Team Core Intelligence at SemEval-2025 Task 8: Multi-hop LLM Agent for Tabular Question Answering

Maryna Chernyshevich
Accuris
Marina.Chernyshevich@accuristech.com

Abstract

Tabular Question Answering (Tabular QA) is a challenging task requiring models to extract, interpret, and reason over structured data. While Large Language Models (LLMs) have demonstrated strong performance in natural language tasks, their ability to process and query tabular data remains inconsistent, particularly in multi-column reasoning and structured output generation. To address these challenges, we propose a multi-hop LLM agent that enhances Tabular QA by analyzing table structure, building step-by-step plan, generating code to extract relevant data, and verifying outputs. Our approach combines proprietary LLM (ChatGPT-3.5-turbo) for code generation and open source LLM (Llama-3.2-3B) for answer validation. We evaluate our method on the SemEval-2025 Task 8 and were ranked 6-th with 87.16% accuracy.

1 Introduction

Retrieval-Augmented Generation (RAG) is an efficient technique for incorporating enterprise knowledge into Large Language Models (LLMs), improving factual accuracy and reducing hallucinations – a persistent problem, when LLMs operate in unfamiliar domains. The RAG system retrieves relevant information from external knowledge bases and provides it as context for LLM for reasoning and answer generation. This approach has been widely adopted for handling unstructured textual data, where relevant documents are stored in vector databases and traditional search engines.

However, a significant portion of public and proprietary knowledge is stored in structured formats, such as Excel spreadsheets or databases. Semantic chunking, a well-established method for retrieval of unstructured information, is not suitable for tables, as structured data is inherently relational, meaning that cells, rows, and columns must be interpreted together to get meaningful insights. RAG systems today primarily focus on text-based retrieval and struggle with structured data integration due to several limitations:

1. Database tables can be too large to process directly. Real-world tables often contain millions of rows and hundreds of columns, which makes passing the whole table into LLM costly and inefficient. A traditional chunking approach (e.g., breaking text into meaningful text chunks of fixed size) does not work well for structured data, as different questions might require different subsets of rows and columns rather than a table fragment.
2. Tabular question answering often requires computation. Unlike text-based question answering, which involves retrieval of relevant information, tabular QA requires sorting, filtering, grouping, aggregations, and mathematical operations. A system working with structured data must be capable of query execution rather than just retrieval.
3. Data inconsistencies and schema mismatches complicate the answer extraction. Data stored in tables or databases often follow inconsistent schemas, contain abbreviations, missing values, diverse types of data such as numbers, strings, categories, timestamps, etc. A robust Tabular QA system must handle these inconsistencies automatically and ensure semantic consistency between queries and table structure.

To overcome these challenges, we propose a multi-step LLM agent designed to serve as a structured data reasoning component of an advanced multi-agent RAG system. Our approach combines four main stages: 1) table exploration to extract data schema and metadata; 2) execution planning to decompose complex question into simpler steps; 3) code generation and execution to extract, transform and summarize tabular data; 4) answer validation to verify the correctness and consistency of the answer.

Our multi-step agent framework enables efficient reasoning over large, structured knowledge sources, while minimizing hallucinations, code execution errors, and ensuring accurate answer generation. The solution was evaluated in the SemEval-2025 Task 8 Question Answering over Tabular Data (Grijalba et al., 2025), where it was ranked 6th with 87.16% accuracy on the test dataset, demonstrating the effectiveness of our structured retrieval and reasoning approach.

2 Task and Dataset Description

Tabular Question Answering (Tabular QA) task involves answering natural language questions based on structured tabular data. Given an input query Q and a table T with N rows and M named columns, the goal is to produce an accurate answer A.

The SemEval-2025 Task 8 organizers offer the DataBench dataset, a large benchmark for the task of question answering on structured or tabular data (Grijalba et al., 2024). This dataset consists of 1300 questions and each question is accompanied by a related tabular dataset. Overall, 65 datasets are presented from 5 different domains, such as business, health, social, sports and travel. The average number of rows in table is 50,300, while maximum is 713,107 rows per table. Average number of columns is 25 per table, and maximum is 123.

The questions are split into different categories depending on the type of expected answer:

Question Type	Example Question	Reasoning Type
Boolean (Yes/No)	*Are all transactions IDs unique?*	Lookup
Category Selection	*Which organization has the patent with the highest number of claims?*	Sorting & comparison
Numerical Value	*How many borrowers have more than 1 existing loan?*	Aggregation
List Output	*Which 5 states have the most number of job listings?*	Filtering & grouping

3 Related Work

Early approaches of Tabular QA relied on semantic parsing, where natural language questions were transformed into SQL-like commands (Zhong et al., 2017; Yu et al., 2018). These methods require extensive training on domain-specific schemas and struggle with generalization across unseen tables.

Recent advances in Large Language Models (LLMs) have enabled zero-shot Tabular QA by leveraging in-context learning (Brown et al., 2020), supervised LLM finetuning (Zha et al., 2023) and code generation (Cao et al., 2023). Models such as GPT-4 and Llama, and CodeLLama can dynamically generate SQL queries or code instructions to retrieve answers from structured data. However, these methods suffer from hallucinations, logical inconsistencies, and execution failures.

To address these limitations, researchers have explored multi-agent systems, where different LLMs specialize in code generation, execution verification, and logical consistency checks (Zhu et al., 2024, Zhou et al, 2024). Our work builds upon this trend and introduces a structured LLM agent, that dynamically plans, executes, and validates tabular queries, ensuring high reliability and accuracy.

4 System overview

We introduce a multi-hop LLM agentic system, that performs tabular QA by decomposing reasoning and execution into separate stages. In contrast to single-pass prompting approaches, which often struggle with multi-column queries, schema inconsistencies, and logical errors, our system follows an iterative process that includes table exploration, execution planning, code generation, and answer validation. The architecture of the agent relies on LangGraph[1], a framework that organizes nodes execution as a directed acyclic graph (DAG), which ensures modularity, reusability, and observability.

Our system orchestrates multiple LLM-based agents, each having a distinct role and underlying implementation, including ChatGPT-3.5-turbo for Python code generation and Llama-3.2-3B-Instruct (fine-tuned) for answer validation and consistency checks. This multi-model collaboration improves both accuracy and efficiency and allows specialized models to handle tasks suited to their strengths. If answer consistency check fails, the system revisits previous reasoning steps, refines queries and execution plans dynamically.

4.1 Table exploration

The system begins processing questions and tables with analyzing the table structure and extracting metadata to establish an accurate understanding of the dataset before attempting to answer a query.
At this stage, the agent executes automatically generated Python code and identifies column names, data types, missing values, and summary statistics and other information.

This step ensures the agent has an accurate understanding of the dataset. For example, it can detect that a "height" column contains values in inches, while the question asks for values in centimeters. It also can recognize that "IBM" in the query refers to "International Business Machines Corp." in the table, preventing incorrect lookups.

The discovered information is stored in memory and passed to the subsequent stages. If the initial metadata extraction is incomplete or inaccurate, the system refines its analysis and re-executes exploration queries before proceeding (Fig 1).

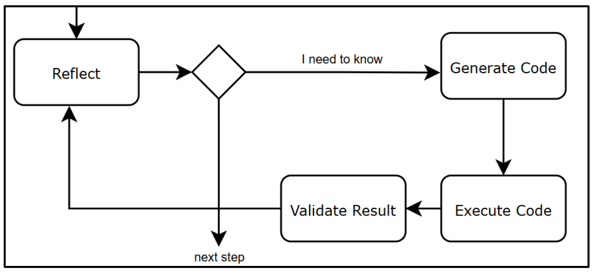


Figure 1: Table Exploration step flow

4.2 Execution planning

Once the table has been analyzed, the agent moves to execution planning, where the question is decomposed into structured subqueries described in natural language and a step-by-step plan is built. This prevents column misinterpretation and ensures proper sequencing of operations such as direct lookup, filtering, aggregation, and mathematical operation.

For example, given the question:

"What are the word counts of the 3 longest posts?"

The agent generates the following plan:

```
1. Create a new column 'word_count' in the DataFrame by splitting the 'text' column by spaces and counting the number of resulting words for each post.
2. Sort the DataFrame by the length of the 'text' column in descending order to find the longest posts.
3. Select the top 3 rows from the sorted DataFrame, which correspond to the 3 longest posts.
4. Extract the 'word_count' values from these top 3 rows to get the word counts of the 3 longest posts.
```

In another case, when asked:

"Which salary level has the least number of employees who had an accident at work?"

The agent produces:

```
1. Filter the table to keep only rows where 'Work Accident' is 'Yes'.
2. Group this filtered table by the 'salary' column and count how many employees in each salary level had an accident.,
3. Identify the salary level(s) with the minimum count from this grouped result.
```

By explicitly identifying the retrieval strategy, the system avoids common errors such as extracting data from irrelevant columns or applying incorrect operations.

4.3 Code generation and execution

Based on the structured execution plan, the agent generates Python (pandas) code to extract the answer from the dataset. The code is executed in a controlled environment to prevent uncontrolled command execution. If an error occurs, such as a KeyError due to an incorrect column name or a TypeError caused by a mismatched data type, the system analyzes the failure and acts accordingly. To mitigate the consequences, each code execution

[1] https://www.langchain.com/langgraph

call is accompanied by a result validation step (Fig. 2), that takes one of the following:

1. If execution succeeds, the system forwards the answer to the validation step.
2. If a recoverable error occurs, it regenerates the Python code with corrections.
3. If the error persists, it replans the execution strategy, modifying the approach before generating new code.

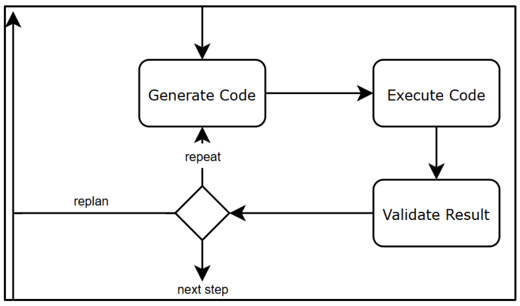
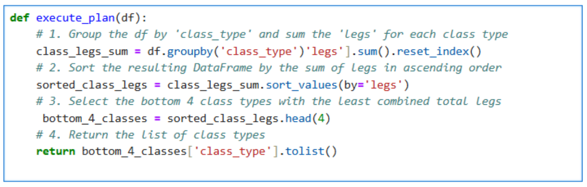


Figure 2: Code Generation & Execution diagram

This iterative code execution reduces overall failure rates and ensures robust handling of schema inconsistencies and dynamic table structures. Figure 3 demonstrates the code generated according to a plan to answer the question: “What are the bottom 4 class types with the least combined total legs?”



```
def execute_plan(df):
    # 1. Group the df by 'class_type' and sum the 'legs' for each class type
    class_legs_sum = df.groupby('class_type')'legs'].sum().reset_index()
    # 2. Sort the resulting DataFrame by the sum of legs in ascending order
    sorted_class_legs = class_legs_sum.sort_values(by='legs')
    # 3. Select the bottom 4 class types with the least combined total legs
     bottom_4_classes = sorted_class_legs.head(4)
    # 4. Return the list of class types
    return bottom_4_classes['class_type'].tolist()
```

Figure 3: Generated code

4.4 Validation and refinement

Even when the code is executed without exceptions, the answer may still contain errors, for example due to incorrect column selection or semantic mismatch with the question. To address this, the extracted response is passed through a verification step, where a fine-tuned Llama-3.2-3B model acts as a supervising agent and checks answer’s correctness.

The validation process includes the following checklist:

1. Do the involved table columns match the question?
2. Is the output type (boolean, numeric, categorical, list) consistent with the question?
3. Does the answer logically align with the original question?
4. Does the answer output contain the expected number of elements?

If the validation model detects an issue, it generates a short explanation describing why the answer is potentially incorrect. This feedback is sent back to the execution planning stage for refinement (Fig. 4). This feedback loop enhances accuracy by incorporating self-correction mechanisms.

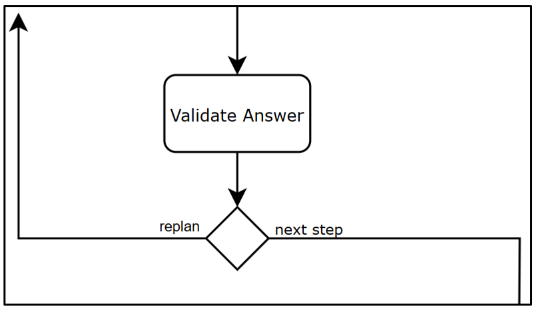


Figure 4: Answer validation diagram

For example, the validation model is generating based on the inputs: “What are the stock codes of the bottom 5 transactions with the lowest quantities ordered? If there is a tie go with numerical order.” (question) and “[556690.0, 556691.0, NaN, NaN]” (answer) following validation message:

```
The answer contains several issues:
1.  'nan' values which are not valid stock
    codes
2.  The answer contains 4 elements, but the
    expected number of elements should be 5.
I suggest to replan the execution.
```

This step catches the incorrect table or question understanding and forces the agent to replan and generate a corrected answer.

To enhance instruction-following and validation capabilities, we leveraged the Low-Rank Adaptation (LoRA), originally introduced in (Hu et al., 2021), to fine-tune the Llama-3.2-3B instruction model on about 1,000 synthetic instructions, generated based on DataBench training question-answer pairs.

This validation step improved the overall quality of the agent, but also introduced a high false positive rate, flagging over 18% of correct answers as incorrect. This led to unnecessary recomputation in some cases but also forced the agent to re-examine its reasoning.

4.5 Conclusion

We evaluated our system on the SemEval-2025 Task 8: Question-Answering over Tabular Data, where it was ranked 6th with 87.16% accuracy, demonstrating its effectiveness compared to other approaches.

Our experiments also proved that the structured agent-based approach significantly outperforms the single-pass LLM prompting approaches. Decomposition of the tabular question answering task into exploration, planning, execution, and validation steps allows handling very complex multi-column tables with millions of records, reducing hallucination and increasing reliability compared to traditional approaches.

While the proposed agent-based solution is showing its efficiency, it also, also highlights areas for further research and improvements, such as evaluating other open models such as Llama-70B or Qwen-72B in zero-shot as well as instruct-tuning settings to execute some of the agent functions. Our findings suggest that distributing reasoning tasks across multiple specialized models is a promising direction, as it allows for more cost-efficient computation while improving answer reliability.

The combination of code-based reasoning, automatic verification, and iterative refinement make a multi-hop LLM agent an effective way to get information from structured data.

References

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *In Advances in Neural Information Processing Systems (NeurIPS).*

Yihan Cao, Shuyi Chen, Ryan Liu, Zhiruo Wang, and Daniel Fried. API-assisted code generation for question answering on varied table structures. In Proc. of EMNLP, 2023.

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

Osés Grijalba, Jorge and Ureña-Lopez, Luis Alfonso and Martinez Camara, Eugenio and Camacho-Collados, Jose. SemEval-2025 Task 8: Question Answering over Tabular Data. In Proceedings of the 19th International Workshop on Semantic Evaluation (SemEval-2025).

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan AllenZhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

Grijalba, J.O., Lopez, L.A.U., Martínez-Cámara, E. and Camacho-Collados, J., 2024, May. Question answering over tabular data with DataBench: A large-scale empirical evaluation of LLMs. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)* (pp. 13471-13488).

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. arXiv preprint arXiv:1809.08887, 2018.

Liangyu Zha, Junlin Zhou, Liyao Li, Rui Wang, Qingyi Huang, Saisai Yang, Jing Yuan, Changbao Su, Xiang Li, Aofeng Su, Tao Zhang, Chen Zhou, Kaizhe Shou, Miao Wang, Wufang Zhu, Guoshan Lu, Chao Ye, Yali Ye, Wentao Ye, Yiming Zhang, Xinglong Deng, Jie Xu, Haobo Wang, Gang Chen, and Junbo Zhao. Tablegpt: Towards unifying tables, nature language and commands into one gpt, 2023.

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

Wei Zhou, Mohsen Mesgar, Annemarie Friedrich, and Heike Adel. "Efficient Multi-Agent Collaboration with Tool Use for Online Planning in Complex Table Question Answering." *arXiv preprint arXiv:2412.20145* (2024).

Jun-Peng Zhu, Peng Cai, Kai Xu, Li Li, Yishen Sun, Shuai Zhou, Haihuang Su, Liu Tang, and Qi Liu. "Autotqa: Towards autonomous tabular question answering through multi-agent large language models." *Proceedings of the VLDB Endowment* 17, no. 12 (2024): 3920-3933.