Dataground at SemEval-2025 Task 8: Small LLMs and Preference Optimization for Tabular QA

Giuseppe Attardi[3], Andrea Nelson Mauro[2], Daniele Sartiano[1,*]

[1]Institute of Informatics and Telematics, National Research Council,
[2]Dataground srls,
[3]Independent researcher

Correspondence: daniele.sartiano@iit.cnr.it

Abstract

We present our submission to SemEval 2025 Task 8: Question Answering on Tabular Data, which challenges participants to develop systems capable of answering natural language questions on real-world tabular datasets. Our approach aims at generating Pandas code that can be run on such datasets to produce the desired answer. The approach consists in fine-tuning a Small Language Model (SLM) through Preference Optimization on both positive and negative examples generated by a teacher model. A base SLM is first elicited to produce the code to compute the answer to a question through a Chain of Thought (CoT) prompt. We performed extensive testing on the DataBench development set, exploring a variety of prompts, eventually settling on a detailed instruction prompt, followed by two-shot examples. Due to hardware constraints, the base model was an SLM with $\leq$ 8 billion parameters. We then fine-tuned the model through Odds Ratio Preference Optimization (ORPO) using as training data the code produced by a teacher model on the DataBench training set. The teacher model was GPT-4o, whose code was labeled preferred, while the code generated by the base model was rejected. This increased the accuracy on the development set from 71% to 85%. Our method demonstrated robust performance in answering complex questions across diverse datasets, highlighting the effectiveness of combining small LLMs with supervised fine-tuning and automated code execution for tabular question answering.

1 Introduction

The ability of Large Language Models (LLMs) to process structured data and generate meaningful responses has become an increasingly important area of research. The SemEval 2025 Task 8: Question Answering on Tabular Data (Osés Grijalba et al., 2025) focuses on assessing the capacity of LLMs to perform question answering (QA) over tabular datasets using the newly developed DataBench benchmark (Osés Grijalba et al., 2024). Unlike traditional QA tasks that operate on unstructured text, this task requires models to interpret structured data, extract relevant information, and generate precise answers. The challenge lies in the complexity of real-world tabular datasets, which contain diverse column types, varying structures, and sometimes millions of rows. The DataBench benchmark was designed to evaluate LLM performance on this task, featuring 65 real-world datasets spanning multiple domains, with 1300 hand-crafted questions and answers.

Our approach to solving this problem relies on code generation rather than in-context learning. Instead of answering questions directly in natural language, our system generates executable Python code that extracts the relevant information from the dataset to compute the answer. The approach consists of fine-tuning a Small Language Model (SLM) through Reinforcement Learning on both positive and negative examples produced by a teacher model. A base SLM is first elicited to produce the Pandas code to compute the answer to a question through a Chain of Thought (CoT) (Wei et al., 2022) prompt consisting of a detailed instruction and two-shot examples (Brown et al., 2020), which helps guide the model through a step-by-step reasoning process. Due to hardware constraints, we had to resort to a small LLM ($\leq$ 8B parameters). We selected *deepseek-coder-6.7b-instruct* as our base model by comparing several models on the DataBench development set. The supervised fine-tuning (SFT) step uses data generated by a teacher model. In our case the teacher model GPT-o4 was used to generate code on the DataBench training set. We used the Odds Ratio Preference Optimization (ORPO) algorithm (Hong et al., 2024) , supplying to it the responses from our base SLM labeled rejected, while the GPT-4o generated responses labeled preferred.

Proceedings of the The 19th International Workshop on Semantic Evaluation (SemEval-2025), pages 1306–1312

To promote transparency and reproducibility, we released our code and fine-tuned model on GitHub: https://github.com/daniele-sartiano/semeval_2025_task_8. This resource includes our prompt templates, code execution script, and fine-tuning scripts enabling further research into improving LLMs for tabular question answering.

Our findings suggest that combining code generation with preference-based fine-tuning offers a promising direction for enhancing LLM capabilities on tabular QA tasks. Future work may explore hybrid approaches, integrating in-context learning with code generation to leverage the strengths of both methodologies.

2 System Overview

The system is designed to generate Python code using the Pandas[1] library to extract information from real-world datasets. We initially used the baseline[2] example provided by the task organizers and then extended the software by integrating prompt engineering, including Chain of Thought (CoT) reasoning and two-shot learning, model selection, supervised fine-tuning, and automated code execution to effectively solve the challenge task.

2.1 Prompt engineering

We designed a structured prompt incorporating Chain of Thought (CoT) instructions and two-shot learning to guide LLMs in generating accurate Python code for data extraction and analysis. The CoT approach encourages step-by-step reasoning in the generated code, improving the model's ability to handle complex queries. Two-shot learning provides the model with two examples of correctly generated code, which helps it infer the proper structure and logic when tackling new questions. Listing 1 shows an example, although the "list of columns" and the "head of the DataFrame in JSON format" of the two-shot examples are omitted for brevity.

2.2 Model Selection

We conducted empirical experiments to find the best model for code generation. We experimented with small models, with a maximum of 30 billion parameters, including general-purpose and code-specific models. The selection of models was guided by the top-ranked entries on Hugging Face's Big Code Models Leaderboard[3], allowing us to focus on state-of-the-art small language models (SLMs) relevant to our task. Table 1 lists some of the models evaluated.

We assessed their performance on the development set from the DataBench dataset, using similar prompts as described in Section 2.1. The model that achieved the highest accuracy was *deepseek-coder-6.7b-instruct*, which we selected as the baseline for our next experiments, without applying fine-tuning.

2.3 Supervised Fine-Tuning with ORPO

To enhance the performance of the baseline model, we applied supervised fine-tuning (SFT) using the Odds Ratio Preference Optimization (ORPO) algorithm. ORPO introduces a loss function that combines the standard negative log-likelihood (NLL) loss with a term based on the log odds ratio. This term effectively contrasts preferred (chosen) responses with less preferred (rejected) ones, guiding the model toward generating outputs that align more closely with human preferences. Unlike Proximal Policy Optimization (PPO) (Schulman et al., 2017), which requires a multi-stage pipeline involving reward modeling and reinforcement learning, ORPO simplifies the process by integrating preference optimization directly into the fine-tuning phase. Similarly, while Direct Preference Optimization (DPO) (Rafailov et al., 2023) aligns preferences with a reference model, ORPO completely eliminates the need for such a model. This streamlined approach reduces computational complexity while also enhancing both the quality and relevance of the generated responses.

Initially, we attempted to generate a fine-tuning dataset by pairing correct answers from the baseline model with incorrect answers, which were generated via a structured prompt. To generate incorrect responses, we exploited the prompt shown in Listing 2.

[1] https://pandas.pydata.org/

[2] https://github.com/jorses/databench_eval/blob/main/examples/competition_baseline.py

[3] https://huggingface.co/spaces/bigcode/bigcode-models-leaderboard

```
Your task is to generate code using pandas to answer a question on a table of data.
You will be provided with a list of table columns, a dataframe in json format and a question.
Choose the relevant information from the table columns and complete the code of function `answer`
    below.
Ensure using compatible types in aggregate comparisons.
Ensure to close expressions before applying further operators.
Use empty to check if there are columns that do not contain any elements.
The output must be concise and directly solve the problem.

Table columns: <list of columns>
Dataframe: <head of the dataframe in json format>
Question: Is the most favorited author mainly communicating in Spanish?
Function:
def answer(df: pd.DataFrame):
    return df[df['author_id'] == df.groupby('author_id')['favorites'].sum().idxmax()]['lang'].mode()
        [0] == 'es'

Table columns: <list of columns>
Dataframe: <head of the dataframe in json format>
Question: Is there a patent containing the word 'method in the title?
Function:
def answer(df: pd.DataFrame):
    return df['title'].str.lower().str.contains('method').any()

Table columns: {{df.columns.to_list()}}
Dataframe: {{df.head().to_json(orient='records')}}
Question: {{question}}
Function:
def answer(df: pd.DataFrame):
```

Listing 1: An example of the prompt where the "list of columns" and the "head of the DataFrame in JSON format" for the two-shot examples are omitted for brevity.

Model	**DataBench**	**DataBench lite**
Mistral-7B-Instruct-v0.3	0.496875	0.528125
Nxcode-CQ-7B-orpo	0.6	0.596875
CodeQwen1.5-7B-Chat	0.625	0.615625
Qwen2.5-Coder-32B-Instruct	0.68125 2	0.68125
OpenCodeInterpreter-DS-6.7B	0.625	0.59375
deepseek-coder-6.7b-instruct	**0.7125**	**0.703125**
DeepSeek-R1-Distill-Llama-8B	0.321875	0.371875

Table 1: Model selection using base SML (without fine tuning).

```
Modify the following Python instruction to
    return an incorrect value for the question:
    '{question}'.
Create an alternative version with the main
    instruction altered, so that it returns an
    incorrect value.
The error can be simple or non-trivial, but must
     remain in one line. Only use pandas and
    numpy.
Do not include any additional text or
    explanations. Write minimum 5 samples.
Respond with only the modified code in the
    following format:

<code>{instruction}</code>
```

Listing 2: The prompt used to create rejected samples.

However, this approach resulted in limited diversity and effectiveness of the fine-tuning dataset. Consequently, we adopted an alternative strategy that leverages GPT-4o (OpenAI, 2024) as a teacher to generate possibly better code as preferred examples. In this revised approach, responses from the baseline LLM were labeled as "rejected," while GPT-4o outputs were labeled as "chosen". This process led to the creation of a preference dataset consisting of 1,079 triples in JSON format: prompt, rejected, and chosen.

This preference-based fine-tuning significantly improved the model's ability to generate more accurate and contextually appropriate code. After fine-tuning, we observed a notable improvement in accuracy on the development set, demonstrat-

ing the effectiveness of ORPO in enhancing small LLMs for code generation tasks. Specifically, the fine-tuning process led to an approximate 6% improvement as detailed in the Experimental Setup Section 3.

2.4 Automated code Execution and Error Handling

Our script automatically extracts the Python function from the response generated by the LLM and executes it on the test dataset. The result of this function is our system's answer to the question. Whenever an error occurs during execution, the system catches the error and submits an extended prompt to the model that includes the wrong generated code and info about the error it caused. This iterative error-handling mechanism enables the model to correct its mistakes.

The overall workflow of the system, also shown in Figure 1, is as follows:

- **Input**: The system retrieves the question and loads the corresponding DataFrame.
- **Prompt**: The prompt is generated using the question and the head of the dataset.
- **LLM**: The SLM is invoked with the generated prompt, and the answer function is extracted from the response.
- **Execution**: The answer function is run on the DataFrame to obtain the answer.
- **Error Handling**: If the execution fails, the error is caught, and the LLM is prompted again, including the error details.
- **Output**: The final output is the answer to the initial question.

This approach enhances our system's ability to reduce errors and improve overall accuracy.

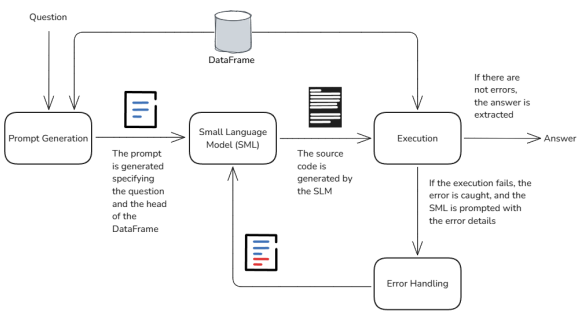

Figure 1: The overall workflow of the system.

3 Experimental Setup

We conducted our experiments on a machine equipped with a single NVIDIA A100 GPU with 80GB VRAM, only suitable to run Small Language Models. We used the Hugging Face Transformers library to interact with the models, retrieving all models from the HuggingFace Hub.

For evaluation, we used the *databench_eval*[4] library provided by the task organizers. The DataBench QA dataset, specifically the development split, which consists of 320 questions, was used in our experiments to validate and optimize our system, measuring the accuracy of several variants and refining the models iteratively.

To fine-tune the models, we used the train split of the DataBench QA dataset, which consists of 988 questions. This dataset was leveraged to generate a preference-based fine-tuning dataset by invoking the GPT-4o model via the OpenAI API. Using the prompt specified in Listing 2, we generated answers, executed the Python functions, and compared their outputs with the ground truth answers from the trainset.

To construct the fine-tuning dataset, we identified the correctly executed answers and paired them with responses generated by the base model selected for fine-tuning. The best performing model at that stage, *deepseek-coder-6.7b-instruct*, was chosen for this process (as shown in Table 1). This process resulted in a fine-tuning dataset containing 805 samples, formatted as JSON records with the structure shown in Listing 3.

```
{
  "prompt": "You are a pandas code generator...
      Question: What's the rank of the
      wealthiest non-self-made billionaire?\
      nFunction:\ndef answer(df: pd.DataFrame):\
      n"",
  "rejected": "import pandas as pd\nimport numpy
       as np\n\ndef answer(df): \treturn df[(df
      ['selfMade'] == False) & (df['finalWorth']
       >= 10**9)]['rank'].min()",
  "chosen": "def answer(df: pd.DataFrame):\n
      return df[df['selfMade'] == False].
      nlargest(1, 'finalWorth')['rank'].iloc[0]"
}
```

Listing 3: An example of one entry of the fine tuning dataset.

ORPO fine-tuning was performed using its implementation from the HuggingFace libraries TRL - Transformer Reinforcement Learning[5] and the

[4] https://github.com/jorses/databench_eval
[5] https://huggingface.co/docs/trl/index

PEFT Parameter-Efficient Fine-Tuning.[6] The hyperparameters used are those listed in the Table 2. We obtained the best results with a maximum of 1000 steps. We also experimented with higher step counts, such as 10,000 and 30,000, but the results on the development set were worse.

Hyperparameter	Value
Batch Size (per device)	2
Max Steps	1,000
Learning Rate	8e-5
Gradient Accumulation Steps	1
Evaluation Steps	500
Optimizer	RMSProp
Warmup Steps	150
LoRA Rank ($lora\_r$)	16
LoRA Alpha ($lora\_alpha$)	16
Max Prompt Length	320

Table 2: Hyperparameters Used in Fine-Tuning

The fine-tuning process, combined with the CoT two-shot prompt resulted in a significant performance improvement. Specifically, accuracy increased from 71.25% to 85.00% on the DataBench QA development dataset and from 70.31% to 80.31% on DataBench Lite QA development dataset. These improvements show the effectiveness of distillation through SFT with a teacher model and Preference Optimization. It was this configuration that we used in our best final submission.

4 Results

Our submission was evaluated by the SemEval 2025 Task 8 organizers using the official *databench_eval* Python script. The final test set originally contained 522 questions, but after identifying errors in the ground truth, corrections were made, and 5 ambiguous questions were removed. These questions were excluded from the final evaluation resulting in a final evaluation set of 517 questions.

In the General ranking, which includes both large and small LMs, as well as both proprietary and open-source models, our submission achieved 31st place in the DataBench QA subtask with an accuracy of 68.97% and 29th place in DataBench Lite QA subtask with an accuracy of 69.35%. This represents an improvement of approximately 43 percentage points over the baseline. In the separate ranking category for Small models ($\leq$ 8B parameters), we achieved second place in both subtasks as shown in Table 3 and in Table 4.

Ranking	Team Name	Score
1	ScottyPoseidon	76.63
2	**Dataground**	**68.97**
3	NexGenius	65.64
4	Tree-Search	64.56
5	Basharat Ali	43.10
-	Baseline	26.00

Table 3: Top 5 final ranking results for DataBench in the small category.

Ranking	Team Name	Score (Lite)
1	ScottyPoseidon	74.71
2	**Dataground**	**69.35**
3	NexGenius	66.22
4	Tree-Search	64.94
5	Basharat Ali	43.87
-	Baseline	27.00

Table 4: Top 5 final ranking results for DataBench Lite in the small category.

After the end of the challenge, we experimented also with a larger model, *deepseek-coder-33b-instruct*, and achieved an accuracy of 72.4% on the test set, using the same prompt as our submission but without SFT.

5 Conclusion

In this paper, we described our submission to SemEval-2025 Task 8: Question Answering on Tabular Data, using a code generation approach, rather than in-context learning with a LLM. We exploit and tune a SML to generate code that extracts the answers from structured datasets. By leveraging a small LLM ($\leq$8B parameters), prompt engineering, and supervised fine-tuning from a teacher model with the Odds Ratio Preference Optimization (ORPO) algorithm, we significantly improved on the baseline model accuracy.

Our experimental results show that:

- Fine-tuning from a large teacher model using ORPO alone improved accuracy by approximately 6%, highlighting the effectiveness of preference optimization.
- Prompt engineering played a crucial role, with structured two-shot learning and Chain of

[6] https://huggingface.co/docs/peft/index

Thought (CoT) reasoning yielding significant performance gains.

- Error handling mechanisms helped refine model outputs, reducing execution failures and increasing robustness.
- Compared to the challenge baseline, our model improved performance by over 43 percentage points on the official test set.

In the official evaluation, our system achieved second place in both tasks, in the category Small Models ($\leq$8B parameters), demonstrating the effectiveness of fine-tuning through preference optimization. Despite the hardware limitations that constrained our experiments, we were able to achieve competitive results. Our results confirm in particular the benefits of distilling the knowledge of a LLM into a smaller model, as reported for the reasoning capabilities of DeepSeek R1 in (DeepSeek-AI et al., 2025) We hope to be able to further explore the distillation technique using a LLM more specialized on coding than the one we could use.

Acknowledgments

We gratefully acknowledge Marco Aldinucci and Sergio Rabellino of the University of Turin for providing us access to a server with NVIDIA H100 GPUs. We also thank Seeweb srl for providing access to a server equipped with four NVIDIA H100 GPUs.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, 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, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. Orpo: Monolithic preference optimization without reference model. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11170–11189.

OpenAI. 2024. Hello GPT-4o. `https://openai.com/index/hello-gpt-4o/`. [Online; accessed 20-February-2025].

Jorge Osés Grijalba, L. Alfonso Ureña-López, Eugenio Martínez Cámara, and Jose Camacho-Collados. 2024. Question answering over tabular data with DataBench: A large-scale empirical evaluation of LLMs. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-*

COLING 2024), pages 13471–13488, Torino, Italia. ELRA and ICCL.

Jorge Osés Grijalba, L. Alfonso Ureña-López, Eugenio Martínez Cámara, and Jose Camacho-Collados. 2025. Semeval-2025 task 8: Question answering over tabular data. In *Proceedings of the 19th International Workshop on Semantic Evaluation (SemEval-2025)*, pages 1015–1022, Vienna, Austria. Association for Computational Linguistics.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.