Howard University-AI4PC at SemEval-2025 Task 8: DeepTabCoder - Code-based Retrieval and In-context Learning for Question-Answering over Tabular Data

Saharsha Tiwari and **Saurav K. Aryal**
EECS, Howard University
Washington, DC 20059, USA
https://howard.edu/
saharsha.tiwari@bison.howard.edu and saurav.aryal@howard.edu

Abstract

Question answering over tabular data requires models to understand diverse table structures and accurately reason over structured information. To address these challenges, we introduce DeepTabCoder, our approach to SemEval 2025 - Task 8: DataBench. We combine a code-based retrieval system with in-context learning to generate and execute Python code for answering questions, leveraging DeepSeek-V3 for code generation. DeepTabCoder outperforms the competition baseline, achieving accuracies of 81.42% on the DataBench dataset and 80.46% on the DataBench Lite dataset. These results demonstrate the potential of in-context learning with code execution methods for improving table reasoning tasks.

1 Introduction

Recent advances in large language models (LLMs) have significantly improved open-domain question answering; however, question answering over structured tabular data remains a challenging problem. Unlike text-based QA, tabular QA requires the model to understand schema structures, handle a variety of data types, and perform logical and numerical operations over cell values. Additionally, models must operate without external knowledge sources, relying solely on the information contained within the tables themselves.

SemEval 2025 Task 8: DataBench (Grijalba et al., 2024) focuses on question-answering over tabular data, introducing a large-scale benchmark designed to evaluate how well models extract and reason over structured information. The dataset includes 65 diverse tables from multiple domains, each varying in size, structure, and data types, making it a comprehensive test for tabular reasoning. Accompanying these datasets are 1,300 manually curated questions, covering different answer types: Boolean (True/False), categorical values, numerical values, and lists. The competition itself features a subset of 15 datasets and 522 questions, providing a focused yet challenging evaluation setting. Unlike open-domain QA, models must (1) derive answers solely from the provided tables, (2) handle heterogeneous table schemas, and (3) execute complex queries without relying on external knowledge. The task is further divided into two categories: DataBench, which uses the full datasets, and DataBench Lite, which provides a smaller 20-row sample for each dataset, testing a model's ability to generalize with limited data.

DeepTabCoder follows a three-step approach leveraging in-context learning to tailor prompts for each dataset. First, we generate dataset-specific prompts that include metadata and relevant schema details to guide the model. Second, we inject the question into the dataset-specific prompt and infer the response from the model. Third, we extract the Python code generated by the model and execute it against the dataset to retrieve the answer. By maintaining modular and reusable functions, our method ensures flexibility across different tabular structures. We extend and modify the approach from *Tool-Augmented Reasoning Framework for Tables (TART)* (Lu et al., 2024), which integrates LLMs with specialized tools to enhance table understanding and numerical reasoning. TART consists of three key components: a table formatter for accurate data representation, a tool maker for constructing computational tools, and an explanation generator for interpretability. We adapt TART's methodology to better align with the specific requirements of the DataBench competition, focusing on structured table reasoning.

Our system combines fixed schema templates with code execution to handle diverse table structures. The system shows particular strength in Boolean reasoning and numerical queries, though challenges remain in complex aggregation tasks requiring multi-hop reasoning. Detailed implementation and results analysis are presented in subsequent sections.

Proceedings of the The 19th International Workshop on Semantic Evaluation (SemEval-2025), pages 1702–1708

2 Related Works

In this section, we review prior research across four key areas that form the basis of DeepTabCoder. For each area, we explain how our work extends existing methods, with special emphasis on our modification to the TART framework.

2.1 Tabular Question Answering

Early work in tabular question answering, such as TAPAS (Herzig et al., 2020) and TaBERT (Yin et al., 2020), converts tables into textual representations to apply semantic parsing and reasoning. These approaches focus on leveraging pre-trained language models to interpret table data. In contrast, DeepTabCoder embeds dataset-specific metadata and schema details directly into the prompt. This design ensures that each table's inherent structure is maintained without exposing full table details, allowing our model to reason more effectively over heterogeneous data formats, as required by the DataBench challenge (Grijalba et al., 2024).

2.2 Code-based Retrieval and Execution Models

Recent work has demonstrated two complementary approaches to code-based table reasoning:

- **Pre-training with synthetic executions**: TAPEX (Liu et al., 2022) introduced table-aware pre-training by exposing the model to 26 million synthetic (SQL query, execution result) pairs. This approach enhances structural reasoning through the learning of SQL execution patterns, though it necessitates expensive, task-specific pre-training instead of relying on general code understanding.
- **Prompt-time decomposition**: DIN-SQL (Pourreza and Rafiei, 2023) showed that decomposing text-to-SQL tasks into subproblems—such as schema linking and classification—can significantly boost the few-shot performance of large language models.

In this work, we integrate these insights through dataset-aware schema prompting combined with code execution.

2.3 In-Context Learning and Prompt Engineering

The performance of large language models is significantly enhanced by in-context learning, as evidenced by works like GPT-3 (Brown et al., 2020) and chain-of-thought prompting (Wei et al., 2023). Typical strategies involve designing generic prompts that guide the model's reasoning. DeepTabCoder extends these strategies by incorporating tailored prompts enriched with structured metadata and function definitions. This targeted prompt engineering enables the model to concentrate on the essential schema characteristics of each dataset without revealing complete table representations, thereby reducing token usage and facilitating more precise code synthesis for query resolution.

2.4 Tool-Augmented Reasoning Frameworks for Tables

The TART [1] framework (Lu et al., 2024) has been influential in integrating external computational tools into table reasoning, combining table formatting, tool creation, and explanation generation. We build upon this foundation by creating dataset-specific metadata and injecting it into the prompt. DeepTabCoder leverages DeepSeek-V3 (DeepSeek-AI et al., 2025), a state-of-the-art Mixture-of-Experts (MoE) language model with a total of 671 billion parameters, of which 37 billion are activated per token for code generation and execution, borrowing TART's capabilities to better handle the nuances of the DataBench task.

3 System Overview

This section presents DeepTabCoder's system architecture and methodology for generating domain-specific prompts that enable models to synthesize Python programs to answer user queries $\mathcal{Q}$ regarding datasets $\mathcal{D}$. The complete implementation of DeepTabCoder can be found in Appendix A.

3.1 In-Context Prompt Generation

We generate tailored prompts that encapsulate key schema characteristics of each dataset while minimizing verbosity. This approach empowers the model to produce precise Python code for query resolution without exposing full table representations.

3.2 Modular Function Definitions

To support dataset manipulation, we define the following modular functions:

- **Data Loading:** `load_data` : $\mathcal{F} \rightarrow \mathcal{D}$, where $\mathcal{F}$ represents the file path space.

[1] https://github.com/XinyuanLu00/TART

- **Row Retrieval:** `get_row_by_name` : $\mathcal{D} \times \mathcal{K} \rightarrow \mathcal{V}$, where $\mathcal{K}$ is the keyspace and $\mathcal{V}$ is the value space.

3.3 Query-Conditioned Inference Pipeline

Given an input tuple $(\mathcal{D}, q)$, DeepTabCoder's inference pipeline consists of the following key components:

3.3.1 Augmented Prompt Construction

We construct an augmented prompt that concatenates structured schema features and the user query:

$$\mathcal{P}(\mathcal{D}, q) = \underbrace{f(\mathcal{D})}_{\text{Schema Features}} \oplus \underbrace{q}_{\text{Query}} \quad (1)$$

where $\oplus$ denotes the concatenation operator.

3.3.2 Inference with Query Injection

DeepTabCoder integrates dataset-specific metadata with the query to ensure the model accurately interprets the task within the dataset context. The process involves:

- **Query Integration:** The dataset-aware prompt is formulated as shown in Equation 1. Here, $f(\mathcal{D})$ encapsulates structured metadata and function definitions extracted from $\mathcal{D}$, ensuring that all pertinent schema details are provided before the model processes the query.
- **LLM Inference:** The constructed prompt $\mathcal{P}(\mathcal{D}, q)$ is passed to the large language model (LLM) $\mathcal{M}$, which generates the corresponding Python code $\mathcal{C}$:

$$\mathcal{C} = \mathcal{M}(\mathcal{P}(\mathcal{D}, q)) \quad (2)$$

  The resulting code $\mathcal{C}$ is a syntactically and semantically structured function designed to compute the answer a based on $\mathcal{D}$.

3.4 Code Execution

After the Python code $\mathcal{C}$ is generated, it is executed and validated to ensure correctness in answering the query q over the dataset $\mathcal{D}$. Code generation is performed using DeepSeek-V3.

- **Code Execution:** The function $\mathcal{C}$ is executed in a controlled runtime environment to produce the answer:

$$a = \mathcal{C}(\mathcal{D}) \quad (3)$$

Figure 1: Overview of the proposed pipeline.

4 Experimental Setup

We use the DataBench and DataBench Lite datasets provided for SemEval 2025 Task 8. Each dataset-specific prompt is constructed by extracting schema metadata (column names and the first row), defining modular utility functions such as `load_data` and `get_row_by_name`, and appending the corresponding query q. The prompts are designed to minimize verbosity while providing sufficient context for code generation, as illustrated in Template A. This is a condensed version; full prompt examples are available in the project's GitHub repository.

During inference, the structured prompt is passed to DeepSeek-V3, which generates a Python code snippet intended to answer the query. The generated code is executed in a sandboxed Python environment to produce the final prediction. The same code $\mathcal{C}$ is used for both the full DataBench datasets and their Lite versions without regeneration.

Evaluation is performed using the `databench_eval`[2] package provided by the organizers. We report overall accuracy along with per-category accuracies across Boolean, Categorical, Numerical, List of categories, and List of numbers.

5 Results

In this section, we present the evaluation results for DeepTabCoder on two datasets: *DataBench* and *DataBench Lite*. We analyze the overall accuracy as well as the category-specific performance for both datasets using databench_eval package. The tables and accompanying analysis provide a detailed overview of the model's performance across different categories.

5.1 DataBench Accuracy Results

The overall accuracy of DeepTabCoder on the *DataBench* dataset is 80.27%. The accuracy values for each category within the dataset are presented in Table 1. From this, we can observe that the model

[2] https://github.com/jorses/databench_eval

excels in the *boolean* category with an accuracy of 86.05%, which is the highest among all categories.

Category	Accuracy
list[category]	0.7639
list[number]	0.7802
category	0.7703
boolean	0.8605
number	0.8013

Table 1: Accuracy results on the DataBench dataset.

As shown in Table 1, the model demonstrates robust performance across the categories, with particularly high accuracy in the *boolean* category.

5.2 DataBench Lite Accuracy Results

When evaluated on the *DataBench Lite* dataset, the overall accuracy of the DeepTabCoder drops slightly to 79.50%, as shown in Table 2. We reuse the same code $\mathcal{C}$ without any modifications for generating answers on the DataBench Lite dataset.

Category	Accuracy
list[category]	0.7222
list[number]	0.7802
category	0.7703
boolean	0.8682
number	0.7885

Table 2: Accuracy results on the DataBench Lite dataset.

Table 2 also shows that DeepTabCoder performs best on the *boolean* category for the Lite dataset, achieving an accuracy of 86.82%.

5.3 Baseline Comparison

The official competition leaderboard reports DeepTabCoder achieving 81.42% on DataBench and 80.46% on DataBench Lite, evaluated manually by task organizers. For comparison, the baseline model, stable-code-3b-GGUF, achieved 26% and 27% respectively. DeepTabCoder outperforms stable-code-3b-GGUF by a significant margin, achieving much higher accuracy across both datasets.

5.4 Evaluation of Accuracy vs. Task Complexity

We evaluated the accuracy of DeepTabCoder using DeepSeek-V3 against the complexity of tasks in both datasets. Figure 2 presents a visual representation of the model's accuracy on the different categories across both datasets.

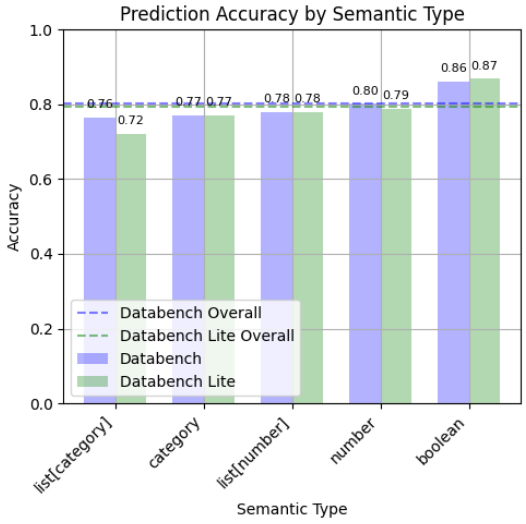

Figure 2: Accuracy of DeepSeek-V3 on different categories for the DataBench and DataBench Lite datasets.

The results highlight DeepTabCoder's strength in handling Boolean and numerical queries, where precision-based conditions and arithmetic computations are critical. The higher accuracy in these categories indicates that executing model-generated Python code offers an effective mechanism for addressing straightforward logical and statistical tasks. However, lower performance on `list[category]` and `category` outputs reveals the difficulty in generalizing over categorical aggregations, particularly when questions require selecting multiple elements or identifying non-unique patterns. In several failure cases, the model either missed entries when filtering a list or returned duplicate values instead of unique ones, suggesting an opportunity to improve aggregation handling.

Overall, while DeepTabCoder significantly outperforms the baseline by leveraging fixed schema templates and code execution, these observations emphasize the need for improvements in structured decoding, multi-hop reasoning, and aggregation strategies to further enhance performance on complex tabular reasoning tasks.

6 Limitations

Despite demonstrating strong performance in tabular question answering, DeepTabCoder still has several limitations that need to be addressed. One major limitation is the handling of complex queries. It struggles with queries that require multi-hop rea-

soning and advanced aggregation. While code execution helps with computation, the model sometimes generates incorrect logic or fails to retrieve the correct subset of data.

Another challenge arises with list-based output generation. As observed in our results, the model's accuracy on questions requiring a list of categories (e.g., `list[category]`) is significantly lower compared to other categories. This indicates difficulties in aggregating and structuring multiple categorical responses correctly.

The model faces challenges with code generation, as it is not always correct, leading to runtime errors. Although executing model-generated Python code improves precision, errors in syntax or logic occasionally occur, requiring additional checks.

7 Future Work

To address these issues, we propose several directions for future research. Enhancing multi-hop reasoning through explicit decomposition and intermediate verification could improve query resolution. For list-based outputs, structured decoding and refined prompt engineering may lead to better aggregation.

To improve code generation accuracy, future work should incorporate additional calls to smaller LLMs that verify and correct errors in the generated code after the initial output, thus enhancing execution reliability. Adaptive schema understanding could also be improved using schema-agnostic or meta-learning techniques. Furthermore, we plan to investigate domain-specific fine-tuning and sophisticated post-processing strategies to improve the handling of aggregation and multi-hop reasoning tasks. Given limited co-location of programming and other languages, special evaluations of multilingual (Aryal et al., 2023a; Aryal and Prioleau, 2023) and code-switched text (Aryal et al., 2023b,c, 2022) will also be considered, especially low-resource languages (Prioleau and Aryal, 2023; Aryal and Adhikari, 2023; Sapkota et al., 2023). These enhancements are expected to bolster the robustness and generalizability of DeepTabCoder in diverse real-world tabular reasoning scenarios

8 Conclusion

In this work, we presented DeepTabCoder to SemEval 2025 Task 8: DataBench, which combines code-based retrieval with in-context learning and dataset-specific prompt engineering for question answering over tabular data. By utilizing DeepSeek-V3 for Python code generation and execution, we achieved an improvement of approximately 3.13 times on the DataBench dataset (81.42% vs. 26%) and 2.98 times on the DataBench Lite dataset (80.46% vs. 27%) compared to the baseline.

While DeepTabCoder demonstrates strong performance in tasks such as Boolean reasoning and numerical queries, we observed challenges with tasks requiring multi-hop reasoning and list-based outputs, particularly in handling aggregation and multi-category responses. This highlights areas for further improvement, including enhanced multi-hop reasoning capabilities, more effective handling of complex aggregations, and improved code generation accuracy through additional error correction and debugging mechanisms.

Overall, our results suggest that DeepTabCoder, with future enhancements, has the potential to offer a robust solution for tabular question answering, addressing both schema diversity and complex query execution. Further work will focus on refining DeepTabCoder's capabilities through multi-hop reasoning enhancements, improved code generation accuracy, adaptive schema understanding, and domain-specific fine-tuning to ensure robustness and generalization across diverse real-world tabular reasoning tasks.

Acknowledgement

This research project was supported in part by the Office of Naval Research grant N00014-22-1-2714. The work is solely the responsibility of the authors and does not necessarily represent the official view of the Office of Naval Research.

References

Saurav Aryal and Howard Prioleau. 2023. Howard university computer science at semeval-2023 task 12: A 2-step system design for multilingual sentiment classification with language identification. In *Proceedings of the 17th International Workshop on Semantic Evaluation (SemEval-2023)*, pages 2153–2159.

Saurav K Aryal, Howard Prioleau, and Surakshya Aryal. 2023a. Sentiment analysis across multiple african languages: A current benchmark. *arXiv preprint arXiv:2310.14120*.

Saurav K Aryal, Howard Prioleau, Surakshya Aryal, and Gloria Washington. 2023b. Baselining performance

for multilingual codeswitching sentiment classification. *Journal of Computing Sciences in Colleges*, 39(3):337–346.

Saurav K Aryal, Howard Prioleau, and Gloria Washington. 2022. Sentiment classification of code-switched text using pre-trained multilingual embeddings and segmentation. *arXiv preprint arXiv:2210.16461*.

Saurav K Aryal, Howard Prioleau, Gloria Washington, and Legand Burge. 2023c. Evaluating ensembled transformers for multilingual code-switched sentiment analysis. In *2023 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 165–173. IEEE.

Saurav Keshari Aryal and Gaurav Adhikari. 2023. Evaluating impact of emoticons and pre-processing on sentiment classification of translated african tweets. *ICLR Tiny Papers*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2025. Deepseek-v3 technical report.

Jorge 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.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. Tapex: Table pre-training via learning a neural sql executor.

Xinyuan Lu, Liangming Pan, Yubo Ma, Preslav Nakov, and Min-Yen Kan. 2024. Tart: An open-source tool-augmented framework for explainable table-based reasoning.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction.

Howard Prioleau and Saurav K Aryal. 2023. Benchmarking current state-of-the-art transformer models on token level language identification and language pair identification. In *2023 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 193–199. IEEE.

Hrishav Sapkota, Saurav Keshari Aryal, and Howard Prioleau. 2023. Zero-shot classification reveals potential positive sentiment bias in african languages translations. *ICLR Tiny Papers*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data.

A Appendix

The code is available at https://github.com/2036saharsha/DeepTabCoder.

Prompt for 066_IBM_HR dataset

Task: Given a table and a question, write a Python program to answer the question.
Steps:

- Define modular, reusable functions that can be used for multiple questions.
- Create a main function `solution(table_data)` that processes the table and answers the query.
- Avoid hallucinating non-existent headers or table structures.
- Ensure no assumptions about missing or empty column headers.
- Keep the code clean, modular, and reusable across queries.

Dataset Schema

Field	Value
Age	41
Attrition	Yes
BusinessTravel	Travel_Rarely
...	...
YearsWithCurrManager	5

Question: What is the average job satisfaction for employees who have worked for more than 5 years?
Solution Example Code:

```
import pandas as pd
def load_data(file_path):
    df = pd.read_parquet(file_path)
    return df
def get_row_by_name(df, key):
    if key in df.columns:
        return df[key].iloc[0]
    return None
def solution(df):
    filtered_df = df[df['YearsAtCompany'] > 5]
   avg_job_satisfaction = filtered_df['JobSatisfaction']
                          .mean()
    return avg_job_satisfaction
df = load_data("./datasets/066_IBM_HR/all.parquet")
print(solution(df))
```

Answer: 2.755
Write a code for this question: [[QUESTION]]