SBU-NLP at SemEval-2025 Task 8: Self-Correction and Collaboration in LLMs for Tabular Question Answering

Rashin Rahnamoun and Mehrnoush Shamsfard
Shahid Beheshti University, Tehran, Iran
rahnamounrashin@gmail.com and m-shams@sbu.ac.ir

Abstract

This paper explains the submission of the SBU-NLP team at SemEval-2025 Task 8: question-answering over tabular data. We present a novel algorithm for this task, aimed at systems capable of interpreting large tables and providing accurate answers to natural language queries. The evaluation uses the DataBench dataset, which covers a wide range of topics and reflects the complexity of real-world tabular data. Our approach incorporates a self-correction mechanism that iteratively refines LLM-generated code to address errors and prevent common mistakes. Additionally, a multi-LLM collaborative strategy is employed to generate answers, where responses from multiple LLMs are compared, and the majority consensus or a valid alternative is selected. The method relies exclusively on open-source, open-weight models, avoiding costly processes like training or fine-tuning. Experimental results demonstrate that combining multiple LLMs with self-correction leads to significant performance improvements. However, challenges arise with list-based answers and responses involving multiple numerical, string, or boolean values, where further refinement is needed. The proposed simple system was among the top performers in both Subtask A and Subtask B among open-source, open-weight models in the competition.

1 Introduction

Task 8(Osés-Grijalba et al., 2025) focuses on question-answering over tabular data. The goal is to evaluate systems that can effectively interpret large tables and provide accurate answers to natural language questions based on the data within those tables. The evaluating dataset, DataBench(Grijalba et al., 2024), is important because many real-world applications rely on tabular data, and everyday datasets can vary significantly in both the subjects they cover and their size. DataBench covers a wide range of topics, making it essential to evaluate systems on diverse data to ensure they can handle a variety of real-world scenarios.

Our system, which we used for this task, employs a novel, simple algorithm based on a self-correction mechanism. In this approach, the large language model (LLM)-generated code from the prompt is iteratively fed back into the LLM to fix errors and avoid common code mistakes. Additionally, we use a multi-LLM collaborative answer generation strategy, where answers from multiple LLMs are compared, and the majority consensus is used. In cases where errors due to code issues prevent the LLM from generating an appropriate answer, alternative answers from other LLMs are utilized. Furthermore, if the generated answers are not in a valid format, a valid answer from another LLM for the same question is adopted. This method uses models that are either open-source or open-weight, avoiding computationally costly procedures like training and fine-tuning.

Participating in this task has led to several key discoveries. The results show that leveraging multiple LLMs and employing a self-correction mechanism significantly improved performance. Additionally, a well-structured Python code generation prompt played a crucial role in obtaining better answers from tables. However, we also discovered that many of the errors in our system were related to cases where the answers were in list format or involved more than one numerical, string, or boolean value. If these specific errors could be better handled, the system's results could be further improved. In terms of ranking, our model performed well among open-source and open-weight models, securing 3rd place out of 37 teams in Subtask B and 6th place out of 37 teams in Subtask A.You can find the code for our system in the fol-

Proceedings of the The 19th International Workshop on Semantic Evaluation (SemEval-2025), pages 703–711

lowing GitHub repository[1].

2 Background

Question answering on tabular data is a critical task in natural language processing. Over the years, researchers have developed diverse methodologies to improve the accuracy and efficiency of this process.

The task is about question answering from tabular data. For this goal, a newly introduced benchmark, DataBench(Grijalba et al., 2024), has been used for evaluation, which consists of different large tables that vary in the topics they cover. As input, like the example in Figure 1, a natural language-based question with a table name is given, and the expectation is to find an appropriate answer from the related tables. For the final competition tests, 15 tables were provided. In Subtask A, the tables can be of any size, while in Subtask B, the row count is fewer than 20 due to the context-length limitations for LLMs. We participated in both subtasks with the same system.

2.1 Training and Fine-tuning the Models

To address the challenges of question answering on tabular data, several models have been developed, each leveraging training and fine-tuning strategies. Below, we highlight some of the most notable approaches:

TAPEX: Built on the BART framework, TAPEX(Liu et al.) is tailored for structured tables. It employs a neural SQL executor, pretrained on a synthetic corpus, to interpret and execute queries effectively.

TaPas: TaPas(Herzig et al., 2020) takes a different approach by directly predicting answers through selecting relevant table cells and applying aggregation operations, bypassing the need for intermediate logical forms. Extending BERTs architecture, it encodes tabular structures and is trained end-to-end for seamless performance.

OmniTab: OmniTab(Jiang et al., 2022) enhances table-based question answering by combining natural and synthetic data during pretraining. It aligns questions with corresponding tables.

2.2 Prompting and In-Context Learning

Similar to the approaches used in this task's paper(Grijalba et al., 2024), two key methodologies have been introduced for evaluating the performance of LLMs on the proposed benchmark:

- **In-Context Learning (ICL):** In this approach, examples with corresponding answers are provided to the model within the prompt, enabling it to infer patterns and generate responses accordingly.

- **Code Generation Prompting:** In this method, the model generates code based on the given prompt, executes it, and derives the final answer from the table.

Furthermore, another paper introduces the **Seek-and-Solve pipeline** (Jiang et al., 2024) , a novel framework for enhancing table-based question answering. This approach instructs LLMs to first seek relevant information before solving the given question, integrating these reasoning steps into a structured **Seek-and-Solve Chain of Thought (SS-CoT)** to improve performance.

Additionally, two other papers propose specific methodologies tailored to particular applications that are worth mentioning. One approach employs a relation graph as an encoder and a tree-based decoder to tackle numerical reasoning questions(Lei et al., 2022). Another approach utilizes the **Multi-TabQA** model, (Pal et al., 2023)which is designed to answer questions based on information from multiple tables and is capable of generalizing to generate tabular answers.

3 System Overview

In the question-answering task on tabular data, several challenges may arise. Due to limitations in hardware infrastructure, we employed a prompt-based approach 2.2 to achieve results while avoiding strategies that rely on fine-tuning or training procedures 2.1.

In our experiments, we exclusively used open-source and open-weight models and avoided third-party and commercial models due to their high costs. To generate code, we utilized prompts in LLMs. However, because of context-length limitations, it was not feasible to include tables in the prompt, particularly in Subtask A. Instead, a code-generating prompt was provided to the LLM, and the generated code was subsequently executed.

One of the main challenges encountered was errors in the generated code. To address this,

[1] https://github.com/rarahnamoun/TabularQA/

we implemented iterative prompts for code self-correction. However, not all errors could be resolved through iterative prompting alone. In such cases, we leveraged responses from alternative LLMs to rectify mistakes. Since different LLMs may not exhibit identical errors for a given question, we used responses from other models.

After all, to avoid mistakes due to data types, a post-processing algorithm was run to ensure the correct format of the expected output data. Our framework consists of three main components:

- **Self-Correction:** First, iterative prompts were used to correct errors in the generated code; if self-correction failed to resolve the issue, the next step was to utilize alternative LLM responses.
- **LLM Collaboration:** When self-correction failed, responses from other LLMs were used to improve the results, as not all LLMs generated incorrect code for the same question.
- **Post-Processing:** Finally, a post-processing algorithm was applied to ensure correct formatting and type validation of the results, aligning them with the expected outputs.

3.1 Problem Formulation

Given a dataset D and a natural language question Q, our goal is to find the function $f(D, Q)$ that returns the correct answer A. Formally:

$$A = f(D, Q) = \text{execute}(\mathcal{M}(\mathcal{P}(D, Q))) \quad (1)$$

where $\mathcal{P}$ is the prompt generator, $\mathcal{M}$ is the LLM generating code, and execute runs the code to retrieve A.

3.2 Prompt

The task consists of two subtasks, both following the same approach with one small difference.

For **Subtask A**, the dataset D can be of any size. Given a natural language question Q, the corresponding dataset name D is also provided. The competition includes 15 different datasets spanning multiple subjects and varying in size.

Examples of questions Q and datasets D are given in Figure 1.

Appendix A provides details on the prompt used for code generation in both subtasks. The primary difference between the two subtasks lies in the table information included in the prompt:

Dataset: 066_IBM_HR
Question: Is our average employee older than 35?

Dataset: 077_Gestational
Question: Which number of pregnancies is most common?

Figure 1: Examples of question (Q) and dataset (D).

- In **Subtask A**, due to context-length limitations of LLMs, only the columns and a few initial rows from D are provided in $\mathcal{P}(D, Q)$.
- In **Subtask B**, since only datasets with fewer than 20 rows are considered, the entire table from D is included in $\mathcal{P}(D, Q)$.

This structured approach ensures that the prompt $\mathcal{P}(D, Q)$ effectively guides $\mathcal{M}$ in generating executable code for obtaining A.

4 Self-Correction Mechanism

Let C_t be the generated code at iteration t, and E_t be the error encountered during execution. The LLM refines C_t iteratively:

$$C_{t+1} = \mathcal{M}(\mathcal{P}(D, Q, C_t, E_t)) \quad (2)$$

until execution produces no errors.

The Self-Correction Mechanism involves iterative refinement of a generated code to handle errors encountered during execution. The process starts with an initial code generation, followed by error detection. If an error is detected, the algorithm refines the code using an error-handling prompt in Appendix B. The error-handling prompt is added to the code generation prompt, which is described in Appendix A. This loop continues until the code executes successfully or a specified maximum number of iterations (in our experiments, 5 iterations) is reached. If errors persist after the limit, the algorithm returns a failure. The detailed steps of this mechanism are outlined in Algorithm 1.

4.1 Multi-LLM Collaborative Answer Generation

Our system integrates multiple LLMs, each providing independent responses to a given question based on tabular data. Formally, given a dataset D and a natural language question Q, each LLM M_i, where $i \in \{1, 2, ..., n\}$, receives a prompt $\mathcal{P}_i(D, Q)$ containing relevant table rows, column descriptions.The steps followed in the sections above are carried out separately for each LLM without any changes. The response A_i is given by:

Algorithm 1 Self-Correcting Mechanism

$C_0 \leftarrow \mathcal{M}(\mathcal{P}(D, Q))$ {Initial code generation}
for $t = 1$ to T **do**
  $A_t, E_t \leftarrow$ execute(C_t) {Run code and check errors}
  if $E_t = \emptyset$ **then**
    `Return` A_t {Return final answer}
  end if
  $C_{t+1} \leftarrow \mathcal{M}(\mathcal{P}(D, Q, C_t, E_t))$ {Refine code}
end for
`Return Failure` =0

$$A_i = M_i(\mathcal{P}_i(D, Q)) \tag{3}$$

where A_i represents the answer produced by model M_i for the given input. The system aggregates these answers for further evaluation and refinement.

To derive the final answer A, a consensus function $\mathcal{O}$ is applied over the set of generated answers:

$$A = \mathcal{O}(\{A_1, A_2, ..., A_n\}) \tag{4}$$

where $\mathcal{O}$ ensures that the generated output has a valid format and does not contain any errors.

4.2 Post-Processing

For each subtask, the expected answer must adhere to a predefined format. Given a dataset D and a natural language question Q, each LLM M_k (where $k \in \{1, 2, \ldots, n\}$) produces an answer A_i^k using the prompt $\mathcal{P}_k(D, Q)$. The responses from different LLMs are collected for further validation and selection.

Since some LLM responses may be empty due to iterative self-correction reaching its limit, we prioritize selecting a valid response. The selection process follows these steps:

1. If at least one answer has the correct expected format and size, we choose the valid response with the highest confidence. 2. If multiple LLMs provide valid answers, we apply a consensus function $\mathcal{O}$ based on majority voting where A represents the final selected answer. 3. If the number of valid answers is tied and n is even, a default LLM M_d is chosen as the tiebreaker, as was done in our experiments where $n = 2$.

Each answer must belong to a predefined category, including Boolean (True/False, Y/N), Category (values from dataset cells), Number (numerical/statistical values), List[category] (fixed-length categorical lists), and List[number] (fixed-length numerical lists). The format and constraints depend on the question's wording.

Since the expected answer type is unknown beforehand, the post-processing step ensures that A belongs to one of these categories.

The full post-processing procedure is detailed in Algorithm 2, which formalizes the steps for merging and filtering LLM outputs before selecting the final answer.

Algorithm 2 Post-Processing: Handling n LLM Outputs

$\mathcal{D}_k \leftarrow$ Read file for model M_k, $\forall k \in \{1, 2, \ldots, n\}$ {Load model outputs}
for each $(r_i, q_i, A_i^k) \in \mathcal{D}_k$ **do**
  $q_i \leftarrow g(q_i)$, $A_i^k \leftarrow g(A_i^k)$ {Pre-process questions and answers}
end for
Merge all $\mathcal{D}_k$ on r_i {Align model outputs}
$\mathcal{D} \leftarrow \{e(r_i) \mid r_i \in \mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_n\}$ {Extract relevant results}
for each $(r_i, q_i, A_1, A_2, \ldots, A_n) \in \mathcal{D}$ **do**
  $A_i^{k'} \leftarrow h(A_i^{k'})$ $\forall k' \in \{1, 2, \ldots, n\}$ {Apply validation and filtering}
  $A \leftarrow \mathcal{O}(\{A_1, A_2, \ldots, A_n\})$ {Select best valid answer}
end for=0

5 Experimental Setup

For generating results, the Together API[2] was used along with two models for experiments.

The models utilized were Llama 3.1 70B[3] (Dubey et al., 2024) and DeepSeek-V3[4] (Liu et al., 2024), The settings for Llama 3.1 70B and DeepSeek-V3 are: Max Tokens = 500 (both), Temperature = 0.7 (both), Top-p = 0.9 (Llama 3.1 70B) and 0.7 (DeepSeek-V3), Stream = False (both).

The evaluation was based on the `databench_eval` Python package introduced by the task organizer in the link[5].

[2] https://www.together.ai/
[3] https://huggingface.co/meta-llama/Llama-3.1-70B
[4] https://huggingface.co/deepseek-ai/DeepSeek-V3
[5] https://github.com/jorses/databench_eval/blob/main/src/databench_eval/eval.py

Metric	Subtask A		Subtask B	
	DeepSeek-V3	Llama 3.1 70B	DeepSeek-V3	Llama 3.1 70B
F1-score	84.9%	80.3%	85.7%	80.7%
Databench_eval	85.6%	80.1%	86.0%	78.9%

Table 1: Performance comparison of DeepSeek-V3 and Llama 3.1 70B on Subtasks A and B.

6 Results

As shown in Table 2, due to higher code errors in Llama 3.1 70B, the improvement percentage after the Self-Correcting step is higher than that of DeepSeek-V3. However, the self-error correction rate ability in DeepSeek-V3 is higher than in Llama 3.1 70B. DeepSeek-V3 outperformed Llama 3.1 70B in the Self-Correcting step across both subtasks. DeepSeek-V3 achieved an error correction rate of 86.67% in Subtask A and 100% in Subtask B, while Llama 3.1 70B showed correction rates of 56.67% and 66.67%, respectively. This demonstrates that DeepSeek-V3 is highly effective in resolving its own code errors. In the Collaborative step, where the outputs of both models were merged to handle cases where one model did not provide an answer, the improvement rates were 5.17% for Subtask A and 4.60% for Subtask B.

The final results in Table 1 also show that although the improvement of each step for Llama 3.1 70B was higher, due to the lack of performance in situations where the Llama 3.1 70B model's answer was accepted as the base model, it performs weaker than DeepSeek-V3.

Step	DeepSeek-V3	Llama 3.1 70B
Subtask A (*Improvement %*)		
Self-Correcting	2.49%	3.25%
Collaborative	5.17%	
Subtask B (*Improvement %*)		
Self-Correcting	1.5%	1.9%
Collaborative	4.60%	

Table 2: The improvement percentages for the Self-Correcting step have been calculated based on the number of corrected code errors relative to the total questions. For the Collaborative step, the percentage represents the number of questions where at least one LLM lacked an answer, and the other LLM's response was used instead, divided by the total number of questions.

As detailed in Appendix C, errors related to list-type outputs (Lists of integers, Lists of strings, and Lists of floats) are the most frequent errors for both DeepSeek-V3 and Llama 3.1 70B across Subtasks A and B. highlighting the difficulty LLMs face when handling structured list-based outputs.

In cases where both LLMs produced error-free answers that differed from each other, one of them was preferred. Because our experiment includes only two LLMs, we simply selected the response from the other LLM. The results are presented in Table 1. DeepSeek-V3 outperforms Llama 3.1 70B across all metrics for both Subtasks A and B. The model achieves higher F1-score and Databench_eval scores, indicating its superior performance in both subtasks.

Table 3 in Appendix D presents the results for open-source, open-weight models category ranking, where TeleAI secured the highest score in both tasks, followed by SRPOL AIS. The SBU-NLP team[6] ranked 6th in Subtask A but improved to 3rd place in Subtask B, demonstrating stronger performance in the second task.[7]

7 Conclusion

This paper presents an innovative algorithm leveraging a self-correction mechanism and a multi-LLM collaborative answer generation approach to address the key challenges in question answering from tabular data. By incorporating iterative error-checking in code generation and utilizing collaborative solutions to ensure valid expected answers, our method significantly reduces errors and inappropriate responses. Our error analysis highlights that most issues arise when the answer is a list rather than a simple data type, such as a number, string, or boolean. Future work will focus on refining the prompting strategies to minimize errors in such scenarios.

References

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman,

[6]Codabench ID: rashrah

[7]All rankings in this paper are based on the latest manual review of the task at the time of writing, within the open-source, open-weight models category.

Akhil Mathur, Alan Schelten, and ... Amy Yang. 2024. The llama 3 herd of models.

Jorge Osés Grijalba, Luis Alfonso Ureña-López, Eugenio Martínez Cámara, and Jose Camacho-Collados. 2024. Question answering over tabular data with databench: A large-scale empirical evaluation of llms. In *Proceedings of LREC-COLING 2024*, Turin, Italy.

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

Ruya Jiang, Chun Wang, and Weihong Deng. 2024. Seek and solve reasoning for table question answering. In *arXiv preprint arXiv:2409.05286*.

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, Seattle, United States. Association for Computational Linguistics.

Fangyu Lei, Shizhu He, Xiang Li, Jun Zhao, and Kang Liu. 2022. Answering numerical reasoning questions in table-text hybrid contents with graph-based encoder and tree-based decoder. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 1379–1390, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

A Liu, B Feng, B Xue, B Wang, B Wu, C Lu, C Zhao, C Deng, C Zhang, C Ruan, and D Dai. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

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

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.

Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. 2023. MultiTabQA: Generating tabular answers for multi-table question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6322–6334, Toronto, Canada. Association for Computational Linguistics.

A Code Generation Prompt

Prompt for Code Generation

The following table is from the dataset `{{dataset_name}}`. The first few rows are:
`{{sample_rows}}`
Your task is to write Python code that answers the question: "{{question}}"
The code should:

- Load the dataset from the appropriate location. The dataset is in the folder named 'competition' which contains subfolders named after the datasets (e.g., `{{dataset_name}}`).
- Load the dataset file (`sample.parquet`) and perform the necessary operations on the DataFrame.

Please do the following:

- Use `pd.read_parquet` to load the dataset from the full data file (`sample.parquet`).
- Process the DataFrame named 'df' accordingly and print the final result.

Please return only the Python code, without any explanation or extra text, and make sure it selects the correct file using the `dataset_name` from the 'competition' folder.
The file path for the full dataset is: `competition/{{dataset_name}}/sample.parquet`
Important Rules:

- Do not include explanations, comments, or code block markers (e.g., `python`).
- If there are multiple answers, format the output as: `['United Kingdom', 'Germany', 'France']`.
- Each code print must be in a single line (no line breaks).
- If the question answer is binary (`True`, `False`), do not include the count.

Types of Answers Expected:
According to the expected answer types:

- **Boolean:** Valid answers include `True/False`, `Y/N`, `Yes/No` (case insensitive).
- **Category:** A value from a cell (or a substring of a cell) in the dataset.
- **Number:** A numerical value from a cell in the dataset, which may represent a computed statistic (e.g., average, maximum, minimum).
- **List[category]:** A list containing a fixed number of categories. The expected format is: `['cat', 'dog']`. Pay attention to the wording of the question to determine if uniqueness is required or if repeated values are allowed.
- **List[number]:** Similar to List[category], but with numbers as its elements.

Prompt for Code Generation

Additional Notes:
Also, import all needed packages.
You will not know the specific type of answer expected, but you can be assured that it will be one of these types.
For the competition, the order of the elements within the list answers will not be taken into account.
The printed output in the code must be one of the above answer types.

B Error Correction Prompt

The code with errors, along with the error information, is sent to the LLM to attempt generating a corrected version. If a previous attempt resulted in an error, the following additional prompt is used to refine the code.

Error Handling Prompt

The previous attempt resulted in an error: `{{previous_error}}`

The previous code was: `{{previous_code}}`

Instructions: Correct the error and return **only** the fixed Python code.

C Error Analysis

As shown in Figures 2, 3, 4, and 5, errors related to list-type outputs (Lists of integers, Lists of strings, and Lists of floats) consistently account for the largest proportion of mistakes across both models and subtasks. . Additionally, numerical errors are notably frequent.

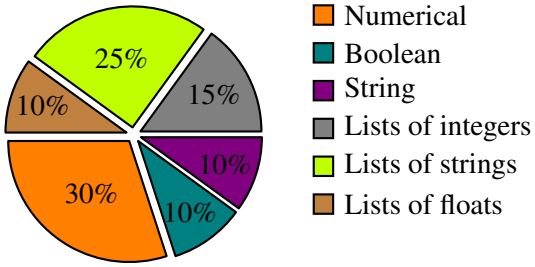

Figure 2: Error Breakdown for DeepSeek-V3 Subtask A

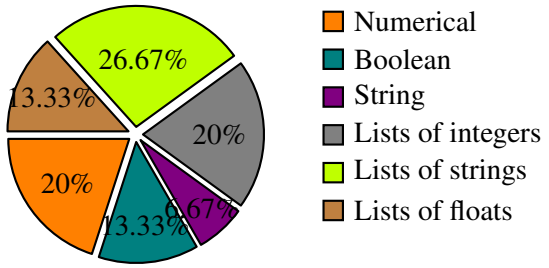

Figure 3: Error Breakdown for DeepSeek-V3 Subtask B

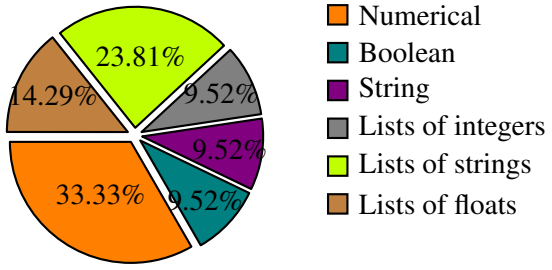

Figure 4: Error Breakdown for Llama 3.1 70B Subtask A

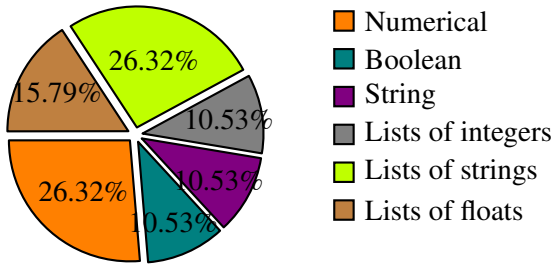

Figure 5: Error Breakdown for Llama 3.1 70B Subtask B

D SemEval Databench 2025 ranking

Subtask A			Subtask B		
Rank	**Team**	**Score (%)**	**Rank**	**Team**	**Score (%)**
1	TeleAI	95.02	1	TeleAI	92.91
2	SRPOL AIS	89.66	2	SRPOL AIS	86.59
6	**SBU-NLP**	85.63	3	**SBU-NLP**	86.02

Table 3: Performance comparison for Subtask A and Subtask B. Among the 37 teams using only open-source or open-weight models, the ranking is in a separate category. The table presents the top 2 rankings along with the result and ranking of SBU-NLP in both subtasks.