sonrobok4 Team at SemEval-2025 Task 8: Question Answering over Tabular Data Using Pandas and Large Language Models

Nguyen Minh Son[1,2] and **Dang Van Thin**[1,2]
[1]University of Information Technology, Ho Chi Minh City, Vietnam
[2]Vietnam National University, Ho Chi Minh City, Vietnam
22521254@gm.uit.edu.vn, thindv@uit.edu.vn

Abstract

This paper describes the system of the sonrobok4 team for the SemEval-2025 Task 8: DataBench, Question-Answering over Tabular Data. The task requires answering questions based on the given question and dataset ID, ensuring that the responses are derived solely from the provided table. We address this task by using large language models (LLMs) to translate natural language questions into executable Python code for querying Pandas DataFrames. Furthermore, we employ techniques such as a rerun mechanism for error handling, structured metadata extraction, and dataset preprocessing to enhance performance. Our best-performing system achieved 89.46% accuracy on Subtask 1 and placed in the top 4 on the private test set. Additionally, it achieved 85.25% accuracy on Subtask 2 and placed in the top 9. We mainly focus on Subtask 1. We analyze the effectiveness of different LLMs for structured data reasoning and discuss key challenges in tabular question answering.

1 Introduction

The SemEval 2025 Task 8 (Osés-Grijalba et al., 2025) focuses on developing systems for Question Answering (QA) over tabular data, a critical subfield of natural language processing (NLP) with applications in business intelligence, automated data analytics, and financial reporting. Unlike traditional QA tasks that rely on retrieving information from free-text documents, this challenge requires models to derive answers solely from structured, tabular data. The DataBench benchmark comprises 65 real-world datasets drawn from diverse domains, with each dataset accompanied by 20 human-generated questions and corresponding answers. This setup demands that models not only retrieve information accurately but also perform the necessary computations and reasoning over multiple table columns.

This paper introduces a system designed to address the challenges of question answering over tabular data (Tabular QA). Our approach translates natural language queries into executable Python code for direct interaction with the provided data. To improve reliability and performance, we incorporate a custom error recovery mechanism and leverage the power of several LLMs: `gpt4o-mini`, `DeepSeek`, and an open-source model. We evaluated these models with our proposed approach using various prompt configurations. Through extensive testing and comparison, we assessed each model's performance on our Tabular QA tasks, ultimately selecting the model that provided the best overall accuracy and robustness for our final system.

2 Related Work

Tabular question answering (QA) has evolved considerably over the past few years, driven by the creation of diverse datasets and innovative methodological approaches. In this section, we discuss methodological advancements, emphasizing recent paradigm shifts enabled by large language models (LLMs).

Early methods translated natural language questions into logical forms (e.g., SQL queries) for structured data retrieval. Systems such as WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018) exemplify this paradigm, achieving high accuracy under constrained conditions. However, their reliance on predefined schemas limits applicability to free-form questions and complex table structures.

End-to-end neural models like TAPAS (Herzig et al., 2020) advanced the field by jointly encoding tables and questions using transformer architectures. TAPAS incorporates specialized positional embeddings to preserve table structure during pretraining, enabling direct answer prediction without intermediate SQL generation. Despite their effec-

Proceedings of the The 19th International Workshop on Semantic Evaluation (SemEval-2025), pages 357–362

tiveness, such models require domain-specific fine-tuning and struggle with numerical reasoning tasks requiring explicit computation.

Recent approaches, notably (Liu et al., 2024b), leverage large language models (LLMs) via **prompting strategies** (e.g., in-context learning) rather than task-specific fine-tuning. Tables are serialized into text sequences and paired with demonstrations (e.g., "Q: [question] A: [answer]"), enabling LLMs like GPT-3.5/4 to perform reasoning across diverse schemas. Although LLMs show strong generalization evidenced by DataBench's multi-domain evaluation they face challenges with large tables, hallucination, and precise numerical operations.

3 Method

Our system is designed to answer natural language questions based solely on the information contained in tabular datasets. The overall pipeline consists of four main components: data preprocessing, context provisioning, natural language-to-code conversion, and error-aware execution. In the following, we detail each step of our approach.

3.1 Data Preprocessing

During our experiments with the development set, we observed that certain datasets contained special cases that led to incorrect answers. Addressing these issues not only resolved errors but also improved model performance. Our pre-processing steps are described as below:

- **Handling Missing Values:** Empty lists (`[]` in the dataset) are treated as missing values and replaced with NaN to ensure consistency in data representation.
- **String Normalization:** Extraneous whitespace is removed by stripping leading and trailing spaces from string values.
- **Datetime Standardization:** Columns containing date values are converted to the `datetime` format to ensure uniformity across datasets.

When evaluated on the test set, we identified specific datasets that frequently exhibited errors. We applied additional preprocessing steps to mitigate these issues:

- **067_TripAdvisor:** Extracted individual rating components (e.g., `'service': 5.0, 'cleanliness': 5.0, 'overall': 5.0`) and author details (e.g., `'username': 'Pressgang', 'num_cities': 8, 'num_helpful_votes': 7, 'num_reviews': 9`) into separate columns.
- **074_Lift:** Generated a `Gender` column by classifying lifter names using an LLM-based approach.
- **079_Coffee:** Fixed currency formatting by converting values such as `6,00 US$` to `6.00`.

3.2 Context Provisioning

To enable accurate code generation, our system first extracts and compiles relevant context from the provided dataset. During testing with the development set, we observed that the context provided to the language model (LLM) significantly impacts its performance. For example, when the baseline method only provided `df.head()`, we observed errors with incorrect column names and data types. This suggests that providing only a few sample rows is insufficient for the LLM to understand the dataset's structure and data types. Based on these insights, we tested several alternative methods of providing context. The following components were included to improve the LLM's understanding of the data:

- **Dataset Preview:** A snapshot of the dataset is created using `df.head()`, which provides a representative sample of rows.
- **Column Data Types:** The data types of each column are retrieved via `df.dtypes`, and the full list of column names is extracted using `df.columns`.
- **Enhanced Column Metadata:** In some experiments, we also provide a dictionary that associates each column name with the type of data in that column. The data types are determined by applying the `type()` function to the first entry in each column.

This comprehensive context is embedded in the prompt template, ensuring that the language model has sufficient information to generate accurate Pandas expressions tailored to the structure of the table.

3.3 Natural Language-to-Code Conversion

The core of our approach employs a PandasQueryEngine that translates a given natural language question into an executable Python expression. The prompt provided to the model contains:

- The dataset context (as described above).
- Explicit instructions on generating a Pandas code snippet that, when executed, produces the answer.
- A directive to output only the final expression, minimizing extraneous text.

This process is performed using multiple Large Language Models (LLMs), we also configured with distinct prompt variants to explore the impact of prompt design on code accuracy.

3.4 Error-Aware Execution and Rerun Mechanism

Recognizing that LLM-generated code may occasionally result in errors, our system integrates an error-detection module that monitors the execution of the generated Pandas code. Upon encountering an error, the system automatically regenerate and execute the new Python code.

3.5 Algorithm

In this section, we present the algorithm for Question-Answering over Tabular Data. The algorithm takes as input a language question and a dataset identifier, then retrieves the corresponding dataframe to generate and execute code for answering the question. The overall workflow involves LLM-based code generation, execution, error handling, and final answer formulation.

Algorithm 1 describes the complete workflow for question-answering over tabular data. Figure 1 illustrates the complete flow of our approach.

Algorithm 1 LLM-Driven Table Question Answering

Input: `question`, `dataset_ID`
Retrieve dataframe `df` using `dataset_ID`.
1. Code Generation: Prompt the LLM with the question and `df` context to generate Python code.
2. Execution and Validation: Execute the code on `df`.
if successful **then**
    Return the result; proceed to Step 4.
else
    Proceed to Step 3.
end if
3. Error Correction:
for up to `k` retries **do**
    Provide code and error to the LLM for correction.
    Re-execute the updated code.
    if successful **then**
        Return the result; proceed to Step 4.
    end if
end for
4. Answer Generation: Use LLM to produce the final answer under competition constraints.

4 Experimental Setup

In this section, we describe the experimental setup, including the models, datasets, and evaluation methodology.

4.1 Models and Prompts

We experiment with different Large Language Models (LLMs):

- **GPT-4o-mini**: A compact and efficient variant of GPT-4o (Achiam et al., 2023) that balances high-quality reasoning and dialogue with reduced computational cost, making it accessible and affordable.
- **DeepSeek-V3** (Liu et al., 2024a): A model that reportedly outperforms other open-source models and achieves performance comparable to leading closed-source models.
- **DeepSeek-R1** (Guo et al., 2025): A reasoning-focused model that matches OpenAI-o1 in tackling complex mathematical, coding, and logical tasks, while offering its capabilities via API at a remarkably low cost.
- **deepseek-r1-distill-qwen-14b**: An open-source model distilled from DeepSeek-R1 based on Qwen

GPT-4o-mini is tested with three different prompts:

- **Prompt A**: A baseline prompt from LlamaIndex with some adjustments to match the competition output.
- **Prompt B**: An improved prompt with some data information addition.

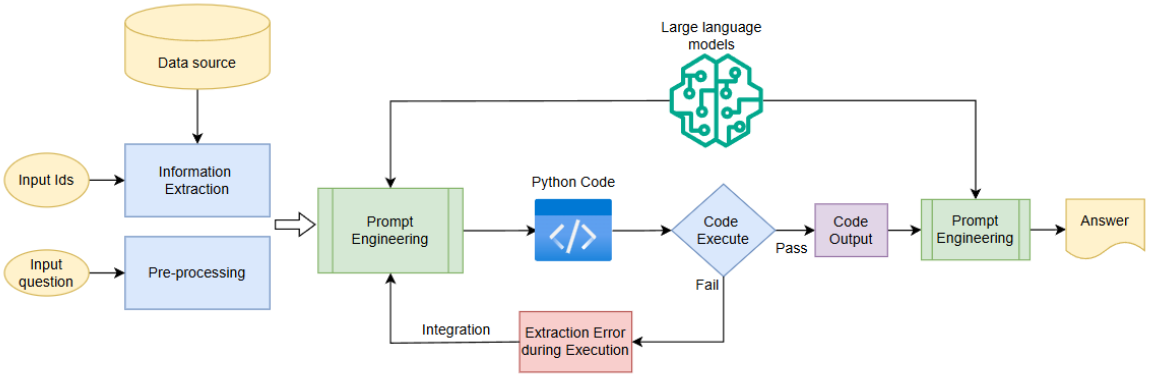

Figure 1: Overview of the Methodology for Question Answering over Tabular Data via Large Language Models

- **Prompt C**: An improved prompt incorporating additional information, including column names and data types, to enhance accuracy.

Each prompt was designed to assess how prompt engineering affects the correctness and consistency of the generated answers. The main differences between the prompts are illustrated in Figure 2.

4.2 Dataset

We conduct experiments on two versions of the dataset:

- **Original Data**: The raw dataset without modifications.
- **Preprocessed Data**: A cleaned version in which we address inconsistencies such as converting missing values, standardizing data formats, and generating additional features.

Further details on the preprocessing steps can be found in Section 3.1.

4.3 Evaluation Metrics and Procedure

We evaluate models based on accuracy, comparing predicted answers to the ground truth.

5 Results and Discussion

In this study, we evaluated four models: *GPT-4o-mini*, *DeepSeek-V3*, *DeepSeek-R1*, and *deepseek-r1-distill-qwen-14b*. Initially, *GPT-4o-mini* was tested with three distinct prompts (A, B, and C) using the original dataset. Based on these preliminary results, we identified the best-performing prompt (Prompt C) and subsequently used it to compare model performance on preprocessed datasets.

5.1 Results Overview

The results summarized in Table 3 present the accuracy of each model under different prompt conditions using the original dataset. As shown, **Prompt C consistently achieved the highest accuracy** across all models, making it the most effective prompt for further experimentation. Based on this finding, we conducted additional evaluations to analyze the impact of data preprocessing while using Prompt C. Table 4 shows the total errors in different question types of the best model which is **DeepSeek-R1**.

To further compare our approach with the top-performing teams, we report rankings separately for Subtask 1 and Subtask 2. Table 1 lists the **top 5 teams in Subtask 1**, where our method ranked **4th**.

For Subtask 2, we used **DeepSeek-V3** instead of **DeepSeek-R1** because the API for DeepSeek-R1 was unavailable at that time. Table 2 presents the **top 5 teams in Subtask 2**, in which our approach ranked **9th**.

Table 1: Top 5 Teams in Subtask 1

Rank	Accuracy (%)
Top 1	95.01
Top 2	89.85
Top 3	89.66
Top 5	88.12
Top 6	87.16
Top 4 (Our Team)	**89.46**

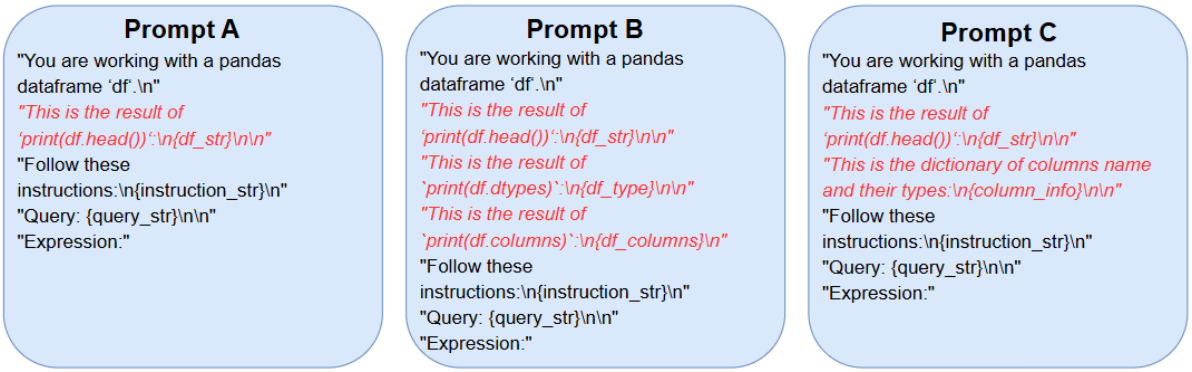

Figure 2: The key differences in how data and information are provided.

Table 2: Top 5 Teams in Subtask 2

Rank	Accuracy (%)
Top 1	92.91
Top 2	88.89
Top 3	88.70
Top 4	86.59
Top 5	86.22
Top 9 (Our Team)	**85.25**

5.2 Discussion

The results presented in Tables 3 highlight several important findings:

- **Prompt Selection:** The comparative analysis across Prompts A, B, and C indicated that Prompt C produced the best overall performance across all models when using the original dataset. This finding underscores the importance of prompt engineering in obtaining correct answers.
- **Impact of Data Preprocessing:** The subsequent comparison using Prompt C demonstrated that preprocessing the data substantially improved model performance. For instance, `gpt4o-mini` model showed increased in accuracy when the preprocessed dataset was used, thereby validating the benefits of a robust data cleaning pipeline.
- **Model Comparison:** After comparing various models, it is clear that the `DeepSeek-R1` outperforms all other models in terms of accuracy. The reasoning capabilities of `DeepSeek-R1` play a significant role in its superior results, allowing it to process and understand complex data in ways that other models cannot.
- **Practical Implications:** The combined analysis of prompt selection and data preprocessing highlights a clear pathway for optimizing model performance. Experiment with multiple prompt designs and invest in thorough data preprocessing to maximize the effectiveness of large language models.

6 Conclusion

In this paper, we explored the challenges and errors encountered when using a large language model to generate executable code for tabular question-answering tasks. Our analysis categorized errors into three key types: numerical precision errors, logical errors, and insufficient information. By systematically evaluating the generated responses and their correctness, we identified the primary causes of errors, including incorrect filtering, precision issues, and reasoning failures.

Our findings highlight the importance of robust error-handling mechanisms when integrating LLMs with structured data processing. Future work should focus on improving the consistency of generated codes, enhancing model understanding of numerical reasoning, and utilizing multiple LLMs to generate and validate answers, selecting the most reliable response based on consensus or predefined criteria.

Through this analysis, we provide valuable insights for researchers and practitioners working on LLM-based data processing systems. Our work underscores the need for a hybrid approach that combines LLM reasoning and prompt engineering

Table 3: Comparison of model performance under different experimental conditions (Subtask 1).

Note: Due to time constraints, experiments for all three DeepSeek models across all prompt engineering configurations were not completed.

Model	**Original Data**			**Preprocessed Data (Prompt C)**
	Prompt A	Prompt B	Prompt C	Prompt C
`GPT-4o-mini`	76.25%	79.31%	81.61%	82.76%
`DeepSeek-V3`	–	–	–	84.67%
`deepseek-r1-distill-qwen-14b`	–	–	–	71.84%
`DeepSeek-R1`	–	–	–	89.46%

Table 4: Error distribution across different question types.

Question Type	**Error Count**
List (category)	17
List (number)	13
Number-type	11
Boolean	7
Category	7

with structured query generation and validation to enhance reliability in real-world applications.

Acknowledgements

This research was supported by The VNUHCM-University of Information Technology's Scientific Research Support Fund.

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*.

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

T. Herzig, M. Weissenborn, S. Ruder, D. Bahdanau, and A. W. Black. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 8253–8263. Association for Computational Linguistics.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Tianyang Liu, Fei Wang, and Muhao Chen. 2024b. Rethinking tabular data understanding with large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 450–482, Mexico City, Mexico. Association for Computational Linguistics.

Jorge Osés-Grijalba, Luis Alfonso Ureña-López, Eugenio Martínez Cámara, and Jose Camacho-Collados. 2025. SemEval-2025 task 8: Question answering over tabular data. In *Proceedings of the 19th International Workshop on Semantic Evaluation (SemEval-2025)*, Vienna, Austria. Association for Computational Linguistics.

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 *EMNLP*.

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