TableRAG: A Novel Approach for Augmenting LLMs with Information from Retrieved Tables

Elvis A. de Souza[1], **Patricia F. da Silva**[2], **Diogo Gomes**[2], **Vitor Batista**[2], **Evelyn Batista**[1], **Marco Pacheco**[1]

[1]Applied Computational Intelligence Lab. (ICA/PUC-Rio)

[2]Petrobras

Abstract. *We present TableRAG, a novel pipeline designed to integrate tabular data into traditional Retrieval-Augmented Generation (RAG) systems. Our approach is composed of three main parts: (i) generating textual representations of tables; (ii) indexing table representations in vector databases for retrieval, and (iii) employing large language models to generate SQL or Python code for data manipulation over a given table. We assessed the effectiveness of TableRAG by comparing retrieval and re-ranking accuracies over the OTT-QA benchmark and by utilizing both open and closed-source LLMs to generate code for answering questions from the WikiTableQuestions benchmark. Our best results show 86.7% HITS@5 for retrieval and 74% accuracy for Q&A, demonstrating the feasibility of integrating tabular data into RAG systems with high accuracy.*

1. Introduction

In recent years, the application of generative models in Question Answering (Q&A) systems has gained substantial traction, particularly in enterprise settings where accurate and efficient information retrieval is crucial for decision-making, customer support, and operational efficiency. Large Language Models (LLMs) have shown remarkable capabilities in generating natural language responses based on vast amounts of textual data, however, these models are not without their challenges—one of the most significant being the issue of hallucination, where the model generates plausible-sounding but incorrect or nonsensical answers [Kandpal et al. 2023, Gao et al. 2023, Lin et al. 2023, Tonmoy et al. 2024]. This problem becomes particularly pronounced when the required information is not purely textual but embedded in semi-structured data, such as tables, which can be stored in large and diversified databases, where the ability to accurately retrieve and interpret them is essential.

To mitigate hallucination and to retrieve relevant information to a particular domain, a widely adopted paradigm known as Retrieval-Augmented Generation (RAG) is employed. This technique is based on the premise that LLMs are more likely to provide accurate responses when supplied with relevant context at runtime, i.e., within the prompt that defines the task instructions, in a strategy called in-context learning [Dong et al. 2024]. Traditional Information Retrieval (IR) methods are utilized to fetch pertinent documents, which are then fed into the language model for text generation.

In this work, we propose a RAG pipeline that integrates tabular data within textual databases for information retrieval, using retrieved tables to generate reliable responses (Figure 1). Building upon traditional RAG pipelines, we implement a table retrieval module based on the same vector similarity strategy commonly used for texts, and another

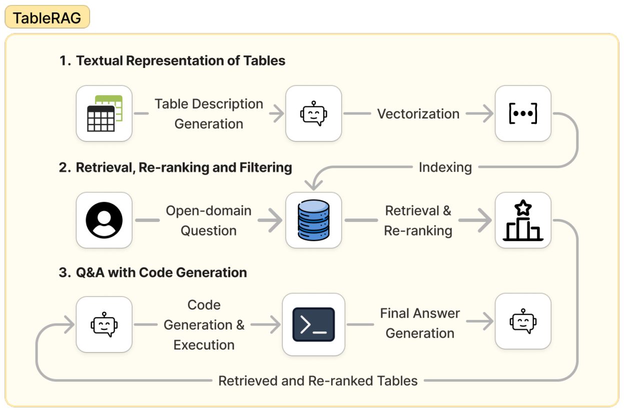

TableRAG
1. Textual Representation of Tables
Table Description Generation
Vectorization
Indexing
2. Retrieval, Re-ranking and Filtering
Open-domain Question
Retrieval & Re-ranking
3. Q&A with Code Generation
Code Generation & Execution
Final Answer Generation
Retrieved and Re-ranked Tables

Figure 1. Overview of the TableRAG Pipeline

module for Q&A based on generating code to manipulate tables. We evaluate the results using two well-known Q&A benchmarks with tabular data, Open Table-and-Text Question Answering (OTT-QA) [Chen et al. 2020a] for the retrieval part of our pipeline, and WikiTableQuestions [Pasupat and Liang 2015] for the Q&A part.

Our results show that it is possible to incorporate tabular data into Retrieval-Augmented Generation systems in an efficient way by representing tables as texts, similarly to how traditional retrieval information systems work. Moreover, we show that code generation yields good results for obtaining answers from retrieved tables, and we highlight the considerable promise for improving the performance of open-source LLMs, which could lead to the development of more robust, adaptable, and affordable RAG systems, capable of managing multimodal data sources.

2. Related Work

Unlike Q&A over texts, which involves reading document excerpts to answer questions using extractive or generative models, Q&A over tables involves additional factors. To answer complex questions using tables, one must interpret the arrangement of rows and columns, perform filtering, joins, mathematical operations, and various other forms of table manipulation.

There are at least three tasks that have been tested in the literature for Q&A with tabular data. First, parsing semantically compositional questions in order to determine the manipulation steps that are needed to obtain a given information from a table. The WikiTableQuestions benchmark [Pasupat and Liang 2015] addresses this task. It comprises semi-structured tables, which may contain textual data, and each question may require operations such as table lookup, aggregation (counting records, summing numerical values, etc.), superlatives (finding the maximum or minimum value), arithmetic operations, among others, which need to be identified in the question in order to determine the manipulations steps that are needed to answer it correctly.

Second, manipulating relational database tables using their relationships with other tables to join information from different sources. The Spider benchmark [Yu et al. 2018] introduced this task. Its specificity involves the use of foreign keys, joining multiple tables, and constructing nested SQL queries.

Third, answering multi-hop open-domain questions that need reasoning over both textual passages and tabular information. The Open Table-and-Text Question Answering (OTT-QA) benchmark [Chen et al. 2020a] introduced this task. Its peculiarity is twofold: the retriever model must find the table that best answers the question from a large collection of tables, and the reader model must simultaneously examine data from two different modalities, texts and tables.

For the Q&A part, our approach seeks to address the challenges proposed by the WikiTableQuestions benchmark, using LLMs to generate code that correctly manipulates tables in order to find answers to semantically compositional questions. In 2015, when the WikiTableQuestions benchmark was proposed, the approach used to solve the task involved converting tables into knowledge graphs and parsing the questions into logical form, followed by selecting the most probable graph candidates to answer the question [Pasupat and Liang 2015]. The results achieved were 37.1% accuracy, considering all candidate answers, and 76.6%, considering that at least one of the candidate answers was correct.

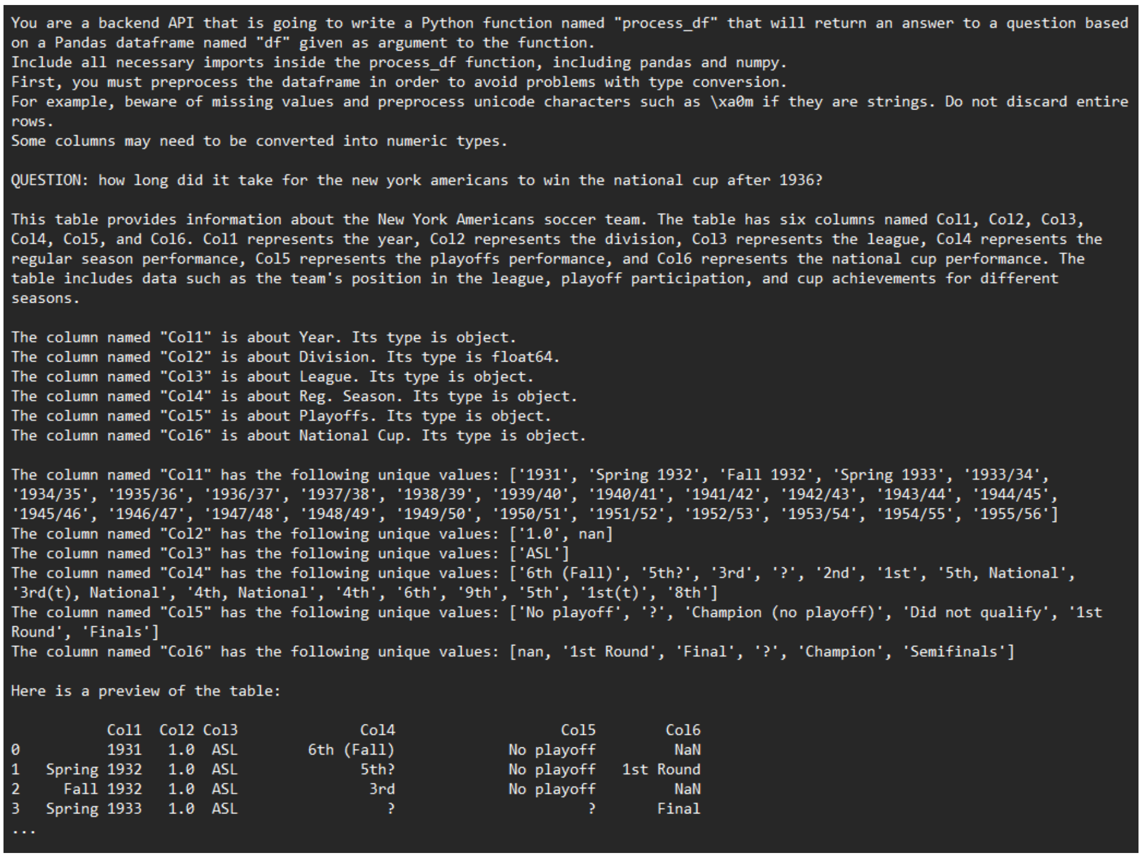
More recent works address the WikiTableQuestions task using large language models. [Yin et al. 2020] achieved a result of 52.3% accuracy by pre-training a new model, called TaBERT, with data from 26 million tables and their respective natural language contexts [Liang et al. 2018]. The work by [Liu et al. 2024] used GPT-3.5 to transpose tables to normalize them and then used the same model to perform two types of inference: with direct prompting (DP), asking the model to reason about the table in textual form, and as a Python agent, asking the model to interact with a Python shell in up to five interactions. They achieve state-of-the-art accuracy of 73.6% on WikiTableQuestions.

3. TableRAG Pipeline Components

The instruction in Figure 2 is an example of the prompt directed to the language models for generating Python code. In this prompt, we can observe, besides the request for code generation, how the tables are represented textually and in consideration of their metadata. These procedures will be explained step by step in the following subsections.

Obtaining Textual Representation of Tables This part of our approach is inspired by the work of [Abraham et al. 2022]. Given that it is not always feasible to load entire tables into memory, the authors employ several strategies to represent the table through its schema, ensuring that the SQL query to be created is based on this indirect form of representation. The idea works particularly well in the context of Retrieval-Augmented Generation, where the initial step is to retrieve specific documents from a diverse document repository. In our case, we aim to retrieve the tables that best answer the user's query, using the same strategy for text indexing and retrieval.

The metadata generated for a table consists of a textual description, generated by an LLM, along with additional information such as column names, unique values, and data types (numeric, textual, dates) for each table column. Other pieces of information, such as title of the table, paragraphs that explain it, acronyms meanings and so on can be inserted here to enrich the textual description of each table. When the table preview is passed to the language model, it is placed as the final element of the prompt, so to truncate it if exceeding the input token limit that the model accepts.



```
You are a backend API that is going to write a Python function named "process_df" that will return an answer to a question based
on a Pandas dataframe named "df" given as argument to the function.
Include all necessary imports inside the process_df function, including pandas and numpy.
First, you must preprocess the dataframe in order to avoid problems with type conversion.
For example, beware of missing values and preprocess unicode characters such as \xa0m if they are strings. Do not discard entire
rows.
Some columns may need to be converted into numeric types.

QUESTION: how long did it take for the new york americans to win the national cup after 1936?

This table provides information about the New York Americans soccer team. The table has six columns named Col1, Col2, Col3,
Col4, Col5, and Col6. Col1 represents the year, Col2 represents the division, Col3 represents the league, Col4 represents the
regular season performance, Col5 represents the playoffs performance, and Col6 represents the national cup performance. The
table includes data such as the team's position in the league, playoff participation, and cup achievements for different
seasons.

The column named "Col1" is about Year. Its type is object.
The column named "Col2" is about Division. Its type is float64.
The column named "Col3" is about League. Its type is object.
The column named "Col4" is about Reg. Season. Its type is object.
The column named "Col5" is about Playoffs. Its type is object.
The column named "Col6" is about National Cup. Its type is object.

The column named "Col1" has the following unique values: ['1931', 'Spring 1932', 'Fall 1932', 'Spring 1933', '1933/34',
'1934/35', '1935/36', '1936/37', '1937/38', '1938/39', '1939/40', '1940/41', '1941/42', '1942/43', '1943/44', '1944/45',
'1945/46', '1946/47', '1947/48', '1948/49', '1949/50', '1950/51', '1951/52', '1952/53', '1953/54', '1954/55', '1955/56']
The column named "Col2" has the following unique values: ['1.0', nan]
The column named "Col3" has the following unique values: ['ASL']
The column named "Col4" has the following unique values: ['6th (Fall)', '5th?', '3rd', '?', '2nd', '1st', '5th, National',
'3rd(t), National', '4th, National', '4th', '6th', '9th', '5th', '1st(t)', '8th']
The column named "Col5" has the following unique values: ['No playoff', '?', 'Champion (no playoff)', 'Did not qualify', '1st
Round', 'Finals']
The column named "Col6" has the following unique values: [nan, '1st Round', 'Final', '?', 'Champion', 'Semifinals']

Here is a preview of the table:

          Col1  Col2 Col3            Col4                 Col5        Col6
0         1931   1.0  ASL       6th (Fall)           No playoff         NaN
1   Spring 1932   1.0  ASL             5th?           No playoff   1st Round
2     Fall 1932   1.0  ASL              3rd           No playoff         NaN
3   Spring 1933   1.0  ASL                ?                    ?       Final
...
```

Figure 2. Prompt for Python Code Generation

Column Renaming: A common error we observed in the construction of Python code and SQL queries by LLMs is the use of non-existent columns, a particular case of hallucination. To address this issue, we implement a strategy to rename columns to temporary placeholders, such as Col1, Col2, Col3, . . . ColN. This forces language models to utilize columns based on the description of the data type they contain, rather than their names.

Table Indexing: The table is then indexed in a vector database through its textual representation. The idea is that the summary generated by the LLM should be sufficient for the table to be retrieved, leveraging the same strategy used in text indexing, by transforming table descriptions into dense contextual vectors.

Retrieval, Re-ranking and Filtering: Table retrieval is performed via similarity measures, such as cosine similarity, comparing the vector representation of the user's query with the vector representations of table descriptions. After retrieval, we employ a re-ranking step by asking an LLM to reorder the retrieved tables by relevance, and a filtering step, by asking it to filter the results, eliminating retrieved tables that despite their similarity to the user's query cannot directly answer the question.

Code Generation: Next, we prompt an LLM to generate code to obtain the answer from the table, as exemplified in Figure 2. The code generation prompt was meticulously adjusted to avoid common pitfalls these models tend to encounter, such as attempting to perform operations on dataframe columns with missing values without first handling these cells. These adjustments were based on the outputs of GPT-4, for both SQL and Python code generation, and improved our accuracy in 16% for the same model, being the previously mentioned column renaming procedure one of the most beneficial changes we have made.

Response Generation: The generated code is then executed and the output is passed to another LLM, which is responsible for providing a natural language response to the user's question, considering the question, the data, and the output of the executed code.

4. Methodology

To evaluate the quality of our pipeline, we utilized two traditional benchmarks, called OTT-QA [Chen et al. 2020a], for the retrieval part, and WikiTableQuestions [Pasupat and Liang 2015], for the Q&A part. The tables from both datasets were sourced from Wikipedia.

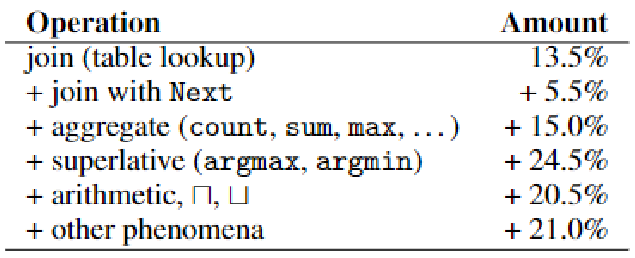
While the WikiTableQuestions dataset is ideal for our purpose of testing code generation for table manipulation, it is not ideal for evaluating the efficiency of our table retrieval module. This is because the questions are constructed in a closed-domain fashion, meaning they can only be answered if it is already known which table they refer to.[1] Therefore, when using the benchmark to evaluate our Q&A capabilities, we provide the language model with the correct table. To assess the quality of our retrieval and re-ranking modules, we will be testing them against the OTT-QA benchmark.

The OTT-QA benchmark is composed of questions based on those from another dataset, HybridQA [Chen et al. 2020b], but they were adapted to become "decontextualized," making them open-domain and therefore suitable for testing retrieval systems. It contains 400K+ tables to retrieve from and 45K human-annotated questions. We use all the 2,214 questions in the development partition of the benchmark to test retrieval and re-ranking. We use HITS@K to measure performance, where K ranges from 1 to 5, and means whether the correct table for each question is in the top-K retrieved or re-ranked tables. Re-ranking is done after the retrieval step, and we compare results with and without re-ranking.

The WikiTableQuestions benchmark contains 2,108 semi-structured tables and 22,033 complex question-answer pairs. The questions and answers were constructed by humans through crowdsourcing, with instructions to create questions that involve various types of operations required to answer them, as shown in the distribution in Figure 3.

Due to operational constraints involving LLMs costs, all tests in the WikiTableQuestions were performed on a sample of 200 questions from the test dataset. As noted, several adjustments were made to the LLM instructions and to the table indexing method to facilitate the inference of correct answers and the inference of code that did not generate exceptions. All the adjustments were based solely on the results observed for the train

[1]For example, in the question "what is the first city sorted alphabetically?", there is no indication of which table should be found–cities of what country? in what state? in what period of time?–, and it can only be correctly answered if the table to which the question refers has already been identified.



Operation	Amount
join (table lookup)	13.5%
+ join with `Next`	+ 5.5%
+ aggregate (`count`, `sum`, `max`, ...)	+ 15.0%
+ superlative (`argmax`, `argmin`)	+ 24.5%
+ arithmetic, ⊓, ⊔	+ 20.5%
+ other phenomena	+ 21.0%

Figure 3. Operations required to answer a sample of 200 questions from WikiTableQuestions. Source: [Pasupat and Liang 2015].

dataset questions, thereby preventing any information leakage from test to train.

The answers in WikiTableQuestions are provided as lists of size greater than or equal to 1, with sizes greater than 1 when more than one term is expected in the predicted answer for it to be considered correct. Thus, we used accuracy to measure the performance of our Q&A module, normalizing the strings in the dataset and the strings inferred by the models, to check if all the expected terms are present in the predicted answer.[2]

In addition to accuracy, we also considered the number of generated codes that produced any runtime exceptions and the time taken to answer each question. In other words, we are also testing the ability of these models to generate functional and correct code in the shortest possible time.

The scenarios in which we tested our pipeline are as follows:

- For generating table descriptions, re-ranking, and the final natural language response, we consistently used the GPT 3.5 Turbo model from OpenAI, due to its good performance and low cost.
- For generating dense vector representations of each table descriptions, we use embeddings generated by a model based on XLM-RoBERTa [Conneau et al. 2019] fine-tuned for information retrieval [Wang et al. 2024], which are then indexed in the Elasticsearch platform[3] and are retrieved using the cosine similarity strategy.
- For generating table manipulation code, we tested:
  - Closed-source models, namely GPT 3.5 Turbo and GPT 4 [OpenAI et al. 2024], accessed from OpenAI API;
  - Open-source models available from the GPT4All hub [Anand et al. 2023], namely LLaMa 3 8B Instruct [Dubey et al. 2024], Nous Hermes 2 Mistral DPO 7B [Jiang et al. 2023], Falcon 7B [Almazrouei et al. 2023], and GPT4All Snoozy 13B [Anand et al. 2023]. We used a single V100 32 GB GPU to infer responses with these models.
- All code generation models were tested by generating SQL queries (executed using the Python library `sqldf`) and code for dataframe manipulation (executed using the Python library `pandas`).

[2]We use accuracy as defined in [Mallen et al. 2022], since other metrics could be harder to be applied due to the nature of generative models answers, which tend to have a lot more words than the needed terms.

[3]`https://www.elastic.co/`

5. Results

Figure 4 shows the performance of the retrieval and the re-ranking modules in TableRAG pipeline to find the most suitable tables to answer questions from OTT-QA. Re-ranking is applied on top of the retrieved tables, and increases the results of retrieval in 3.7 points in HITS@1, when the table ranked first is the correct one. Re-ranking is still better than just retrieval when looking at HITS@2, but after that, just retrieval becomes better than when re-ranking is applied. When looking at HITS@5, the results from retrieval are 1.4 point better than those with re-ranking. This decrease happens mainly because of lower-ranked tables that the LLM judged that needed to be filtered out of the list during re-ranking, when they were in fact the correct ones.

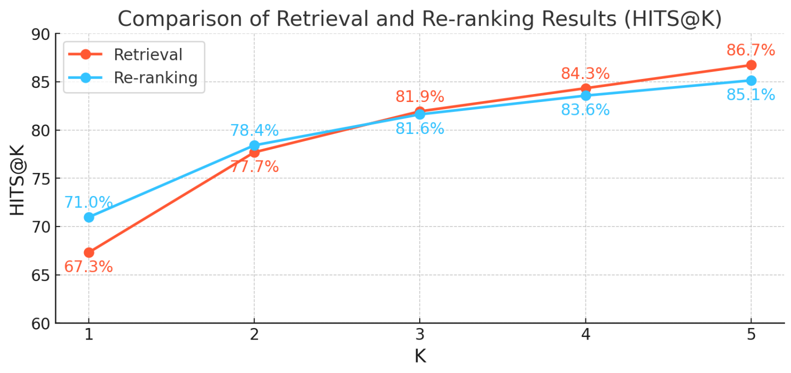


Figure 4. Comparison of Retrieval and Re-ranking Results over OTT-QA

Figure 5 shows the performance of the different models tested for generating Python code or SQL queries to answer the WikiTableQuestions. The yellow bars represent the accuracy value, while the orange ones represent the number of codes that produced exception at runtime, both in % of questions. The green line represents the average time in seconds taken to answer each question.

The best results come from OpenAI's closed-source models, both GPT-4 and GPT-3.5. The disparity between the best closed-source model and the best open-source model can be easily explained, among other factors, by the size of the models: while Nous Hermes 2 has 7 billion parameters in its neural architecture and achieved 40% accuracy, GPT-4 is speculated to have over 1 trillion[4], reaching 74%.

The difference in results between generating SQL and Python code for closed-source models is relatively small (74% in Python versus 72.5% in SQL for GPT-4). However, the difference in time is striking: while GPT-4 took an average of 11.5 seconds to generate Python code per question, generating SQL code was much faster, averaging 2.8 seconds. This is due to the fact that more preprocessing is required to construct functional Python code, and the larger the code, the longer the inference time. For open-source models, in general, better results are obtained when generating SQL code. Nous Hermes 2 achieved 40% accuracy building SQL queries and only 20% building Python codes, while the better open-source Python code generator, LLaMA, obtained only 26%.

[4]Sources suggest that GPT-4 could be a mixture of several models that, together, total 1.76 trillion parameters (`https://en.wikipedia.org/wiki/GPT-4`).

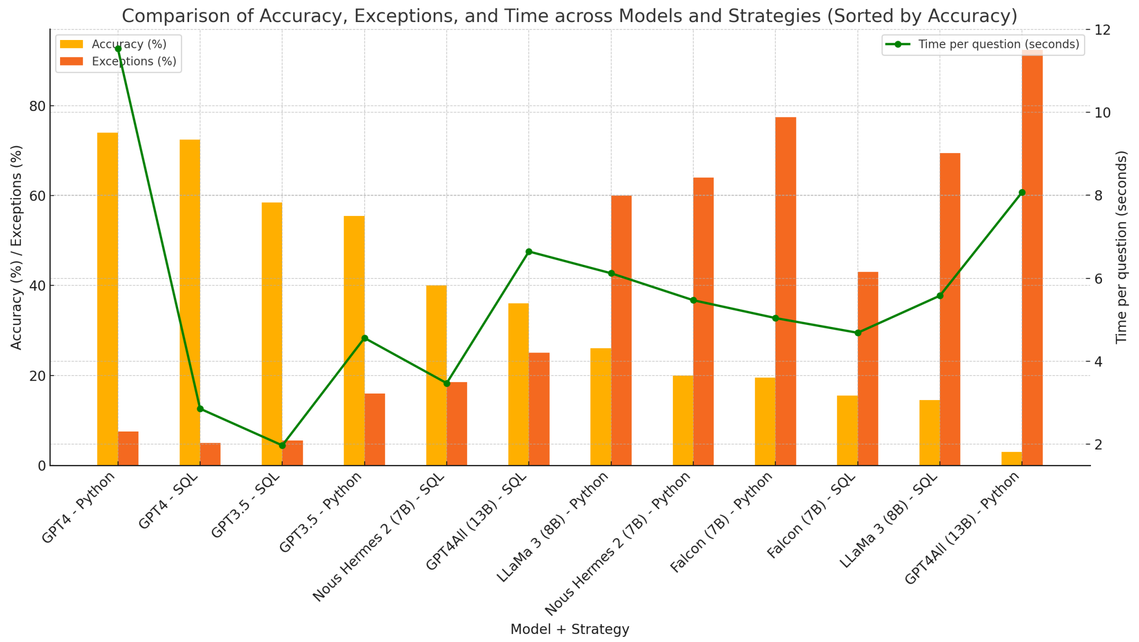


Figure 5. Comparison of LLMs Results in Code Generation for Q&A over WikiTableQuestions

6. Conclusions

We presented TableRAG, a pipeline for integrating tabular data into traditional Retrieval-Augmented Generation systems. The pipeline consists of obtaining textual representations of tables, indexing and retrieving them as dense contextual vectors, generating SQL or Python code for table manipulation, and generating a candidate answer in natural language. With the described pipeline, we achieved results of up to 86.7% HITS@5 in retrieval and 74% accuracy in Q&A using GPT-4.

Some limitations of this work should be considered. Due to computational cost and processing time, the results for Q&A were obtained using a sample of 200 questions from the WikiTableQuestions test set, making it difficult to compare our results with those of other works using the same benchmark. Additionally, the instructions provided to the language models for code generation were adjusted based on the outputs of the GPT-4 model, as it was the best-performing model, but they could also have been adjusted considering the most frequent errors of each of the other models individually. The open-source models did not undergo any fine-tuning process, which could significantly improve their results, and we did not test open-source models larger than 13B parameters. Finally, we did not perform any preprocessing on the tables to make them more easily interpretable, such as the table transposition procedure performed by [Liu et al. 2024], which yielded the state-of-the-art with 73.6% accuracy using GPT-3.5.

Acknowledgments

The work was carried out with assistance granted by the National Agency of Petroleum, Natural Gas and Biofuels (ANP), Brazil, associated with the investment of resources originating from the R,D&I Clauses, through the Cooperation Agreement between Petrobras and PUC-Rio.

References

[Abraham et al. 2022] Abraham, A. N., Rahman, F., and Kaur, D. (2022). Tablequery: Querying tabular data with natural language. *arXiv preprint arXiv:2202.00454*.

[Almazrouei et al. 2023] Almazrouei, E., Alobeidli, H., Alshamsi, A., Cappelli, A., Cojocaru, R., Debbah, M., Goffinet, E., Heslow, D., Launay, J., Malartic, Q., Noune, B., Pannier, B., and Penedo, G. (2023). Falcon-40B: an open large language model with state-of-the-art performance.

[Anand et al. 2023] Anand, Y., Nussbaum, Z., Duderstadt, B., Schmidt, B., and Mulyar, A. (2023). Gpt4all: Training an assistant-style chatbot with large scale data distillation from gpt-3.5-turbo. `https://github.com/nomic-ai/gpt4all`.

[Chen et al. 2020a] Chen, W., Chang, M.-W., Schlinger, E., Wang, W., and Cohen, W. W. (2020a). Open question answering over tables and text. *arXiv preprint arXiv:2010.10439*.

[Chen et al. 2020b] Chen, W., Zha, H., Chen, Z., Xiong, W., Wang, H., and Wang, W. (2020b). Hybridqa: A dataset of multi-hop question answering over tabular and textual data. *arXiv preprint arXiv:2004.07347*.

[Conneau et al. 2019] Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2019). Unsupervised cross-lingual representation learning at scale. *CoRR*, abs/1911.02116.

[Dong et al. 2024] Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., Xia, H., Xu, J., Wu, Z., Chang, B., Sun, X., Li, L., and Sui, Z. (2024). A survey on in-context learning.

[Dubey et al. 2024] Dubey, A. et al. (2024). The llama 3 herd of models.

[Gao et al. 2023] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., and Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

[Jiang et al. 2023] Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. (2023). Mistral 7b.

[Kandpal et al. 2023] Kandpal, N., Deng, H., Roberts, A., Wallace, E., and Raffel, C. (2023). Large language models struggle to learn long-tail knowledge. In *International Conference on Machine Learning*, pages 15696–15707. PMLR.

[Liang et al. 2018] Liang, C., Norouzi, M., Berant, J., Le, Q. V., and Lao, N. (2018). Memory augmented policy optimization for program synthesis and semantic parsing. *Advances in Neural Information Processing Systems*, 31.

[Lin et al. 2023] Lin, X. V., Chen, X., Chen, M., Shi, W., Lomeli, M., James, R., Rodriguez, P., Kahn, J., Szilvasy, G., Lewis, M., et al. (2023). Ra-dit: Retrieval-augmented dual instruction tuning. *arXiv preprint arXiv:2310.01352*.

[Liu et al. 2024] Liu, T., Wang, F., and Chen, M. (2024). Rethinking tabular data understanding with large language models. In Duh, K., Gomez, H., and Bethard, S., editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long*

Papers), pages 450–482, Mexico City, Mexico. Association for Computational Linguistics.

[Mallen et al. 2022] Mallen, A., Asai, A., Zhong, V., Das, R., Khashabi, D., and Hajishirzi, H. (2022). When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511*.

[OpenAI et al. 2024] OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., et al. (2024). Gpt-4 technical report.

[Pasupat and Liang 2015] Pasupat, P. and Liang, P. (2015). Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

[Tonmoy et al. 2024] Tonmoy, S., Zaman, S., Jain, V., Rani, A., Rawte, V., Chadha, A., and Das, A. (2024). A comprehensive survey of hallucination mitigation techniques in large language models. *arXiv preprint arXiv:2401.01313*.

[Wang et al. 2024] Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., and Wei, F. (2024). Multilingual e5 text embeddings: A technical report. *arXiv preprint arXiv:2402.05672*.

[Yin et al. 2020] Yin, P., Neubig, G., Yih, W.-t., and Riedel, S. (2020). TaBERT: Pretraining for joint understanding of textual and tabular data. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online. Association for Computational Linguistics.

[Yu et al. 2018] Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.