YNU-HPCC at SemEval-2025 Task 8: Enhancing Question-Answering over Tabular Data with TableGPT2

Kaiwen Hu, Jin Wang and Xuejie Zhang
School of Information Science and Engineering
Yunnan University
Kunming, China
12024215001@stu.ynu.edu.cn, {wangjin, xjzhang}@ynu.edu.cn

Abstract

This paper describes our systems for SemEval 2025 Task8, Question Answering over Tabular Data. This task encourages us to develop a system that answers questions of the kind present in DataBench over day-to-day datasets, where the answer is either a number, a categorical value, a boolean value, or lists of several types. Participating in Task 8, we engage in all subtasks. The challenge lies in the multi-step reasoning process of converting natural language queries into executable code. This challenge is exacerbated by the limitations of current methods, such as chaining reasoning, which have difficulty handling complex multi-step reasoning paths due to difficulty evaluating intermediate steps. On the final competition test set, our DataBench accuracy is 65.64%, and DataBench Lite accuracy is 66.22%. Both exceed the baseline. The competitive results in two subtasks demonstrate the effectiveness of our system.[1]

1 Introduction

The rise of large language models (LLMs) has transformed research in natural language processing (NLP). Their ability to learn from little data has made them useful for tasks like summarization (Zhang et al., 2024a) , machine translation, and text sentiment analysis (Zhang et al., 2024b). This progress has been driven by the development of general-purpose LLMs and the discovery of their unexpected abilities. However, the rapid growth of these models hasn't been matched by high-quality benchmarks for comparing their performance.

Question answering (QA) has been a key NLP task focused on finding the best answer from unstructured text. The issue of structured knowledge grounding has been widely researched for years. Furthermore, tables, a common form of (semi)-structured data that stores world knowledge, have garnered considerable attention from the natural language processing (NLP) community. Traditionally, accessing the information inside the structured data (like tables) has depended on synthesizing executable languages like SQL or SPARQL. Still, these methods don't understand the meaning of text in the fields or allow natural language questions. Such challenges have led to interest in using LLMs to answer questions from structured or tabular data. Recently, the ability of LLMs to perform table question answering (Chen, 2023; Chen et al., 2020; Nan et al., 2021; Zhu et al., 2021) has emerged as a valuable skill. This highlights the need for a reliable benchmark to evaluate LLM performance.

Our contributions can be summarized as follows: We propose an approach exploring using the TableGPT2-7B (Su et al., 2024) model to solve the corresponding subtasks. The model develops a tabular encoder that processes the entire table, generating compact embeddings for each column.

The rest of this paper is structured as follows: Section 2 covers related work, Section 3 presents an overview of the model and our system, and Sections 4, 5, and 6 detail the experiments, key results, and conclusions, respectively.

2 Related Work

2.1 Semantic Parsing

In table question answering (QA) context, semantic parsing refers to converting a natural language question into a formal, structured representation that can be understood and processed by a machine. This structured representation often takes the form of a query (like SQL) that can be executed to retrieve the relevant data from the table like WikiTableQuestions (Pasupat and Liang, 2015), WikiSQL (Zhong et al., 2017), and Spider (Yu et al., 2018). For example, for the question *What is the total sales for Product X in January?*, the semantic parser would convert it into a query: *SELECT*

[1]The code of the paper is available at https://github.com/ywh5/semeval2025-task8

Proceedings of the The 19th International Workshop on Semantic Evaluation (SemEval-2025), pages 2276–2282

SUM(sales) FROM table WHERE product = 'Product X' AND month = 'January';. Semantic parsing in table QA aims to accurately interpret the intent behind the natural language question and maps it to the appropriate data manipulation or retrieval query that can be executed on the structured table. In short, it's about bridging the gap between the natural language used by the user and the structured data format needed for querying.

2.2 Table question answering

Table question answering or question answering on tabular data aims to provide answers to natural language questions from data in tables (Jin et al., 2022), which is a spin-off task of QA. The user's question involves table-based question answering, which focuses on delivering accurate responses by comprehending and reasoning through tables.

Table QA tasks generally originate from querying relational databases using natural language, where the tables are well-structured. This approach addresses the table QA task by employing a semantic parser to convert natural language into a structured query (such as SQL), which is then executed to retrieve the answers (Zhong et al., 2017).

3 System Overview

Our system is designed to generate Python code using the Pandas library to extract information from these datasets.

3.1 Prompt Generator

The prompt generator creates prompts for the LLM, which include three components: (1) a general task description (creating a query to answer a question), (2) examples of questions, table summaries, and expected answers, and (3) the actual question and table summary for the LLM to process. We show a summarized prompt in the follow Listing.

Listing 1: Example of a prompt used in our experiments.

```
Your task is to generate pandas code to
    answer a specific question based on
    a provided table of data.
You will receive a list of column names,
     a DataFrame in JSON format, and the
     question itself.
Select the relevant columns and complete
     the 'answer' function accordingly.
Ensure proper type compatibility in
    aggregate operations, and always
    close expressions before applying
    further operations.
Use 'empty' to check if any columns are
    missing data.
Provide the answer to the last question.
Keep the output straightforward and
    focused on solving the problem.

TODO: complete the following function in
     one line.
Question: How many rows are there in
    this dataframe?
Function:
def example(df: pd.DataFrame) -> int:
    df.columns=["A"]
    return df.shape[0]

TODO: complete the following function in
     one line.
Question: {question}
Function:
def answer(df: pd.DataFrame) -> {row["
    type"]}:
    df.columns = {list(df.columns)}
    return
```

3.2 Model Selection

We conducted empirical experiments to find the best model for code generation. TableGPT2's language framework is built on the Qwen2.5 (Yang et al., 2024) architecture. It undergoes continual pretraining (CPT), supervised fine-tuning (SFT), and an agent framework for production-level ability. These steps set it apart from traditional LLMs, as the pretraining and fine-tuning focus more on coding, multi-turn reasoning, and tool utilization. This approach ensures the model excels in natural language processing and addressing the complex demands of table-related tasks.

TableGPT2 introduces a unique modality module designed specifically for reading and interpreting tabular data. Similar to vision-language models (VLMs), it features a tabular data reading module that generates specialized embeddings, which are then combined with token embeddings from textual input. This module enhances TableGPT2's ability to understand the structure and semantics of tabular data, improving its performance in complex business intelligence scenarios.

The conceptual diagram of the TableGPT2 model revolves around continuous pretraining

(CPT) and supervised fine tuning (SFT).

Continual Pretraining (CPT). CPT focuses on enhancing coding and reasoning abilities. 80% of the CPT (Ke et al., 2023) data was well-commented code, aligned with DeepSeek-v2 (DeepSeek-AI et al., 2024), ensuring strong coding capabilities. The remaining data included domain-specific knowledge to improve reasoning. A two-level filtering strategy was used: categorizing documents with 54 labels for diverse coverage and fine-tuning token selection with the RHO-1 (Lin et al., 2024) technique. Additionally, strategies for code length and context windows were introduced (Kim and Lee, 2024), optimizing model performance for handling different code segments. After filtering, the CPT data comprised 86 billion tokens, boosting the model's coding and reasoning for complex BI tasks.

Supervised Fine-Tuning (SFT). This stage focuses on adapting the model for BI-specific tasks and addressing its prior limitations. The dataset coveres multi-turn conversations, complex reasoning, tool usage, and business-specific queries. It includes 2.36 million samples and billions of tokens. Key areas of specialization includes table-based tasks such as code generation (Wang et al., 2021; Ahmad et al., 2021) (Python, SQL), table querying, data visualization, and predictive modeling. The dataset ensures variety by incorporating different input formats and table metadata combinations. A multi-step filtering pipeline is employed, with rule-based checks for code correctness, anomaly detection, and scoring using models like GPT-4o. Only high-quality samples pass, with manual calibration and validation against a fixed validation set of 94.9K cases.

The overall system framework is shown in Figure 1, which also demonstrates how QA works over tables from a particular industry and how the LLM encodes the tabular data. As shown in the figure, the table encoder takes the entire table as input and generates compact embeddings for each column. This architecture is optimized for the unique properties of tabular data, which differs significantly from text, images, or other data types. The semantics of the table are represented in four key dimensions: cells, rows, columns, and the entire table structure, all of which exhibit permutation invariance. To address this, a bi-dimensional attention mechanism is used without positional embeddings, along with a hierarchical feature extraction process to capture both row-wise and column-wise relationships effectively. Additionally, a column-wise contrastive learning approach is employed to encourage the model to learn meaningful, structure-aware semantic representations of the table.

3.3 Automatic code execution

Our system automatically extracts the Python function from the LLM's response and runs it on the validation and test dataset. The output from the function serves as the system's answer to the query. If an error occurs during execution, the system captures it and sends an updated prompt to the model, including the erroneous code and details about the error.

The overall workflow of the system is as follows:

- **Input:** Our system retrieves the question and loads the relevant DataFrame.
- **Prompt:** The prompt is produced through the utilization of the question and the dataset header.
- **LLM:** Utilize the generated prompt to make a call to the LLM and extract the corresponding answer out of the response.
- **Execution:** The answer function is operated on the DataFrame so as to acquire the answer.
- **Error Handling:** In case the execution fails, the error is caught, and the LLM is prompted once more with the inclusion of the error details.
- **Output:** The final output is to answer the initial question.

The above approach enhances our system's ability to reduce errors and improve entire accuracy.

4 Experimental Settings

4.1 Dataset

There are 65 publicly available datasets summarized in Table 1 and Table 2. Each dataset for this task includes natural language questions, relevant column information, and corresponding answers. Moreover, it has 20 hand-made questions per dataset, with a total number of 1300 questions. Questions are further split in different types depending on the type of answer (i.e., true/false, categories from the dataset, numbers or lists), and their corresponding gold standard answer accompanies each question.

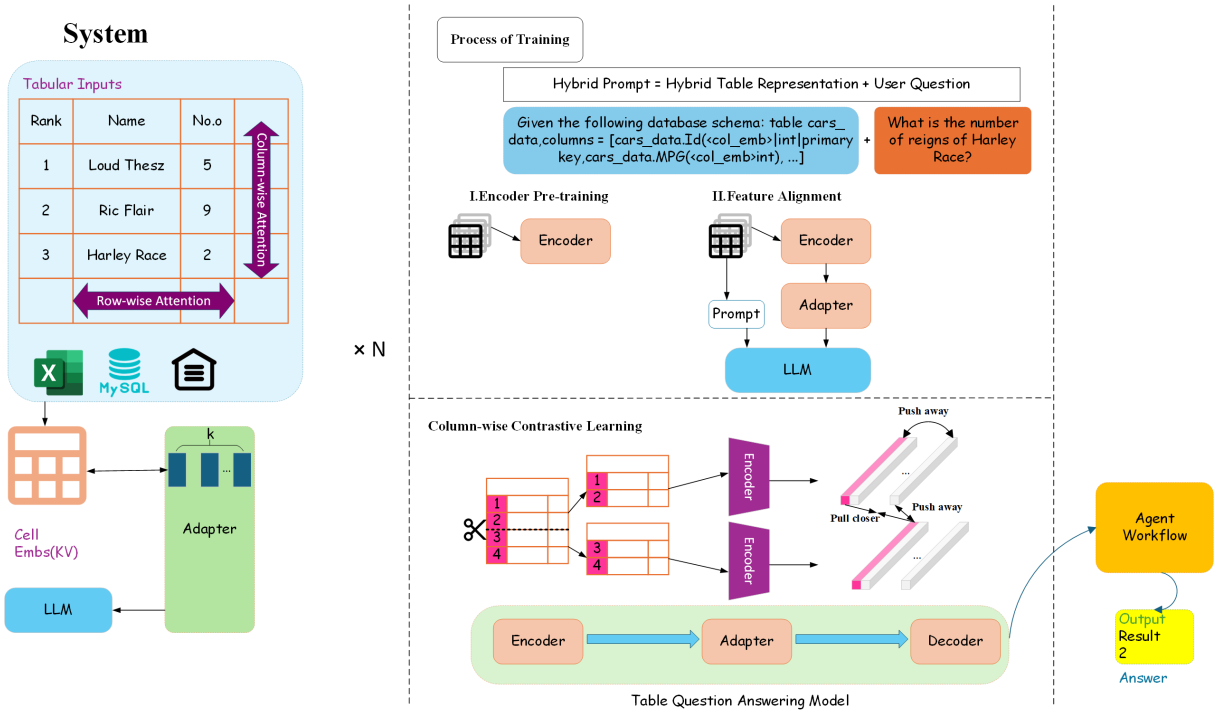

Figure 1: An illustration example of table QA system. The number (2) is the target answer.

Domain	Datasets	Rows	Columns
Business	26	1,156,538	534
Health	7	98,032	123
Social	16	1,189,476	508
Sports	6	389778	177
Travel	10	427,151	273
Total	65	3,269,975	1615

Table 1: DataBench domain taxonomy.

Type	Columns	Example
number	788	583600
category	548	Flat
date	50	1934-02-09
text	46	Such a ...
url	31	apple.com
boolean	18	False
list[number]	14	[21, 14, 13, 11]
list[category]	112	[sales, technical]
list[url]	8	[apple.com, ...]

Table 2: Column types present in DataBench.

4.2 Baselines

In these experiments, we mainly consider the following baseline models.

Stable-Code. Stable-Code-3B (Pinnaparaju et al.) is a 2.7B billion parameter decoder-only language model pre-trained on 1.3 trillion tokens of diverse textual and code datasets. It is trained on 18 programming languages (selected based on the 2023 StackOverflow Developer Survey) and demonstrates state-of-the-art performance (compared to models of similar size) on the MultiPL-E metrics across multiple programming languages tested using BigCode's Evaluation Harness.

CodeLlama. CodeLlama, a specialized version of Llama2 designed for coding tasks (Rozière et al., 2023), excels in coding benchmarks and benefits from a 16K token window. We utilized its instruction models with 7B parameter sizes, emphasizing its ability to generate and understand code.

Deepseek-Coder. Deepseek Coder comprises a series of code language models, each trained from scratch on 2T tokens, with a composition of 87% code and 13% natural language in both English and Chinese. It provides various sizes of the code model, ranging from 1B to 33B versions. Each model is pre-trained on project-level code corpus by employing a window size of 16K and an extra fill-in-the-blank task to support project-level code completion and infilling. For coding capabilities, Deepseek Coder achieves state-of-the-art performance among open-source code models on multiple programming languages and various benchmarks.

4.3 Implementation Details

The deployment of Stable-Code, CodeLlama, Deepseek-Coder, and TableGPT2 used the official checkpoints provided by HuggingFace. The experiments were run on a machine equipped with a single NVIDIA Tesla P100 GPU with 16GB VRAM, only suitable to run these models.

4.4 Evaluation Metrics

The evaluation focuses on accuracy and is further split by question types and other factors. The accuracy is obtained by comparing predicted answers with the correct answers across various domains (such as booleans, categories, numbers, etc.). However, one issue with LLMs is that their responses often lack a consistent formatting pattern. Relaxing the criteria for a correct answer to allow small format variations is an effective solution to address this. For example, answers like "true," "True," or "Yes" will all be considered correct for a boolean question that is meant to be true. This flexibility has a minimal impact on code-based models. Furthermore, for lists, the order of elements is not considered in contrast to the ground truth, which may be important in some cases.

5 Experimental Results

Results on Dev Set. When processing input prompts in batches and generating text, we adjust some parameters to change the diversity and length of the generated text. For example, the parameter *temperature* controls the randomness or creativity of the generated text. When the *temperature* is low, the text generated by the model is more deterministic and tends to choose words with higher probability; when the *temperature* is high, the generated text is more diverse and random. We use two *temperatures* of 0.1 and 0.2. As shown in Table 3, the model performs better when the *temperature* is lower, especially on DataBench Lite. If *do_sample* is set to True, the sampling strategy is enabled. The model will generate the next token based on probability distribution sampling. This method will increase the diversity of generated text. If set to False, the model will use a greedy strategy to select the next token with the highest probability.

Competition Results. Our experimental results on the test set for the task competition are summarized in Figure 2. It can be observed that the TableGPT2-7B model achieves the highest performance scores in all subtasks. Also, we observe a large performance gap between these models, although their number of parameters is similar.

Analysis. As indicated in Figure 2, TableGPT2-7B achieves 0.65% accuracy on the DataBench benchmark and 0.66% accuracy on the DataBench Lite benchmark. The consistency across both benchmarks (DataBench and DataBench Lite) further validates its robustness and generalizability.

When comparing it with other models like the Deepspeek-Coder-6.7B-Base, which shows an accuracy of 0.51% on DataBench and 0.50% on DataBench Lite, the TableGPT2-7B outperforms them by a notable margin, indicating that it benefits from more refined training or a more effective architecture for this type of task. In contrast, the Stable-Code-3B and CodeLlama-7B-hf models exhibit comparatively lower accuracy rates, highlighting the potential advantages of using larger or more specialized models like TableGPT2-7B for similar tasks. The result suggests that further fine-tuning or enhancement of such models could lead to even more significant improvements in performance.

Meanwhile, the results of Deepseek-Coder's three different parameter models (Deepspeek-Coder-1.3B-Base, Deepseek-Coder-5.7Bmqa-Base and Deepspeek-Coder-6.7B-Base) show that the accuracy improves as the model size increases.

The accuracy of Deepspeek-Coder-1.3B-Base on DataBench is 0.40%, and slightly lower on DataBench Lite, 0.39%. The accuracy of Deepspeek-Coder-6.7B-Base is 0.51% and 0.50%, respectively, which is more than 27% higher than the 1.3B-version. Increasing the number of model parameters (from 1.3B to 6.7B) impacts model performance. Generally speaking, larger models can capture more data features and subtle patterns and thus perform better in complex tasks. This suggests that the 6.7B version of Deepseek-Coder may have obtained more contextual information during training and can better cope with the challenges in the task.

However, an important point can also be drawn from this, that is, simply increasing the parameters of the model does not always guarantee unlimited improvement. Taking the Deepspeek-Coder model as an example, when the number of parameters increased from 5.7B to 6.7B, the results do not improve much. After a certain scale, the performance improvement of the model gradually stabilizes; in other words, other factors such as data quality and training strategy are equally important in affecting the model's performance.

Model	Temperature	Do_sample	Accuracy	
			DataBench	DataBench Lite
Stable-Code-3B	0.1	True	0.48	0.47
	0.2	True	0.47	0.44
	0.1	False	0.48	0.48
TableGPT-7B	0.2	True	**0.72**	**0.68**

Table 3: Comparison on different parameters.Evaluation is based on accuracy(%).

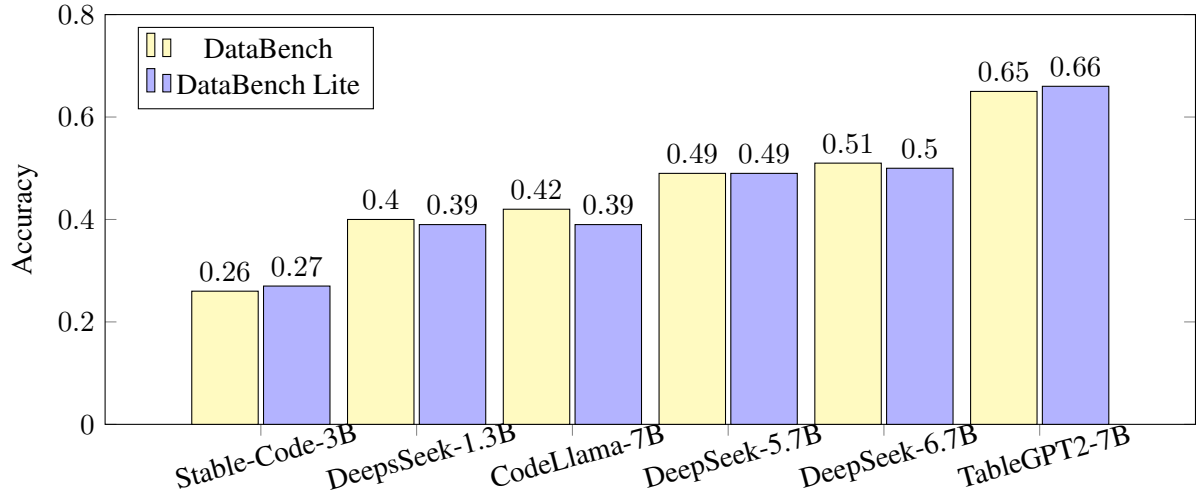

Figure 2: Comparison of model accuracy on DataBench and DataBench Lite.

In general, the increase in parameter scale brings about performance improvement, but this improvement is not endless, so optimizing other factors is also the key to improving model performance.

6 Conclusions

During Task 8 in SemEval 2025, we made prediction for each question provided by final competition. Conducting preliminary exploration on the training set and validation set, we finally decided to use the TableGPT2 model to complete the competition. Due to the powerful table understanding ability of this model, it has an inherent advantage in handling Task 8. Our approach proved to be highly effective by outstanding performance in this task. In future work, we may adjust some basic parameters or explore models with larger parameters to optimize the experimental results (which have been proven to be effective in above work). Thus, our goal is to continue progressing in this area.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant Nos. 61966038 and 62266051. We would like to thank the anonymous reviewers for their constructive comments.

References

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2668. Association for Computational Linguistics.

Wenhu Chen. 2023. Large language models are few(1)-shot table reasoners. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1120–1130. Association for Computational Linguistics.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Wang. 2020. Hybridqa: A dataset of multi-hop question answering over tabular and textual data.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, and 138 others. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model.

Nengzheng Jin, Joanna Siebert, Dongfang Li, and Qingcai Chen. 2022. A survey on table question answering: Recent advances.

Zixuan Ke, Yijia Shao, Haowei Lin, Tatsuya Konishi, Gyuhak Kim, and Bing Liu. 2023. Continual pretraining of language models.

Jisu Kim and Juhwan Lee. 2024. Strategic data ordering: Enhancing large language model performance through curriculum learning.

Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, and Weizhu Chen. 2024. Rho-1: Not all tokens are what you need.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Nick Schoelkopf, Riley Kong, Xiangru Tang, Murori Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, and Dragomir Radev. 2021. Fetaqa: Free-form table question answering.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480. Association for Computational Linguistics.

Nikhil Pinnaparaju, Reshinth Adithyan, Duy Phung, Jonathan Tow, James Baicoianu, and Nathan Cooper. Stable code 3b.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, and 7 others. 2023. Code llama: Open foundation models for code.

Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Gang Chen, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, Haoze Li, Haoxuan Lan, Jiaming Tian, Jing Yuan, Junbo Zhao, Junlin Zhou, Kaizhe Shou, Liangyu Zha, Lin Long, and 14 others. 2024. Tablegpt2: A large multimodal model with tabular data integration.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708. Association for Computational Linguistics.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024. Qwen2 technical report.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921. Association for Computational Linguistics.

Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B. Hashimoto. 2024a. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57.

Wenxuan Zhang, Yue Deng, Bing Liu, Sinno Pan, and Lidong Bing. 2024b. Sentiment analysis in the era of large language models: A reality check. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3881–3906. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance.