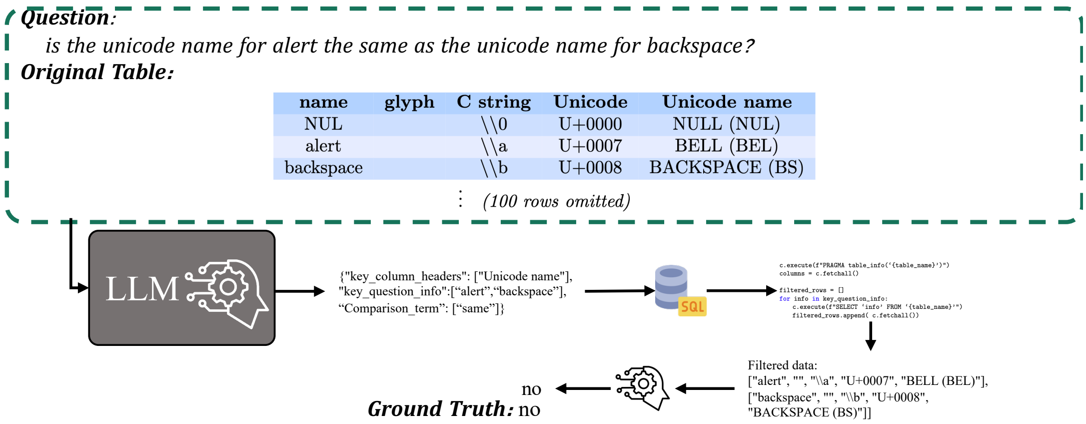

Figure 10: The example of TABLE CALL for a question with a comparison type.

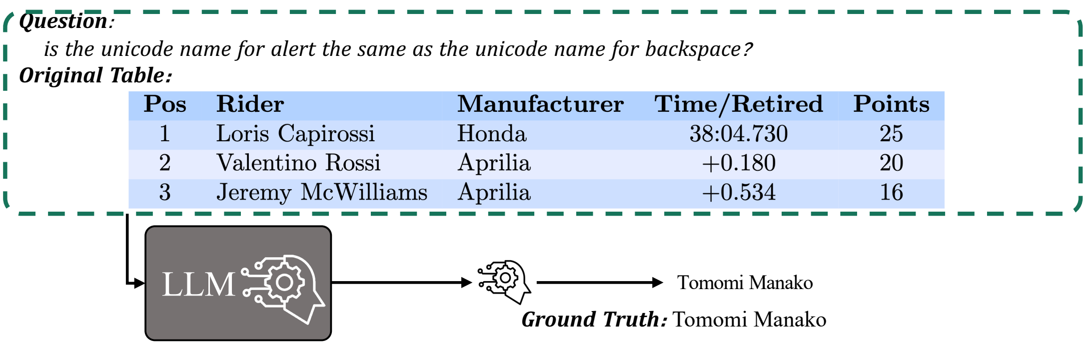

Figure 11: The example of TABLE CALL for a question with a sequential/relative positioning type or a question yielding exceptions.