# Samsung Research Poland at SemEval-2025 Task 8: LLM ensemble methods for QA over tabular data

Paweł Bujnowski*, Tomasz Dryjański, Christian Goltz, Bartosz Świderski,
Natalia Paszkiewicz, Bartłomiej Kuźma, Jacek Rutkowski, Jakub Stępka, Miłosz Dudek,
Wojciech Siemiątkowski, Weronika Plichta, Bartłomiej Paziewski, Maciej Grabowski,
Katarzyna Beksa*, Zuzanna Bordzicka, Filip Ostrowski, Grzegorz Sochacki

**Samsung Research Poland, Warsaw**

*{p.bujnowski, k.beksa}@samsung.com

## Abstract

Question answering using Large Language Models has gained significant popularity in both everyday communication and at the workplace. However, certain tasks, such as querying tables, still pose challenges for commercial and open-source chatbots powered by advanced deep learning models. Addressing these challenges requires specialized approaches.
During the SemEval-2025 Task 8 competition focused on tabular data, our solution achieved 86.21% accuracy and took 2nd place out of 100 teams. In this paper we present ten methods that significantly improve the baseline solution. Our code is available as open-source software at the link: https://github.com/samsungnlp/semeval2025-task8.

## 1 Introduction

In recent years, Large Language Models (LLMs) have made significant advancements, emerging as powerful tools for extracting, interpreting, and generating insights from textual data. One of their most significant applications is Question Answering (QA), where LLMs provide contextually relevant responses to user queries. Although LLMs excel in natural language understanding, they still face challenges in processing and reasoning over tabular data, particularly in understanding relationships, identifying relevant columns, and answering complex queries. With a substantial amount of real-world data stored in tabular formats, the ability to efficiently interpret and utilize structured information seems more critical than ever.

### 1.1 Related methods

Tabular QA has gained significant attention in recent years, with various approaches being explored. Ye et al. (2024) generated pandas queries using only column names. Giang et al. (2024) introduced The Plan-of-SQL (POS), which enhances transparency by breaking down questions into SQL sub-queries. Zhang et al. (2023) proposed ReAcTable, which iteratively generates intermediate tables (through SQL or Python code) for step-by-step reasoning. Abhyankar et al. (2024) presented H-STAR, which extracts relevant table rows and columns before reasoning, reducing noise but risking error propagation if key columns are missed.

### 1.2 System overview

Our solution is based on an ensemble of carefully prompted models built around generative LLMs, where each model contributes to the prompt or verifies the result. Each of these models votes on the final answer. The system overview is illustrated in Figure 1a.

Although the models differ significantly – which is essential to leverage voting – they share a common structure composed of essential blocks, as shown in Figure 1b. Key components include table preprocessing and summary, identifying necessary columns and answer types, question paraphrases, few-shot learning and a correction loop.

## 2 Data

The training and test data we used was DataBench, a benchmark dataset for tabular data (Osés Grijalba et al., 2024; Os'es Grijalba et al., 2025). The authors of DataBench emphasised their intention to create a benchmark using "real-life" datasets, which also resulted in challenges in interpretation. The issues related to tables involved two areas (please refer to Table 10 in Appendix D for examples): (1) Multiple types of values within a single column (e.g. integers, floats and NaNs), (2) Unclear column names – acronyms or shortened words.

The analysis of the questions also showed that some posed greater challenges than others, which corresponded to the models' performance. We identified the following groups of issues: (1) The need



*Proceedings of the The 19th International Workshop on Semantic Evaluation (SemEval-2025)*, pages 1223–1232

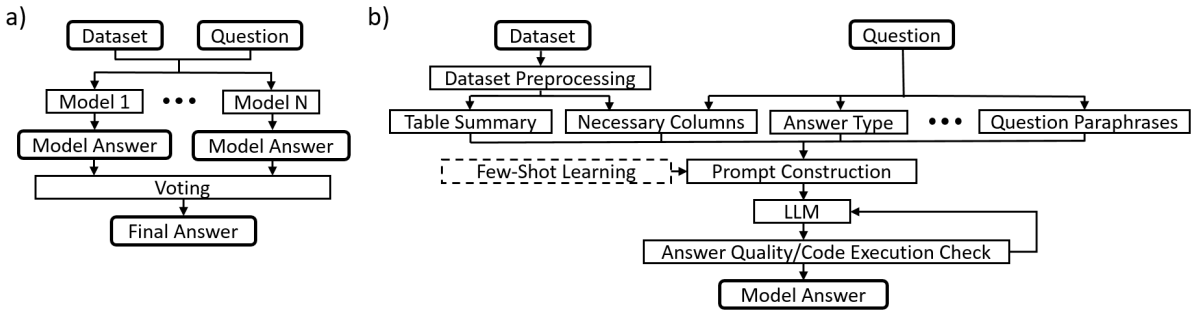


Figure 1: System overview. a) Top level overview of the system – multiple models are used to produce answers further used for voting. b) Overview of elements building a single model.

for external knowledge not present in the table, (2) The necessity to operate on substrings/altered strings, (3) Ambiguity in question phrasing that allows for multiple interpretations, (4) Multiple possible answers that are equally correct, (5) The need to convert units, such as weight or currency, (6) Complex phrasing or language mistakes. See examples in Table 11 in Appendix D.

## 3 Methods

### 3.1 Prompt Construction

During our experimentation we found that the following approaches significantly enhanced system performance and proved effective on our tasks:

**Data preprocessing:** LLM's performance declines without preprocessing due to issues like emojis in column names and table content interfering with code execution. We discuss it in Section 3.2.

**Table summary:** As LLMs struggle with numerical data, and passing an entire table would require a large context window, we included only a summary of the table in the prompt (see Section 3.4.)

**Detecting necessary columns:** To simplify the LLM tasks, we include only essential column names in the prompt. The detection process is detailed in Section 3.3.

**Question paraphrases:** We used question paraphrases in the prompts (see details in Section 3.8).

**Code output:** Our LLMs return code for answerers, as this solution works well with tables (Osés Grijalba et al., 2024). See Section 3.7 for details.

**Output formatting:** To ensure compatibility with the validation function, we specify the required answer formatting in the prompt for each question. We determine the necessary formatting by analyzing the questions with a separate model, as described in Section 3.6.

**Few-shot learning:** To enhance prompting, for some models, we used around 10 carefully selected QA examples.

### 3.2 Data preprocessing

We encountered irregularities in column names and issues when reading datasets with *pandas* `read_csv`, particularly with advanced data types, such as lists or dictionaries, which were incorrectly converted to strings. We implemented a multi-stage preprocessing pipeline involving:

(1) **Column name cleanup**: Removing emojis, HTML tags, and excess whitespaces.

(2) **Value Parsing**: At first we attempted literal evaluation. If this approach failed, we then parsed values as JSON objects. Finally, we transformed list-like values (those lacking quotation marks or containing extra brackets) into valid lists.

These steps improved data parsing for effective operations.

### 3.3 Detecting necessary columns

To reduce dimensionality, we used LLMs to identify the essential columns for specific questions. Through several experiments and iterative prompt engineering, we discovered that the best results were achieved by breaking down the task of extracting the appropriate pandas query into three steps:

1. Filtering the dataset with relevant columns (time-based, categorical, or entity-related filters).
2. Sorting, ranking, or aggregating data based on specific columns.
3. Returning the final answer by selecting the necessary columns.

We provided the LLM with a list of dataset columns, including data types and three random example values for each.

To ensure that the model returned only original column names from the dataset, the prompt restricted outputs to the provided column list. However, a postprocessing loop was added as a fallback, where each proposed column was checked against the input list. If a column was missing, preprocessing steps like removing double spaces, trimming underscores, and eliminating trailing whitespace were applied, followed by a re-check.

Ultimately, Llama 3.3 achieved approximately 95% accuracy[1] on this task, where accuracy was defined as the inclusion of *at least* all required columns for a given query.

### 3.4 Table summary

In order to provide LLM with additional knowledge about a given table, we created a script that extracts key information about the table, including column names, variable types, empty values, and statistics for numeric columns (standard deviation, mean, min and max).

We also checked whether each row of the column is unique and what are the most common values. The generated report, passed to the prompt, helped inform the LLM about the table's structure and potential difficulties in the analysis.

### 3.5 Raw data in markdown format

We found it valuable to include both the column headers and a sample of row data in markdown format. Typically, we fed the prompt with 20 rows.

### 3.6 Answer type prediction

To achieve balance between classification task metrics and GPU usage we utilized paraphrase-albert-small-v2 ALBERT based model (Lan et al., 2019) from the Sentence BERT model family (Reimers and Gurevych, 2019). The model was first fine-tuned on DataBench dataset. Given a query tokenized into subwords using ALBERT's tokenizer, the model then processed the text through the transformers layer, allowing its neural network to classify the given query into one of the answer types.

ALBERT' accuracy on the training set was approximately 96%, exceeding Llama 3.3's performance of 86% on the same task. See Appendix A for more ALBERT's result details.

To further improve the classification, a voting system incorporating Llama 3.3 and Qwen 2.5 was deployed. In case of ALBERT and Llama disagreeing, Qwen is inferenced. Thanks to this voting, the overall accuracy of answer type prediction increased to 98.28%.

### 3.7 Python pandas and SQL code generation

Our approach involved generating single, one-line commands in pandas and SQL. At first, we prompted LLM to generate pandas code answering a question. We constructed our prompts iteratively, as described in Section 3.9 It was specified that the model should generate a plain command, without any additional explanation. Tests revealed that for some questions LLM continued to make similar mistakes in pandas commands.

For stronger contribution to the ensemble of models, we asked LLM to write SQL queries. The prompt construction mirrored that written in pandas case, introducing as an add-on SQL schema of a table. For generating and executing SQL code we used SQLite and DuckDB.

### 3.8 Question paraphrasess

Using paraphrase generation as an auxiliary method to increase accuracy is a common approach applied in various AI systems, e.g.: text style transfer (Bujnowski et al., 2020) or open domain question answering (Siriwardhana et al., 2023). In our experiments we generated paraphrases of questions using Qwen 2.5 and used them in various answerer models. Input to the model was a prompt with a task to return 5 paraphrases of a question (in a JSON format) and included the table headline and a few examples (e.g. 3 rows) from the dataset in a markdown format.

Question paraphrases seem to be beneficial for LLMs in case of ambiguous questions, e.g. by using column names directly or reformulating a question in a less complex way.

### 3.9 Loops for code correction

We employed an LLM for QA tasks by generating pandas or SQL code iteratively. The process involved querying the LLM to propose a code snippet within a loop, which was set to a maximum number of iterations ($max\_iter$). Each proposed code was then executed to evaluate its response. If the code was executed without errors, the response was accepted. If an error occurred, the information about the error was fed back into the LLM as feedback, allowing it to refine the next proposed code snippet. This iterative process continued until either

[1]This result is nontrivial to calculate precisely, as the task is inherently nondeterministic, and some questions may have multiple valid solutions.

executable code was generated or the $max\_iter$ limit was reached.

The next stage of our pipeline focused on improving the generated queries. Common errors included the absence of methods such as `.to_list()`, `.any()`, `.iloc[0]`, `.item()`, `.index.to_list()` at the end of a query, problems with redundant or missing brackets, as well as unnecessary artifacts of LLM's responses such as ```` ```python ````. The auxiliary LLM received input that included a pre-generated pandas query along with details about possible issues, and was tasked with generating a corrected version of the query based on this information.

### 3.10 Limiting inference tokens

Many questions demanded thorough understanding of both the question and dataset, prompting us to use reasoning models. We adhered to established prompt structures and temperature recommendations (Guo et al., 2025). However, for ambiguous or highyly dataset-specific questions, reasoning models often generated excessively long thinking processes without arriving at correct solutions. We addressed this by implementing a token cap for the thinking process, which forced the models to provide final answers after reaching a predetermined token limit.

### 3.11 Ensemble models

To improve predictions we used ensemble models, selecting the best-performing ones based on training set results. For each question we took answers inferred by selected models and removed these flagged as invalid. If there was a single answer left, it was returned as the ensemble result. Otherwise, we next applied simple majority voting. The vote was considered conclusive if more than the half of the answers were consistent. If not, we used Qwen 2.5 for arbitration. Its input included: the question text, column names, inferred necessary columns (Section 3.3), table summary (Section 3.4), predicted answer type (Section 3.6), and valid model results. Additionally, we assessed the complexity of pandas queries where applicable, based on factors such as the number of executable functions in a query, occurrence of a custom function like `lambda` or `.apply()`, and presence of data type conversions. For each criterion the query received a penalty, which was then added up to the final score and fed to the LLM to support voting.

## 4 Results

### 4.1 Experimental setup

| Model name | Model size |
|---|---|
| Llama-3-Instruct | 70B |
| Llama-3.3 | 70B |
| Qwen-2.5-Instruct | 72B |
| DeepSeek-R1-Distill-Qwen | 32B |
| DeepSeek-R1-Distill-Llama | 70B |
| DeepSeek-R1 | 671B |

Table 1: LLMs: models used in experiments and for final predictions, with their respective parameter counts.

We evaluated various LLMs and selected them based on their high scores on coding and reasoning benchmarks (Dubey et al., 2024; Guo et al., 2025; Yang et al., 2025). The specific models that we used are detailed in Table 1. All models were implemented in 4-bit quantized versions due to hardware limitations, and executed on GPUs via the llama.cpp interface (further details provided in the Appendix in Table 7).

### 4.2 Results of separate and ensemble models

In Table 2 we present our results for single and ensemble models for "FULL" task. Our top-performing system achieved an accuracy of 86.21% and consisted of 8 various models using combinations of the methods outlined in Section 3.

In Table 8 in Appendix D we present examples of the most challenging questions from the test dataset, which none of our models with accuracy over 80% could answer correctly. Additionally, the Lite task results are shown in Appendix C.2.

### 4.3 Ablation studies for "FULL" task

To better understand how various methods presented in Section 3 impact the final results, we conducted an ablation study using Qwen 2.5 model. We changed the parameters of one model, starting with the base methods and progressively adding more sophisticated ones. While the choice of methods was somewhat arbitrary, calculating the full permutation of methods was difficult. The results for two types of code generation queries – Python pandas and SQL – are shown in Table 3.

The baseline method consisted of a simple prompt with a single code generation attempt. The original pandas and SQL prompts used in the ablation studies are presented in Appendix B. To this simple prompt we added just three sentences (we

| Experiment | Accuracy | Description |
|---|---|---|
| Voting | **0.8621** (0.8736) | Voting from models S10, S8, S9, S11, S7, S6, S4, S1. The winning model submitted to the competition |
| Voting | **0.8602** (0.8716) | Voting from models S12, S10, S8, S9, S11. The better single model (S12) added after submission |
| Voting | **0.8563** (0.8678) | Voting from models S12, S10, S8, S9, S11, S7, S6, S4, S1. Long list of models to vote from (9 models) |
| Voting | **0.8506** (0.8621) | Voting from models S12, S10, S8, S9, S11, S7, S6, S4, S1. Only LLM voting, majority voting not used |
| S12 | **0.8333** (0.8448) | Qwen 2.5; generation of pandas code using methods 3.1 – 3.9 |
| S11 | **0.8161** (0.8276) | Qwen 2.5; generation of SQL code using methods 3.1 – 3.9 |
| S10 | **0.8161** (0.8276) | Qwen 2.5; generation of pandas code using methods 3.1 – 3.9; shorter prompt |
| S9 | **0.8142** (0.8276) | Qwen 2.5; generation of SQLite code using methods 3.5, 3.8 |
| S8 | **0.8123** (0.8238) | Llama 3.3; generation of pandas code using methods 3.1 – 3.9 |
| S7 | **0.8084** (0.8199) | DeepSeek R1; generation of pandas code; 3.1 – 3.7, 3.9, 3.10 (inference limit: 3000 tokens) |
| S6 | **0.7950** (0.8065) | Qwen 2.5; generation of pandas code using methods 3.1 – 3.4 and 3.6 – 3.9; shorter prompt |
| S5 | **0.7893** (0.8008) | Qwen 2.5; generation of pandas code using methods 3.3, 3.4, 3.6 |
| S4 | **0.7874** (0.7989) | Llama 3.3; generation of pandas code using methods 3.1 – 3.9; shorter prompt |
| S3 | **0.7510** (0.7625) | Llama 3.3; generation of pandas code using methods 3.3, 3.4, 3.6 |
| S2 | **0.7356** (0.7471) | DeepSeek R1; generation of pandas code; 3.1 – 3.4, 3.6, 3.7, 3.10 (inference limit: 1200 tokens) |
| S1 | **0.729**9 (0.7395) | Qwen 2.5; generation of DuckDB code instead of pandas |

Table 2: Performance of single models and their ensembles on the test set.
In brackets: results with the final evaluation function updated by the task organizers.

| Methods of one model for FULL testset | Accuracy (pandas) | Accuracy (SQL) |
|---|---|---|
| Simple prompt, 1 LLM request | 0.4770 | 0.6743 |
| Extended prompt, 1 LLM request | 0.6782 | 0.6897 |
| ... + up to 3 LLM requests (3.9) | 0.6801 | 0.7050 |
| ... + up to 10 LLM requests (3.9) | 0.6877 | 0.7088 |
| ... + added 20 table rows in markdown (3.5) | 0.7759 | 0.7682 |
| ... + added table summary (3.4) | 0.7893 | 0.7510 |
| ... + answer type classifier (3.6) | 0.8218 | 0.8199 |
| ... + necessary column detector (3.3) | 0.8238 | 0.8046 |
| ... + LLM-gen. 5 paraphrases of question (3.8) | 0.8333 | 0.8218 |
| ... + LLM-gen. 5 paraph. of question w/o column detector | 0.8429 | 0.8314 |

Table 3: Ablation studies: the impact of methods on one-model results: Python pandas or SQL query generation.

call it "the complex prompt"), achieving a 20% increase in the performance of the pandas code model. Next, we increased the number of LLM inferences, up to 3 or 10 when necessary (described in Section 3.9), resulting in a performance gain of between 1% and 1.9% (for 10 loops).

Two factors had the greatest impact on accuracy in the later stages: (1) Adding the number of table rows in markdown format (+8.8% for pandas and +5.9% for SQL), (2) Including answer type prediction (described in Section 3.6; +3.3% for pandas and +6.9% for SQL).

The necessary columns selector, presented in Section 3.3, slightly improved the pandas code results and worsened the SQL results.

Finally, using paraphrases (Section 3.8) improved outcomes by +1% for SQL and 1.9% for pandas (with the column selector removed).

Interestingly, both pandas and SQL models reached similar maximum accuracy of 84.3% and 83.1% respectively, with larger differences when applying various methods in the earlier stages.

## 5 Limitations

Our system was built and fine-tuned using the DataBench dataset. Although it includes a diverse set of tables and questions, our experiments were conducted on a limited sample of real data. Additional research is necessary to determine how well the proposed methods generalize to other domains.

## 6 Conclusion

Despite the availability of verified open-source LLMs, answering questions over massive tabular data is still a challenging task. Designing an effective prompt is undoubtedly a crucial method, but can be difficult to control. Interestingly, simple feature and system engineering, combined with common classifiers, continue to be valuable and can significantly improve QA accuracy. Through our experiments in the SemEval Task 8, we demonstrated that using multiple models and smart voting

can result in creating an effective, general-purpose tabular QA system.

## References


Nikhil Abhyankar, Vivek Gupta, Dan Roth, and Chandan K. Reddy. 2024. H-star: Llm-driven hybrid sql-text adaptive reasoning on tables. *Preprint*, arXiv:2407.05952.

Pawel Bujnowski, Kseniia Ryzhova, Hyungtak Choi, Katarzyna Witkowska, Jaroslaw Piersa, Tymoteusz Krumholc, and Katarzyna Beksa. 2020. An empirical study on multi-task learning for text style transfer and paraphrase generation. In *Proceedings of the 28th International Conference on Computational Linguistics: Industry Track*, pages 50–63, Online. International Committee on Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, and 516 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Giang, Nguyen, Ivan Brugere, Shubham Sharma, Sanjay Kariyappa, Anh Nguyen, and Freddy Lecue. 2024. Interpretable llm-based table question answering.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, and Radu Soricut Piyush Sharma. 2019. Albert: a lite bert for self-supervised learning of language representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

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.

Jorge Osés Grijalba, L. 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 the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 13471–13488, Torino, Italia. ELRA and ICCL.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana, and Suranga Nanayakkara. 2023. Improving the domain adaptation of retrieval augmented generation (RAG) models for open domain question answering. *Transactions of the Association for Computational Linguistics*, 11:1–17.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Junyi Ye, Mengnan Du, and Guiling Wang. 2024. Dataframe qa: A universal llm framework on dataframe question answering without data exposure. *Preprint*, arXiv:2401.15463.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2023. Reactable: Enhancing react for table question answering.

## A Answer type prediction – ALBERT classifier results

Appendix A provides detailed performance metrics for the classification model discussed in Section 3.6.

| | |
|---|---|
| Accuracy | 0.9618 |
| F1 Score | 0.9620 |
| F1 Micro Score | 0.9618 |
| F1 Macro Score | 0.9643 |

Table 4: ALBERT model metrics.

Table 4 presents the overall performance of the ALBERT model, fine-tuned on the DataBench dataset, evaluated using accuracy and F1 score.

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| boolean | 1.00 | 1.00 | 1.00 | 44 |
| category | 1.00 | 1.00 | 1.00 | 42 |
| list[category] | 0.85 | 0.96 | 0.90 | 48 |
| list[number] | 0.97 | 0.88 | 0.92 | 65 |
| number | 1.00 | 1.00 | 1.00 | 63 |
| accuracy | | | 0.96 | 262 |
| macro avg | 0.96 | 0.97 | 0.96 | 262 |
| weighted avg | 0.96 | 0.96 | 0.96 | 262 |

Table 5: ALBERT classes metrics.

Table 5 breaks down ALBERT's classification report for answer types, highlighting class-specific strengths and weaknesses.

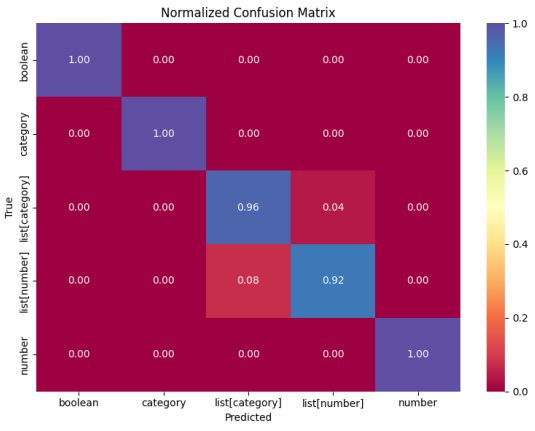


Figure 2: ALBERT Confusion Matrix.

Figure 2 visualizes the correctness of predictions made by the ALBERT model, pretrained on the questions from the dev and train sets. The matrix has been normalized to simplify analysis.

Table 6 compares the accuracy of ALBERT, Llama 3.3 and Qwen 2.5 on the DataBench training set.

| Model | Accuracy |
|---|---|
| ALBERT | 0.9618 |
| Llama 3.3 | 0.8624 |
| Qwen 2.5 | 0.9067 |

Table 6: Classification accuracy.

## B Ablations study – supplement

We present both a simple and a complex prompt (the latter with added purple sentences) for pandas and SQL generation output. The red color indicates string variables that were added. Some sentences contain syntax errors (such as the repetition of the word "executable" in the last sentence). However, this version worked better compared to the correct version, where "executable" was used only once.

**pandas_question_prompt** = f"""{You are given a pandas DataFrame named 'df' which contains the following columns: {*all_column_names*}. Based on this information, generate a query in Python Pandas to answer the question: {*question_text*}. Specify only the code needed to calculate the answer using pandas (don't write anything else and do not write anything else. Also, return the code as a string, without any characters marking that this is code. Make sure it is an executable command, not a print statement. Be attentive to units of measurement, currencies, and notation systems, as data can be represented in various ways (using numbers, words, abbreviations, or symbols). Verify if conversion is needed. For a currency column it could be better to transform values into floats (on fly) and answer using it. Finally, make sure the code you produce will return an answer in the proper format – it should never be a DataFrame, a dictionary or a series. Make sure the answer is an executable one line executable command!!!} """

**sql_question_prompt** = f"""{You are given a pandas DataFrame named 'df' created from famous dataset: {*dataset_name*} with columns: {*all_column_names*}. In the next step 'df' DataFrame is converted to SQLite table with schema: {*schema*}. Please generate a query in SQLite answering the question: {*question_text*}. Specify only SQL query. Don't write anything else and do not write any explanation! Also, return the query as a string, without any characters marking that this is code. Be attentive to units of measurement, currencies, and notation systems, as data can be represented in various ways (using numbers, words, abbreviations, or symbols). Verify if conversion is needed. For a currency column it could be better to transform values into floats (on fly) and answer using it. Query should be as simple as possible, avoid nested queries and joins whenever possible! """

# C Results

## C.1 The most challenging questions

Out of the 522 question in the test set, 21 turned out to be the most difficult. It emerges that none of our models with accuracy exceeding 80% managed to return the correct answers to them. Table 8 depicts 5 examples from this set.

## C.2 Results on the DataBench Lite QA subtask

We applied the same methods as for the full version subtask, and received results presented in Table 9.

# D Data

Table 10 illustrates challenges in table interpretation based on columns (either their names or the type of values). Table 11 provides examples of potentially problematic questions, followed by a short discussion of the applicable issue.

| Model name | Parameters | Quantization | Hardware | Source |
|---|---|---|---|---|
| Llama-3-Instruct | 70B | Q4_K_M | 2 x Quadro RTX 8000 | bartowski |
| Llama-3.3 | 70B | Q4_K_M | 2 x NVIDIA RTX A6000 | unsloth |
| Qwen-2.5-Instruct | 72B | Q4_K_M | 2 x NVIDIA L20 | bartowski |
| DeepSeek-R1-Distill-Qwen | 32B | Q4_K_M | NVIDIA L20 | unsloth |
| DeepSeek-R1-Distill-Llama | 70B | Q4_K_M | 2 x Quadro RTX 8000 | unsloth |
| DeepSeek-R1 | 671B | Q4_K_M | 8 x NVIDIA H100 | unsloth |

Table 7: LLMs: specifications of the models used, the hardware they were deployed on, and the source of quantized weights.

| Question | Comment | Challenge |
|---|---|---|
| *What is the name of the animal involved in the production of the most expensive coffee-related product that we offer? Answer with a value present in a cell of the database.* | It was impossible to create a valid query to search for an unspecified animal within the column values. | external knowledge |
| *How many suns were there in the title of Hosseinis' novel? Answer with a number* | The book title is *A Thousand Splendid Suns*. The number in this cell is a string, but an integer is required as the answer. An additional difficulty is that the models must search for an unspecified number. | substring/altered string |
| *List the first (by number of appearance) 3 different values in the highest tier of the dataset. If there are less than 3 list as many as there are.* | The expression "highest tier of the dataset" was confusing for the models – the majority of them chose the column named "Tier 4" instead of "Tier 1", as the former name include the highest number. | ambiguity |
| *Is Barbados considered overall more expensive than the country ranked in the 10th place?* | "Overall more expensive" refers to the most general index, i.e. "Cost of Living Plus Rent Index". Meanwhile, models took various approaches, e.g. they tried to sum several random indices and compare these sums. | ambiguity |
| *Is the average age of all lifting records in the weight class of someone who weights 103000 grams above 40?* | Converting weight units turned out to be an issue. | converting units |

Table 8: Examples of questions to which none of the models answered correctly.

| Experiment | Accuracy | Description |
|---|---|---|
| Voting | **0.8563** (0.8659) | SL9, SL8, SL6, SL5, SL4, SL3 |
| Voting | **0.8506** (0.8602) | SL8, SL6, SL5, SL4, SL3 |
| SL9 | **0.8467** (0.8582) | Qwen 2.5; generation of pandas code using methods 3.1 – 3.9 |
| SL8 | **0.8276** (0.8372) | SL7 with code correction (Section 3.9) |
| SL7 | **0.8218** (0.8314) | Qwen 2.5; generation of pandas code using methods 3.1 – 3.9; shorter prompt |
| SL6 | **0.8218** (0.8314) | Qwen 2.5; generation of SQL code instead of pandas |
| SL5 | **0.8161** (0.8257) | Qwen 2.5; generation of pandas code using methods 3.1 – 3.4 and 3.6 – 3.9; shorter prompt |
| SL4 | **0.7893** (0.7989) | SL1 with code correction (Section 3.9) |
| SL3 | **0.7778** (0.7874) | SL2 with code correction (Section 3.9) |
| SL2 | **0.6897** (0.6973) | Llama 3.3; generation of pandas code using methods 3.5 and few-shot learning |
| SL1 | **0.6743** (0.6839) | Llama 3.3; generation of pandas code using methods 3.5 |

Table 9: Performance of single models and their ensembles on the test set for the DataBench Lite QA subtask. In brackets: results with the final evaluation function updated by the task organizers.

| Dataset | Column name | Meaning | Question | Comment |
|---|---|---|---|---|
| 069_Taxonomy | Parent | The Parent ID number. | *List the 3 Parent values associated with the 3 highest number of descendants (direct or otherwise).* | The column contains both integers (e.g. "483") and strings (e.g. "SPSHQ5"). |
| 072_Admissions | Research | Indicates if a candidate has any research experience. | *Do most students have some research experience prior to the application?* | The answer contains integers (0 and 1), not strings ("yes", "no"). |
| 072_Admissions | LOR | The score for Letter of Recommendation (a statement given by a university or college.). | *What is the score for the recommendation letters presented by the student with the lowest English score?* | In the column name, there is an abbreviation, whereas in the question the term is paraphrased. |
| 023_Climate | racha | The maximum wind speed. | *Did any day with maximum wind speed above 15 also have average wind speed below 5?* | The column name is in Spanish. |
| 022_Airbnbs | host_total_listings_count | The number of properties owned by a host. | *Are there any hosts who have listed more than 10 properties?* | The wording in the column name differs from the phrasing used in the question. |

Table 10: Examples of challenging columns. The first two concern value types and the following three illustrate unclear column names.

| Challenge | Dataset | Question | Comment |
|---|---|---|---|
| External knowledge | 058_US | *How many respondents have a high school degree or less as their highest level of education?* | Knowledge about the US education system is necessary. |
| External knowledge | 074_Lift | *Are there more than 100 lifters in the weight class someone that weighs 82kg would compete in?* | The model needs to understand that weight categories are fixed weight ranges and then find the closest weight category. |
| Substring/altered string | 018_Staff | *Were there any employees hired in 2019?* | The content of the applicable cell is a date from which the year must be extracted. |
| Substring/altered string | 080_Books | *How much stock (in number of books) of Ben Graham's work is there in this store?* | There is "Benjamin Graham" in the dataset, not "Ben". |
| Ambiguous questions | 017_Hacker | *List the top 4 most frequent terms in the "Clusters II" column.* | An exemplary value for the "Clusters II" column is "year, work, new". It is unclear whether to list 4 most frequent sets of terms or 4 single-word terms. |
| Ambiguous question | 077_Gestational | *How many teen pregnancies are there in this dataset?* | The name of the applicable column is "Pregnancy No", which may refer to either cardinal or ordinal numbers. |
| Ambiguous question | 078_Fires | *What is the name of the month that recorded the driest day when a fire took place?* | There are three columns with dryness metrics: RH (relative humidity), DC (Drought Code), and DMC (Duff Moisture Code). |
| Multiple possible answers | 074_Lift | *List 5 lifters from the "74 kg" weight class.* | There are 10 lifters in this weight class. |
| Multiple possible answers | 076_NBA | *List the 5 players with the least games played.* | There are 26 players who played just 1 game. |
| Converting units | 074_Lift | *Is the biggest lift performed greater than 880 pounds?* | The content of the applicable column is in kilograms, so conversion is necessary. |
| Converting units | 077_Gestational | *List the weights of women with a height of exactly 1m and 45cm.* | The numbers in the "Height" column range from 135 to 196, so they are in centimeters (though not specified). |
| Language mistakes | 020_Real | *What are the 2 types of properties which are listed more frequently?* | It is not specified what "more frequently" refers to, leading one to assume the question is about the "most frequently" listed properties. |
| Language mistakes | 077_Gestational | *What is the most value of the status marking hereditary diabetes risk in the dataset?* | It is unclear whether the question refers to "the highest value" or "the most common value" (the dataset authors admit a word is missing, indicating that it is the latter). |
| Lexically challenging | 072_Admissions | *List the best 2 graduate record scores of applicants whose stated motivation to enter got a rating better than 4.* | The phrase "graduate record scores" refers to "GRE Score" (Graduate Record Examinations), and "stated motivation to enter" refers to "SOP" (statement of purpose). |

Table 11: Challenging questions examples.