Question Answering Over Tabular Data with DataBench: Evaluating A Large Language Model

Zuriel Gonzalez
gonzalez@union.edu

Abstract

Large Language Models (LLMs) have demonstrated remarkable performance in natural language understanding and reasoning, particularly in question answering from tabular data. In this paper, we evaluate the ChatGPT 4o-mini model with DataBench, a large-scale benchmark carefully crafted for tabular question answering. We compare three various approaches: zero-shot in-context model, where the model is merely presented with the dataset and question, zero-shot code-based model, where the model generates and executes Python code to get the answer, few-shot code-based model, which includes example question-answer pairs to enhance code generation accuracy. Our results indicate that the zero-shot code-based approach is significantly better than the in-context and few-shot code-based approaches. These results highlight the strength of the zero-shot code-based method for tabular reasoning and point toward exploring further evaluation of LLMs.

1 Introduction

In recent years, the rapid advancement of Large Language Models (LLMs) has transformed natural language processing (NLP) and has shown significant improvement in handling complex tasks such as sentiment analysis, machine translation, and, in particular, their ability to reason and answer questions from tabular data. Despite these advances, a key challenge remains in evaluating and comparing specific capacities of LLMs. DataBench (Grijalba et al., 2024), a large benchmark for evaluating LLMs as tabular reasoners, has been proposed as a benchmark for evaluating and comparing LLMs. In this research, we leveraged the recently introduced DataBench dataset to conduct a deeper empirical evaluation of LLMs' ability to interpret and reason with real-world tabular data to see: how effectively can LLMs reason over and answer questions about structured tabular data, including generating data manipulation code? We aim to assess how well LLMs generate accurate answers and write effective data manipulation code when queried about complex tabular datasets. In this research, we will be focusing on the ChatGPT 4o-mini model. The results show that the model has shown significant improvements over past ChatGPT models to be used on tabular data however, there is still room for improvement for all types of questions.

1.1 DataBench

DataBench is a benchmark designed to assess the performance of LLMs in answering questions over tabular data. The benchmark consists of 65 real-world datasets and 1,300 questions. The datasets represent a wide range of real-world scenarios, including finance, healthcare, and retail, and vary in size and complexity. Figure 1 presents an overview of the datasets. The expected output in DataBench follows a specific format depending on the type of question asked. The possible output types include: a boolean, a categorical label represented as a string, an integer, a list of categories, strings, or integers.

2 Related Work

This study builds upon the foundational work in table question answering (QA) and the development of benchmarks for evaluating the performances of LLMs. In the past, studies have explored different methods for tackling this task, like using semantic parsing to turn natural language queries into formal language like SQL to extract data from databases. Prior to DataBench, existing benchmarks such as WikiTableQuestions, mainly focused on Wikipedia tables, did not fully capture the complexity of real-world tables. Grijalba et al. completed an extensive evaluation on both open-source and closed-source LLMs (ChatGPT 3.5, Llama -2-7b, Llama-2-13b) using Databench. Their study examined two pri-

```
def zeroshot_prompt(self, dataset_csv, question):
    """Constructs the prompt for zero-shot in-context learning."""
    return f"""
    You are an AI assistant that answers questions based on tabular data.

    Here is the dataset: {dataset_csv}
    Answer the following question based on this data:
    Question: "{question}

    Provide your answer in JSON format with these keys:
    - "answer": The best possible answer based on the dataset.
    - "columns_used": List of relevant columns for answering the question.
    - "explanation": A short reasoning behind your answer.

    Only return the JSON object. Do not include any other text.
    Ensure the answer must be one of these type: boolean, category, number, List[category], List[number]

    """
```

Figure 2: TabularQAAgent (Zero-Shot Prompting)

	Name	Rows	Cols	Domain
1	Forbes	2668	17	Business
2	Titanic	887	8	Travel and Locations
3	Love	373	35	Social Networks and Surveys
4	Taxi	100000	20	Travel and Locations
5	NYC Calls	100000	46	Business
6	London Airbnbs	75241	74	Travel and Locations
7	Fifa	14620	59	Sports and Entertainment
8	Tornados	67558	14	Health
9	Central Park	56245	6	Travel and Locations
10	ECommerce Reviews	23486	10	Business
11	SF Police	713107	35	Social Networks and Surveys
12	Heart Failure	918	12	Health
13	Roller Coasters	1087	56	Sports and Entertainment
14	Airbnb Madrid	20776	75	Travel and Locations
15	Food Names Embeddings	906	4	Business
16	Holiday Package Sales	4888	20	Travel and Locations
17	Hacker News	9429	20	Social Networks and Surveys
18	Staff Satisfaction	14999	11	Business
19	Aircraft Accidents	23519	23	Health
20	Real Estate Madrid	26026	59	Business
21	Telco Customer Churn	7043	21	Business
22	Airbnbs Listings NY	37012	33	Travel and Locations
23	Madrid Climate	36858	26	Travel and Locations
24	Salary Survey Spain 2018	216726	29	Business
25	Data Driven SEO	62	5	Business
26	Predicting Wine Quality	1599	12	Business
27	Supermarket Sales	1000	17	Business
28	Predict Diabetes	768	9	Health
29	NYTimes World In 2021	52588	5	Travel and Locations
30	Professionals Kaggle Survey	19169	64	Business
31	TrustPilot Reviews	8020	6	Business
32	Delicatessen Customers	2240	29	Business
33	Employee Attrition	14999	11	Business
34	World Happiness Report 2020	153	20	Social Networks and Surveys
35	Billboard Lyrics	5100	6	Sports and Entertainment
36	US Migrations 2012-2016	288300	9	Social Networks and Surveys
37	Ted Talks	4005	19	Social Networks and Surveys
38	Stroke Likelihood	5110	12	Health
39	Happy Moments	100535	11	Social Networks and Surveys
40	Speed Dating	8378	123	Social Networks and Surveys
41	Airline Mentions X	14640	15	Social Networks and Surveys
42	Predict Student Performance	395	33	Business
43	Loan Defaults	83656	20	Business
44	IMDb Movies	85855	22	Sports and Entertainment
45	Spotify Songs	21000	19	Sports and Entertainment
46	120 Years Olympics	271116	15	Sports and Entertainment
47	Bank Customer Churn	7088	15	Business
48	Data Science Salary Data	742	28	Business
49	Boris Johnson UK PM Tweets	3220	34	Social Networks and Surveys
50	ING 2019 X Mentions	7244	22	Social Networks and Surveys
51	Pokemon Feature Correlation	1072	13	Business
52	Professional Map	1227	12	Business
53	Google Patents	9999	20	Business
54	Joe Biden Tweets	491	34	Social Networks and Surveys
55	German Loans	1000	18	Business
56	Emoji Diet	58	35	Health
57	Spain Survey 2015	20000	45	Social Networks and Surveys
58	US Polls 2020	3523	52	Social Networks and Surveys
59	Second Hand Cars	50000	21	Business
60	Bakery Purchases	20507	5	Business
61	Disneyland Customer Reviews	42656	6	Travel and Locations
62	Trump Tweets	15039	20	Social Networks and Surveys
63	Influencers	1039	14	Social Networks and Surveys
64	Clustering Zoo Animals	101	18	Health
65	RFM Analysis	541909	8	Business

Figure 1: The 65 datasets included in DataBench

mary prompting strategies: an in-context learning prompt, where the entire dataset is included in the prompt along with the question, and a code prompt, where the LLM is given an incomplete function containing the question and dataset metadata, instructing it to generate executable code to extract the answer. The results highlighted significant performance gaps between different model types and showed that even the strongest models struggled with seemingly simple tasks, such as boolean questions or queries involving only a single column. For this study, we used DataBench to evaluate a newer LLM model, ChatGPT 4o-mini. Additionally, we investigated the effectiveness of few-shot prompting in enhancing the model's ability to reason over tabular data and generate code that extracted accurate answers. It is important to note that the evaluation conducted by the previous paper utilized a reduced version of DataBench, which had a smaller sample with the first 20 rows and was called **DataBench_lite**.

3 Approach

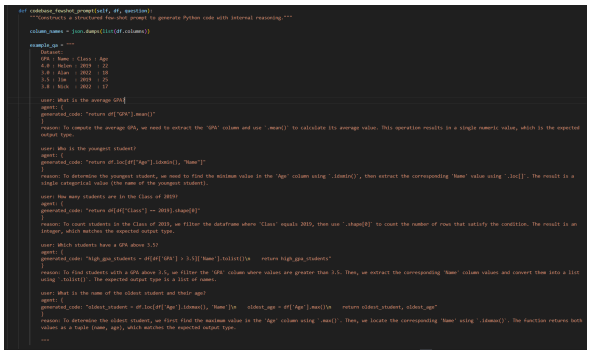
Our approach to question answering over tabular data involved three distinct agents, each employing a different strategy for interpreting and answering queries. The TabularQAAgent used zero-shot prompting, in-context learning, where the model directly analyzes a dataset within the prompt to generate an answer. The CodeBasedQAAgent also used zero-shot prompting, but with a code generation approach, prompting the model to produce and execute Python code to derive the answer. Lastly, the FewShotCodeBasedAgent was also a code generation agent but with few-shot prompting, incorporating example question-answer pairs with corresponding code to improve response accuracy.

3.1 Zero-shot In-Context Model

The TabularQAAgent utilized a zero-shot in-context learning approach, where the agent was provided with a sample dataset in CSV format

```
def codebase_prompt(self, df, question):
    """Constructs a structured prompt to generate Python code"""

    column_names = json.dumps(list(df.columns))

    return f"""
    You are an AI assistant that answers questions about tabular data by completing a Python function.

    Task
    You must complete the function below to return the answer
    Only generate the necessary query or computation.
    Try to generate only one line of code that could provide the answer
    Multi-line code should not have extra spaces at the start of lines.
    Ensure that if we're to run the code, the answer must be one of these type: boolean, category, number, list[category], List[number]

    Provided Information
    The dataset is given as a Pandas DataFrame named `df`.
    The dataset has the following column names: {column_names}
    Column names have already been assigned inside the function.
    Do not include df.columns = ... in your response.

    Function Definition:
    import pandas as pd
    import numpy as np

    def answer(df: pd.DataFrame):
        \"\"\"Returns: {question}\"\"\"
        df.columns = {column_names}

        # Your code starts here:
    """
```

Figure 3: CodeBasedQAAgent (Zero-Shot Prompting)



```
def codebase_fewshot_prompt(self, df, question):
    """Constructs a structured few-shot prompt to generate Python code with internal reasoning."""

    column_names = json.dumps(list(df.columns))

    example_qa = """
        Dataset:
        GPA : Name : Class : Age
        4.0 : Helen : 2019 : 22
        3.0 : Alan : 2022 : 18
        3.5 : Tim : 2019 : 25
        3.0 : Nick : 2022 : 17

        user: What is the average GPA?
        agent: {
        generated_code: "return df["GPA"].mean()"
        }
        reason: To compute the average GPA, we need to extract the 'GPA' column and use `.mean()` to calculate its average value. This operation results in a single numeric value, which is the expected output type.

        user: Who is the youngest student?
        agent: {
        generated_code: "return df.loc[df["Age"].idxmin(), "Name"]"
        }
        reason: To determine the youngest student, we need to find the minimum value in the 'Age' column using `.idxmin()`, then extract the corresponding 'Name' value using `.loc[]`. The result is a single categorical value (the name of the youngest student).

        user: How many students are in the Class of 2019?
        agent: {
        generated_code: "return df[df["Class"] == 2019].shape[0]"
        }
        reason: To count students in the Class of 2019, we filter the dataframe where 'Class' equals 2019, then use `.shape[0]` to count the number of rows that satisfy the condition. The result is an integer, which matches the expected output type.

        user: Which students have a GPA above 3.5?
        agent: {
        generated_code: "high_gpa_students = df[df['GPA'] > 3.5]['Name'].tolist()\n    return high_gpa_students"
        }
        reason: To find students with a GPA above 3.5, we filter the 'GPA' column where values are greater than 3.5. Then, we extract the corresponding 'Name' column values and convert them into a list using `.tolist()`. The expected output type is a list of names.

        user: What is the name of the oldest student and their age?
        agent: {
        generated_code: "oldest_student = df.loc[df['Age'].idxmax(), 'Name']\n    oldest_age = df['Age'].max()\n    return oldest_student, oldest_age"
        }
        reason: To determine the oldest student, we first find the maximum value in the 'Age' column using `.max()`. Then, we locate the corresponding 'Name' using `.idxmax()`. The function returns both values as a tuple (name, age), which matches the expected output type.
        """
```

Figure 4: FewShotCodeBase (Few-shot Code Generation Prompt)

along with the user's question. The agent constructs a structured prompt that instructs the model to extract information directly from the dataset and generate a JSON-formatted response containing: the answer, the columns used, and an explanation for the answer.

3.2 Zero-shot Code-based Model

The CodeBasedQAAgent followed a code generation approach, also utilizing zero-shot prompting, where the agent generated Python code to compute the answer based on the given dataset. The agent constructed a prompt that provided the dataset's column names and the users' questions, instructing the model to generate a single line of code that would return the answer. The generated code is executed, and the output is returned as the final answer.

3.3 Few-shot Code-based Model

The FewShotCodeBase agent also used a code generation approach, however, it utilized few-shot prompting, where the agent was provided with example question-answer pairs and Python code to compute the answer. The agent generated code based on similar examples and applied internal reasoning to the problem. The prompt is structured to include examples, where the model is encouraged to replicate the style of generated code and the reasoning process for the current question.

4 Experiment

To evaluate the performance of each agent on the Databench benchmark, we followed the same evaluation methodology as outlined in the Grijalba et al. paper to ensure a consistent evaluation . For each model, we processed a set of tabular QA tasks and compared the predicted answers to the correct answers. The evaluation measured the overall accuracy and the accuracy per question type. Our evaluation relied on a standardized comparison function that handled various response formats, including booleans, categorical values, numerical values, and lists, the same evaluation as outlined in the Grijalba et al. study. This function also incorporated preprocessing techniques such as string normalization, numerical rounding, and date parsing to ensure a fair comparison. Errors from model failures (e.g., parsing errors or invalid executions) were counted as incorrect predictions. It is important to note that for this study, we utilized all of the datasets in Databench, while Grijalba et al. used **DataBench_lite**.

5 Results

Prompt,Model	Average
Zero-shot In-Context Model	20.41%
Zero-shot Code-based Model	72.78%
Few-shot Code-based Model	20.18%

Table 1: Average Accuracy for each model

Prompt, Model	**Boolean**	**Category**	**Number**	**List[Category]**	**List[Num]**
Zero-shot In-Context	50.00%	34.60%	7.69%	7.28%	2.29%
Zero-shot Code-based	75.19%	79.85%	80.77%	65.52%	62.60%
Few-shot Code-based	48.85%	34.22%	7.69%	7.66%	2.29%

Table 2: Average Accuracy for each type of question

Table 1 provides the overall accuracy of the models that were measured, with Table 2 presenting a breakdown of performance by question type. The highest average accuracy belonged to the zero-shot code-based model at 72.78%, which is significantly higher than the zero-shot in-context model and the few-shot code-based model. On a breakdown by

question type, the zero-shot code-based model performed extremely well on boolean (75.19%), categorical (79.85%), and numerical (80.77%) questions. However, its performance suffered on categorical (65. 52%) and numerical (62. 60%) questions based on lists, with relative weakness in more complex answer forms. Both zero-shot in-context and few-shot code-based models had low accuracy in all question types throughout, particularly with weak performance on list-based questions (<8%) and numerical reasoning tasks (<8%).

6 Discussion

The results highlight a clear performance gap between code-based approaches and in-context learning approaches in tabular QA tasks. Zero-shot code-based model outperforms the other two models by a wide margin, demonstrating that direct execution of generated code is a viable way of answering structured questions. The few-shot code-based approach was not a significant improvement over the zero-shot code-based model, suggesting that few-shot examples could be insufficient for instruction models on how to reason over tables. This is an indication that simply providing examples does not equip models with the necessary logic to perform operations on tables such as aggregation, filtering, or numerical computations. Despite the overall better performance, the code-based model still did not perform as well in list-based numeric and list category types of questions. This suggest the potential challenge in handling multi-step operations or in correctly parsing properly formatted results. Addressing these issues could improve the overall handling of errors in generated code.

7 Conclusion

In this paper, we evaluated ChatGPT 4o-mini using DataBench, a question-answering benchmark over tabular data using different approaches to evaluate the performance of the ChatGPT 4o-mini model. The results demonstrate that the zero-shot code-based approach performed considerably better than the zero-shot in-context approach and the few-shot code-based approach. The results highlight the limitation of the in-context approach for structured data reasoning, where the averages consistently remained low. In contrast, code generation and execution of Python code enable the model to perform explicit computations, logical operations, and handling of structured data more effectively. However, list-type output handling remains an issue, showing potential for improvement in code optimization and structured response shaping. Additionally, the few-shot code-based model did not perform better as we expected. This suggests that few-shot prompting alone does not enhance the performance in code writing for tabular QA, however, further testing is required to see if few-shot prompting could have an impact on the performance of LLM models. While this study focused on ChatGPT 4o-mini, other models can still be evaluated through the use of DataBench and can utilize different approaches to QA over tabular data. In future work, further experimentation will be needed to evaluate more LLMs.

8 Appendix

For this project, I was worked with the evaluation process with the code provided in the previous paper to evaluate out models. I wrote code for evaluating each category of Databench output, I did not modify any of the provided code. My teammates were responsible for writing the prompts for each model and writing code for the different agents. Everyone contributed to the group reports and presentations. Everyone gave their best efforts and worked on the project. There were no issues during this project. the project was implemented using Python and concepts from previous courses were important when writing and designing the code for the project. I gained a better understanding of the amount of effort that goes into developing LLMs and AI models. I also learned more about how LLMs could be evaluated. Previous to this course, I had not written any reports related to coding, this was my first time writing reports like this and it is an important skill to know. If I were to do the project again, I would include other agents to evaluate the ChatGPT 4o-mini model, and also, it would also be good to evaluate other LLM models to see how each compares to the other. I would do the same evaluation, using DataBench and similar approaches as we did in this project.

References

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.