TableWise at SemEval-2025 Task 8: LLM Agents for TabQA

Harsh Bansal [†]
IIIT Hyderabad
harsh.bansal@students.iiit.ac.in

Aman Raj [†]
IIIT Hyderabad
aman.r@students.iiit.ac.in

Akshit Sharma [†]
IIIT Hyderabad
akshit.sharma@students.iiit.ac.in

Parameswari Krishnamurthy
IIIT Hyderabad
param.krishna@iiit.ac.in

Abstract

In this study, we present our approach to SemEval Task 8: Question Answering over Tabular Data (Os'es Grijalba et al., 2025), where we develop a Large Language Models(LLM)-based agent capable of answering questions over tabular data. Our agent leverages a set of custom defined tools incorporating structured table parsing and reasoning mechanisms to enhance semantic understanding of tabular data. Extending our methodology, we apply chain-of-thought prompting to further refine it's understanding of the task. Our findings suggest that LLM-based agents, when properly adapted, can significantly improve their table-based question answering capabilities.

1 Introduction

The task of Question Answering on Tabular Data (TabQA) involves answering natural language queries using structured tabular data. Given a natural language query, the goal is to generate accurate responses based on the tabular data.

Conventional TabQA methods typically involve providing large language models (LLMs) with the entire table alongside the query, under the assumption that full-table context enables models to reason over any potentially relevant information. However, this strategy faces significant challenges, including limited scalability due to context length constraints, poor generalization and high computational overhead.

Retrieval-Augmented Generation (RAG) methods have partially tried to address these issues by retrieving semantically similar chunks of the table. However, they rely on embeddings that often fail to capture relational structures and numerical precision, resulting in poor retrieval performance and difficulty in handling diverse column types.

To overcome these limitations, we propose an LLM-agent-based framework that abstracts tables into SQL representations, enabling LLMs to leverage structured information more effectively. This approach combines structured query execution with natural language understanding, offering a scalable and efficient solution for TabQA.

[†]The authors contributed equally to this work.

2 Related Work

Early approaches for TabQA primarily relied on semantic parsing, translating queries into executable programs such as SQL, as seen in models like Neural Programmer (Neelakantan et al., 2017) and SQLNet (Xu et al., 2017). While effective, these approaches required annotated SQL queries and struggled to generalize across diverse table schemas.

Later on, models such as TAPAS (Herzig et al., 2020) and TaBERT (Yin and Neubig, 2020) used weakly supervised learning over table-text pairs, removing the dependence on explicit SQL annotations. While these models advanced the state of the art, they exhibited limitations in numerical reasoning capabilities and scalability to long or complex tables.

To further address these challenges, Retrieval-Augmented Generation (RAG) techniques, exemplified by models like RASAT (Kim et al., 2022), incorporated retrieval mechanisms to identify relevant table segments before answer generation. However, despite improved scalability, these methods often failed to capture the structured and relational aspects inherent to tables.

Recent developments in agentic LLMs, such as Toolformer (Schick et al., 2023) and ReAct (Yao et al., 2022), introduced frameworks where language models interact with external tools (e.g., SQL engines, calculators) to perform more accurate and interpretable reasoning.

Building upon these advancements, we propose a novel framework that integrates agentic reasoning with SQL-based execution. By abstracting tables into SQL databases and enabling LLMs to interact

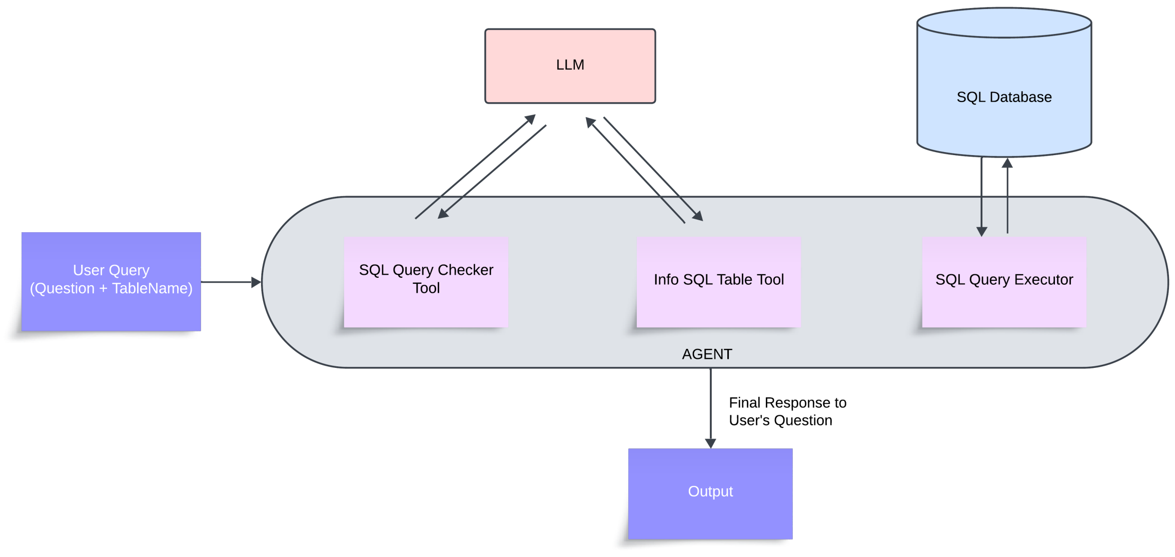

Figure 1: Proposed Agentic Framework for TabQA

with them, our approach enhances generalization across domains, and strengthens numerical reasoning capabilities in TabQA tasks.

3 Dataset

Each table is provided in two formats: `all.parquet`, containing the complete set of rows, and `sample.parquet`, containing the first 20 rows of the corresponding `all.parquet` file. The collection of all `all.parquet` files across every table constitutes the Databench dataset, while the set of all `sample.parquet` files forms the Databench-lite dataset.

Additionally, for each table, a set of natural language queries has been provided, which can be answered using the table's information.

4 System Overview

We introduce a novel pipeline for Question Answering over Tables that allows LLMs to interact with a set of specialized tools to deliver precise, reliable responses. The end-to-end architecture is depicted in Figure 1 and comprises two primary stages:

4.1 Data Ingestion

All input tables, originally stored in `.parquet` format, are imported into an SQL database. This conversion enables efficient, schema-driven querying using SQL.

4.2 Query Processing

The core of our methodology is a framework that allows an LLM to interact with multiple tools for query interpretation, SQL generation, validation, and execution. The description of the tools is as follows:

- **SQL Query Checker:** Ensures generated SQL statements are syntactically correct, safe, and restricted to read-only operations, thereby preventing unintended modifications.
- **Info SQL Table:** Retrieves table metadata (e.g., column names, data types) and sample rows to inform accurate and context-aware SQL formulation.
- **SQL Query Executor:** Executes validated SQL statements against the database and returns the results.

Throughout execution, the agent systematically records intermediate information—such as table schemas, draft queries, and execution outputs—which are then used by the LLM to produce the final, formatted answer.

5 Experimental Setup

To evaluate the system's performance, we integrate multiple open-source LLMs into our framework. Experiments are conducted using standardized evaluation scripts from external evaluators. Table 1

LLM Provider	No. of Parameters	Model
Meta	70B	Llama-3.3-70B
Codestral	22B	Codestral-22B-v0.1
Mistral	7B	Mistral-7B-v0.3
Meta	3B	Llama-3.2-3B

Table 1: Open-source LLMs used for Experiments

Model	Databench-lite	Databench
Llama-3.3-70B	67.43	62.07
Codestral-22B-v0.1	60.21	56.73
Mistral-7B-v0.3	49.42	44.29
Llama-3.2-3B	39.65	36.17

Table 2: LLM Performance Metrics

summarizes the LLM variants tested, indicating their provider, parameter count, and model used. Each model interacts with the defined set of tools to answer the natural language queries.

6 Results and Analysis

We integrated four open-source LLMs—Llama-3.3-70B, Codestral-22B-v0.1, Mistral-7B-v0.3, and Llama-3.2-3B—into our agentic framework and evaluated each on both the Databench and Databench-lite datasets.

6.1 Effect of Dataset Scale

All models achieved higher accuracy on Databench-lite than on Databench. The reduced row count in Databench-lite simplifies multi-step SQL queries, decreases execution times, and lowers the risk of exceeding tool-call limits or entering infinite reasoning loops. In contrast, Databench's larger search space for intermediate operations—such as retrieving unique values before aggregation—leads to longer query pipelines and a higher error rate in query formulation and execution.

6.2 Comparative Model Performance

Llama-3.3-70B achieved the highest accuracy (62.07% on Databench), followed by Codestral-22B-v0.1, Mistral-7B-v0.3, and finally Llama-3.2-3B. The larger parameter count of Llama-3.3-70B enables more precise SQL generation and robust reasoning over complex queries. Codestral-22B-v0.1 and Mistral-7B-v0.3 performed competitively but showed occasional failures on nested or multi-step queries. Llama-3.2-3B's lower accuracy indicates that smaller models struggle with the reasoning depth required for intricate tabular QA tasks.

6.3 Error Analysis

6.3.1 Multi-Step Query Failures

In multi-step queries, the agent frequently misaligns natural language predicates with corresponding schema values, resulting in omitted filtering clauses or syntactic errors. An example from the `066_IBM_HR` table illustrates this behavior:

> "Are there more employees who travel frequently than those in the HR department?"

The predicate "travel frequently" should translate to:

```
WHERE BusinessTravel = 'Travel_Frequently'
```

However, the agent either omits the WHERE clause entirely or generates invalid SQL, such as:

```
SELECT COUNT("travel frequently")
FROM 066_IBM_HR;
```

This misalignment prevents correct filtering and yields an incorrect answer.

6.3.2 Infinite Tool-Call Loops

A Small fraction of the queries processed by Llama-3.3-70B entered infinite refinement loops due to overly strict or misspelled filters (e.g., `EmployeID`). Smaller models like Llama-3.2-3B were found to be more susceptible to infinite refinement loops highlighting the trade-off between model capacity and reasoning capabilities.

6.4 Summary

Our results indicate that dataset scale and model size are critical determinants of TabQA performance. Databench-lite facilitates efficient and accurate querying, while larger LLMs produce superior SQL formulation and complex reasoning. Future work will explore methods to reduce query complexity and improve the resilience of smaller models.

7 Conclusion

Our research highlights the advantages of using an agentic approach for QA on tabular data. It demonstrates its ability to dynamically construct and refine queries for precise information retrieval, which is essential for QA performance. By outperforming RAG-based and full-table context methods in handling mixed data types and scalability challenges, it proves more adaptable and efficient. Additionally, its iterative reasoning surpasses direct SQL querying and reinforces the potential of LLM-powered agents in table-aware AI systems.

8 Limitations

Despite our progress, this work has several limitations. The multi-step reasoning process necessitates multiple tool calls, which increases response latency—especially when the agent engages in excessive query refinement—and can occasionally lead to infinite reasoning loops that exhaust the tool-call budget and prevent valid answers. The agent also sometimes deviates from the expected answer format, undermining consistency. To address these issues, we plan to incorporate termination conditions that detect repetitive tool calls and enforce early exits, enforce stricter output validation to guarantee format adherence, and optimize query execution to reduce latency. Finally, our reliance on off-the-shelf open-source LLMs may limit performance; fine-tuning these models on domain-specific training data could further improve accuracy and robustness.

References

Jonathan Herzig, Pawel Nowak, Thomas Muller, Francesco Piccinno, and Julian Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pretraining. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4320–4333. Association for Computational Linguistics.

Jinhyuk Kim, Wonjin Park, and Jaewoo Lee. 2022. Rasat: Integrating relational structures into pretrained language models for table-based question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics.

Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. 2017. Neural programmer: Inducing latent programs with gradient descent. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. International Conference on Learning Representations.

Jorge Os'es Grijalba, Luis Alfonso Ure na-L'opez, Eugenio Mart'inez C'amara, 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.

Timo Schick, Arun Tejasvi Dwivedi-Yu, Jaap Jumelet, Nafise Sadat Moosavi, and Iryna Gurevych. 2023. Toolformer: Language models can teach themselves to use tools. In *arXiv preprint arXiv:2302.04761*.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 681–691. Association for Computational Linguistics.

Shinn Yao, Jiong Zhao, Dian Yu, Kaixin Yang, Maarten Bosma, and Denny Zhou. 2022. React: Synergizing reasoning and acting in language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Pengcheng Yin and Graham Neubig. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 8413–8426. Association for Computational Linguistics.